Denotational Semantics

10 lectures for Part II CST 2012/13

Marcelo Fiore

Course web page:

http://www.cl.cam.ac.uk/teaching/1213/DenotSem/

Lecture 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. IB Operational Semantics

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

Axiomatic. II Hoare logic

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

Denotational. This course

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

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

$$P \quad \mapsto \quad [\![P]\!] \rightsquigarrow \text{the meaning of } P$$

Basic idea of denotational semantics

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

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

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

Basic idea of denotational semantics

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

Basic idea of denotational semantics

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

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

$$\text{Boolean circuit} \mapsto \text{Boolean function}$$

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

Concerns:

- Abstract models (*i.e.* implementation/machine independent).

  $\leadsto$ Lectures 2, 3 and 4.

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 (*i.e.* implementation/machine independent).

  $\rightsquigarrow$ Lectures 2, 3 and 4.

- Compositionality.

  $\rightsquigarrow$ Lectures 5 and 6.

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 (*i.e.* implementation/machine independent).

  $\rightsquigarrow$ Lectures 2, 3 and 4.

- Compositionality.

  $\rightsquigarrow$ Lectures 5 and 6.

- Relationship to computation (*e.g.* operational semantics).

  $\rightsquigarrow$ 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

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

where n ranges over *integers* and L over a specified set of *locations* $\mathbb{L}$

Boolean expressions

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

Commands

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

Basic example of denotational semantics (II)

For a state $s \in State = (\mathbb{L} \to \mathbb{Z})$ and $L \in \mathbb{L}$, $s(L) \in \mathbb{Z}$ is the value of L in state s.

The space of all functions from the set State to the set of integers $\mathbb{Z}$.

Semantic functions

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

$\mathcal{A}[\![A]\!] : State \to \mathbb{Z}$

$\forall s \in State.$ $\mathcal{A}[\![A]\!](s) \in \mathbb{Z}$

where

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

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

If X and Y are sets then $(X \to Y)$ is the set of functions from X to Y.

Basic example of denotational semantics (II)

Semantic functions

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

$$\mathcal{B}: \quad \mathbf{Bexp} \rightarrow (State \rightarrow \mathbb{B})$$

$$\mathcal{B}[\![b]\!](s) \in \mathbb{B}$$

where

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

Basic example of denotational semantics (II)

$\forall c \in$ Comm.

$\mathcal{C}[\![c]\!]$: State → State ⇝ A state transformer

Semantic functions

$$\mathcal{A}: \quad \textbf{Aexp} \to (State \to \mathbb{Z})$$

$$\mathcal{B}: \quad \textbf{Bexp} \to (State \to \mathbb{B})$$

$$\mathcal{C}: \quad \textbf{Comm} \to \underbrace{(State \rightharpoonup State)}$$

The set of partial functions from State to State.

where

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

$$\mathbb{B} \quad = \quad \{\, true, false \,\}$$

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

Basic example of denotational semantics (III)

Semantic function $\mathcal{A}$

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

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

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

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

where $n \in \mathbb{Z}$

$s \in State$

$$\mathcal{A}[\![n]\!] : State \to \mathbb{Z}$$

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

syntax for addition

The addition operation on integers.

$$\mathcal{A}[\![L]\!](s) = s(L) \in \mathbb{Z} \qquad s \in State = (\mathbb{L} \to \mathbb{Z})$$

Basic example of denotational semantics (IV)

Semantic function $\mathcal{B}$

$$
\begin{aligned}
\mathcal{B}[\![\mathbf{true}]\!] &= \lambda s \in State.\, true \\
\mathcal{B}[\![\mathbf{false}]\!] &= \lambda s \in State.\, false \\
\mathcal{B}[\![A_1 = A_2]\!] &= \lambda s \in State.\, eq\big(\mathcal{A}[\![A_1]\!](s), \mathcal{A}[\![A_2]\!](s)\big)
\end{aligned}
$$

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

$[\![C]\!] : State \to State$

Basic example of denotational semantics (V)

Semantic function $\mathcal{C}$

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

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

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

A simple example of compositionality

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

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

where

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

$[\![L := A]\!] = \ldots [\![A]\!] \ldots$

$[\![L := A]\!](s) =$ a new state where

for all locations $L' \neq L$ the value of L' is that in s

and for L we have the value $[\![A]\!](s)$

Basic example of denotational semantics (VI)

Semantic function $\mathcal{C}$

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

$[\![L := A]\!](s) : \mathbb{L} \to \mathbb{Z}$

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.

sequencing $\mapsto$ composition

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.

They match

Cf. operational semantics of sequential composition:

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

Prop: $C, s \Downarrow s'$ iff $[\![C]\!](s) = s'$. $\forall C \in Comm$, $s, s' \in State$

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