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 $\sigma, \sigma' \in \textbf{Store}$. 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:

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

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.

$$\mathcal{A}[\![a]\!] : \textbf{Store} \to \textbf{Int}$$
$$\mathcal{B}[\![b]\!] : \textbf{Store} \to \{\textbf{true}, \textbf{false}\}$$

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

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

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

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:

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

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.

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

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: **while** foo $<$ bar **do** foo := foo $+ 1$. Here $b =$ foo $<$ bar and $c =$ foo := foo $+ 1$.

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

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

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:

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

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

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

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

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

2 Equivalence with Operational Semantics

How do we justify that our solution for $\mathcal{C}[\![\textbf{while } b \textbf{ do } c]\!]$ (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 $a \in \textbf{\textit{AExp}}, \sigma \in \textbf{\textit{Store}}, n \in \textbf{\textit{Int}}$, $\mathcal{A}[\![a]\!]\sigma = n$ *if and only if* $\langle a, \sigma\rangle \Downarrow n$.

(ii) For all $b \in \textbf{\textit{BExp}}, \sigma \in \textbf{\textit{Store}}, b_0 \in \{\textsf{true}, \textsf{false}\}$, $\mathcal{B}[\![b]\!]\sigma = b_0$ *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. □

3 Fixed Points

We turn to the problem of finding solution to equations of the form:

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

Since the denotational semantics of IMP is in term of partial functions, we seek a partial function solution to the above equation, but we are in fact going to solve the problem in a more general setting, for a broad class of mathematical structures that may be meaningfully used to give denotations to programs.

Definition 1. A partial order $(P, \leq)$ is a set P with a binary relations $\leq$ that is

(i) reflexive: $\forall x \in P, x \leq x$

(ii) transitive: $\forall x, y, z \in P, x \leq y \wedge y \leq z \Rightarrow x \leq z$

(iii) antisymmetric: $\forall x, y \in P, x \leq y \wedge y \leq x \Rightarrow x = y$

Examples of partial orders include the powerset of an arbitrary set X, ordered by inclusion $(\wp(X), \subseteq)$, and the set of natural numbers $(\mathbb{N}, \leq)$ with the usual ordering.

The set of partial functions $(X \rightharpoonup X, \sqsubseteq)$ over a set X also form a partial order, where $f \sqsubseteq g$ holds exactly when for all $x \in X$, either f is not defined at x or $f(x) = g(x)$. (Equivalently, $f \sqsubseteq g$ exactly when the graph of f is a subset of the graph of g.)

Definition 2. Let $(P, \leq)$ be a partial order. Given $X \subseteq P$, we say u is an *upper bound* of X if $\forall x \in X, x \leq u$. We say u is a *least upper bound* (lub) of X if (i) u is an upper bound of X, and (ii) whenever v is an upper bound of X, then $u \leq v$. We write the least upper bound of X (when it exists) as $\sqcup X$. When $X = \{x_1, \ldots, x_n\}$, we sometimes write $x_1 \sqcup x_2 \sqcup \cdots \sqcup x_n$ for $\sqcup X$.

In $(\wp(X), \subseteq)$, the least upper bound of a set X is simply $\cup X$. (Check!) Similarly, in $(X \rightharpoonup X, \sqsubseteq)$, the least upper bound of a set F of partial functions if the partial functions whose graph is the union of the graphs of the functions in F.

Definition 3. Let $(D, \leq)$ be a partial order. An ω-chain in D is an increasing sequence $d_0 \leq d_1 \leq \cdots \leq d_n \leq \ldots$ of elemenst of D.

(i) $(D, \leq)$ is a *complete partial order* (cpo) if all ω-chains have a least upper bound.

(ii) $(D, \leq)$ is a *cpo with bottom* if it is a cpo with an element $\bot$ such that for all $d \in D$, $\bot \leq d$.

Both $(\wp(X), \subseteq)$ and $(X \rightharpoonup X, \sqsubseteq)$ are cpos with bottom, where $\bot$ is the empty set in the case of $\wp(X)$, and the nowhere-defined function in the case of $X \rightharpoonup X$. Note that $(\mathbb{N}, \leq)$ where $\leq$ is the usual ordering is *not* a cpo. (Why?)

Definition 4. Let $(D, \leq_D)$ and $(E, \leq_E)$ be cpos.

(i) A function $f : D \to E$ is *monotonic* if it preserves order: $\forall d, d' \in D, d \leq_D d' \Rightarrow f(d) \leq_E f(d')$.

(ii) A function $f : D \to E$ is *continuous* if it is monotonic and it preserves least upper bounds: for all ω-chains $d_0 \leq_D d_1 \leq_D \cdots \leq_D d_n \leq_D$ in D,

$$\bigsqcup_{n\geq 0} f(d_n) = f\left(\bigsqcup_{n\geq 0} d_n\right)$$

Definition 5. A *fixed point* of function $f : X \to X$ is an element $x \in X$ such that $f(x) = x$.

The main result of this section is that continuous functions between cpos with bottom all have fixed points, and they can be computed.

Lemma 2. *let $(D, \leq)$ be a cpo with bottom, and let $f : D \to D$ be a monotonic function. The following is an ω-chain:*

$$\bot \leq f(\bot) \leq f^2(\bot) \leq \cdots \leq f^n(\bot) \leq \ldots$$

Proof. An easy induction on n shows that for all $n \geq 0$, $f^n(\bot) \leq f^{n+1}(\bot)$. For $n = 0$, the result is immediate from $\bot \leq d$ for all $d \in D$. For the inductive case, assume $f^k(\bot) \leq f^{k+1}(\bot)$. By monotonicity of f, $f(f^k(\bot)) \leq f(f^{k+1}(\bot))$, that is, $f^{k+1}(\bot) \leq f^{k+2}(\bot)$, as required. □

Theorem 2. *Let $(D, \leq)$ be a cpo with bottom, and let $f : D \to D$ be a continuous function. Define*

$$\mathit{fix}(f) = \bigsqcup_{n\geq 0} f^n(\bot).$$

Then $\mathit{fix}(f)$ is a fixed point of f. In fact, it is the least fixed point of f: if d is a fixed point of f, then $\mathit{fix}(f) \leq d$.

Proof. The definition of $\mathit{fix}(f)$ is allowed because by the previous lemma $\{f^n(\bot) \mid n \geq 0\}$ is an ω-chain.

That $\mathit{fix}(f)$ is a fixed point follows directly by continuity:

$$\begin{aligned}
f(\mathit{fix}(f)) &= f(\sqcup_{n\geq 0} f^n(\bot)) \\
&= \bigsqcup_{n\geq 0} f(f^n(\bot)) \\
&= \bigsqcup_{n\geq 1} f^n(\bot) \\
&= \bot \sqcup \bigsqcup_{n\geq 1} f^n(\bot) \\
&= \bigsqcup_{n\geq 0} f^n(\bot) \\
&= \mathit{fix}(f)
\end{aligned}$$

The $\mathit{fix}(f)$ is the least fixed point follows by monotonicity. Let d be a fixed point of f, so that $f(d) = d$. An easy induction on n establishes that $f^n(\bot) \leq d$: for $n = 0$, $f^0(\bot) = \bot \leq d$ by definition of $\bot$. For $n = k$, assume that $f^k(\bot) \leq d$; then by monotonicity $f^{k+1}(\bot) = f(f^k(\bot)) \leq f(d) = d$, as required.

Since $f^n(\bot) \leq d$ for all $n \geq 0$, d is an upper bound of $\{f^n(\bot) \mid n \geq 0\}$. Since $\mathit{fix}(f)$ is the least upper bound of that set, by definition of least upper bound, $\mathit{fix}(f) \leq d$. □

And now we see where we came up with the solution to the equation for **while** b **do** c. We know that $(\mathbf{Store} \rightharpoonup \mathbf{Store}, \sqsubseteq)$ is a cpo with bottom, and we want to solve

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

A solution to the above equation is a fixed point of the function

$$F_{b,c} : (\mathbf{Store} \rightharpoonup \mathbf{Store}) \to (\mathbf{Store} \rightharpoonup \mathbf{Store})$$

defined by

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

If we can show that this function is continuous, then we can derive its fixed point as

$$\bigsqcup_{n\geq 0} {F_{b,c}}^i(\bot)$$

and since $\sqcup$ is union of graphs in the cpo of partial functions and $\bot$ is the function with graph $\varnothing$, we get the formulation:

$$\bigcup_{n\geq 0} {F_{b,c}}^i(\varnothing).$$

How do we know that $F_{b,c}$ is continuous? We could prove it directly, although it is a painful proof. It is actually easier to show a more general result, namely that a whole class of functions over partial functions are continuous.

Theorem 3. *Let $(X \rightharpoonup X, \sqsubseteq)$ be the cpo with bottom of partial functions over X. We represent partial functions using their graph.*

(a) The identity function $I : (X \rightharpoonup X) \to (X \rightharpoonup X)$ defined by $I(S) = S$ is continuous.

(b) For all $T \in X \rightharpoonup X$, the constant function $K_T : (X \rightharpoonup X) \to (X \rightharpoonup X)$ defined by $K_T(S) = T$ is continuous.

(c) For all continuous functions $F, G : (X \rightharpoonup X) \to (X \rightharpoonup X)$ such that for all $S \in X \rightharpoonup X$, $F(S) \sqcup G(S)$ exists, the function $H : (X \rightharpoonup X) \to (X \rightharpoonup X)$ defined by $H(S) = F(S) \cup G(S)$ is continuous.

(d) For all continuous functions $F, G : (X \rightharpoonup X) \to (X \rightharpoonup X)$, the function $H : (X \rightharpoonup X) \to (X \rightharpoonup X)$ defined by $H(S) = G(S) \circ F(S)$ is continuous.

(e) For all continuous functions $F : (X \rightharpoonup X) \to (X \rightharpoonup X)$ and all subsets $Y \subseteq X$, the function $H : (X \rightharpoonup X) \to (X \rightharpoonup X)$ defined by $H(S) = F(S)|_Y = \{(x,x') \in F(S) \mid x \in Y\}$ is continuous.

Why is this helpful? I leave it to you to check that $F_{b,c}$ can be defined as:

$$F_{b,c}(S) = K_{\{(\sigma,\sigma') \mid \mathcal{B}[\![b]\!]\sigma = \mathsf{false}\}}(S) \cup \big(I(S) \circ K_{\mathcal{C}[\![c]\!]}(S)\big)\,|_{\{\sigma \mid \mathcal{B}[\![b]\!]\sigma = \mathsf{true}\}}$$

and is therefore continuous.

4 Fixed Points for Other Recursive Equations

The above techniques are not only useful to solve recursive equations when defining denotational semantics, but also apply in other domains.

Here's a simple example. Consider the following equation for a function $f : \mathbb{N} \to \mathbb{N}$.

$$f(x) = \begin{cases} 0 & \text{if } x = 0 \\ f(x-1) + 2x - 1 & \text{otherwise} \end{cases} \tag{1}$$

This is not a definition for f, but rather an equation that we want f to satisfy. What function, or functions, satisfy this equation for f? The only solution to this equation is the function $f(x) = x^2$.

We can construct this solution by considering the cpo $(\mathbb{N} \rightharpoonup \mathbb{N}, \sqsubseteq)$ and finding a fixed point to the following function:

$$F(S) = \{(0,0)\} \cup \{(n, m+2n-1) \mid (n-1, m) \in S\}$$

(We need an argument for this function being continuous. Can you find it? Again, there is a general argument that says that the function derived from an equation like (1) is continuous.)

A fixed point of F is obtained by

$$\bigsqcup_{k \geq 0} F^k(\bot)$$

that is,

$$\bigcup_{k \geq 0} F^k(\varnothing)$$

and we compute

$$\begin{aligned}
F^0(\varnothing) &= \varnothing \\
F^1(\varnothing) &= F(\varnothing) \\
&= \{0,0\} \\
F^2(\varnothing) &= F(F^1(\varnothing)) \\
&= F(\{0,0\}) \\
&= \{(0,0),(1,1)\} \\
F^3(\varnothing) &= F(F^2(\varnothing)) \\
&= F(\{(0,0),(1,1)\}) \\
&= \{(0,0),(1,1),(2,4)\} \\
F^4(\varnothing) &= F(F^3(\varnothing)) \\
&= F(\{(0,0),(1,1),(2,4)\}) \\
&= \{(0,0),(1,1),(2,4),(3,9)\} \\
F^5(\varnothing) &= \ldots
\end{aligned}$$

We see how this sequence gradually builds successive approximations F^k of the function $f(x) = x^2$.