Denotational Semantics

Meven Lennon-Bertrand

Lectures for Part II CST 2023/2024

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**: tools for the specification, development, analysis and verification of software and hardware systems.

What is this course about?

- **Formal methods**: tools for the specification, development, analysis and verification of software and hardware systems.
- **Programming language theory**: how to design, implement and reason about programming languages?

What is this course about?

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

What is this course about?

- Formal methods: tools for the specification, development, analysis and verification of software and hardware systems.
- Programming language theory: how to design, implement and reason 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 should we care?

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

Why should we care?

- **Insight**: exposes the mathematical "essence" of programming language concepts.
- **Language design**: feedback from semantic concepts (monads, algebraic effects & effect handlers...).

Why should we care?

- **Insight**: exposes the mathematical "essence" of programming language concepts.
- **Language design**: feedback from semantic concepts (monads, algebraic effects & effect handlers...).
- **Rigour**: semantics is necessary to specify/justify formal methods (compilers, type systems, code analysis, certification...).

- Operational
- Axiomatic
- Denotational

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

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

- **Operational**: meaning of a program in terms of the *steps of computation* it takes during execution (see Part IB Semantics).
- **Axiomatic**: indirect meaning of a program in terms of a *program logic* to reason about its properties (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

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

Denotational semantics in a nutshell

$$
\begin{array}{rcl}
\text{Syntax} & \xrightarrow{[\![-]\!]} & \text{Semantics} \\
\text{Program } P & \mapsto & \text{Denotation } [\![P]\!] \\
\\
\text{Recursive program} & \mapsto & \text{Partial recursive function} \\
\text{Boolean circuit} & \mapsto & \text{Boolean function} \\
& \cdots & \\
\text{Type} & \mapsto & \text{Domain} \\
\text{Program} & \mapsto & \text{Continuous functions between domains}
\end{array}
$$

Properties of denotational semantics

Abstraction

- mathematical object, implementation/machine independent;
- captures the abstract essence of programming language concepts;
- should relate to practical implementations, though...

Properties of denotational semantics

Abstraction

- mathematical object, implementation/machine independent;
- captures the abstract essence of programming language concepts;
- should relate to practical implementations, though...

Compositionality

- The denotation of a phrase is defined using the *denotation* of its sub-phrases.
- $[\![P]\!]$ represents the contribution of P to *any* program containing P.
- Much more flexible than whole-program semantics.

Introduction

A basic example

Commands

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

IMP syntax

Commands

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

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

Arithmetic expressions

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

Commands

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

ranges over *integers* →

Arithmetic expressions

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

Commands

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

Arithmetic expressions

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

Boolean expressions

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

Commands

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

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

where

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

$$\begin{aligned} \mathcal{A} &: \quad \mathbf{Aexp} \to \mathbb{Z} \\ \mathcal{B} &: \quad \mathbf{Bexp} \to \mathbb{B} \end{aligned}$$

where

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

Arithmetic expressions?

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

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

Arithmetic expressions?

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

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

$$\mathcal{A}[\![L]\!] = ???$$

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

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

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

where

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

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

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

where $\rightharpoonup$ denotes partial functions and

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

Semantics of arithmetic expressions

$$
\begin{aligned}
\mathcal{A}\llbracket \underline{n} \rrbracket &= \lambda s \in \text{State}.\ n \\
\mathcal{A}\llbracket A_1 + A_2 \rrbracket &= \lambda s \in \text{State}.\ \mathcal{A}\llbracket A_1 \rrbracket (s) + \mathcal{A}\llbracket A_2 \rrbracket (s)
\end{aligned}
$$

Semantics of arithmetic expressions

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

Semantics of boolean expressions

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

Semantics of commands

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

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

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

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

Semantics of commands

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

This is compositionality!

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

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

Semantics of commands

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

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

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

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

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

Semantics of commands

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

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:

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

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

Loop as a fixpoint

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

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

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

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

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

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

Our occupation for the next few lectures...

Introduction

A taste of domain theory

An example

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

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

should be some w such that:

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

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

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 $F : D \to D$, where D is $(\text{State} \rightharpoonup \text{State})$:

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

The poset of partial functions

Partial order $\sqsubseteq$ on D ($= \mathrm{State} \rightharpoonup \mathrm{State}$):

$w \sqsubseteq w'$ if for all $s \in \mathrm{State}$, if w is defined at s then so is w' and moreover $w(s) = w'(s)$.

if the graph of w is included in the graph of w'.

Partial order $\sqsubseteq$ on D ($= \mathrm{State} \rightharpoonup \mathrm{State}$):

$w \sqsubseteq w'$ if for all $s \in \mathrm{State}$, if w is defined at s then so is w' and moreover $w(s) = w'(s)$.

if the graph of w is included in the graph of w'.

Least element $\perp \in D$:

$\perp$ = totally undefined partial function

= partial function with empty graph

Approximating the 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 fixed point

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

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

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

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

Approximating the fixed point

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

$$w_3[X \mapsto x, Y \mapsto y] = F(w_2)[X \mapsto x, Y \mapsto y] = \begin{cases} [X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\ [X \mapsto 0, Y \mapsto y] & \text{if } x = 1 \\ [X \mapsto 0, Y \mapsto 2y] & \text{if } x = 2 \\ \text{undefined} & \text{if } x \geq 3 \end{cases}$$

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 $\begin{cases} w_0 & = \bot \\ w_{n+1} & = F(w_n) \end{cases}$.

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

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

Approximating the fixed point

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

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

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

Approximating the 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

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

We have our semantics

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

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

We have our semantics

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

We have our semantics

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

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

$$= w_\infty[X \mapsto x, Y \mapsto y]$$

- w_∞ is a fixed point
- which moreover agrees with the operational semantics (!)

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

$$\text{REFL}\ \frac{}{x \sqsubseteq x} \qquad \text{TRANS}\ \frac{x \sqsubseteq y \qquad y \sqsubseteq z}{x \sqsubseteq z} \qquad \text{ASYM}\ \frac{x \sqsubseteq y \qquad y \sqsubseteq x}{x = y}$$

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 $\mathrm{graph}(f) \subseteq \mathrm{graph}(g)$.

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 $\bot$.

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

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 $\bot$.

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

Pre-fixed point

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

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

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

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:

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

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

The least pre-fixed point is a fixed point.

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

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

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

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

$$\text{ASYM}\ \frac{\text{LFP-FIX}\ \dfrac{}{f(\mathrm{fix}(f)) \sqsubseteq \mathrm{fix}(f)} \qquad \dfrac{}{\mathrm{fix}(f) \sqsubseteq f(\mathrm{fix}(f))}}{f(\mathrm{fix}(f)) = \mathrm{fix}(f)}$$

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

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

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

Least Fixed Points

Least upper bounds

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:

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

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, $\bigsqcup_n d = d$.

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).

Diagonalisation

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

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

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

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

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

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

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

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

Moreover, again assuming they 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.

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!

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 domain is a cpo with a least element $\bot$.

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

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

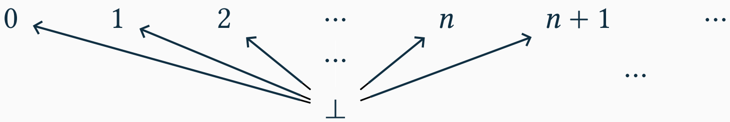
$$f(x) = \begin{cases} f_n(x) & \text{if } x \in \mathbf{dom}(f_n) \text{ for some } n \\ \text{undefined} & \text{otherwise} \end{cases}$$

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

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

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


0
1
2
...
n
n + 1
...
...
...
⊥

Least Fixed Points

Continuous functions

Given two cpos D and E, a function $f\colon D \to E$ 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

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

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

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

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

All computable functions are continuous.

Thesis

All computable functions are continuous.

All computable functions are continuous.

The typical non-continuous function: “is a sequence the constant $\mathbf{0}$”?

$$
\begin{array}{llllll}
0 & 0 & \bot & \dots & & \mapsto \bot \\
0 & 0 & 0 & 0 & 1 \quad \dots & \mapsto 1
\end{array}
$$

$$
0 \quad 0 \quad 0 \quad 0 \quad 0 \quad \overline{0} \qquad \mapsto 0
$$

All computable functions are continuous.

The typical non-continuous function: “is a sequence the constant $\mathbf{0}$”?

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

All computable functions are continuous.

The typical non-continuous function: “is a sequence the constant $\mathbf{0}$”?

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

All computable functions are continuous.

The typical non-continuous function: “is a sequence the constant 0”?

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

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

All computable functions are continuous.

The typical non-continuous function: “is a sequence the constant $\mathbf{0}$”?

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

Intuition: non-continuity ≈ “jump at infinity” ≈ non-computability

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

Least Fixed Points

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

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

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

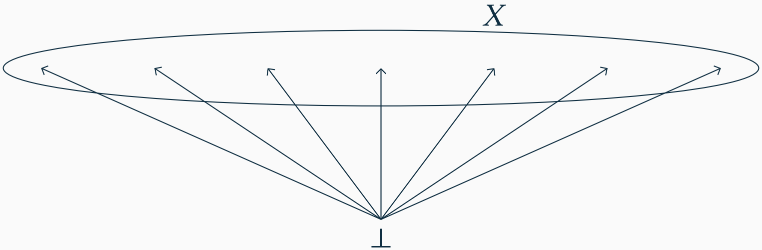
Constructions on Domains

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 $x = x'$.



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 continuous function between the corresponding flat domains.

Constructions on Domains

Products of domains

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

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

Binary product

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

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

and partial order $\sqsubseteq$ defined by

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

$$\text{POX}\ \frac{d_1 \sqsubseteq_1 d_1' \qquad d_2 \sqsubseteq_2 d_2'}{(d_1, d_2) \sqsubseteq (d_1', d_2')}$$

Componentwise lubs and least elements

lubs of chains are computed componentwise:

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

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

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

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:

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

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

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

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

Let D_1 and D_2 be cpos. The projections

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

are continuous functions.

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.

The **conditional** function

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

is continuous.

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 $D_{\mathrm{true}} \times D_{\mathrm{false}}$.

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 $p(i) \in X_i$.

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

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

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

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 $\prod_{i \in I} D_i$;

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

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

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:

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

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:

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

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

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

$$\begin{array}{rccc} \mathrm{eval} : & (D \to E) \times D & \to & E \\ & (f, d) & \mapsto & f(d) \end{array}$$

$$\begin{array}{rccc} \mathrm{cur}(f) : & D' & \to & (D \to E) \\ & d' & \mapsto & \lambda d \in D.\ f(d', d) \end{array}$$

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

are all well-defined and continuous.

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

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:

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

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

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 stable under f, *i.e.* $f(S) \subseteq S$,

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

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

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 stable under f, *i.e.* $f(S) \subseteq S$,

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

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

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

All the following are chain-closed:

Building chain-closed sets

All the following are chain-closed:

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

Building chain-closed sets

All the following are chain-closed:

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

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

All the following are chain-closed:

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

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

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

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 $f : D \to E$ 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

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:

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

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

Terms and types

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:

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

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

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

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

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

Typing for PCF (II)

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

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

Typing for PCF (II)

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

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

$$\mathrm{PCF}_{\Gamma,\tau} \overset{\text{def}}{=} \{t \mid \Gamma \vdash t : \tau\} \qquad\qquad \mathrm{PCF}_{\tau} \overset{\text{def}}{=} \mathrm{PCF}_{\cdot,\tau}$$

PCF

Operational Semantics

PCF evaluation

Values:

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

PCF evaluation

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

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

PCF evaluation

Values:

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

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

PCF evaluation

Values: $$v ::= \underbrace{\texttt{0} \mid \texttt{succ}(v)}_{\underline{n}} \mid \texttt{true} \mid \texttt{false} \mid \texttt{fun}\, x{:}\,\tau.\, t$$

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

$$\textsc{ZeroZ}\ \frac{t \Downarrow_{\texttt{nat}} \texttt{0}}{\texttt{zero?}(t) \Downarrow_{\texttt{bool}} \texttt{true}} \qquad \ldots \qquad \textsc{IfT}\ \frac{b \Downarrow_{\texttt{bool}} \texttt{true} \qquad t_1 \Downarrow_\tau v}{\texttt{if}\ b\ \texttt{then}\ t_1\ \texttt{else}\ t_2 \Downarrow_\tau v} \qquad \ldots$$

PCF EVALUATION

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

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

$$\textsc{ZeroZ}\ \frac{t \Downarrow_{\mathtt{nat}} \mathtt{0}}{\mathtt{zero?}(t) \Downarrow_{\mathtt{bool}} \mathtt{true}} \qquad \dots \qquad \textsc{IfT}\ \frac{b \Downarrow_{\mathtt{bool}} \mathtt{true} \qquad t_1 \Downarrow_\tau v}{\mathtt{if}\ b\ \mathtt{then}\ t_1\ \mathtt{else}\ t_2 \Downarrow_\tau v} \qquad \dots$$

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

PCF evaluation

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

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

$$\textsc{ZeroZ}\ \frac{t \Downarrow_{\mathtt{nat}} \mathtt{0}}{\mathtt{zero?}(t) \Downarrow_{\mathtt{bool}} \mathtt{true}} \qquad \dots \qquad \textsc{IfT}\ \frac{b \Downarrow_{\mathtt{bool}} \mathtt{true} \qquad t_1 \Downarrow_\tau v}{\mathtt{if}\ b\ \mathtt{then}\ t_1\ \mathtt{else}\ t_2 \Downarrow_\tau v} \qquad \dots$$

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

Alternatively: small-step $t \rightsquigarrow_\tau u$, we have $t \Downarrow_\tau v$ iff $t \rightsquigarrow_\tau^\star u$.

Examples

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

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

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

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

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

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

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

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

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

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

Try it out!

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

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

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

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

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 $\Downarrow$:

$$\{(t, \tau, v) \mid t \Downarrow_\tau v \wedge \forall v'.(t \Downarrow_\tau v' \Rightarrow v = v')\}$$

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.

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

Evaluation contexts

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

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

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

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

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

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 $\cdot \vdash t \cong_{\mathrm{ctx}} t' : \tau$.

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.

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.

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 $t \Downarrow_\gamma v$.

Proof principle: to show

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

it suffices to establish

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

The power of denotational semantics

Proof principle: to show

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

it suffices to establish

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

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

Proof principle: to show

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

it suffices to establish

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

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

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

Proof principle: to show

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

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

Denotational Semantics for PCF

Types and contexts

Semantics of types

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

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

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

Semantics of contexts

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

Semantics of contexts

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

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

Denotational Semantics for PCF

Terms

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,

$$[\![-]\!] : \mathrm{PCF}_{\Gamma,\tau} \to [\![\Gamma]\!] \to [\![\tau]\!]$$

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

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

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

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

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

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

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

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

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

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

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

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

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

$$
[\![\texttt{succ}(t)]\!] = \text{succ}_\bot \circ [\![t]\!]
$$

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

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

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

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

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

Denotation of the λ-calculus operations

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

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

Denotation of the λ-calculus operations

$$\begin{aligned} [\![x]\!](\rho) &\stackrel{\text{def}}{=} \rho(x) && \in [\![\Gamma(x)]\!] \\ [\![t_1\ t_2]\!](\rho) &\stackrel{\text{def}}{=} ([\![t_1]\!](\rho))\ ([\![t_2]\!](\rho)) \end{aligned}$$

$$[\![t_1\ t_2]\!] = \mathrm{eval} \circ \langle [\![t_1]\!], [\![t_2]\!] \rangle$$

Denotation of the λ-calculus operations

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

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

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

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 $\llbracket t \rrbracket$
is well-defined and a continuous function $\llbracket t \rrbracket : \llbracket \Gamma \rrbracket \to \tau$.

If $t \in \mathrm{PCF}_\tau$: $\quad \llbracket t \rrbracket \quad \in \quad \llbracket \cdot \rrbracket \to \llbracket \tau \rrbracket \quad = \quad \mathbb{1} \to \llbracket \tau \rrbracket \quad \cong \quad \llbracket \tau \rrbracket$

Denotational Semantics for PCF

Compositionality

Suppose $t, u \in \mathrm{PCF}_{\Gamma,\tau}$, such that

$$[\![t]\!] = [\![u]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$$

Suppose moreover that $\mathcal{C}[-]$ is a PCF context such that $\Gamma' \vdash_{\Gamma,\tau} \mathcal{C} : \tau'$. Then

$$[\![\mathcal{C}[t]]\!] = [\![\mathcal{C}[u]]\!] : [\![\Gamma']\!] \to [\![\tau']\!] \,.$$

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

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

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 $\Gamma \vdash_{\Delta,\sigma} \mathcal{C} : \tau$, 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

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

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

Relating Denotational and Operational Semantics

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

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

but

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

Relating Denotational and Operational Semantics

Formal approximation relation

How to prove adequacy

Proof idea: introduce a relation R such that

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

How to prove adequacy

Proof idea: introduce a relation R such that

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

Assume $t, v \in \mathrm{PCF}_{\mathtt{nat}}$, $\llbracket t \rrbracket = \llbracket v \rrbracket$, and v is a value.

Proof idea: introduce a relation R such that

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

Assume $t, v \in \mathrm{PCF}_{\mathtt{nat}}$, $\llbracket t \rrbracket = \llbracket v \rrbracket$, and v is a value.

Thus $v = \underline{n}$ for some $n \in \mathbb{N}$, and $\llbracket v \rrbracket = n$.

Proof idea: introduce a relation R such that

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

Assume $t, v \in \mathrm{PCF}_{\mathtt{nat}}$, $\llbracket t \rrbracket = \llbracket v \rrbracket$, and v is a value.

Thus $v = \underline{n}$ for some $n \in \mathbb{N}$, and $\llbracket v \rrbracket = n$.

$$\begin{aligned} \llbracket t \rrbracket = \llbracket \underline{n} \rrbracket &= n \\ &\Rightarrow R(n, t) \\ &\Rightarrow t \Downarrow \underline{n} = v \end{aligned}$$

Proof idea: introduce a relation R such that

1. if $t \in \mathrm{PCF}_{\mathtt{nat}}$, $n \in \mathbb{N}$, and $R(n, t)$, then $t \Downarrow_{\gamma} \underline{n}$ (same for booleans);
2. for any well-typed term t, $R([\![t]\!], t)$;

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

Proof idea: introduce a relation R such that

1. if $t \in \mathrm{PCF}_{\mathtt{nat}}$, $n \in \mathbb{N}$, 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.

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

$$\lhd_\tau \subseteq \llbracket \tau \rrbracket \times \mathrm{PCF}_\tau$$

Formal approximation at base types

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

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

Formal approximation at base types

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

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

Exactly what we need to get 1.

Formal approximation at base types

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

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

Exactly what we need to get 1.

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

Formal approximation at function types

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

Formal approximation at function types

1. if $t \in \mathrm{PCF}_{\mathtt{nat}}$, $n \in \mathbb{N}$, 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 $t \in \mathrm{PCF}_{\mathtt{nat}}$, $n \in \mathbb{N}$, 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 $t \in \mathrm{PCF}_{\mathtt{nat}}$, $n \in \mathbb{N}$, 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.

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

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

Formal approximation at function types

1. if $t \in \mathrm{PCF}_{\mathtt{nat}}$, $n \in \mathbb{N}$, 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 $[\![t\ u]\!] = [\![t]\!]([\![u]\!]) \lhd_\tau t\ u$?

Define

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

Formal approximation for open terms

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

To prove Item 2, we need to talk about 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.

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

Semantic application ≈ syntactic substitution

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

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

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

Fundamental property of formal approximation

Given a term t such that $\Gamma \vdash t : \tau$ for some Γ and τ, for any environment ρ and substitution σ such that $\rho \lhd_\Gamma \sigma$, we have $[\![t]\!]\,(\rho) \lhd_\tau t[\sigma]$.

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.

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

Fundamental property of formal approximation

Given a term t such that $\Gamma \vdash t : \tau$ for some Γ and τ, for any environment ρ and **substitution** σ such that $\rho \lhd_\Gamma \sigma$, we have $[\![t]\!]\,(\rho) \lhd_\tau t[\sigma]$.

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

Relating Denotational and Operational Semantics

Proof of the fundamental property of formal approximation

Properties of formal approximation

1. The least element approximates any program: for any τ and $t \in \mathrm{PCF}_\tau$, $\bot_{[\![\tau]\!]} \lhd_\tau t$;

2. the set $\{d \in [\![\tau]\!] \mid d \lhd_\tau t\}$ is chain-closed;

1. The least element approximates any program: for any τ and $t \in \mathrm{PCF}_\tau$, $\perp_{[\![\tau]\!]} \lhd_\tau t$;

2. the set $\{d \in [\![\tau]\!] \mid d \lhd_\tau t\}$ is chain-closed;

3. if $\forall v.\, t \Downarrow_\tau v \Rightarrow t' \Downarrow_\tau v$, and $d \lhd_\tau t$, then $d \lhd_\tau t'$.

Relating Denotational and Operational Semantics

Extensionality

Characterizing formal approximation

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,

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

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

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,

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

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

It **corresponds to formal approximation**: for all PCF types τ and closed terms $t_1, t_2 \in \mathrm{PCF}_\tau$

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

For contextual preorder between closed terms, applicative contexts are enough.

For contextual preorder between closed terms, applicative contexts are enough.

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

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

Extensionality properties of contextual preorder

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

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

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

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

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

$$\forall t \in \mathrm{PCF}_\tau \,.\, (t_1\, t \leq_{\mathrm{ctx}} t_2\, t : \tau').$$

Full abstraction

Full abstraction

Failure of full abstraction

Full abstraction

A denotational model is **fully abstract** if

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

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	$\perp$	$\perp$

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

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

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	$\bot$
true	true	true	true
false	true	false	$\bot$
$\bot$	true	$\bot$	$\bot$

or	true	false	$\bot$
true	true	true	true
false	true	false	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$

por	true	false	$\bot$
true	true	true	true
false	true	false	$\bot$
$\bot$	true	$\bot$	$\bot$

or	true	false	$\bot$
true	true	true	true
false	true	false	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$

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 $T_{\mathtt{false}}$,

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

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

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

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

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?

PCF+por

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

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

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

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

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

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.

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 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
- ...

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

Categorical semantics

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:

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: $T(A) \stackrel{\text{def}}{=} (A \times \mathrm{State})^{\mathrm{State}}$)

Beyond pure languages

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

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

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

More semantics

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

More semantics

Easter: axiomatic semantic (Hoare Logic and Model Checking)

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