Comp Sci 704 | Lecture 7: Big-Step Semantics | Ethan Cecchetti
Fall 2024

As an alternative to small-step structural operational semantics, which specifies the operation of the program one step at a time, we now consider *big-step* operational semantics (or *large-step*), in which we specify the entire transition from a configuration (an ⟨expression, store⟩ pair) to a final value in one "big step." As with the small-step semantics, a big-step semantics for Imp must include rules for all three types of expressions—arithmetic, boolean, and commands. For arithmetic expressions, the final value is an integer $n \in \mathbb{Z}$; for boolean expressions, it is a boolean truth value $b \in \mathbb{2} = \{\mathit{true}, \mathit{false}\}$; and for commands, it is a store $\sigma : \mathbf{Var} \to \mathbb{Z}$. We usually represent big-step semantics using a double down arrow $\Downarrow$, so

$$\begin{aligned}\Downarrow_a &\subseteq (\mathbf{AExp} \times \mathbf{Store}) \times \mathbb{Z}\\ \Downarrow_b &\subseteq (\mathbf{BExp} \times \mathbf{Store}) \times \mathbb{2}\\ \Downarrow_c &\subseteq (\mathbf{Com} \times \mathbf{Store}) \times \mathbf{Store}\end{aligned}$$

Intuitively, these three relations have the following meanings.

- $\langle a, \sigma\rangle \Downarrow_a n$ means that evaluating expression a in context σ produces integer $n \in \mathbb{Z}$.
- $\langle b, \sigma\rangle \Downarrow_b t$ means that evaluating expression b in context σ produces boolean $t \in \mathbb{2}$.
- $\langle c, \sigma\rangle \Downarrow_c \sigma'$ means that command c terminates when evaluated in environment σ, and σ' is the final environment after c is finished executing.

Unlike in our small-step semantics, where integers, booleans, and skip were unable to step, here arithmetic and boolean expressions will *always* produce a result, meaning $\Downarrow_a$ and $\Downarrow_b$ are actually total functions. However, $\Downarrow_c$ remains partial, this time because commands may not terminate.

1 Defining Big-Step Semantics

We will use the same Imp language as in the small-step operational semantics, which, as a reminder, has the following BNF grammar.

$$\begin{array}{rrcl}\mathbf{AExp}: & a & ::= & n \mid x \mid a_1 + a_2 \mid a_1 * a_2 \mid a_1 - a_2\\ \mathbf{BExp}: & b & ::= & \mathsf{true} \mid \mathsf{false} \mid a_1 = a_2 \mid a_1 \le a_2 \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \mid \neg b\\ \mathbf{Com}: & c & ::= & \mathsf{skip} \mid x := a \mid c_1 \mathbin{;} c_2 \mid \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \mid \mathsf{while}\ b\ \mathsf{do}\ c\end{array}$$

1.1 Arithmetic and Boolean Expressions

For the arithmetic expressions, the semantics take the following form.

$$[\text{B-Const}]\ \frac{}{\langle n, \sigma\rangle \Downarrow_a n} \qquad [\text{B-Var}]\ \frac{}{\langle x, \sigma\rangle \Downarrow_a \sigma(x)} \qquad [\text{B-Op}]\ \frac{\begin{array}{cc}\langle a_1, \sigma\rangle \Downarrow_a n_1 & \langle a_2, \sigma\rangle \Downarrow_a n_2\\ \otimes \in \{+, *, -\} & m = n_1 \otimes n_2\ \text{(mathematically)}\end{array}}{\langle a_1 \otimes a_2, \sigma\rangle \Downarrow m}$$

The rules for boolean operators and comparisons are similar.

Note the difference here from small-step semantics. Instead of taking one step at a time and only invoking mathematical operators when the arguments are already integers, we are now immediately evaluating both sides of the operator to an integer and applying them immediately. These rules follow our intuition for the full behavior of a whole program, not just for a single step in execution.

1.2 Commands

The big-step operational semantic rules for commands are as follows.

$$\text{[B-SKIP]}\ \frac{}{\langle \mathsf{skip}, \sigma\rangle \Downarrow_c \sigma} \qquad \text{[B-ASSIGN]}\ \frac{\langle a, \sigma\rangle \Downarrow_a n}{\langle x := a, \sigma\rangle \Downarrow_c \sigma[x \mapsto n]} \qquad \text{[B-SEQ]}\ \frac{\langle c_1, \sigma\rangle \Downarrow_c \sigma' \qquad \langle c_2, \sigma'\rangle \Downarrow_c \sigma''}{\langle c_1 \mathbin{;} c_2, \sigma\rangle \Downarrow_c \sigma''}$$

$$\text{[B-IFT]}\ \frac{\langle b, \sigma\rangle \Downarrow_b \mathit{true} \qquad \langle c_1, \sigma\rangle \Downarrow_c \sigma'}{\langle \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2, \sigma\rangle \Downarrow_c \sigma'} \qquad \text{[B-IFF]}\ \frac{\langle b, \sigma\rangle \Downarrow_b \mathit{false} \qquad \langle c_2, \sigma\rangle \Downarrow_c \sigma'}{\langle \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2, \sigma\rangle \Downarrow_c \sigma'}$$

$$\text{[B-WHILEF]}\ \frac{\langle b, \sigma\rangle \Downarrow_b \mathit{false}}{\langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma\rangle \Downarrow_c \sigma} \qquad \text{[B-WHILET]}\ \frac{\begin{array}{c}\langle b, \sigma\rangle \Downarrow_b \mathit{true} \qquad \langle c, \sigma\rangle \Downarrow_c \sigma' \\ \langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma'\rangle \Downarrow_c \sigma''\end{array}}{\langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma\rangle \Downarrow_c \sigma''}$$

2 Big-Set vs Small-Step Semantics

The big-step semantics above and the small-step semantics from last time both describe the same language, so we would expect them to agree. In particular, we would expect that if some configuration $\langle c, \sigma\rangle$ evaluates to $\langle \mathsf{skip}, \sigma'\rangle$ in the small-step semantics, it should also evaluate to σ' in the big-step semantics, and vice versa. Formally,

Theorem 1. *For all commands $c \in \mathbf{Com}$ and stores $\sigma, \sigma' \in \mathbf{Store}$,*

$$\langle c, \sigma\rangle \longrightarrow_c^* \langle \mathsf{skip}, \sigma'\rangle \iff \langle c, \sigma\rangle \Downarrow_c \sigma'$$

Proof. We can express the idea that two semantics should agree on terminating executions by connecting the $\longrightarrow^*$ and $\Downarrow$ relations:

$$\langle a, \sigma\rangle \longrightarrow_a^* n \iff \langle a, \sigma\rangle \Downarrow_a n \tag{1}$$

$$\langle b, \sigma\rangle \longrightarrow_b^* \bar{t} \iff \langle b, \sigma\rangle \Downarrow_b t \tag{2}$$

$$\langle c, \sigma\rangle \longrightarrow_c^* \langle \mathsf{skip}, \sigma'\rangle \iff \langle c, \sigma\rangle \Downarrow_c \sigma' \tag{3}$$

Here $\bar{t}$ represents conversion between syntactic booleans in IMP and mathematical booleans in $\mathbb{2}$. That is, $\overline{\mathit{true}} = \mathsf{true}$ and $\overline{\mathit{false}} = \mathsf{false}$. We can prove (1) and (2) as lemmas separately and use them to prove (3).

To prove the forward implication of (3), we proceed by structural induction on c, though the while case requires a separate induction on the number of loop iterations. The converse requires induction on the derivation of the big-step evaluation. We will show a few cases of that proof here.

Suppose we are given $\langle c, \sigma\rangle \Downarrow_c \sigma'$. We aim to prove $\langle c, \sigma\rangle \longrightarrow_c^* \langle \mathsf{skip}, \sigma'\rangle$. The derivation of the big-step relation depends on the form of c.

Case B-SKIP ($c = \mathsf{skip}$)**:** Here $\sigma' = \sigma$, $\langle \mathsf{skip}, \sigma\rangle \longrightarrow_c^* \langle \mathsf{skip}, \sigma\rangle$, and $\langle \mathsf{skip}, \sigma\rangle \Downarrow_c \sigma$ all immediately.

Case B-ASSIGN ($c = x := a$)**:** By the big-step rule B-ASSIGN, $\langle a, \sigma\rangle \Downarrow_a n$ and $\sigma' = \sigma[x \mapsto n]$. By our assumed lemma proving (1), $\langle a, \sigma\rangle \longrightarrow_a^* n$.

We now need a lemma that proves if $\langle a, \sigma\rangle \longrightarrow_a^* n$, then $\langle x := a, \sigma\rangle \longrightarrow_c^* \langle x := n, \sigma\rangle$, which can prove by induction on the number of $\longrightarrow_a$ steps. From there,

$$\langle x := a, \sigma\rangle \longrightarrow_c^* \langle x := n, \sigma\rangle \longrightarrow_c \langle \mathsf{skip}, \sigma[x \mapsto n]\rangle,$$

which completes the case.

Case B-WHILEF ($c = \mathsf{while}\ b\ \mathsf{do}\ c_0$ **where** $\langle b, \sigma \rangle \Downarrow_b \mathit{false}$)**:** Then $\sigma' = \sigma$. In the small-step operational semantics,

$$\langle \mathsf{while}\ b\ \mathsf{do}\ c_0, \sigma \rangle \longrightarrow_c \langle \mathsf{if}\ b\ \mathsf{then}\ (c_0\ ;\ \mathsf{while}\ b\ \mathsf{do}\ c_0)\ \mathsf{else}\ \mathsf{skip}, \sigma \rangle.$$

By the lemma proving (2), $\langle b, \sigma \rangle \longrightarrow_b^* \mathsf{false}$, so by a similar lemma to in the previous case,

$$\langle \mathsf{if}\ b\ \mathsf{then}\ (c_0\ ;\ \mathsf{while}\ b\ \mathsf{do}\ c_0)\ \mathsf{else}\ \mathsf{skip}, \sigma \rangle \longrightarrow_c^* \langle \mathsf{if}\ \mathsf{false}\ \mathsf{then}\ (c_0\ ;\ \mathsf{while}\ b\ \mathsf{do}\ c_0)\ \mathsf{else}\ \mathsf{skip}, \sigma \rangle.$$

Putting that together with the existing facts and one more step at the end arrives at

$$\begin{aligned} \langle c, \sigma \rangle = \langle \mathsf{while}\ b\ \mathsf{do}\ c_0, \sigma \rangle &\longrightarrow_c \langle \mathsf{if}\ b\ \mathsf{then}\ (c_0\ ;\ \mathsf{while}\ b\ \mathsf{do}\ c_0)\ \mathsf{else}\ \mathsf{skip}, \sigma \rangle \\ &\longrightarrow_c^* \langle \mathsf{if}\ \mathsf{false}\ \mathsf{then}\ (c_0\ ;\ \mathsf{while}\ b\ \mathsf{do}\ c_0)\ \mathsf{else}\ \mathsf{skip}, \sigma \rangle \\ &\longrightarrow_c \langle \mathsf{skip}, \sigma \rangle. \end{aligned}$$

This completes the case.

Case B-WHILET ($c = \mathsf{while}\ b\ \mathsf{do}\ c_0$ **where** $\langle b, \sigma \rangle \Downarrow_b \mathit{true}$)**:** This is the most interesting case in the entire proof. For the small-step semantics, we have the same initial case as in the previous proof, and again the condition on the if statement evaluates down, but this time to true, meaning the last step of the previous case instead produces $\langle c_0\ ;\ \mathsf{while}\ b\ \mathsf{do}\ c_0, \sigma \rangle$.

We need another lemma for stitching together small-step executions.

Lemma 1. *For any* $c_1, c_2 \in \mathbf{Com}$ *and* $\sigma, \sigma', \sigma'' \in \mathbf{Store}$,

$$\langle c_1, \sigma \rangle \longrightarrow_c^* \langle \mathsf{skip}, \sigma' \rangle \ \mathit{and}\ \langle c_2, \sigma' \rangle \longrightarrow_c^* \langle \mathsf{skip}, \sigma'' \rangle \implies \langle c_1\ ;\ c_2, \sigma \rangle \longrightarrow_c^* \langle \mathsf{skip}, \sigma'' \rangle.$$

Proof of Lemma 1. By induction on the number of steps needed to prove $\langle c_1, \sigma \rangle \longrightarrow_c^* \langle \mathsf{skip}, \sigma' \rangle$. □

The premises of the B-WHILET rule include $\langle c_0, \sigma \rangle \Downarrow_c \sigma'$ and $\langle \mathsf{while}\ b\ \mathsf{do}\ c_0, \sigma' \rangle \Downarrow_c \sigma''$. Because these are both sub-derivations of the derivation $\langle \mathsf{while}\ b\ \mathsf{do}\ c_0, \sigma \rangle \Downarrow_c \sigma''$ on which we are doing induction, the inductive hypotheses immediately prove

$$\langle c_0, \sigma \rangle \longrightarrow_c^* \langle \mathsf{skip}, \sigma' \rangle \quad \text{and} \quad \langle \mathsf{while}\ b\ \mathsf{do}\ c_0, \sigma' \rangle \longrightarrow_c^* \langle \mathsf{skip}, \sigma'' \rangle.$$

Lemma 1 is then enough to show

$$\langle c_0\ ;\ \mathsf{while}\ b\ \mathsf{do}\ c_0, \sigma \rangle \longrightarrow_c^* \langle \mathsf{skip}, \sigma'' \rangle$$

which, in combination with the facts stated at the top of this case, completes the case. □

Note that this proof relied on structural induction on the derivation of $\langle c, \sigma \rangle \Downarrow_c \sigma'$, *not* induction on c. This choice was critical. The last case relied on an inductive hypothesis applied to a smaller derivation, but the same command, meaning induction on c would not have been well-founded.

Also note that this theorem about the agreement between big-step and small-step semantics talks only about the behavior of *terminating* programs. This omission of nonterminating programs is inherently necessary because the big-step relation $\Downarrow_c$ cannot talk directly about nontermination. Indeed, if $\langle c, \sigma \rangle$ does not terminate, there is no σ' such that $\langle c, \sigma \rangle \Downarrow_c \sigma'$. Small-step semantics can model more complex features such as nonterminating programs and concurrency. However, in many cases it involves unnecessary extra work.

If we do not care about modeling nonterminating computations, it can be easier to reason in terms of big-step semantics. Moreover, big-step semantics more closely models an actual recursive interpreter. However, because evaluation skips over intermediate steps, all programs without final configurations are indistinguishable.