Harvard School of Engineering and Applied Sciences — CS 152: Programming Languages

Notes on Denotational Semantics

Spring 2017

1 Denotational semantics of IMP

We have seen two operational models for programming languages: small-step and large-step. We now consider a different semantic model, called *denotational semantics.*

By way of motivation, consider our definition of command equivalence in IMP: $c \sim c'$ when c and c' evaluate to the same resulting store when executed in the same initial store. In terms of large-step semantics, when $\langle c, \sigma\rangle \Downarrow \sigma'$ exactly when $\langle c', \sigma\rangle \Downarrow \sigma'$ for all $\sigma, \sigma' \in \mathbf{Store}$. Therefore, as far as command equivalence is concerned, the only thing we need from the semantics is which store results from executing a command in a given initial store. The details of how that execution proceeds is irrelevant.

The idea in denotational semantics is to associate with every program a *mathematical object* that captures what the program computes. For IMP, we can think of a program c as a function from stores to stores: given an an initial store, the program produces a final store. For example, the program $\mathsf{foo} := \mathsf{bar} + 1$ can be thought of as a function that when given an input store σ, produces a final store σ' that is identical to σ except that it maps foo to the integer $\sigma(\mathsf{bar}) + 1$; that is, $\sigma' = \sigma[\mathsf{foo} \mapsto \sigma(\mathsf{bar}) + 1]$.

As opposed to operational models, which tell us *how* programs execute, denotational model shows us *what* programs compute. One benefit that this kind of model will give us is *compositionality*: the objects corresponding to a complex program will be constructed as a function of the objects corresponding to the parts of the program.

For IMP, we are going to model programs as functions from input stores to output stores. (Denotational semantics for other programming languages may associate different mathematical objects with programs: relations, probability distributions, sets of traces, trees, etc.)

For a program c (a piece of syntax), we write $\mathcal{C}[\![c]\!]$ for the *denotation* of c, that is, the mathematical function that c represents:

$$\mathcal{C}[\![c]\!] : \mathbf{Store} \rightharpoonup \mathbf{Store}.$$

Note that $\mathcal{C}[\![c]\!]$ is actually a partial function (as opposed to a total function), because the program may not terminate for certain input stores; $\mathcal{C}[\![c]\!]$ is not defined for those inputs, since they have no corresponding output stores.

We write $\mathcal{C}[\![c]\!]\sigma$ for the result of applying the function $\mathcal{C}[\![c]\!]$ to the store σ. That is, if f is the function $\mathcal{C}[\![c]\!]$, then we write $\mathcal{C}[\![c]\!]\sigma$ to mean the same thing as $f(\sigma)$.

We must also model expressions as functions, this time from stores to the values they represent. We will write $\mathcal{A}[\![a]\!]$ for the denotation of arithmetic expression a, and $\mathcal{B}[\![b]\!]$ for the denotation of boolean expression b. Note that $\mathcal{A}[\![a]\!]$ and $\mathcal{B}[\![b]\!]$ are total functions.

$$\begin{aligned}\mathcal{A}[\![a]\!] &: \mathbf{Store} \to \mathbf{Int}\\ \mathcal{B}[\![b]\!] &: \mathbf{Store} \to \{\mathsf{true}, \mathsf{false}\}\end{aligned}$$

Now we want to define these functions. We start with the denotations of expressions:

$$\begin{aligned}\mathcal{A}[\![n]\!]\sigma &= n\\ \mathcal{A}[\![x]\!]\sigma &= \sigma(x)\\ \mathcal{A}[\![a_1 + a_2]\!]\sigma &= \mathcal{A}[\![a_1]\!]\sigma + \mathcal{A}[\![a_2]\!]\sigma\\ \mathcal{A}[\![a_1 \times a_2]\!]\sigma &= \mathcal{A}[\![a_1]\!]\sigma \times \mathcal{A}[\![a_2]\!]\sigma\end{aligned}$$

$$
\begin{aligned}
\mathcal{B}[\![\textsf{true}]\!]\sigma &= \textsf{true}\\
\mathcal{B}[\![\textsf{false}]\!]\sigma &= \textsf{false}\\
\mathcal{B}[\![a_1 < a_2]\!]\sigma &= \begin{cases}\textsf{true} & \text{if } \mathcal{A}[\![a_1]\!]\sigma < \mathcal{A}[\![a_2]\!]\sigma\\ \textsf{false} & \text{otherwise}\end{cases}
\end{aligned}
$$

The denotations for commands are partial functions, and to make it easier to write down those definitions, we will express functions as sets of pairs—namely, as the *graphs* of the functions. More precisely, we will represent a partial map $f : A \rightharpoonup B$ as a set of pairs $F = \{(a, b) \mid a \in A \text{ and } b = f(a) \in B\}$ such that, for each $a \in A$, there is at most one pair of the form $(a, \_)$ in the set. Hence $(a, b) \in F$ is the same as $b = f(a)$. (We could have used graphs of functions to write the denotations for $\mathcal{A}[\![a]\!]$ and $\mathcal{B}[\![b]\!]$ as well, though it is less compelling for total functions.)

The denotations for commands are as follows:

$$
\begin{aligned}
\mathcal{C}[\![\textsf{skip}]\!] &= \{(\sigma, \sigma)\}\\
\mathcal{C}[\![x := a]\!] &= \{(\sigma, \sigma[x \mapsto n]) \mid \mathcal{A}[\![a]\!]\sigma = n\}\\
\mathcal{C}[\![c_1; c_2]\!] &= \{(\sigma, \sigma') \mid \exists\sigma''.\ ((\sigma, \sigma'') \in \mathcal{C}[\![c_1]\!] \wedge (\sigma'', \sigma') \in \mathcal{C}[\![c_2]\!])\}
\end{aligned}
$$

Note that $\mathcal{C}[\![c_1; c_2]\!] = \mathcal{C}[\![c_2]\!] \circ \mathcal{C}[\![c_1]\!]$, where $\circ$ is the composition of relations. (Composition of relations is defined as follows: if $R_1 \subseteq A \times B$ and $R_2 \subseteq B \times C$ then $R_2 \circ R_1 \subseteq A \times C$ is $R_2 \circ R_1 = \{(a, c) \mid \exists b \in B.\ (a, b) \in R_1 \wedge (b, c) \in R_2\}$.) If $\mathcal{C}[\![c_1]\!]$ and $\mathcal{C}[\![c_2]\!]$ are total functions, then $\circ$ is function composition.

$$
\begin{aligned}
\mathcal{C}[\![\textsf{if } b \textsf{ then } c_1 \textsf{ else } c_2]\!] &= \{(\sigma, \sigma') \mid \mathcal{B}[\![b]\!]\sigma = \textsf{true} \wedge (\sigma, \sigma') \in \mathcal{C}[\![c_1]\!]\} \ \cup\\
&\qquad \{(\sigma, \sigma') \mid \mathcal{B}[\![b]\!]\sigma = \textsf{false} \wedge (\sigma, \sigma') \in \mathcal{C}[\![c_2]\!]\}\\
\mathcal{C}[\![\textsf{while } b \textsf{ do } c]\!] &= \{(\sigma, \sigma) \mid \mathcal{B}[\![b]\!]\sigma = \textsf{false}\} \ \cup\\
&\qquad \{(\sigma, \sigma') \mid \mathcal{B}[\![b]\!]\sigma = \textsf{true} \wedge \exists\sigma''.\ ((\sigma, \sigma'') \in \mathcal{C}[\![c]\!] \wedge (\sigma'', \sigma') \in \mathcal{C}[\![\textsf{while } b \textsf{ do } c]\!])\}
\end{aligned}
$$

But now we've got a problem: the last "definition" is not really a definition, it expresses $\mathcal{C}[\![\textsf{while } b \textsf{ do } c]\!]$ in terms of itself! It is not a definition, but a recursive equation. What we want is the solution to this equation, i.e., we want to find a function f, such that f satisfies the following equation, and we will take the semantics of a while loop to be that function f.

$$
\begin{aligned}
f = \{&(\sigma, \sigma) \mid \mathcal{B}[\![b]\!]\sigma = \textsf{false}\} \ \cup\\
&\{(\sigma, \sigma') \mid \mathcal{B}[\![b]\!]\sigma = \textsf{true} \wedge \exists\sigma''.\ ((\sigma, \sigma'') \in \mathcal{C}[\![c]\!] \wedge (\sigma'', \sigma') \in f)\}
\end{aligned}
$$

I claim that the following is a solution to the equation for $\mathcal{C}[\![\textsf{while } b \textsf{ do } c]\!]$, and moreover, the *right* solution for our purposes:

$$
\begin{aligned}
\mathcal{C}[\![\textsf{while } b \textsf{ do } c]\!] &= \bigcup_{i \geq 0} {F_{b,c}}^i(\varnothing)\\
&= \varnothing \cup F_{b,c}(\varnothing) \cup F_{b,c}(F_{b,c}(\varnothing)) \cup F_{b,c}(F_{b,c}(F_{b,c}(\varnothing))) \cup \ldots
\end{aligned}
$$

where

$$
\begin{aligned}
F_{b,c}(f) = \{&(\sigma, \sigma) \mid \mathcal{B}[\![b]\!]\sigma = \textsf{false}\} \ \cup\\
&\{(\sigma, \sigma') \mid \mathcal{B}[\![b]\!]\sigma = \textsf{true} \wedge \exists\sigma''.\ ((\sigma, \sigma'') \in \mathcal{C}[\![c]\!] \wedge (\sigma'', \sigma') \in f)\}
\end{aligned}
$$

This looks like it's coming out of left field. We will come back to this next time. The first thing to do is to check that this is a solution to the equation. In other words, we have to check that this definition satisfies:

$$
\begin{aligned}
\mathcal{C}[\![\textsf{while } b \textsf{ do } c]\!] = \{&(\sigma, \sigma) \mid \mathcal{B}[\![b]\!]\sigma = \textsf{false}\} \ \cup\\
&\{(\sigma, \sigma') \mid \mathcal{B}[\![b]\!]\sigma = \textsf{true} \wedge \exists\sigma''.\ ((\sigma, \sigma'') \in \mathcal{C}[\![c]\!] \wedge (\sigma'', \sigma') \in \mathcal{C}[\![\textsf{while } b \textsf{ do } c]\!])\}
\end{aligned}
$$

This is left as an exercise to the reader.

This definition completes the definition of $\mathcal{C}[\![c]\!]$. It is not entirely clear that this defines a partial function **Store** $\rightharpoonup$ **Store**, but that fact is reasonably easy to establish.

Proposition 1. *For all $c \in$ **Com** and all $\sigma, \sigma_1, \sigma_2 \in$ **Store**, if $(\sigma, \sigma_1) \in \mathcal{C}[\![c]\!]$ and $(\sigma, \sigma_2) \in \mathcal{C}[\![c]\!]$, then $\sigma_1 = \sigma_2$.*

Proof. An easy structural induction on commands. □

Let's consider an example: $\textsf{\textbf{while}}\ \mathsf{foo} < \mathsf{bar}\ \textsf{\textbf{do}}\ \mathsf{foo} := \mathsf{foo} + 1$. Here $b = \mathsf{foo} < \mathsf{bar}$ and $c = \mathsf{foo} := \mathsf{foo} + 1$.

$$\begin{aligned}
F_{b,c}(\varnothing) &= \{(\sigma,\sigma) \mid \mathcal{B}[\![b]\!]\sigma = \textsf{\textbf{false}}\} \ \cup \\
&\qquad \{(\sigma,\sigma') \mid \mathcal{B}[\![b]\!]\sigma = \textsf{\textbf{true}} \wedge \exists\sigma''.\ ((\sigma,\sigma'') \in \mathcal{C}[\![c]\!] \wedge (\sigma'',\sigma') \in \varnothing)\} \\
&= \{(\sigma,\sigma) \mid \sigma(\mathsf{foo}) \geq \sigma(\mathsf{bar})\}
\end{aligned}$$

$$\begin{aligned}
{F_{b,c}}^2(\varnothing) &= \{(\sigma,\sigma) \mid \mathcal{B}[\![b]\!]\sigma = \textsf{\textbf{false}}\} \ \cup \\
&\qquad \{(\sigma,\sigma') \mid \mathcal{B}[\![b]\!]\sigma = \textsf{\textbf{true}} \wedge \exists\sigma''.\ ((\sigma,\sigma'') \in \mathcal{C}[\![c]\!] \wedge (\sigma'',\sigma') \in F_{b,c}(\varnothing))\} \\
&= \{(\sigma,\sigma) \mid \sigma(\mathsf{foo}) \geq \sigma(\mathsf{bar})\} \ \cup \\
&\qquad \{(\sigma,\sigma[\mathsf{foo} \mapsto \sigma(\mathsf{foo}) + 1]) \mid \sigma(\mathsf{foo}) < \sigma(\mathsf{bar}) \wedge \sigma(\mathsf{foo}) + 1 \geq \sigma(\mathsf{bar})\}
\end{aligned}$$

But if $\sigma(\mathsf{foo}) < \sigma(\mathsf{bar}) \wedge \sigma(\mathsf{foo}) + 1 \geq \sigma(\mathsf{bar})$ then $\sigma(\mathsf{foo}) + 1 = \sigma(\mathsf{bar})$, so we can simplify further:

$$\begin{aligned}
&= \{(\sigma,\sigma) \mid \sigma(\mathsf{foo}) \geq \sigma(\mathsf{bar})\} \ \cup \\
&\qquad \{(\sigma,\sigma[\mathsf{foo} \mapsto \sigma(\mathsf{foo}) + 1]) \mid \sigma(\mathsf{foo}) + 1 = \sigma(\mathsf{bar})\}
\end{aligned}$$

$$\begin{aligned}
{F_{b,c}}^3(\varnothing) &= \{(\sigma,\sigma) \mid \mathcal{B}[\![b]\!]\sigma = \textsf{\textbf{false}}\} \ \cup \\
&\qquad \{(\sigma,\sigma') \mid \mathcal{B}[\![b]\!]\sigma = \textsf{\textbf{true}} \wedge \exists\sigma''.\ ((\sigma,\sigma'') \in \mathcal{C}[\![c]\!] \wedge (\sigma'',\sigma') \in {F_{b,c}}^2(\varnothing))\} \\
&= \{(\sigma,\sigma) \mid \sigma(\mathsf{foo}) \geq \sigma(\mathsf{bar})\} \ \cup \\
&\qquad \{(\sigma,\sigma[\mathsf{foo} \mapsto \sigma(\mathsf{foo}) + 1]) \mid \sigma(\mathsf{foo}) + 1 = \sigma(\mathsf{bar})\} \ \cup \\
&\qquad \{(\sigma,\sigma[\mathsf{foo} \mapsto \sigma(\mathsf{foo}) + 2]) \mid \sigma(\mathsf{foo}) + 2 = \sigma(\mathsf{bar})\}
\end{aligned}$$

$$\begin{aligned}
{F_{b,c}}^4(\varnothing) &= \{(\sigma,\sigma) \mid \sigma(\mathsf{foo}) \geq \sigma(\mathsf{bar})\} \ \cup \\
&\qquad \{(\sigma,\sigma[\mathsf{foo} \mapsto \sigma(\mathsf{foo}) + 1]) \mid \sigma(\mathsf{foo}) + 1 = \sigma(\mathsf{bar})\} \ \cup \\
&\qquad \{(\sigma,\sigma[\mathsf{foo} \mapsto \sigma(\mathsf{foo}) + 2]) \mid \sigma(\mathsf{foo}) + 2 = \sigma(\mathsf{bar})\} \ \cup \\
&\qquad \{(\sigma,\sigma[\mathsf{foo} \mapsto \sigma(\mathsf{foo}) + 3]) \mid \sigma(\mathsf{foo}) + 3 = \sigma(\mathsf{bar})\}
\end{aligned}$$

If we take the union of all ${F_{b,c}}^i(\varnothing)$, we get the expected semantics of the loop.

$$\begin{aligned}
\mathcal{C}[\![\textsf{\textbf{while}}\ \mathsf{foo} < \mathsf{bar}\ \textsf{\textbf{do}}\ \mathsf{foo} := \mathsf{foo} + 1]\!] = \{(\sigma,\sigma) \mid \sigma(\mathsf{foo}) \geq \sigma(\mathsf{bar})\} \ \cup& \\
\{(\sigma,\sigma[\mathsf{foo} \mapsto \sigma(\mathsf{foo}) + n]) \mid \sigma(\mathsf{foo}) + n = \sigma(\mathsf{bar}) \wedge n \geq 1\}&
\end{aligned}$$

2 Equivalence with Operational Semantics

How do we justify that our solution for $\mathcal{C}[\![\textsf{\textbf{while}}\ b\ \textsf{\textbf{do}}\ c]\!]$ (and therefore, our definition of $\mathcal{C}[\![c]\!]$) is indeed the right definition? One way is to make sure that the denotational semantics agrees with the operational semantics we have already given for IMP.

Theorem 1. *For all $c \in$ **Com** and all $\sigma, \sigma' \in$ **Store**, $(\sigma, \sigma') \in \mathcal{C}[\![c]\!]$ if and only if $\langle c, \sigma \rangle \Downarrow \sigma'$.*

The proof of this result relies on the following lemma, which establishes a similar result for expressions.

Lemma 1.

(i) For all $a \in \boldsymbol{AExp}, \sigma \in \boldsymbol{Store}, n \in \boldsymbol{Int}$, $\mathcal{A}[\![a]\!]\sigma = n$ *if and only if* $\langle a, \sigma \rangle \Downarrow n$.

(ii) For all $b \in \boldsymbol{BExp}, \sigma \in \boldsymbol{Store}, b_0 \in \{\textsf{true}, \textsf{false}\}$, $\mathcal{B}[\![b]\!]\sigma = b_0$ *if and only if* $\langle b, \sigma \rangle \Downarrow b_0$.

This lemma is an easy proof by structural induction on expressions for both (i) and (ii), in both directions.

Proof of Theorem 1. We prove the ($\Rightarrow$) direction by structural induction on commands. The reverse ($\Leftarrow$) direction is proved by structural induction on derivations. $\square$