Denotational semantics ctd.; Axiomatic semantics

Lecture 6 Thursday, February 11, 2010

1 Denotational semantics

1.1 Fixed-point semantics for loops

Returning to our original problem: we want to find $\mathcal{C}[\![\textbf{while } b \textbf{ do } c]\!]$, the (partial) function from stores to stores that is the denotation of the loop **while** b **do** c. We will do this by expressing $\mathcal{C}[\![\textbf{while } b \textbf{ do } c]\!]$ as the fixed point of a higher-order function $F_{b,c}$.

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

Compare the definition of our higher-order function $F_{b,c}$ to our recursive equation for $\mathcal{C}[\![\textbf{while } b \textbf{ do } c]\!]$.

The higher-order function $F_{b,c}$ takes in the partial function f, and acts like one iteration of the while loop, except that, instead of invoking itself when it needs to go around the loop again, it instead calls f.

We can now define the semantics of the while loop:

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

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

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

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

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:

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

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

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

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

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

2 Introduction to axiomatic semantics

The idea in axiomatic semantics is to give specifications for what programs are supposed to compute. This contrasts with operational model (which show how programs execute) or denotational models (which show what programs compute). Axiomatic semantics defines the meaning of programs in terms of logical formulas satisfied by the program.

This approach to reasoning about programs and expressing program semantics was originally proposed by Floyd and Hoare, and then pushed further by Dijkstra and Gries.

Program specifications can be expressed using pre-conditions and post-conditions:

$$\{Pre\}\ c\ \{Post\}$$

where c is a program, and Pre and $Post$ are logical formulas that describe properties of the program state (usually referred to as assertions). Such a triple is referred to as a *partial correctness statement* (or partial correctness assertion triple) and has the following meaning:

"If Pre holds before c, and c terminates, then $Post$ holds after c."

In other words, if we start with a store σ where Pre holds, and the execution of c in store σ terminates and yields store σ', then $Post$ holds in store σ'.

Pre- and post-conditions can be regarded as interfaces or contracts between the program and its clients. They help users to understand what the program is supposed to yield without needing to understand how the program executes. Typically, programmers write them as comments for procedures and functions, for better program understanding, and to make it easier to maintain programs. Such specifications are especially useful for library functions, for which the source code is, in many cases, unavailable to the users. In this case, pre- and post-conditions serve as contracts between the library developers and users of the library.

However, there is no guarantee that pre- and post-conditions written as informal code comments are actually correct: the comments specify the intent, but give no correctness guarantees. Axiomatic semantics addresses this problem: we will look at how to rigorously describe partial correctness statements and how to prove and reason about program correctness.

Note that partial correctness doesn't ensure that the given program will terminate – this is why it is called "partial correctness". In contrast, total correctness statements ensure that the program terminates whenever the precondition holds. Such statements are denoted using square brackets:

$$[\ Pre\ ]\ c\ [\ Post\ ]$$

meaning:

"If Pre holds before c then c will terminate and $Post$ will hold after c."

In general a pre-condition specifies what the program expects before execution; and the post-conditions specifies what guarantees the program provides when the program terminates. Here is a simple example:

$$\{\mathsf{foo} = 0 \wedge \mathsf{bar} = i\}\ \mathsf{baz} := 0; \textbf{while}\ \mathsf{foo} \neq \mathsf{bar}\ \textbf{do}\ (\mathsf{baz} := \mathsf{baz} - 2; \mathsf{foo} := \mathsf{foo} + 1)\ \{\mathsf{baz} = -2i\}$$

If, before the program executes, the store maps foo to zero, and maps bar to i, then, if the program terminates, then the final store will map baz to $-2i$, that is, -2 times the initial value of bar. Note that i is a logical variable: it doesn't occur in the program, and is just used to express the initial value of bar.

This partial correctness statement is valid. That is, it is indeed the case that if we have any store σ such that $\sigma(\mathsf{foo}) = 0$, and

$$\mathcal{C}[\![\mathsf{baz} := 0; \textbf{while}\ \mathsf{foo} \neq \mathsf{bar}\ \textbf{do}\ (\mathsf{baz} := \mathsf{baz} - 2; \mathsf{foo} := \mathsf{foo} + 1)]\!]\sigma = \sigma',$$

then $\sigma'(\mathsf{baz}) = -2\sigma(\mathsf{bar})$.

Note that this is a *partial* correctness statement: if the pre-condition is true before c, **and** c **terminates** then the post-condition holds after c. For some initial stores, the program will fail to terminate.

The following total correctness statement is true.

$$[\mathsf{foo} = 0 \wedge \mathsf{bar} = i \wedge i \geq 0]\ \ \mathsf{baz} := 0; \textbf{while}\ \mathsf{foo} \neq \mathsf{bar}\ \textbf{do}\ (\mathsf{baz} := \mathsf{baz} - 2; \mathsf{foo} := \mathsf{foo} + 1)\ \ [\mathsf{baz} = -2i]$$

That is, if we have a store σ such that $\sigma(\mathsf{foo}) = 0$, and $\sigma(\mathsf{bar})$ is non-negative, then the execution of the command will terminate, and if σ' is the final store then $\sigma'(\mathsf{baz}) = -2\sigma(\mathsf{bar})$.

The following partial correctness statement is not valid. (Why not?)

$$\{\mathsf{foo} = 0 \wedge \mathsf{bar} = i\}\ \mathsf{baz} := 0; \textbf{while}\ \mathsf{foo} \neq \mathsf{bar}\ \textbf{do}\ (\mathsf{baz} := \mathsf{baz} + \mathsf{foo}; \mathsf{foo} := \mathsf{foo} + 1)\ \{\mathsf{baz} = i\}$$

In the rest of our presentation of axiomatic semantics we will focus exclusively on partial correctness. We will formalize and address the following:

- What logic do we use for writing assertions? That is, what can we express in pre- and post-condition?
- What does it mean that an assertion is valid? What does it mean that a partial correctness statement $\{Pre\}\ c\ \{Post\}$ is valid?
- How can we prove that a partial correctness statement is valid?