Denotational Semantics

Slides mostly follow

John C. Reynolds' book *Theories of Programming Languages* and

Xinyu Feng's lecture notes

Denotational semantics

- Idea: programs → mathematical objects
- Finding domains that represent what programs do
  - Partial functions
  - Games between environtment and the system
- Should be compositional
  - Built out of the denotations of sub-programs
- Should be abstract
  - Syntax independence, full abstraction

This class

- Formulating the denotational semantics for the simple imperative programming language (IMP)
- Basics of domain theory

Recall the syntax of IMP

$$(IntExp)\ \ e \ ::= \mathbf{n} \mid x \mid e + e \mid e - e \mid \ldots$$

$$\begin{aligned}(BoolExp)\ \ b \ ::= \ & \mathbf{true} \mid \mathbf{false} \\ & \mid e = e \mid e < e \\ & \mid \neg b \mid b \wedge b \mid b \vee b \mid \ldots\end{aligned}$$

$$\begin{aligned}(Comm)\ \ c \ ::= \ & \mathbf{skip} \\ \mid \ & x := e \\ \mid \ & c; c \\ \mid \ & \mathbf{if}\ b\ \mathbf{then}\ c\ \mathbf{else}\ c \\ \mid \ & \mathbf{while}\ b\ \mathbf{do}\ c\end{aligned}$$

Denotational semantics for Exps

$$(IntExp)\ \ e \ ::= \mathbf{n} \mid x \mid e + e \mid e \ - e \mid \ldots$$

$$\begin{aligned}(BoolExp)\ \ b \ ::= & \ \mathbf{true} \ \mid \mathbf{false} \\ & \mid e = e \mid e < e \\ & \mid \ \neg b \mid b \wedge b \mid b \vee b \mid \ldots\end{aligned}$$

$$(State)\ \ \sigma \quad \in \ Var \to \mathbb{Z}$$

$$\begin{aligned}[\![-]\!]_I \quad & \in \ IntExp \to State \to \mathbb{Z} \\ [\![-]\!]_B \quad & \in \ Bool \to State \to \mathbb{B}\end{aligned}$$

Denotational semantics for Exps 1

$$(IntExp)\ \ e\ ::= \mathbf{n} \mid x \mid e + e \mid e\ - e \mid \ldots$$

$$\begin{aligned}(BoolExp)\ \ b\ ::= &\ \mathbf{true}\ \mid \mathbf{false} \\ &\mid e = e \mid e < e \\ &\mid\ \neg b \mid b \land b \mid b \lor b \mid \ldots\end{aligned}$$

$$(State)\ \ \sigma \quad \in\ Var \to \mathbb{Z}$$

$$[\![-]\!]_I\ ::= \lambda e.\lambda\sigma.n\ , if\ (e, \sigma) \to^* (\mathbf{n}, \sigma)\ and\ \lfloor \mathbf{n} \rfloor = n$$

$$[\![-]\!]_B ::= \lambda b.\lambda\sigma.\begin{cases} true, & if\ (b, \sigma) \to^* (\mathbf{true}, \sigma) \\ false, & if\ (b, \sigma) \to^* (\mathbf{false}, \sigma) \end{cases}$$

Denotational semantics for Exps 2

$$(IntExp)\ \ e\ ::= \mathbf{n} \mid x \mid e + e \mid e\ - e \mid \ldots$$

$$\begin{aligned}(BoolExp)\ \ b\ ::= &\ \mathbf{true}\ \mid \mathbf{false} \\ &\mid e = e \mid e < e \\ &\mid\ \neg b \mid b \wedge b \mid b \vee b \mid \ldots\end{aligned}$$

$$(State)\ \ \sigma \quad \in\ Var \rightarrow \mathbb{Z}$$

$$[\![-]\!]_I\ ::= \lambda e.\lambda\sigma.n\ , if\ (e,\sigma) \Downarrow n$$

$$[\![-]\!]_B ::= \lambda b.\lambda\sigma.\begin{cases} true, & if\ (b,\sigma) \Downarrow true \\ false, & if\ (b,\sigma) \Downarrow false \end{cases}$$

Denotational semantics for Exps 3

$(IntExp)\ \ e\ \ ::= \mathbf{n} \mid x \mid e + e \mid e\ - e \mid \ldots$

$(BoolExp)\ \ b\ \ ::= \mathbf{true}\ \mid \mathbf{false}$
$\qquad\qquad\qquad \mid e = e \mid e < e$
$\qquad\qquad\qquad \mid\ \neg b \mid b \wedge b \mid b \vee b \mid \ldots$

$(State)\ \ \sigma\ \ \ \in\ Var \rightarrow \mathbb{Z}$

$[\![\mathbf{n}]\!]_I\, \sigma ::= \lfloor \mathbf{n} \rfloor \qquad\qquad [\![x]\!]_I\, \sigma ::= \sigma(x)$
$[\![e_1 + e_2]\!]_I\, \sigma ::= [\![e_1]\!]_I\, \sigma + [\![e_2]\!]_I \sigma \qquad \ldots$

$[\![\mathbf{true}]\!]_B \sigma ::= true \qquad [\![\mathbf{false}]\!]_B \sigma ::= false$
$[\![\neg b]\!]_B \sigma ::= \text{if } [\![b]\!]_B \sigma \text{ then } true \text{ else } false \quad \ldots$

Denotational semantics for $Comm$

$$\llbracket - \rrbracket_C \in IntExp \to State \to ?$$

- Either
  - **Terminate**, with a final $State$;
  - **Nonterminating**, without a final state, e.g., **while true do skip**
- Must be partial if $? = State$

Denotational semantics for $Comm$

$$\llbracket - \rrbracket_C \quad \in \; IntExp \to State \to State_\perp$$

- For any set S, let $S_\perp = S \cup \{\perp\}$ (assuming $\perp \notin S$)
  - $\perp$, usually called “bottom”, for nontermination
- The denotational semantics of $Comm$ made total

Semantics for skip and assign.

- $[\![\mathbf{skip}]\!]_C\, \sigma \quad ::= \sigma$
- $[\![x \coloneqq e]\!]_C\, \sigma \;\; ::= \sigma\{x \leadsto [\![e]\!]_I \sigma\}$

- E.g.,

$$
\begin{aligned}
& [\![x \coloneqq x + \mathbf{10}]\!]_C \{(x, 32)\} \\
& = \{(x, 32)\}\{x \leadsto [\![x + \mathbf{10}]\!]_I \{(x, 32)\}\ \} \\
& = \{(x, 32)\}\{x \leadsto ([\![x]\!]_I \{(x, 32)\} + [\![\mathbf{10}]\!]_I \{(x, 32)\})\} \\
& = \{(x, 32)\}\{x \leadsto (32 + 10)\} \\
& = \{(x, 32)\}\{x \leadsto 42\} \\
& = \{(x, 42)\}
\end{aligned}
$$

Semantics for conditionals

- $[\![\textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2]\!]_C\, \sigma \quad ::= \begin{cases} [\![c_1]\!]_C\, \sigma, & \text{if } [\![b]\!]_B \sigma = true \\ [\![c_2]\!]_C\, \sigma, & \text{if } [\![b]\!]_B \sigma = false \end{cases}$
- E.g.,

$[\![\textbf{if } x < 0 \textbf{ then } x = 0 - x \textbf{ else skip}]\!]_C\, \{(x, -3)\}$

$= [\![x = 0 - x\,]\!]_C\, \{(x, -3)\}$ since $[\![x < 0]\!]_B\, \{(x, -3)\} = true$
$= \{(x, -3)\}\{x \rightsquigarrow [\![0 - x\,]\!]_I\, \{(x, -3)\}\}$
$= \{(x, 3)\}$

$[\![\textbf{if } x < 0 \textbf{ then } x = 0 - x \textbf{ else skip}]\!]_C\, \{(x, 5)\}$

$= [\![\textbf{skip}]\!]_C\, \{(x, 5)\}$ since $[\![x < 0]\!]_B\, \{(x, 5)\} = false$
$= \{(x, 5)\}$

Semantics for sequential composition

- $[\![c_1;\ c_2]\!]_C\,\sigma \quad ::= \begin{cases} \bot & , \quad \text{if } [\![c_1]\!]_C\sigma = \bot \\ [\![c_2]\!]_C \circ [\![c_1]\!]_C\,\sigma, & otherwise \end{cases}$

- We extend $f \in S \to T_\bot$ to $f_{\perp\!\!\!\perp} \in S_\bot \to T_\bot$

$$f_{\perp\!\!\!\perp} x \ ::= \begin{cases} \bot, if\ x = \bot \\ f\ x, otherwise \end{cases}$$

- Effectively it defines a ***lift*** operator

$$(-)_{\perp\!\!\!\perp} \in (S \to T_\bot) \to (S_\bot \to T_\bot)$$

- So $[\![c_1;\ c_2]\!]_C\,\sigma \ = ([\![c_2]\!]_C)_{\perp\!\!\!\perp}([\![c_1]\!]_C\,\sigma)$

Semantics of loops

- Idea: define the meaning of $\textbf{while } b \textbf{ do } c$ as that of

$$\textbf{if } b \textbf{ then } (c; \textbf{while } b \textbf{ do } c) \textbf{ else skip}$$

- That is,

$$\begin{aligned}
&[\![\textbf{while } b \textbf{ do } c]\!]_C \sigma \\
&\quad = [\![\textbf{if } b \textbf{ then } (c; \textbf{while } b \textbf{ do } c) \textbf{ else skip}]\!]_C\ \sigma \\
&\quad = \begin{cases} ([\![\textbf{while } b \textbf{ do } c]\!]_C)_{\perp\!\!\perp}([\![c]\!]_C \sigma), \text{if } [\![b]\!]_B \sigma = true \\ \sigma \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad , otherwise \end{cases}
\end{aligned}$$

- Not syntax directed, not compositional

Semantics of loops

- We may view $[\![\textbf{while}\ b\ \textbf{do}\ c]\!]_C$ as a sulotion for this equation:

$$[\![\textbf{while}\ b\ \textbf{do}\ c]\!]_C\sigma =$$
$$\begin{cases} ([\![\textbf{while}\ b\ \textbf{do}\ c]\!]_C)_{\perp\!\!\perp}([\![c]\!]_C\sigma), \text{if } [\![b]\!]_B\sigma = true \\ \sigma \qquad\qquad\qquad\qquad\qquad\qquad\qquad , otherwise \end{cases}$$

- That is, a fixed-point of

$$F ::= \lambda f \in State \to State_{\perp}.$$
$$\lambda\sigma \in State. \begin{cases} f_{\perp\!\!\perp}([\![c]\!]_C\sigma), & if\, [\![b]\!]_B\sigma = true \\ \sigma & , \quad otherwise \end{cases}$$

Semantics of loops

- That is, a fixed-point of

$$F ::= \lambda f \in State \to State_{\perp}.$$

$$\lambda\sigma \in State. \begin{cases} f_{\perp\!\!\perp}(\llbracket c \rrbracket_C \sigma), & if \llbracket b \rrbracket_B \sigma = true \\ \sigma & , \ otherwise \end{cases}$$

- However, not every $F \in (State \to State_{\perp}) \to (State \to State_{\perp})$ has a fixed-point, and some may have more than one.
- Example: for any σ', $\lambda\sigma.\sigma'$ (a constant function) is a solution for

$$\llbracket \textbf{while true do } x \coloneqq x + 1 \rrbracket_C$$

- We need to guarantee the meaning is uniquely determined by the equation.

Semantics of loops

- Intuition: the limit of approximations W_n
- First and least accurate approximation (0-iteration)
$$W_0 ::= \lambda\sigma \in State. \bot$$
- 1 iteration
$$W_1 ::= F\ W_0 = \lambda\sigma \in State. \text{if } [\![b]\!]_B\sigma \text{ then } (W_0)_{\perp\!\!\!\perp}([\![c]\!]_C\sigma) \text{ else } \sigma$$
$$= \lambda\sigma \in State. \text{if } [\![b]\!]_B\sigma \text{ then } \bot \text{ else } \sigma$$
- 2 iterations
$$W_2 ::= F\ W_1 = \lambda\sigma \in State. \text{if } [\![b]\!]_B\sigma \text{ then } (W_1)_{\perp\!\!\!\perp}([\![c]\!]_C\sigma) \text{ else } \sigma$$
- …
- n+1 iterations
$$W_{n+1} ::= F\ W_n$$

Semantics of loops

- Intuition: the limit of finite approximations W_n
- First and least accurate approximation (0-iteration)

$$W_0 ::= \lambda\sigma \in State.\perp$$

- n+1 iterations

$$W_{n+1} ::= F\ W_n$$

- The limit $W ::= \lim_{n\to\infty} W_n$

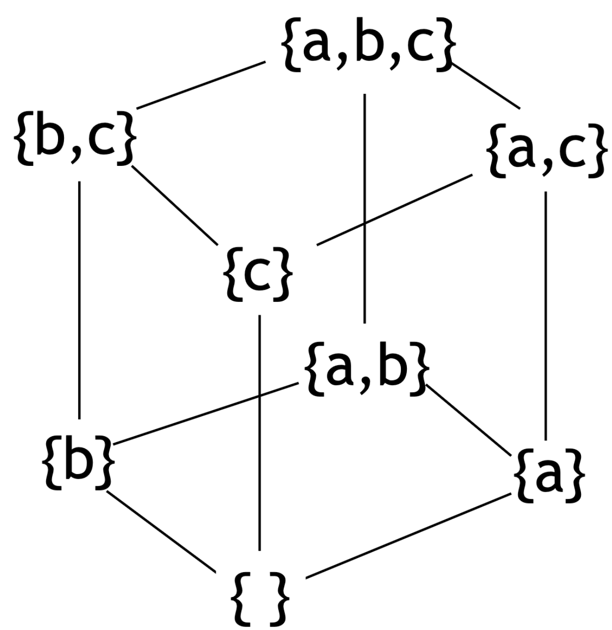
- How do we take limits in a space of functions?
- Monotonicity + bound
  - An ***ordering*** $\sqsubseteq$ such that $W_0 \sqsubseteq W_1 \sqsubseteq W_2 \sqsubseteq$ ...
  - ***Least upper bound*** of the sequence

Partially ordered sets

- A binary relation ρ on S is
  - Reflexive iff $\forall x \in S.\, x \;\rho\; x$
  - Transitive iff $x \;\rho\; y \land y \;\rho\; z \Rightarrow x \;\rho\; z$
  - Antisymmetric iff $x \;\rho\; y \land y \;\rho\; x \Rightarrow x = y$
  - Symmetric iff $x \;\rho\; y \Rightarrow y \;\rho\; x$
- $\sqsubseteq$ is a ***preorder*** on S iff $\sqsubseteq$ is reflexive and transitive
- $\sqsubseteq$ is a ***partial order*** on S iff $\sqsubseteq$ is a preorder on S and antisymmetric
- A ***poset*** S: S with a partial order $\sqsubseteq$ on S
- A ***discretely ordered*** S: S with Id_S as a partial order

Hasse diagrams

- Picturize partial orders
  - Points – elements; lines – direct predecessor
- E.g., $\subseteq$ as the partial order on set $2^{\{a,b,c\}}$



Monotonicity and upper bound

- $f \in S \to T$ is ***monotone*** iff $x \sqsubseteq y \Rightarrow f\ x \sqsubseteq f\ y$
- y is ***upper bound*** of $X \subseteq S$ iff $\forall x \in X.\ x \sqsubseteq y$

Least upper bound

- y is a ***least upper bound (lub)*** of $X \subseteq S$ iff
  - y is ***upper bound*** of X, and
  - $\forall z \in S.\, z$ is an upper bound of $X \Rightarrow y \sqsubseteq z$
- If S is a poset and $X \subseteq S$, there is at most one lub of X (denoted by $\sqcup X$)
- $\sqcup \emptyset = \perp$, the least element of S (if exists)
- Let $\mathcal{X} \subseteq \mathcal{P}(S)$ such that $\sqcup X$ exists forall $X \in \mathcal{X}$,

$$\sqcup \{ \sqcup X \mid X \in \mathcal{X} \} = \sqcup (\cup \mathcal{X})$$

if either of these lub exists

Domains

- A ***chain*** C is a countably infinite non-decreasing sequence

$$x_0 \sqsubseteq x_1 \sqsubseteq \ \dots$$

- We may also use C to represent the set of elements on the chain

- The ***limit*** of a chain C is the lub of all its elements when it exists
- A chain C is interesting if $(\sqcup\, C) \notin C$

- A poset D is a ***predomain*** (or ***complete partial order*** – ***cpo***) if every chain elements in D has a limit in D
- A predomain D is a ***domain*** (or ***pointed cpo***) if D has a least element $\perp$

Lifting

- $D_\perp$ is a ***lifting*** of the predomain D if:
  - $\perp \notin D$
  - $x \sqsubseteq_{D_\perp} y$ iff either $x = \perp$ or $x \sqsubseteq_D y$

- Any set S can be viewed as a predomain with ***discrete partial order*** $\sqsubseteq ::= \mathrm{Id}_S$

- D is a ***flat domain*** if $D - \{\perp\}$ is discretely ordered

Continuous Functions

- If D and D' are predomains, $f \in D \to D'$ is a ***continuous function*** if it maps limits to limis:
  $f\ (\sqcup\ C) = \sqcup'\ \{\ f\ x_i \mid x_i \in C\ \}$ for every chain C in D

- Continuous functions are monotone ($x \sqsubseteq y \sqsubseteq y$ ...)
- Monotone functions may not be continuous
  - Suppose $C = x_0 \sqsubseteq x_1 \sqsubseteq \cdots$ is an insteresting chain in D with a limit x, and $D' = \{\bot, \top\}$ such that $\bot \sqsubseteq' \top$
  - Consider $f = \lambda y.\, \text{if}\ y = x\ \text{then}\ \top\ \text{else}\ \bot$

Monotone vs continuous

- A monotone function $f \in D \to D'$ is continuous iff forall interesting chains $x_0 \sqsubseteq x_1 \sqsubseteq \cdots$ we have
$$f(\sqcup_{i=0}^{\infty} x_i) \sqsubseteq \sqcup'^{\infty}_{i=0} (f\ x_i)$$
- Proof.

The (pre)domain of continuous functions

- ***Pointwise ordering*** of functions in $P \to P'$, where P' is a predomain:

$$f \sqsubseteq_{\to} g \ ::= \forall x \in P.\, f\ x \sqsubseteq_{P'} g\ x$$

- **Proposition**:

If P and P' are predomains, then the set $[P \to P']$ of continuous functions in $P \to P'$ with partial order $\sqsubseteq_{\to}$ is a predomain, such that for any chain $f_0 \sqsubseteq_{\to} f_1 \sqsubseteq_{\to} \ldots$, we have

$$\sqcup_i f_i = \lambda x \in P. \sqcup_i' (f_i\ x)$$

If P' is a domain, then $[P \to P']$ is a domain with

$$\bot_{\to} = \lambda x \in P.\, \bot_{P'}$$

Examples: continuous functions

- For predomains P, P' and P'',
  - If $f \in P \to P'$ is constant, then $f \in [P \to P']$
  - $\mathrm{Id}_P \in [P \to P]$
  - If $f \in [P \to P']$ and $g \in [P' \to P'']$, $g \circ f \in [P \to P'']$
  - If $f \in [P \to P']$, $(- \circ f) \in \big[[P' \to P''] \to [P \to P'']\big]$

Strict functions and lifting

- If D and D' are domains, $f \in D \to D'$ is ***strict*** if $f\ \bot = \bot'$
- If P and P' are predomains, $f \in P \to P'$, then the strict funcion

$$f_\bot ::= \lambda x \in P_\bot .\, if\ x = \bot\ then\ \bot'\ \ else\ f\ x$$

is the ***lifting*** of f to $P_\bot \to P'_{\bot'}$.

- If P' is a domain, then the strict function

$$f_{\perp\!\!\!\perp} ::= \lambda x \in P_\bot .\, if\ x = \bot\ then\ \bot'\ \ else\ f\ x$$

is the ***source lifting*** of f to $P_\bot \to P'$

- If f is continuous, so are $f_\bot$ and $f_{\perp\!\!\!\perp}$.
- $(-)_\bot$ and $(-)_{\perp\!\!\!\perp}$ are also continuous.

Least fixed-point

- **Theorem** [*Kleene fixed-point theorem*]: If D is a domain and $f \in [D \to \mathrm{D}]$ then $x ::= \sqcup_{i=0}^{\infty} \left(f^{i} \perp\right)$ is the ***least fixed-point*** of f.
- Proof.

  x is well-defined because $\perp \sqsubseteq f \sqsubseteq f^{2} \sqsubseteq \cdots$ is a chain.

  x is a fixed-point because

  $$f\ x = f\ \left(\sqcup_{i=0}^{\infty} \left(f^{i} \perp\right)\right) = \sqcup_{i=0}^{\infty} \left(f\left(f^{i} \perp\right)\right) = x$$

  For any fixed-point y of f, $\perp \sqsubseteq y \Rightarrow f \perp \sqsubseteq f\ y = y$.

  By induction, $\forall i \in \mathbb{N}.\ f^{i} \sqsubseteq y$. So y is an upper bound of the chain $\perp \sqsubseteq f \perp \sqsubseteq \cdots$. Since x is a lub, $x \sqsubseteq y$.

The least fixed-point operator

- Let

$$\mathbf{Y}_D = \lambda f \in [D \to D]. \sqcup_{i=0}^{\infty} (f^i \perp)$$

- $\forall f \in [D \to D]. \mathbf{Y}_D\ f$ is the least fixed-point of f.

- $\mathbf{Y}_D \in \big[[D \to D] \to D\big]$

Back to semantics of loops

- Recall $[\![\textbf{while } b \textbf{ do } c]\!]_C \sigma =$
$$\begin{cases} ([\![\textbf{while } b \textbf{ do } c]\!]_C)_{\perp\!\!\perp}([\![c]\!]_C\sigma), \text{if } [\![b]\!]_B\sigma = true \\ \sigma \qquad\qquad\qquad\qquad\qquad\qquad , otherwise \end{cases}$$

- It implies that $[\![\textbf{while } b \textbf{ do } c]\!]_C$ is a fixed-point of
$$F ::= \lambda f \in State \rightarrow State_\perp . \lambda\sigma \in State. \text{if } [\![b]\!]_B\sigma \text{ then } f_{\perp\!\!\perp}([\![c]\!]_C\sigma) \text{ else } \sigma$$

- We pick the least fixed-point
$$[\![\textbf{while } b \textbf{ do } c]\!]_C ::= \mathbf{Y}_{[State \rightarrow State_\perp]} F$$

- Coincides with our intuition based on operational semantics:
$$W ::= \lim_{n\rightarrow\infty} W_n = \lim_{n\rightarrow\infty} F^n W_0$$

Abstractness of semantics

- Abstract semantics are an attempt to separate the important properties of a language (what computations can it express) from the unimportant (how exactly computations are represented).

- The more terms are considered equal by a semantics, the more abstract it is.

- A semantic function $[\![-]\!]_1$ is ***at least as abstract as*** $[\![-]\!]_0$ if

$$\forall c, c'. [\![c]\!]_0 = [\![c']\!]_0 \Rightarrow [\![c]\!]_1 = [\![c']\!]_1$$

Observation and context

- If there are other means of observing the result of a computation, a semantics may be incorrect if it equates too many terms.

- Observation: “needs of the user”
- Let O be an observation, and $\mathcal{O}$ be a set of observations, i.e.

$$O \in \mathcal{O} \subseteq Comm \rightarrow Outcomes$$

- A ***context*** C is a command with a ***hole*** $[\,]$. Use $\mathcal{C}$ for all contexts.
- A command c can be ***placed in the hole*** of C, yielding $C[c]$ (not substitution – name capture is allowed).
- E.g., $C = (\mathbf{newvar}\; x \coloneqq 1\; \mathbf{in}\; [\,]\,; y \coloneqq x)$

Soundness and full abstractness

- A semantic function $[\![-]\!]$ is ***sound (with respect to $\mathcal{O}$)*** iff $\forall c, c'. [\![c]\!] = [\![c']\!] \Rightarrow \forall O \in \mathcal{O}. \forall C \in \mathcal{C}. O(C[c]) = O(C[c'])$

- A semantic function $[\![-]\!]$ is ***fully abstract (with respect to $\mathcal{O}$)*** iff
  $\forall c, c'. [\![c]\!] = [\![c']\!] \Leftrightarrow \forall O \in \mathcal{O}. \forall C \in \mathcal{C}. O(C[c]) = O(C[c'])$

i.e. $[\![-]\!]$ is the "most abstract" sound semantics.

- **Proposition**: if $[\![-]\!]_0$ and $[\![-]\!]_1$ are both fully abstract semantics w.r.t. $\mathcal{O}$, then $[\![-]\!]_0 = [\![-]\!]_1$

Full abstractness of semantis for IMP

- Let $O_{\sigma,x} ::= \lambda c.\,\mathrm{if}\ [\![c]\!]_C\sigma = \bot\ \mathrm{then}\ \bot\ \mathrm{else}\ ([\![c]\!]_C\sigma)\ \ x$
- Let $\mathcal{O}$ be the set of all such observations, i.e.

$$\mathcal{O} = \left\{ O_{\sigma,x} \mid \sigma \in State, x \in Var \right\} \subseteq Comm \to \mathbb{Z}_\bot$$

- **Proposition**: $[\![-]\!]_C$ is fully abstract w.r.t. $\mathcal{O}$.
  - $[\![-]\!]_C$ is sound: by compositionality, if $[\![c]\!]_C = [\![c']\!]_C$, then for any context C, $[\![C[c]]\!]_C = [\![C[c']]\!]_C$ (induction). So $O_{\sigma,x}(C[c]) = O_{\sigma,x}(C[c'])$ for any observation $O_{\sigma,x}$.
  - $[\![-]\!]_C$ is most abstract: consider the empty context $C = \cdot$. If $O_{\sigma,x}(c) = O_{\sigma,x}(c')$ holds for all $x \in Var$ and $\sigma \in State$, we know by definition $[\![c]\!]_C = [\![c']\!]_C$.

Main points of denotational semantics

- Idea: programs → mathematical objects
- Theoretical foundation: domain theory
  - Poset, lub
  - Predomain (cpo), domain (pointed cpo)
  - Continuous functions, least fixed-point
- Compositional and abstract

More on this topic

- Denotations for **newvar**, …
- Observing termination of closed commands
- Extensions, e.g., the **fail** command
- …

- Please refer to Chapter 2 of *Theories of Programming Languages* by Reynolds