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

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**: indirect meaning of a program in terms of a *program logic* to reason about its properties (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**: 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

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

Denotational semantics in a nutshell

$$\text{Syntax} \xrightarrow{[\![-]\!]} \text{Semantics}$$

$$\text{Program } P \mapsto \text{Denotation } [\![P]\!]$$

Recursive program $\mapsto$ Partial recursive function

Boolean circuit $\mapsto$ Boolean function

...

Type $\mapsto$ Domain

Program $\mapsto$ Continuous functions between domains

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

IMP SYNTAX

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*

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

IMP SYNTAX

Arithmetic expressions

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

Commands

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

ranges over *integers* → $\underline{n}$

Arithmetic expressions

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

Commands

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

IMP syntax

Arithmetic expressions

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

Boolean expressions

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

Commands

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

Denotation functions – naïvely

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

where

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

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

where

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

Arithmetic expressions?

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

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

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

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

Denotation functions

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

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

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

where

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

Denotation functions

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

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

where $\rightharpoonup$ denotes partial functions and

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

Semantics of arithmetic expressions

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

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

Semantics of boolean expressions

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

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

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

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

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

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

This is compositionality!

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

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

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

Semantics of commands

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

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

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

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

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

$$\begin{aligned} \mathcal{C}[\![C; C']\!] &= \mathcal{C}[\![C']\!] \circ \mathcal{C}[\![C]\!] \\ &= \lambda s \in \text{State}.\ \mathcal{C}[\![C']\!]\left(\mathcal{C}[\![C]\!](s)\right) \end{aligned}$$

Introduction

A semantics for 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 $[\![B]\!]$

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

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

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

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

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

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

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

Our occupation for the next few lectures...

Introduction

A taste of domain theory

An example

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

An example

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

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 $(\mathrm{State} \rightharpoonup \mathrm{State})$:

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

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

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 fixed point

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

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

Approximating the fixed point

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

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

Approximating the fixed point

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

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

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

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

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

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

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

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

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

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

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

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

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

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

We have our semantics

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

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

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

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

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

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

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

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 $\bot$.

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

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:

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

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

The least pre-fixed point is a 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 $f(d) \sqsubseteq d$.

Proofs with least fixed points

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

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.

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