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?
Why do we care?

- Rigour.
  
  • specification of programming languages
  • justification of program transformations
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
• Feedback into language design.
  … continuations
  … monads

• Reasoning principles.
  … Scott induction
  … Logical relations
  … Co-induction
Styles of formal semantics

Operational.

Axiomatic.

Denotational.
Styles of formal semantics

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

Axiomatic.

Denotational.
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.
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

Syntax \[ \to \] Semantics

\[ P \leftrightarrow \llbracket P \rrbracket \]
Basic idea of denotational semantics

Syntax \[\rightarrow\] Semantics

Recursive program \[\iff\] Partial recursive function

\[ P \iff [P] \]
Basic idea of denotational semantics

Syntax \[\rightarrow\] Semantics

Recursive program \[\mapsto\] Partial recursive function

Boolean circuit \[\mapsto\] Boolean function

\[P\] \[\mapsto\] \([P]\)
Basic idea of denotational semantics

Syntax \[\rightarrow\] Semantics

Recursive program \[\mapsto\] Partial recursive function

Boolean circuit \[\mapsto\] Boolean function

\[P\] \[\mapsto\] \([P]\)

Concerns:

- Abstract models (i.e. implementation/machine independent).
  \(\leadsto\) Lectures 2, 3 and 4.
Basic idea of denotational semantics

Syntax $\rightarrow$ Semantics

Recursive program $\mapsto$ Partial recursive function

Boolean circuit $\mapsto$ Boolean function

$P \mapsto \lbrack P \rbrack$

Concerns:

- Abstract models (i.e. implementation/machine independent).
  $\mapsto$ Lectures 2, 3 and 4.

- Compositionality.
  $\mapsto$ Lectures 5 and 6.
Basic idea of denotational semantics

Syntax $\xrightarrow{\llbracket \cdot \rrbracket}$ Semantics

Recursive program $\mapsto$ Partial recursive function

Boolean circuit $\mapsto$ Boolean function

$P \mapsto \llbracket P \rrbracket$

Concerns:

- Abstract models (i.e. implementation/machine independent).
  $\leadsto$ Lectures 2, 3 and 4.

- Compositionality.
  $\leadsto$ Lectures 5 and 6.

- Relationship to computation (e.g. operational semantics).
  $\leadsto$ Lectures 7 and 8.
Characteristic features of a denotational semantics

• Each phrase (= part of a program), $P$, is given a denotation, $\llbracket P \rrbracket$ — 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

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

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

Boolean expressions

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

\[ \mid \neg B \mid \ldots \]

Commands

\[ C \in \text{Comm} ::= \ \text{skip} \mid L := A \mid C; C \]

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

Semantic functions

\[ A : \text{Aexp} \rightarrow (\text{State} \rightarrow \mathbb{Z}) \]

where

\[ \mathbb{Z} = \{ \ldots, -1, 0, 1, \ldots \} \]

\[ \text{State} = (\mathbb{L} \rightarrow \mathbb{Z}) \]
Basic example of denotational semantics (II)

Semantic functions

\[ A : \text{Aexp} \rightarrow (\text{State} \rightarrow \mathbb{Z}) \]
\[ B : \text{Bexp} \rightarrow (\text{State} \rightarrow \mathbb{B}) \]

where

\[ \mathbb{Z} = \{ \ldots, -1, 0, 1, \ldots \} \]
\[ \mathbb{B} = \{ \text{true}, \text{false} \} \]
\[ \text{State} = (\mathbb{L} \rightarrow \mathbb{Z}) \]
Semantic functions

\[ \mathcal{A} : \text{Aexp} \rightarrow (\text{State} \rightarrow \mathbb{Z}) \]
\[ \mathcal{B} : \text{Bexp} \rightarrow (\text{State} \rightarrow \mathbb{B}) \]
\[ \mathcal{C} : \text{Comm} \rightarrow (\text{State} \rightarrow \text{State}) \]

where

\[ \mathbb{Z} = \{ \ldots, -1, 0, 1, \ldots \} \]
\[ \mathbb{B} = \{ \text{true}, \text{false} \} \]
\[ \text{State} = (\mathcal{L} \rightarrow \mathbb{Z}) \]
Basic example of denotational semantics (III)

Semantic function $\mathcal{A}$

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

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

$$\mathcal{A}[A_1 + A_2] = \lambda s \in State. \ \mathcal{A}[A_1](s) + \mathcal{A}[A_2](s)$$
Basic example of denotational semantics (IV)

Semantic function $\mathcal{B}$

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

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

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

where $eq(a, a') = \begin{cases} 
  true & \text{if } a = a' \\
  false & \text{if } a \neq a' 
\end{cases}$
Basic example of denotational semantics (V)

Semantic function $\mathcal{C}$

$$\llbracket \text{skip} \rrbracket = \lambda s \in \text{State}. s$$

**NB:** From now on the names of semantic functions are omitted!
A simple example of compositionality

Given partial functions \([C], [C'] : \text{State} \rightarrow \text{State}\) and a function \([B] : \text{State} \rightarrow \{\text{true}, \text{false}\}\), we can define

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

where

\[
\text{if} (b, x, x') = \begin{cases} 
  x & \text{if } b = \text{true} \\
  x' & \text{if } b = \text{false} 
\end{cases}
\]
Basic example of denotational semantics (VI)

Semantic function $C$

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

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

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

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

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

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

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

Cf. operational semantics of sequential composition:

$$C, s \Downarrow s' \quad C', s' \Downarrow s''$$

$\quad C; C', s \Downarrow s''$
[while $B$ do $C$]
Fixed point property of
\([\text{while } B \text{ do } C]\)

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

where, for each \(b : \text{State} \rightarrow \{\text{true, false}\}\) and \(c : \text{State} \rightarrow \text{State}\), we define

\[f_{b,c} : (\text{State} \rightarrow \text{State}) \rightarrow (\text{State} \rightarrow \text{State})\]

as

\[f_{b,c} = \lambda w \in (\text{State} \rightarrow \text{State}). \lambda s \in \text{State}. \text{if}(b(s), w(c(s)), s)\].
Fixed point property of 
\[
[\text{while } B \text{ do } C]\]

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

where, for each \( b : \text{State} \rightarrow \{\text{true, false}\} \) and 
\( c : \text{State} \rightarrow \text{State} \), we define

\[
f_{b,c} : (\text{State} \rightarrow \text{State}) \rightarrow (\text{State} \rightarrow \text{State})
\]

as

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

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

$$f_{[B],[C]}^{n}(\bot) = \lambda s \in \text{State.} \begin{cases} [C]^k(s) & \text{if } \exists 0 \leq k < n. [B]([C]^k(s)) = \text{false} \\
& \text{and } \forall 0 \leq i < k. [B]([C]^i(s)) = \text{true} \\
\uparrow & \text{if } \forall 0 \leq i < n. [B]([C]^i(s)) = \text{true} \end{cases}$$
\[ D \overset{\text{def}}{=} (\text{State} \rightarrow \text{State}) \]

- **Partial order \( \sqsubseteq \) on \( D \):**
  \[ \forall w, w' \quad w \sqsubseteq w' \iff \text{for all } s \in \text{State}, \text{ if } w \text{ is defined at } s \text{ then so is } w' \text{ and moreover } w(s) = w'(s). \]
  \[ \Downarrow \text{iff the graph of } w \text{ is included in the graph of } w'. \]

- **Least element \( \bot \in D \) w.r.t. \( \sqsubseteq \):**
  \[ \bot = \text{totally undefined partial function} \]
  \[ = \text{partial function with empty graph} \]
  (satisfies \( \bot \sqsubseteq w \), for all \( w \in D \)).