1 Denotational Semantics

So far we have looked at

- *operational semantics* involving rules for state transitions,
- *definitional semantics* involving translations from a source language to a target language, where the target language is simpler or better understood, and
- *axiomatic semantics* involving logical rules for deriving relations between preconditions and postconditions.

Another approach, known as *denotational semantics*, involves translations to mathematical objects. The objects in question are functions with well-defined extensional meaning in terms of sets. The main challenge is getting a precise understanding of the sets over which these function operate.

2 Diagonalization

For example, consider the identity function $\lambda x.x$. This clearly represents some kind of function that takes any input object x to itself. But what is its domain? An even more interesting example is the function $\lambda x.\, xx$. Let's say that the domain of this function is D. Then x represents some element of D, since x is an input to the function. But in the body, x is applied to x, so x must also represent some function $D \to E$. For this to make sense, it must be possible to interpret every element of D as an element of $D \to E$. Thus there must be a function $f : D \to (D \to E)$ taking $x \in D$ to the function $f(x) : D \to E$ that it represents.

It is conceivable that f could actually be a bijection between D and the function space $D \to E$. However, this is impossible if E contains more than one element. In fact, f cannot even be onto. This follows by a diagonalization argument. Let $e_0, e_1 \in E$, $e_0 \neq e_1$. For any function $f : D \to (D \to E)$, we can define $d : D \to E$ by $d = \lambda x.\, \mathsf{if}\ f\,x\,x = e_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_0$. Then for all x, $d\,x \neq f\,x\,x$, so $d \neq f\,x$ for any x.

This type of argument is called *diagonalization* because for countable sets D, the function d is constructed by arranging the values $f\,x\,y$ for $x, y \in D$ in a countable matrix and going down the diagonal, creating a function that is different from every $f\,x$ on at least one input (namely x).

	0	1	2	
f_0	$f_0\,0$	$f_0\,1$	$f_0\,2$	$\ldots$
f_1	$f_1\,0$	$f_1\,1$	$f_1\,2$	$\ldots$
f_2	$f_2\,0$	$f_2\,1$	$f_2\,2$	$\ldots$
$\vdots$	$\vdots$			

Thus for any function $f : D \to (D \to E)$, where E contains more than one element, there always exists a function $d : D \to E$ that is not $f\,x$ for any $x \in D$. This says that the cardinality (size) of the set of functions $D \to E$ is strictly larger than the cardinality of D, no matter what the sets D and E are, as long as E contains at least two elements.

3 Denotational Semantics of IMP

When defining denotational semantics, we will use the notation $\lambda x \in D.\, e$ to indicate that the domain of the function is the set D. This will ensure that we are precise in identifying the extension of functions.

This is not really a type declaration. Later, we will introduce types and write them as $\lambda x : \tau.\, e$. The distinction is that types are pieces of language syntax, whereas sets are semantic objects.

The syntax of IMP was

$$\begin{array}{lcl} a & ::= & n \mid x \mid a_0 \oplus a_1 \\ b & ::= & \mathsf{true} \mid \mathsf{false} \mid \neg b \mid b_0 \wedge b_1 \mid a_0 = a_1 \mid \cdots \\ c & ::= & \mathsf{skip} \mid x := a \mid c_0; c_1 \mid \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \mid \mathsf{while}\ b\ \mathsf{do}\ c \end{array}$$

The syntactic categories of a, b, c are arithmetic expressions *AExp*, Boolean expressions *BExp*, and commands *Com*, respectively.

To define the denotational semantics, we will refer to *states*, which are functions $\Sigma = \mathsf{Var} \to \mathbb{Z}$.

$$\begin{array}{rcll} \mathcal{A}[\![a]\!] & \in & \Sigma \to \mathbb{Z} & \\ \mathcal{B}[\![b]\!] & \in & \Sigma \to \mathbb{2} & \text{where } \mathbb{2} = \{\mathit{true}, \mathit{false}\} \\ \mathcal{C}[\![c]\!] & \in & \Sigma \to ? & \end{array}$$

Intuitively, we would like the meaning of commands to be functions from states to states. Given an initial state, the function produces the final state reached by applying the command. However, there will be no such final state if the program does not terminate (e.g., $\mathsf{while\ true\ do\ skip}$). Thus the function would have to be partial. However, we can make it a total function by including a special element $\perp$ (called *bottom*) denoting nontermination. For any set S, let $S_\perp \triangleq S \cup \{\perp\}$. Then $\mathcal{C}[\![c]\!] \in \Sigma \to \Sigma_\perp$, where $\mathcal{C}[\![c]\!]\sigma = \tau$ if c terminates in state τ on input state σ, and $\mathcal{C}[\![c]\!]\sigma = \perp$ if c does not terminate on input state σ.

Now we can define the denotational semantics of expressions by structural induction. This induction is a little more complicated since we are defining all three functions at once. However, it is still well-founded because we only use the function value on subexpressions in the definitions. For numbers,

$$\mathcal{A}[\![n]\!] \triangleq \lambda\sigma \in \Sigma.n = \{(\sigma, n) \mid \sigma \in \Sigma\}.$$

For the remaining definitions, we use the shorthand of defining the value of the function given some $\sigma \in \Sigma$.

$$\begin{array}{rcl} \mathcal{A}[\![x]\!]\sigma & \triangleq & \sigma(x) \\ \mathcal{A}[\![a_1 \oplus a_2]\!]\sigma & \triangleq & \mathcal{A}[\![a_1]\!]\sigma \oplus \mathcal{A}[\![a_2]\!]\sigma \\ \mathcal{B}[\![\mathsf{true}]\!]\sigma & \triangleq & \mathit{true} \\ \mathcal{B}[\![\mathsf{false}]\!]\sigma & \triangleq & \mathit{false} \\ \mathcal{B}[\![\neg b]\!]\sigma & \triangleq & \begin{cases} \mathit{true}, & \text{if } \mathcal{B}[\![b]\!]\sigma = \mathit{false}, \\ \mathit{false}, & \text{if } \mathcal{B}[\![b]\!]\sigma = \mathit{true}. \end{cases} \end{array}$$

We can express negation more compactly with a meta-conditional expression:

$$\mathcal{B}[\![\neg b]\!]\sigma \triangleq \mathit{if}\ \mathcal{B}[\![b]\!]\sigma\ \mathit{then\ false\ else\ true}.$$

Alternatively, we can write down the function extensionally as a set of ordered pairs:

$$\{(\sigma, \mathit{true}) \mid \sigma \in \Sigma \wedge \neg\mathcal{B}[\![b]\!]\sigma\} \cup \{(\sigma, \mathit{false}) \mid \sigma \in \Sigma \wedge \mathcal{B}[\![b]\!]\sigma\}.$$

For commands, we can define

$$\begin{aligned}
\mathcal{C}[\![\mathsf{skip}]\!]\sigma &\triangleq \sigma \\
\mathcal{C}[\![x := a]\!]\sigma &\triangleq \sigma[\mathcal{A}[\![a]\!]\sigma/x] \\
\mathcal{C}[\![\mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2]\!]\sigma &\triangleq \begin{cases} \mathcal{C}[\![c_1]\!]\sigma, & \text{if } \mathcal{B}[\![b]\!]\sigma = \mathit{true}, \\ \mathcal{C}[\![c_2]\!]\sigma, & \text{if } \mathcal{B}[\![b]\!]\sigma = \mathit{false} \end{cases} \\
&= \mathit{if}\ \mathcal{B}[\![b]\!]\sigma\ \mathit{then}\ \mathcal{C}[\![c_1]\!]\sigma\ \mathit{else}\ \mathcal{C}[\![c_2]\!]\sigma \\
\mathcal{C}[\![c_1\ ;\ c_2]\!]\sigma &\triangleq \begin{cases} \mathcal{C}[\![c_2]\!]\left(\mathcal{C}[\![c_1]\!]\sigma\right), & \text{if } \mathcal{C}[\![c_1]\!]\sigma \neq \bot, \\ \bot, & \text{if } \mathcal{C}[\![c_1]\!]\sigma = \bot \end{cases} \\
&= \mathit{if}\ \mathcal{C}[\![c_1]\!]\sigma = \bot\ \mathit{then}\ \bot\ \mathit{else}\ \mathcal{C}[\![c_2]\!]\left(\mathcal{C}[\![c_1]\!]\sigma\right).
\end{aligned}$$

For the last case involving sequential composition $c_1\ ;\ c_2$, another way of achieving this effect is by defining a *lift* operator $(\,\cdot\,)^* : (D \to E_\bot) \to (D_\bot \to E_\bot)$ on functions:

$$(f)^* \triangleq \lambda x.\mathit{if}\ x = \bot\ \mathit{then}\ \bot\ \mathit{else}\ f(x).$$

With this notation, we have

$$\mathcal{C}[\![c_1\ ;\ c_2]\!]\sigma \triangleq \left(\mathcal{C}[\![c_2]\!]\right)^*\left(\mathcal{C}[\![c_1]\!]\sigma\right).$$

We have one command left: $\mathsf{while}\ b\ \mathsf{do}\ c$. This is equivalent to $\mathsf{if}\ b\ \mathsf{then}\ (c\ ;\ \mathsf{while}\ b\ \mathsf{do}\ c)\ \mathsf{else}\ \mathsf{skip}$, so a first guess at a definition might be:

$$\begin{aligned}
\mathcal{C}[\![\mathsf{while}\ b\ \mathsf{do}\ c]\!]\sigma &\triangleq \mathit{if}\ \mathcal{B}[\![b]\!]\sigma\ \mathit{then}\ \mathcal{C}[\![c\ ;\ \mathsf{while}\ b\ \mathsf{do}\ c]\!]\sigma\ \mathit{else}\ \sigma \\
&= \mathit{if}\ \mathcal{B}[\![b]\!]\sigma\ \mathit{then}\ (\mathcal{C}[\![\mathsf{while}\ b\ \mathsf{do}\ c]\!])^*(\mathcal{C}[\![c]\!]\sigma)\ \mathit{else}\ \sigma,
\end{aligned}$$

but this appears to be circular. However, we can fix this by taking a fixpoint in some domain. Abbreviating $\mathcal{C}[\![\mathsf{while}\ b\ \mathsf{do}\ c]\!]$ by W, we would like

$$W\,\sigma = \mathit{if}\ \mathcal{B}[\![b]\!]\sigma\ \mathit{then}\ (W)^*(\mathcal{C}[\![c]\!]\sigma)\ \mathit{else}\ \sigma,$$

or rather

$$W = \lambda\sigma \in \Sigma.\,\mathit{if}\ \mathcal{B}[\![b]\!]\sigma\ \mathit{then}\ (W)^*(\mathcal{C}[\![c]\!]\sigma)\ \mathit{else}\ \sigma.$$

Define $\mathcal{F}$ as

$$\mathcal{F} \triangleq \lambda w \in \Sigma \to \Sigma_\bot.\,\lambda\sigma \in \Sigma.\,\mathit{if}\ \mathcal{B}[\![b]\!]\sigma\ \mathit{then}\ (w)^*(\mathcal{C}[\![c]\!]\sigma)\ \mathit{else}\ \sigma.$$

Then we would like $W = \mathcal{F}\,W$; that is, we are looking for a fixpoint of $\mathcal{F}$. But how do we take fixed points without using the dreaded Y combinator? Eventually we will have a function fix with $W = \mathsf{fix}\,\mathcal{F}$, where $\mathcal{F} \in (\Sigma \to \Sigma_\bot) \to (\Sigma \to \Sigma_\bot)$. The solution will be to think of a while statement as the limit of a sequence of approximations. Intuitively, by running through the loop more and more times, we will get better and better approximations.

The first and least accurate approximation is the totally undefined function

$$W_0 \triangleq \lambda\sigma \in \Sigma.\,\bot.$$

The next approximation is

$$\begin{aligned}
W_1 &\triangleq \mathcal{F}\,W_0 \\
&= \lambda\sigma \in \Sigma.\,\mathit{if}\ \mathcal{B}[\![b]\!]\sigma\ \mathit{then}\ (W_0)^*(\mathcal{C}[\![c]\!]\sigma)\ \mathit{else}\ \sigma \\
&= \lambda\sigma \in \Sigma.\,\mathit{if}\ \mathcal{B}[\![b]\!]\sigma\ \mathit{then}\ \bot\ \mathit{else}\ \sigma.
\end{aligned}$$

This simulates one iteration of the loop. We could then simulate two iterations by:

$$W_2 \stackrel{\triangle}{=} \mathcal{F}\, W_1 = \lambda\sigma \in \Sigma.\, \mathit{if}\ \mathcal{B}[\![b]\!]\sigma\ \mathit{then}\ (W_1)^*(\mathcal{C}[\![c]\!]\sigma)\ \mathit{else}\ \sigma.$$

In general,

$$W_{n+1} \stackrel{\triangle}{=} \mathcal{F}\, W_n = \lambda\sigma \in \Sigma.\, \mathit{if}\ \mathcal{B}[\![b]\!]\sigma\ \mathit{then}\ (W_n)^*(\mathcal{C}[\![c]\!]\sigma)\ \mathit{else}\ \sigma.$$

Then denotation of the while statement should be the limit of this sequence. But how do we take limits in spaces of functions? To do this, we need some structure on the space of functions. We will define an ordering $\sqsubseteq$ on these functions such that $W_0 \sqsubseteq W_1 \sqsubseteq W_2 \sqsubseteq \cdots$, then find the least upper bound of this sequence.

4 Partial Orders

A *partial order* (also known as a *partially ordered set* or *poset*) consists of a set S and a binary relation $\sqsubseteq$ on S that is

- *reflexive*: for all $d \in S$, $d \sqsubseteq d$;
- *transitive*: for all $d, e, f \in S$, if $d \sqsubseteq e$ and $e \sqsubseteq f$, then $d \sqsubseteq f$; and
- *antisymmetric*: for all $d, e \in S$, if $d \sqsubseteq e$ and $e \sqsubseteq d$, then $d = e$.

Examples

- $(\mathbb{Z}, \leq)$, where $\mathbb{Z}$ is the set of integers and $\leq$ is the usual ordering.
- $(\mathbb{Z}, =)$. Note that unequal elements are incomparable in this order.
- $(2^S, \subseteq)$. Here 2^S denotes the *powerset* of S, or the set of all subsets of S, often written $\mathcal{P}(S)$.
- $(2^S, \supseteq)$. In fact, if $(S, \sqsubseteq)$ is a partial order, then so is $(S, \sqsupseteq)$, where $s \sqsupseteq t \stackrel{\triangle}{=} t \sqsubseteq s$.
- $(\mathbb{N}, |)$, where $\mathbb{N} = \{0, 1, 2 \ldots\}$ and $a \mid b \stackrel{\triangle}{\Longleftrightarrow} (a \text{ divides } b) \Leftrightarrow (b = ka \text{ for some } k \in \mathbb{N})$. Note that for any $n \in \mathbb{N}$, we have $n \mid 0$; we call 0 an *upper bound* for $\mathbb{N}$ (but only in this ordering, of course!).

Non-examples

- $(\mathbb{Z}, <)$ is not a partial order, because $<$ is not reflexive.
- $(\mathbb{Z}, \sqsubseteq)$, where $m \sqsubseteq n \stackrel{\triangle}{\Longleftrightarrow} |m| \leq |n|$, is not a partial order because $\sqsubseteq$ is not anti-symmetric: $-1 \sqsubseteq 1$ and $1 \sqsubseteq -1$, but $-1 \neq 1$.

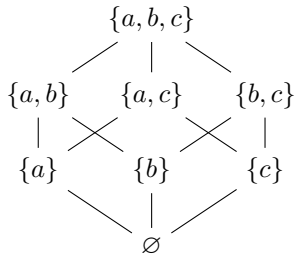
The "partial" in partial order comes from the fact that our definition does not require these orders to be total. For example, in the partial order $(2^{\{a,b\}}, \subseteq)$, the elements $\{a\}$ and $\{b\}$ are incomparable: neither $\{a\} \subseteq \{b\}$ nor $\{b\} \subseteq \{a\}$.

4.1 Hasse Diagrams

Partial orders can be described pictorially using *Hasse diagrams*[1]. In a Hasse diagram, each element of the partial order is displayed as a (possibly labeled) point, and lines are drawn between these points, according to these rules:

1. If x and y are elements of the partial order, and $x \sqsubseteq y$, then the point corresponding to x is drawn lower in the diagram than the point corresponding to y.
2. A line is drawn between the points representing x and y iff $x \sqsubseteq y$ and there does not exist a z strictly between x and y in the partial order; that is, the ordering relation between x and y is not due to transitivity.

Here is an example of a Hasse diagram for the subset relation on the set $2^{\{a,b,c\}}$:



A partial order like $(\mathbb{Z}, =)$ in which no distinct elements are related to each other is called a *discrete* partial order.

Given any partial order $(S, \sqsubseteq)$, we can define a new partial order $(S_\perp, \sqsubseteq_\perp)$ such that $S_\perp = S \cup \{\perp\}$, $d_1 \sqsubseteq_\perp d_2$ if $d_1, d_2 \in S$ and $d_1 \sqsubseteq d_2$, and $\perp \sqsubseteq_\perp d$ for all $d \in S_\perp$. Thus $S_\perp$ is the set S with a new least element $\perp$ added below everything in S.

In our semantic domains, we can think of $\sqsubseteq$ as "less information than". Thus nontermination $\perp$ contains less information than any element of S.

If we lift a discrete partial order (e.g., $\mathbb{Z}_\perp$), we get a *flat partial order*. The only relationships among distinct elements are between $\perp$ and every other element.

If a partial order has a least element $\perp$, that partial order is called *pointed*. All lifted partial orders, including flat partial orders, are pointed.

[1]Named after Helmut Hasse, 1898-1979. Hasse published fundamental results in algebraic number theory, including the Hasse (or "local-global") principle. He succeeded Hilbert and Weyl as the chair of the Mathematical Institute at Göttingen.