Denotational Semantics

10 lectures for Part II CST 2018/19

Andrew Pitts

Course web page:
http://www.cl.cam.ac.uk/teaching/1819/DenotSem/
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 \[\llbracket - \rrbracket\] \rightarrow Semantics

Recursive program \[\mapsto\] Partial recursive function
Boolean circuit \[\mapsto\] Boolean function

\[P \mapsto [P]\]

Concerns:

- Abstract models (i.e. implementation/machine independent).
  \[\leadsto\] first third

- Compositionality.
  \[\leadsto\] middle third

- Relationship to computation (e.g. operational semantics).
  \[\leadsto\] last third
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

\[ A : \ A_{exp} \rightarrow (\text{State} \rightarrow \mathbb{Z}) \]
\[ B : \ B_{exp} \rightarrow (\text{State} \rightarrow \mathbb{B}) \]
\[ C : \ \text{Comm} \rightarrow (\text{State} \rightarrow \text{State}) \]

where

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

Semantic functions

$$A : \text{Aexp} \to (\text{State} \to \mathbb{Z})$$

where

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

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

set of all (total) functions from set State to set $\mathbb{Z}$
Basic example of denotational semantics (III)

Semantic function $\mathcal{A}$

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

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

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

Semantic function $A$

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

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

$A[A_1 + A_2] = \lambda s \in State. A[A_1](s) + A[A_2](s)$

**Syntax**

**Semantics**
Basic example of denotational semantics (IV)

Semantic function $\mathcal{B}$

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

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

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

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 (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

\[ \mathbb{Z} = \{ \ldots, -1, 0, 1, \ldots \} \]
\[ \mathbb{B} = \{ \text{true, false} \} \]
\[ \text{State} = (\mathbb{L} \rightarrow \mathbb{Z}) \]
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 $\llbracket C \rrbracket, \llbracket C' \rrbracket : State \rightarrow State$ and a function $\llbracket B \rrbracket : State \rightarrow \{true, false\}$, we can define

$$\llbracket \text{if } B \text{ then } C \text{ else } C' \rrbracket = \lambda s \in \text{State}. \text{if} (\llbracket B \rrbracket (s), \llbracket C \rrbracket (s), \llbracket C' \rrbracket (s))$$

where

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

($x$ & $x'$ are states, or undefined)
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

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

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.

$[[C']([C](s))]$ is undefined if

- either $[[C](s)]$ is undefined
- or $[[C](s)] = s'$, say, and $[[C']](s')$ is undefined.
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:

$C, s \Downarrow s' \quad C', s' \Downarrow s'' \quad \Rightarrow \quad C; C', s \Downarrow s''$. 

[while $B$ do $C$]

Extend the language $\text{IMP}$ to a language $\text{IMP}'$ by extending the grammar of commands:

$$C \in \text{Comm} ::= \ldots | \text{while } B \text{ do } C$$
[while \( B \) do \( C \)]

Operational semantics of while-loops

\[ \langle \text{while } B \text{ do } C, s \rangle \rightarrow \langle \text{if } B \text{ then } C ; (\text{while } B \text{ do } C) \text{ else skip }, s \rangle \]

Suggests looking for a denotation \([\text{while } B \text{ do } C]\)

Satisfying

\[ [\text{while } B \text{ do } C] = [\text{if } B \text{ then } C ; (\text{while } B \text{ do } C) \text{ else skip }] \]
Fixed point property of

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

where, for each \( b : \text{State} \rightarrow \{ \text{true}, \text{false} \} \) 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 \( \text{[while } B \text{ do } C \text{]} \)?
\[ D \overset{\text{def}}{=} (State \to State) \]

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

- **Least element \( \bot \in D \) w.r.t. \( \sqsubseteq \):**
  \[ \bot = \text{totally undefined partial function} \]
  \[ = \text{partial function with empty graph} \]
  \[ (\text{satisfies } \bot \sqsubseteq w, \text{ for all } w \in D). \]
while $X > 0$ do $(Y := X \times Y ; X := X - 1)$

Let

$\text{State} \overset{\text{def}}{=} (\mathbb{L} \rightarrow \mathbb{Z})$ \hspace{1cm} \text{integer assignments to locations}

$D \overset{\text{def}}{=} (\text{State} \rightarrow \text{State})$ \hspace{1cm} \text{partial functions on states}

For $\left[ \text{while } X > 0 \text{ do } Y := X \times Y ; X := X - 1 \right] \in D$ we seek a minimal solution to $w = f(w)$, where $f : D \rightarrow D$ is defined by:

$$f(w)\left( [X \mapsto x, Y \mapsto y] \right) = \begin{cases} 
[X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\
 w\left( [X \mapsto x - 1, Y \mapsto x \times y] \right) & \text{if } x > 0.
\end{cases}$$
$f : D \rightarrow D$ is given by

$$f(w) [x \mapsto x, y \mapsto y] = \begin{cases} [x \mapsto x, y \mapsto y] & \text{if } x \leq 0 \\ w [x \mapsto x-1, y \mapsto x\cdot y] & \text{if } x > 0 \end{cases}$$

Want to find $w \in D$ s.t. $w = f(w)$

Define $w_0 = 1$, $w_1 = f(1)$, $w_2 = f(f(1))$, etc.

$$w_0 [x \mapsto x, y \mapsto y] = \text{undefined}$$
$f : D \rightarrow D$ is given by

$$f(w) [x \mapsto x, y \mapsto y] = \begin{cases} [x \mapsto x, y \mapsto y] & \text{if } x \leq 0 \\ w [x \mapsto x-1, y \mapsto x \cdot y] & \text{if } x > 0 \end{cases}$$

Want to find $w \in D$ s.t. $w = f(w)$

Define $w_0 = 1$, $w_1 = f(1)$, $w_2 = f(f(1))$, etc.

$$w_1 [x \mapsto x, y \mapsto y] = \begin{cases} [x \mapsto x, y \mapsto y] & \text{if } x \leq 0 \\ \text{undefined} & \text{if } x \geq 1 \end{cases}$$
\( f : D \rightarrow D \) is given by

\[
f(w) [x\mapsto x, y\mapsto y] = \begin{cases} 
[x\mapsto x, y\mapsto y] & \text{if } x \leq 0 \\
 w [x\mapsto x-1, y\mapsto xy] & \text{if } x > 0
\end{cases}
\]

Want to find \( w \in D \) s.t. \( w = f(w) \)

Define \( w_0 = 1, w_1 = f(1), w_2 = f(f(1)), \) etc.

\[
w_2 [x\mapsto x, y\mapsto y] = \begin{cases} 
[x\mapsto x, y\mapsto y] & \text{if } x \leq 0 \\
[x\mapsto 0, y\mapsto y] & \text{if } x = 1 \\
\text{undefined} & \text{if } x > 2
\end{cases}
\]
$f : D \rightarrow D$ is given by

$$f(w) [x \mapsto x, y \mapsto y] = \begin{cases} [x \mapsto x, y \mapsto y] & \text{if } x \leq 0 \\ w [x \mapsto x-1, y \mapsto xy] & \text{if } x > 0 \end{cases}$$

Want to find $w \in D$ s.t. $w = f(w)$

Define $w_0 = 1$, $w_1 = f(1)$, $w_2 = f(f(1))$, etc.

$$w_2 [x \mapsto x, y \mapsto y] = \begin{cases} [x \mapsto x, y \mapsto y] & \text{if } x \leq 0 \\ [x \mapsto 0, y \mapsto y] & \text{if } x = 1 \\ [x \mapsto 0, y \mapsto 2y] & \text{if } x = 2 \\ \text{undefined} & \text{if } x > 3 \end{cases}$$
\( f : D \to D \) is given by
\[
f(w) [x \mapsto x, y \mapsto y] = \begin{cases} 
[X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\
W [X \mapsto x-1, Y \mapsto xy] & \text{if } x > 0 
\end{cases}
\]

Want to find \( w \in D \) s.t. \( w = f(w) \)

Define \( w_0 = 1, w_1 = f(1), w_2 = f(f(1)), \) etc.

\[
w_4 [x \mapsto x, y \mapsto y] = \begin{cases} 
[X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\
[X \mapsto 0, Y \mapsto y] & \text{if } x = 1 \\
[X \mapsto 0, Y \mapsto 2y] & \text{if } x = 2 \\
[X \mapsto 0, Y \mapsto 6y] & \text{if } x = 3 \\
\text{undefined} & \text{if } x \geq 4
\end{cases}
\]
\( f : D \rightarrow D \) is given by
\[
f(w) [x \mapsto x, y \mapsto y] = \begin{cases} 
[X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\
W [X \mapsto x-1, Y \mapsto x*y] & \text{if } x > 0
\end{cases}
\]

Want to find \( w \in D \) s.t. \( w = f(w) \)

Define \( w_0 = 1 \), \( w_1 = f(1) \), \( w_2 = f(f(1)) \), etc.

Union \( w_\infty = w_0 \cup w_1 \cup w_2 \cup \ldots \) is the function
\[
w_\infty [x \mapsto x, y \mapsto y] = \begin{cases} 
[X \mapsto x, Y \mapsto y] & \text{if } x \leq 0 \\
[X \mapsto 0, Y \mapsto !x*y] & \text{if } x > 0
\end{cases}
\]
$f: D \to D$ is given by

$$
f(w) = \begin{cases} 
[x \mapsto x, y \mapsto y] & \text{if } x \leq 0 \\
 w[x \mapsto x-1, y \mapsto x \times y] & \text{if } x > 0
\end{cases}
$$

Want to find $w \in D$ s.t. $w = f(w)$

Define $w_0 = 1$, $w_1 = f(1)$, $w_2 = f(f(1))$, etc.

Union $w_\infty = w_0 \cup w_1 \cup w_2 \cup \ldots$ is the function

$$
w_\infty[x \mapsto x, y \mapsto y] = \begin{cases} 
[x \mapsto x, y \mapsto y] & \text{if } x \leq 0 \\
 [x \mapsto 0, y \mapsto \lnot x \times y] & \text{if } x > 0
\end{cases}
$$

It satisfies $w_\infty = f(w_\infty)$ — fixed point we seek for definition of $\lceil \text{while } \text{true} \rightarrow \text{do } (y := y \times x; x := x - 1) \rfloor$
\[ f : D \rightarrow D \text{ is given by} \]
\[
f(w) \left[ x \mapsto x, \ y \mapsto y \right] = \begin{cases} 
[x \mapsto x, \ y \mapsto y] & \text{if } x \leq 0 \\
 w \left[ x \mapsto x-1, \ y \mapsto x \ast y \right] & \text{if } x > 0
\end{cases}
\]

Want to find \( w \in D \) s.t. \( w = f(w) \)

Define \( w_0 = 1 \), \( w_1 = f(1) \), \( w_2 = f(f(1)) \), etc.

Union \( w_\infty = w_0 \cup w_1 \cup w_2 \cup \ldots \) is the function
\[
w_\infty \left[ x \mapsto x, \ y \mapsto y \right] = \begin{cases} 
[x \mapsto x, \ y \mapsto y] & \text{if } x \leq 0 \\
 [x \mapsto 0, \ y \mapsto \lfloor x \ast y \rfloor] & \text{if } x > 0
\end{cases}
\]

It satisfies \( w_\infty = f(w_\infty) \) and \( (\forall w) \ w = f(w) \Rightarrow w_\infty \subseteq w \) — \( w_\infty \) is a least fixed point for \( f \)