Big-Step Semantics and Errors

Stefan Muller, based on material by Jim Sasaki

CS 536: Science of Programming, Fall 2023
Lecture 6

1 Big-Step Operational Semantics

So far, we've seen two kinds of operational semantics:

- Small-step operational semantics, $\langle S, \sigma \rangle \to \langle S', \sigma' \rangle$ where we show each step the evaluation takes.
- $\sigma(e)$ for expressions, where we only care about the result and we get there in "one big step." This is a big step semantics.

It turns out we can also define a big-step semantics for statements (and a small-step semantics for expressions, but we won't do that one). We'll write it a little differently from the expression one, though:

$$M(S, \sigma) = \{\sigma'\}$$

means that if we start running S in state σ, the result is σ' (we don't care about the final statement, because it's always skip).

- Why is the result a set? (We'll use Σ to refer to sets of states)
- Intuitively, if $M(S, \sigma) = \Sigma$, then Σ is the set of states that might result from the evaluation.
- We'll see today cases where the evaluation doesn't result in a state, and the set is empty. We might see later in the course cases where there are multiple states.

We have:

$$M(S, \sigma) = \{\sigma'\} \Leftrightarrow \langle S, \sigma \rangle \to^* \langle \mathsf{skip}, \sigma' \rangle$$

We'll define this like a function, like we did with the big-step semantics for expressions. We could use inference rules too, but this is a little more concise. The cases for everything except while are pretty straightforward:

$$\begin{array}{rcll} M(\mathsf{skip}, \sigma) & = & \{\sigma\} & \\ M(S_1; S_2, \sigma) & = & \bigcup_{\sigma' \in M(S_1, \sigma)} M(S_2, \sigma') & \\ M(x := e, \sigma) & = & \{\sigma[x \mapsto \sigma(e)]\} & \\ M(a[e_1] := e_2, \sigma) & = & \{\sigma[a[\sigma(e_1)] \mapsto \sigma(e_2)]\} & 0 \le \sigma(e_1) < |\sigma(a)| \\ M(\mathsf{if}\ e\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}, \sigma) & = & M(S_1, \sigma) & \sigma(e) = T \\ M(\mathsf{if}\ e\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}, \sigma) & = & M(S_2, \sigma) & \sigma(e) = F \end{array}$$

Let's try a first (wrong) attempt at defining $M(\mathsf{while}\ e\ \mathsf{do}\ S\ \mathsf{od}, \sigma)$. We'll use the same trick of turing it into a conditional:

$$M(\mathsf{while}\ e\ \mathsf{do}\ S\ \mathsf{od}, \sigma) = M(\mathsf{if}\ e\ \mathsf{then}\ S; \mathsf{while}\ e\ \mathsf{do}\ S\ \mathsf{od}\ \mathsf{else}\ \mathsf{skip}\ \mathsf{fi}, \sigma)$$

But this isn't a valid recursive definition![1]

[1] It would be fine in programming, but in math, if you define something recursively, the "argument" needs to get smaller in "recursive calls."

Instead, let's look at progressive iterations of the loop. Let Σ_k be the set of states we might have after running the loop k times:

$$\begin{array}{rcl} \Sigma_0 & = & \{\sigma\} \\ \Sigma_k + 1 & = & \bigcup_{\sigma \in \Sigma_k} M(S, \sigma) \end{array}$$

Take $\textsf{while } x \geq \overline{0} \textsf{ do } x := x - \overline{1} \textsf{ od}$ in the state $\{x = 3\}$.

$$\begin{array}{lclcl} \Sigma_0 & = & \{\{x = 3\}\} & & \\ \Sigma_1 & = & \bigcup_{\sigma \in \Sigma_0} M(x := x - \overline{1}, \sigma) & = & M(x := x - \overline{1}, \{x = 3\}) = \{\{x = 2\}\} \\ \Sigma_2 & = & \bigcup_{\sigma \in \Sigma_1} M(x := x - \overline{1}, \sigma) & = & M(x := x - \overline{1}, \{x = 2\}) = \{\{x = 1\}\} \\ \Sigma_3 & = & \bigcup_{\sigma \in \Sigma_2} M(x := x - \overline{1}, \sigma) & = & M(x := x - \overline{1}, \{x = 1\}) = \{\{x = 0\}\} \\ \Sigma_4 & = & \bigcup_{\sigma \in \Sigma_3} M(x := x - \overline{1}, \sigma) & = & M(x := x - \overline{1}, \{x = 0\}) = \{\{x = -1\}\} \\ \ldots & & & & \end{array}$$

We could keep going like this forever, but note that (in this case) we don't have to. Once we reach a Σ_k such that for all $\sigma \in \Sigma_k$, the conditional expression is false, the loop stops, and so can we.

This gives us the formal definitions:

$$M(\textsf{while } e \textsf{ do } S \textsf{ od}, \sigma) = \Sigma_k \quad \Sigma_k \text{ is the lowest } k \text{ such that if } \sigma \in \Sigma_k, \text{ then } \sigma(e) = F$$

Of course, there may not be such a k (e.g., if $S = \textsf{while } x \geq \overline{0} \textsf{ do } x := x + \overline{1} \textsf{ od}$). In this case, $M(S, \sigma) = \emptyset$.

Example

$$\begin{array}{rcl} M(x := \overline{5}; y := x + \overline{1}, \emptyset) & = & \bigcup_{\sigma \in M(x := \overline{5}, \emptyset)} M(y := x + \overline{1}, \sigma) \\ & = & \bigcup_{\sigma \in \{\{x=5\}\}} M(y := x + \overline{1}, \sigma) \\ & = & M(y := x + \overline{1}, \{x = 5\}) \\ & = & \{x = 5, y = 6\} \end{array}$$

2 Runtime Errors

Fill in the blank: $\sigma(\overline{42}/\overline{0}) = ?$

It has to be a semantic integer, but there... isn't one. In any reasonable programming language (and a lot of unreasonable ones), this is an error/exception. So we should model those. We'll say $\sigma(\overline{42}/\overline{0}) = \perp$. So now $\sigma(e)$ is either a semantic value or $\perp$.

Other things that raise errors:

- $\{x = [1; 2]\}(a[\overline{3}]) = \perp$
- $\{x = -1\}(\textsf{sqrt}(x)) = \perp$ (if we assume we have a $\textsf{sqrt}$ function).
- $\sigma(\textsf{true} + \textsf{false}) = \perp$ (runtime type error)

2.1 Hereditary Failure

Fill in the blank: $\sigma(3 + (\overline{42}/\overline{0})) = ?$

This still fails even though the failure isn't at the "outer level." Let's add some cases to handle this:

$$\begin{array}{rcll} \sigma(e_1 \; op \; e_2) & = & \perp & \sigma(e_1) = \perp \vee \sigma(e_2) = \perp \\ \sigma(\textsf{if } e_1 \textsf{ then } e_2 \textsf{ else } e_3) & = & \perp & \sigma(e_1) = \perp \end{array}$$

This is called *hereditary failure*: we propagate errors up to outer expressions.

Note: We *don't* fail if the not-taken branch fails. This lets us do things like

$$\sigma(\textsf{if } x = \overline{0} \textsf{ then } \overline{1} \textsf{ else } y/x)$$

2.2 Errors in Statements

For small-step semantics, we'll write $\langle S, \sigma \rangle \to \langle \mathsf{skip}, \perp \rangle$ if a step leads to an error.

$$\frac{\langle s_1, \sigma \rangle \to \langle \mathsf{skip}, \perp \rangle}{\langle s_1; s_2, \sigma \rangle \to \langle \mathsf{skip}, \perp \rangle} \qquad \frac{\sigma(e) = \perp}{\langle x := e, \sigma \rangle \to \langle \mathsf{skip}, \perp \rangle} \qquad \frac{\sigma(e_1) = \perp \vee \sigma(e_2) = \perp}{\langle a[e_1] := e_2, \sigma \rangle \to \langle \mathsf{skip}, \perp \rangle}$$

$$\frac{\sigma(e_1) \geq |\sigma(a)| \vee \sigma(e_1) < 0}{\langle a[e_1] := e_2, \sigma \rangle \to \langle \mathsf{skip}, \perp \rangle} \qquad \frac{\sigma(e) = \perp}{\langle \mathsf{if}\ e\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}, \sigma \rangle \to \langle \mathsf{skip}, \perp \rangle}$$

Note: $\perp$ appears in some of the places a state does, but it's **not** a state (or a value). In particular, here are some things we'll **never** write:

- $\perp[x \mapsto 0]$
- $\perp(x)$
- $\perp(e)$
- $\sigma[x \mapsto \perp]$
- $M(S, \perp)$

We'll do something equivalent in big-step semantics:

$$\begin{array}{rcll} M(x := e, \sigma) & = & \{\perp\} & \sigma(e) = \perp \\ M(a[e_1] := e_2, \sigma) & = & \{\perp\} & \sigma(e_1) = \perp \vee \sigma(e_2) = \perp \vee \sigma(e_1) < 0 \vee \sigma(e_1) \geq |\sigma(a)| \\ M(\mathsf{if}\ e\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}, \sigma) & = & \{\perp\} & \sigma(e) = \perp \end{array}$$

Or equivalently, we can say that $\sigma(S) = \{\perp\}$ if $\langle S, \sigma \rangle \to^* \langle \mathsf{skip}, \perp \rangle$.