Denotational Semantics

10 lectures for Part II CST 2019/20

Marcelo Fiore

Course web page:
http://www.cl.cam.ac.uk/teaching/1920/DenotSem/
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

Syntax \[\rightarrow\] Semantics

\[
P \quad \mapsto \quad [P]
\]

Examples:
- Propositional formula \(\rightarrow\) Truth value
- Partial recursive function \(\rightarrow\) \(\mathbb{N} \rightarrow \mathbb{N}\)
Basic idea of denotational semantics

Syntax \( \rightarrow \) Semantics

Recursive program \( \leftrightarrow \) Partial recursive function

Boolean circuit \( \leftrightarrow \) Boolean function

\( P \) \( \leftrightarrow \) \([P]\)

Concerns:

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

- Compositionality.
  \( \leadsto \) Lectures 5 and 6.

- Relationship to computation (\textit{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, \([P]\) — 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

\[ A \in A_{\text{exp}} ::= n \mid L \mid A + A \mid \ldots \]
where \( n \) ranges over integers and \( L \) over a specified set of locations \( \text{IL} \)

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

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

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

the set of functions from the set of states to the set of integers.

where

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

The set of locations.

\[ \text{State} = (\mathbb{L} \rightarrow \mathbb{Z}) \]

Idea: \( s \in \text{State} \)

Then, for a location \( L \in \mathbb{L} \), \( s(L) \in \mathbb{Z} \) is the value of \( L \) in \( s \).
Basic example of denotational semantics (II)

Semantic functions

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

where

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

Semantic function $\mathcal{A}$

- $\mathcal{A}[n] = \lambda s \in \text{State}. n$
- $\mathcal{A}[L] = \lambda s \in \text{State}. s(L)$
- $\mathcal{A}[A_1 + A_2] = \lambda s \in \text{State}. \mathcal{A}[A_1](s) + \mathcal{A}[A_2](s)$
Semantics function $B$

\begin{align*}
B[\text{true}] &= \lambda s \in \text{State}. \text{true} \\
B[\text{false}] &= \lambda s \in \text{State}. \text{false} \\
B[A_1 = A_2] &= \lambda s \in \text{State}. \text{eq}(A[A_1](s), A[A_2](s))
\end{align*}

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

Semantic function $C$

\[ [\text{skip}] = \lambda s \in \text{State}. s \]

**NB:** From now on the names of semantic functions are omitted!
Given partial functions $[C], [C'] : \text{State} \rightarrow \text{State}$ and a function $[B] : \text{State} \rightarrow \{\text{true, false}\}$, 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 $C$

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

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

$$\begin{align*}
[C; C'] &= [C'] \circ [C] = \lambda s \in State. [C']([C](s))
\end{align*}$$

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

Syntax $\quad$ Semantics

Sequencing $\quad$ Composition

$\text{skip}; C \equiv C \equiv C; \text{skip}$

$(C_1; C_2); C_3 \equiv C_1; (C_2; C_3)$

$id \circ f = f = f \circ \text{id}$

$(f \circ f_2 \circ f_3 = f \circ (f_2 \circ f_3)$
Denotational semantics of sequential composition

Denotation of sequential composition \( C; C' \) of two commands

\[
[C; C'] = [C'] \circ [C] = \lambda s \in \text{State}. [C']( [C](s))
\]

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

Cf. operational semantics of sequential composition:

\[
\begin{array}{c}
C, s \Downarrow s' \quad C', s' \Downarrow s'' \\
\hline
C; C', s \Downarrow s''
\end{array}
\]
\[ \text{while } B \text{ do } C \]