Topic 1

Introduction

What is this course about?

- General area.

  Formal methods: Mathematical techniques for the specification, development, and verification of software and hardware systems.

- Specific area.

  Formal semantics: Mathematical theories for ascribing meanings to computer languages.

Why do we care?

- Rigour.

  . . . specification of programming languages

  . . . justification of program transformations

- Insight.

  . . . generalisations of notions computability

  . . . higher-order functions

  . . . data structures

- Feedback into language design.

  … continuations

  … monads

- Reasoning principles.

  … Scott induction

  … Logical relations

  … Co-induction

Styles of formal semantics

Operational.

Meanings for program phrases defined in terms of the *steps of computation* they can take during program execution.

Axiomatic.

Meanings for program phrases defined indirectly via the *axioms and rules* of some logic of program properties.

Denotational.

Concerned with giving *mathematical models* of programming languages. Meanings for program phrases defined abstractly as elements of some suitable mathematical structure.

Basic idea of denotational semantics

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

semantic brackets

phrase $\qquad P \quad \mapsto \quad [\![P]\!] \qquad$ meaning.

Basic idea of denotational semantics

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

$$\text{Recursive program} \mapsto \text{Partial recursive function}$$

$$P \mapsto [\![P]\!]$$

Basic idea of denotational semantics

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

Concerns

- Abstract models.
- Compositionality.
- Relationship to computation.

Characteristic features of a denotational semantics

- Each phrase (= part of a program), P, is given a denotation, $[\![P]\!]$ — a mathematical object representing the contribution of P to the meaning of *any* complete program in which it occurs.

- The denotation of a phrase is determined just by the denotations of its subphrases (one says that the semantics is compositional).

Basic example of denotational semantics (I)

IMP$^-$ syntax

Arithmetic expressions

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

where n ranges over *integers* and

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

Boolean expressions

$$\begin{array}{rcl} B \in \mathbf{Bexp} & ::= & \mathbf{true} \mid \mathbf{false} \mid A = A \mid \ \ldots \\ & \mid & \neg B \mid \ \ldots \end{array}$$

Commands

$$\begin{array}{rcl} C \in \mathbf{Comm} & ::= & \mathbf{skip} \mid L := A \mid C ; C \\ & \mid & \mathbf{if}\ B\ \mathbf{then}\ C\ \mathbf{else}\ C \end{array}$$

Basic example of denotational semantics (II)

Semantic functions

$$\mathcal{A}: \quad \mathbf{Aexp} \rightarrow (State \rightarrow \mathbb{Z})$$

$\mathcal{A}[\![E]\!] : State \rightarrow \mathbb{Z}$ $\qquad$ $s \in State$

$E \in \underline{\underline{Aexp}}$ $\qquad$ $\mathcal{A}[\![E]\!](s) \in \mathbb{Z}$

where

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

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

Basic example of denotational semantics (II)

Semantic functions

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

$E \in \underline{BExp}$, $s \in \underline{State}$, $\mathcal{B}[\![E]\!](s) \in \mathbb{B}$

where

$$\begin{aligned} \mathbb{Z} &= \{\ldots, -1, 0, 1, \ldots\} \\ \mathbb{B} &= \{\, true, false \,\} \\ State &= (\mathbb{L} \to \mathbb{Z}) \end{aligned}$$

$\mathcal{C}$: commands $\mapsto$ state transformers

Basic example of denotational semantics (II)

formally, a partial function from State to State.

Semantic functions

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

where $P \in \underline{\mathrm{Comm}}$, $s \in \underline{\mathrm{State}}$: $\mathcal{C}[\![P]\!](s) \in \underline{\mathrm{State}}$.

The state resulting from the computation of P on states.

$$\begin{aligned}
\mathbb{Z} &= \{\ldots, -1, 0, 1, \ldots\} \\
\mathbb{B} &= \{\mathit{true}, \mathit{false}\} \\
\mathit{State} &= (\mathbb{L} \to \mathbb{Z})
\end{aligned}$$

Basic example of denotational semantics (II)

Semantic functions

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

where

$$\begin{aligned}
\mathbb{Z} &= \{\ldots, -1, 0, 1, \ldots\} \\
\mathbb{B} &= \{\, true, false \,\} \\
State &= (\mathbb{L} \to \mathbb{Z})
\end{aligned}$$

$L \in \mathbb{L}$
$s \in State$
$s(L) \in \mathbb{Z}$

or
memory
or
store

= functions from locations to integers

Recall $\lambda x. e[x]$ denotes the function that on input, say a, results in output $e[a]$.

Basic example of denotational semantics (III)

Semantic function $\mathcal{A}$

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

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

$\mathcal{A}[\![L]\!](s) = s(L)$

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

compositionality

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

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

Abuse of notation

plus symbol

integer addition

Basic example of denotational semantics (IV)

Semantic function $\mathcal{B}$

$\mathcal{B}[\![\underline{true}]\!](s) = \underline{true}$

$$\mathcal{B}[\![\mathbf{true}]\!] = \lambda s \in State.\, true$$

$$\mathcal{B}[\![\mathbf{false}]\!] = \lambda s \in State.\, false$$

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

where $eq(a, a') = \begin{cases} true & \text{if } a = a' \\ false & \text{if } a \neq a' \end{cases}$

$$\mathcal{B}[\![A_1 = A_2]\!](s) = \begin{cases} true & [\![A_1]\!](s) = [\![A_2]\!](s) \\ false & \text{otw.} \end{cases}$$

state transformer

$\mathcal{C}[\![P]\!] : \underline{State} \to \underline{State}$

PeComm

Basic example of denotational semantics (V)

Semantic function $\mathcal{C}$

$$[\![\mathbf{skip}]\!] \quad = \quad \lambda s \in State.\, s$$

identity function

$$[\![skip]\!](s) = s$$

NB: From now on the names of semantic functions are omitted!

$$[\![\text{If } B \text{ then } C \text{ else } C']\!](s) = \begin{cases} [\![C]\!](s) & [\![B]\!](s) = \text{true} \\ [\![C']\!](s) & \text{otw} \end{cases}$$

A simple example of compositionality

Given partial functions $[\![C]\!], [\![C']\!] : \mathit{State} \rightharpoonup \mathit{State}$ and a function $[\![B]\!] : \mathit{State} \to \{\mathit{true}, \mathit{false}\}$, we can define

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

where

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

Basic example of denotational semantics (VI)

Semantic function $\mathcal{C}$

$$[\![L := A]\!] \quad = \quad \lambda s \in State.\, \lambda \ell \in \mathbb{L}.\, if\left(\ell = L, [\![A]\!](s), s(\ell)\right)$$

$$[\![L := A]\!](s) = \lambda \ell \in \mathbb{L}. \begin{cases} [\![A]\!](s) & \ell = L \\ s(\ell) & \text{otw} \end{cases}$$

Denotational semantics of sequential composition

Denotation of sequential composition $C; C'$ of two commands

$$[\![C; C']\!] = [\![C']\!] \circ [\![C]\!] = \lambda s \in State.\, [\![C']\!]\big([\![C]\!](s)\big)$$

given by composition of the partial functions from states to states $[\![C]\!], [\![C']\!] : State \rightharpoonup State$ which are the denotations of the commands.

$$[\![C; C']\!](s) = [\![C']\!]\big([\![C]\!](s)\big)$$

Denotational semantics of sequential composition

Denotation of sequential composition $C; C'$ of two commands

$$[\![C;C']\!] = [\![C']\!] \circ [\![C]\!] = \lambda s \in State.\, [\![C']\!]\big([\![C]\!](s)\big)$$

given by composition of the partial functions from states to states $[\![C]\!], [\![C']\!] : State \rightharpoonup State$ which are the denotations of the commands.

Cf. operational semantics of sequential composition:

$$\frac{C, s \Downarrow s' \quad C', s' \Downarrow s''}{C; C', s \Downarrow s''}\ .$$

One can show

$\forall C, s, s'.\ [\![C]\!](s) = s'$ iff $C, s \Downarrow s'$.

$[\![\textbf{while}\ B\ \textbf{do}\ C]\!]$: State $\rightarrow$ State

$C \in Comm ::= \ldots \mid \underline{while}\ B\ \underline{do}\ C \mid \ldots$

$[\![\underline{while}\ B\ \underline{do}\ C]\!](s)$

$= \ldots [\![B]\!](s) \ldots [\![C]\!](s) \ldots$

Program equivalence

while B do C
≡ if B then (C; while B do C) else skip.

Expect

$[\![$while B do C$]\!]$
= $[\![$if B then (C; while B do C) else skip$]\!]$

That is,

for all states s

$$[\![\text{while } B \text{ do } C]\!](s)$$

$$= [\![\text{if } B \text{ then } (C; \text{while } B \text{ do } C) \text{ else skip}]\!](s)$$

$$= \text{if}\left([\![B]\!](s), [\![\text{while } B \text{ do } C]\!]([\![C]\!]s), s\right)$$

Tempting to define

$$[\![\text{while } B \text{ do } C]\!](s)$$

$$\stackrel{\text{def}}{=} \text{if}\left([\![B]\!](s), [\![\text{while } B \text{ do } C]\!]([\![C]\!]s), s\right)$$

but this is non-sense!

$[\![\text{while } B \text{ do } C]\!]$ is a fixed point

recall that a fixed of a function f is an element x such that $x = f(x)$.

$[\![\text{while } B \text{ do } C]\!]$

$= \lambda s \in State.\ if\ ([\![B]\!](s), [\![\text{while } B \text{ do } C]\!]([\![C]\!]s),$
$s)$

$f_{[\![B]\!],[\![C]\!]} : (State \to State) \to (State \to State)$

s.t. $f_{[\![B]\!],[\![C]\!]}([\![\text{while } B \text{ do } C]\!]) = [\![\text{while } B \text{ do } C]\!]$.

Fixed point property of $[\![\textbf{while } B \textbf{ do } C]\!]$

$$[\![\textbf{while } B \textbf{ do } C]\!] = f_{[\![B]\!],[\![C]\!]}([\![\textbf{while } B \textbf{ do } C]\!])$$

where, for each $b : State \to \{true, false\}$ and $c : State \rightharpoonup State$, we define

$$f_{b,c} : (State \rightharpoonup State) \to (State \rightharpoonup State)$$

as

$$\boxed{f_{b,c} = \lambda w \in (State \rightharpoonup State).\, \lambda s \in State.\, if\big(b(s), w(c(s)), s\big).}$$

$$[\![\text{while } B \text{ do } C]\!] = \underbrace{fix}_{\boxed{?}}(f_{[\![B]\!],[\![C]\!]})$$

Fixed point property of $[\![\textbf{while } B \textbf{ do } C]\!]$

$$[\![\textbf{while } B \textbf{ do } C]\!] = f_{[\![B]\!],[\![C]\!]}([\![\textbf{while } B \textbf{ do } C]\!])$$

where, for each $b : \mathit{State} \to \{\mathit{true}, \mathit{false}\}$ and $c : \mathit{State} \rightharpoonup \mathit{State}$, we define

$$f_{b,c} : (\mathit{State} \rightharpoonup \mathit{State}) \to (\mathit{State} \rightharpoonup \mathit{State})$$

as

$$f_{b,c} = \lambda w \in (\mathit{State} \rightharpoonup \mathit{State}).\, \lambda s \in \mathit{State}.\, \mathit{if}\big(b(s), w(c(s)), s\big).$$

- Why does $w = f_{[\![B]\!],[\![C]\!]}(w)$ have a solution?
- What if it has several solutions—which one do we take to be $[\![\textbf{while } B \textbf{ do } C]\!]$?

$fix(f_{[\![B]\!],[\![C]\!]})$ by approximation

Approximating $[\![\textbf{while } B \textbf{ do } C]\!]$

- $w_0 : State \to State$

  $w_0 = \emptyset$ empty partial function

  $w_0(s) = \uparrow$ ("undefined")

- $w_1 = f_{[\![B]\!],[\![C]\!]}(w_0)$

  $w_1(s) = if([\![B]\!](s), w_0([\![C]\!]s), s)$

  $= if([\![B]\!](s), \uparrow, s)$

- $w_2 = f_{[\![B]\!],[\![C]\!]}(w_1)$

$$w_2(s) = f([\![B]\!](s), w_1([\![C]\!](s)), s)$$

$$= f([\![B]\!](s), f([\![B]\!]([\![C]\!](s)), \uparrow, [\![C]\!](s)), s)$$

$\vdots$

- $w_n = f_{[\![B]\!],[\![C]\!]}(w_{n-1})$

approximates $[\![\text{while } B \text{ do } C]\!]$ up to n iterations.

union
join $\bigcup_{n \in \mathbb{N}} w_n = \bigcup_{n \in \mathbb{N}} f_{[\![B]\!],[\![C]\!]}{}^n(\perp)$
$\quad \shortparallel$

Approximating $[\![\textbf{while } B \textbf{ do } C]\!]$

$\perp$ — empty partial function

$$w_n \stackrel{\text{def}}{=} f_{[\![B]\!],[\![C]\!]}{}^n(\perp)$$

$$= \lambda s \in \mathit{State}.$$

$$\begin{cases} [\![C]\!]^k(s) & \text{if } \exists\, 0 \le k < n.\, [\![B]\!]([\![C]\!]^k(s)) = \mathit{false} \\ & \text{and } \forall\, 0 \le i < k.\, [\![B]\!]([\![C]\!]^i(s)) = \mathit{true} \\ \\ \uparrow & \text{if } \forall\, 0 \le i < n.\, [\![B]\!]([\![C]\!]^i(s)) = \mathit{true} \end{cases}$$

- **while** **true** **do** c

$$w_0 = \perp$$

$$w_1 = \lambda s.\ \text{if}\,(\llbracket \text{true} \rrbracket(s), \uparrow, s)$$

$$= \lambda s.\ \uparrow\ = w_0$$

$$\llbracket \underline{\text{while}}\ \underline{\text{true}}\ \underline{\text{do}}\ c \rrbracket = \bigcup_n w_n = \bigcup_n \perp$$

$$= \perp$$

- <u>while</u> <u>false</u> <u>do</u> c

$w_0 = \perp$

$w_1 = \lambda s.\ \text{if}\ ([\![\text{false}]\!](s), \uparrow, s) = \lambda s.s$

for all $n \geq 1$ $\quad w_n = \lambda s.s.$

$$[\![\underline{\text{while}}\ \underline{\text{false}}\ \text{do}\ c]\!] = \bigcup_n (\perp, \lambda s.s, \ldots)$$

$$= \lambda s.s$$

$$= [\![\text{skip}]\!]$$

The domain of state transformers.

$$D \stackrel{\text{def}}{=} (State \rightharpoonup State)$$

- **Partial order $\sqsubseteq$ on D:**

$w \sqsubseteq w'$	iff	for all $s \in State$, if w is defined at s then so is w' and moreover $w(s) = w'(s)$.
	iff	the graph of w is included in the graph of w'.

- **Least element $\bot \in D$ w.r.t. $\sqsubseteq$:**

$\bot$	=	totally undefined partial function
	=	partial function with empty graph

(satisfies $\bot \sqsubseteq w$, for all $w \in D$).

For a chain of approximations of state transformers

$$\omega_0 \sqsubseteq \omega_1 \sqsubseteq \cdots \sqsubseteq \omega_n \sqsubseteq \cdots \quad (n \in \omega)$$

We let

$$\bigsqcup (\omega_0 \sqsubseteq \omega_1 \sqsubseteq \cdots \sqsubseteq \omega_n \sqsubseteq \cdots) = \bigsqcup_{n \in \omega} \omega_n$$

be the state transformer whose graph is

$$\bigcup_{n \in \omega} \underline{\text{graph}}(\omega_n).$$