Lecture 1

Introduction
What is this course about?

General area.

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

Specific area.

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

General area.

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

Specific area.
Why do we care?

• Rigour.
  ... specification of programming languages
  ... justification of program transformations
  ... data structures
  ... higher-order functions

• Insight.
  ... generalisations of notions computability
  ... justifications of program transformations
  ... specification of programming languages

• Rigour.
Feedback into language design.

- Continuations
- Monads
- Scott induction
- Logical relations
- Co-induction

Reasoning principles.
Styles of formal semantics

Operational. Meanings for program phrases defined in terms of the steps of computation 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

- Syntax
  \[
  [\[ \cdot \]] \quad \rightarrow \quad \cdot
  \]

- Semantics
  
Concerns:

- Abstract models (i.e., implementation/machine independent).
  \[\rightarrow\text{Lectures 2, 3 and 4.}\]

- Compositionality.
  \[\rightarrow\text{Lectures 5 and 6.}\]

- Relationship to computation (e.g., operational semantics).
  \[\leftrightarrow\text{Lectures 7 and 8.}\]

Booleans:

- Boolean function
- Boolean circuit
- Recursive function
- Partial recursive function
- Syntax
- Semantics
Each phrase (= part of a program), $d$, is given a denotation, $[d]$, a mathematical object representing the contribution of $d$ to the meaning of any complete program in which it occurs.

Characteristic features of a denotational semantics is

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

Denotational semantics
Basic example of denotational semantics (I)

IMP syntax

Arithmetic expressions

\[ A \in A_{exp} ::= n | L | A + A | \ldots \]

where \( n \) ranges over integers and \( L \) over a specified set of locations.

\[ T \in T_\text{prog} ::= \text{skip} | C : T | \bar{u} \]

\[ B \in B_{exp} ::= \text{true} | \text{false} | \bar{A} = A | \ldots \]

\[ C \in C ::= \text{if } B \text{ then } C \text{ else } C \]

Commands

Boolean expressions

Arithmetic expressions

IMP syntax
(\( \mathbb{Z} \leftarrow \top \)) = \text{State} \\
\{ \text{true, false} \} = \mathbb{B} \\
\{ \ldots, -1, 0, 1, \ldots \} = \mathbb{Z}

\text{where}

(B \leftarrow \text{State}) \leftarrow \text{Bexp} : \mathbb{B} \\
(A \leftarrow \text{State}) \leftarrow \text{Aexp} : A

\text{Semantic functions}

\text{Basic example of denotational semantics (II)}
\[
(s)[[\mathcal{A}_2] \mathcal{A}] + (s)[[\mathcal{A}_1] \mathcal{A}] \lambda s \in \text{State}\cdot \mathcal{A} = [[\mathcal{A}_2 + \mathcal{A}_1] \mathcal{A}]
\]

\[
(s)[[\mathcal{T}] \mathcal{A}] \lambda s \in \text{State}\cdot \mathcal{A} = [[\mathcal{T}] \mathcal{A}]
\]

\[
\text{Semantic function } \mathcal{A}
\]

Basic example of denotational semanitcs (III)
Basic example of denotational semantics (IV)

Semantic function $B$

- $B[\text{true}] = \lambda s \in \text{State}. \text{true}$
- $B[\text{false}] = \lambda s \in \text{State}. \text{false}$

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

$((s)[\text{false}], (s)[\text{true}])$ if $s \in \text{State. eq}$

$B[\text{false}] = [\text{false}]$

$B[\text{true}] = [\text{true}]$

Semantic function $B$
Basic example of denotational semantics (V)

Semantic function $C$

$\forall s \in State. \ skip = \ [\skip]\$
A simple example of compositionality

Given partial functions

\[
\begin{align*}
&\text{false} = q \downarrow \ x \\
&\text{true} = q \downarrow \ x
\end{align*}
\]

and a function

\[
\begin{array}{c} 
\text{if} \ B \text{ then } C \text{ else } C \\
\end{array}
\]

we can define

\[
\left((s)[C], (s)[C], (s)[B]\right) \text{ if } B \in \text{ STATE} \text{. if } B \in \text{ STATE} \text{ then } C \text{ else } C
\]

Given partial functions \( B : \text{ STATE} \rightarrow \text{ STATE} \) and a

function \( \{\text{false, true}\} \rightarrow \text{ STATE} \), we can define

\[\text{if } B \text{ then } C \text{ else } C\]
Basic example of denotational semantics (VI)

Semantic function $C$

$$\llbracket L := A \rrbracket = \lambda s \in \text{State. } \lambda \ell \in \mathbb{L}. \text{if } (\ell = L, \llbracket A \rrbracket(s), s(\ell))$$
Denotational semantics of sequential composition

\[ \text{[C; C]} = \text{[C]} \circ \text{[C]} \]

\[ \text{[C; C]}(s) = \text{[C]}(\text{[C]}(s)) \]

Denotation of sequential composition of two commands

Cf. operational semantics of sequential composition:

\[ C; C \Downarrow s \Downarrow C; C \Downarrow s \Downarrow C; C \Downarrow s \]

Cf. operational semantics of sequential composition:

\[ \text{[C; C]} \Downarrow s \Downarrow \text{[C; C]} \Downarrow s \Downarrow \text{[C; C]} \Downarrow s \]

Cf. operational semantics of sequential composition:

\[ \text{[C; C]} \Downarrow s \Downarrow \text{[C; C]} \Downarrow s \Downarrow \text{[C; C]} \Downarrow s \]

Cf. operational semantics of sequential composition:

\[ \text{[C; C]} \Downarrow s \Downarrow \text{[C; C]} \Downarrow s \Downarrow \text{[C; C]} \Downarrow s \]

Cf. operational semantics of sequential composition:

\[ \text{[C; C]} \Downarrow s \Downarrow \text{[C; C]} \Downarrow s \Downarrow \text{[C; C]} \Downarrow s \]
\[ \text{while } B \text{ do } C \]

What if it has several solutions—which one do we take to be

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

Why does \( f = w \) have a solution?

\[ f = \lambda w \in \text{State} \rightarrow \text{State}. \lambda s \in \text{State}. \text{if } b(s) \in w(c(s)) \in s \text{.} \]

\[ (s, ((s)c)_w, (s)b) \in \text{State} \rightarrow \text{State}. \text{if } s \in \text{State} \rightarrow \text{State}. \]  

\[ (s, ((s)c)_w, (s)b) \in \text{State} \rightarrow \text{State}. \text{if } s \in \text{State} \rightarrow \text{State}. \]

\[ f : \text{State} \rightarrow \text{State}, \text{ we define} \]

\[ c : \text{State} \rightarrow \text{State}, \text{ we define} \]

\[ \{ \text{true, false} \} \rightarrow \text{State} \rightarrow \text{State}. \text{ where, for each } b : \text{State} \rightarrow \text{State} \]

\[ (\text{while } B \text{ do } C)[C][B]f = \text{while } B \text{ do } C \]

Fixed point property of
\[\text{true} = ((s)_i[C])[B]. u > i \geq 0 \land \downarrow \]

\[\text{true} = ((s)_i[C])[B]. u > i \geq 0 \land \text{and} \]

\[\text{false} = ((s)_i[C])[B]. u > i \geq 0 \land \uparrow \text{if } \forall 0 \leq i < n. (s)_i[C]\]

\[\forall s \in \text{State}. (\exists t (u)[C][B]_t) \]

Approxiimating While $B$ do $C$
\[ D \in \mathcal{D} \subseteq \mathcal{D} \, \text{for all } m \in \mathcal{D} \, \text{iff there exists a partially undefined partial function} \]

\[ \text{iff the graph of } m \text{ is included in the graph of } m'. \]

\[ (s,m) = (s,m') \text{ and moreover } m \text{ is undefined at } s \text{ then } m \subseteq \mathcal{D} \text{ for all } s \in \text{State}, \text{ if } m \text{ is defined at } s \text{ then } \]

\[ \text{Partial order on } \mathcal{D} \in \text{State} \]

\[ \text{Def } \mathcal{D} \]

Least element \( \bot \in \mathcal{D} \text{ wrt. } \mathcal{D} \in \mathcal{T} \)

= totally undefined partial function

\[ \bot = \text{partial function with empty graph} \text{ satisfies } \bot \sqsubseteq w, \text{ for all } w \in \mathcal{D} \text{ wrt. } \mathcal{D} \in \mathcal{T} \text{.} \]