Harvard School of Engineering and Applied Sciences — Computer Science 152

Large-step semantics; IMP: an imperative language

Lecture 3 Tuesday, February 2, 2010

1 Large-step semantics

So far we have defined the small step evaluation relation $\longrightarrow \subseteq$ **Config** $\times$ **Config**, and used its transitive and reflexive closure $\longrightarrow^*$ to describe the execution of multiple steps of evaluation. In particular, if $\langle e, \sigma\rangle$ is some start configuration, and $\langle n, \sigma'\rangle$ is a final configuration, the evaluation $\langle e, \sigma\rangle \longrightarrow^* \langle n, \sigma'\rangle$ shows that by executing expression e starting with the store σ, we get the result n, and the final store σ'.

Large-step semantics is an alternative way to specify the operational semantics of a language. Large-step semantics directly give the final result.

We'll use the same configurations as before, but define a large step evaluation relation:

$$\Downarrow \subseteq \textbf{Exp} \times \textbf{Store} \times \textbf{Int} \times \textbf{Store}.$$

We write $\langle e, \sigma\rangle \Downarrow \langle n, \sigma'\rangle$ to mean that $(e, \sigma, n, \sigma') \in \Downarrow$. In other words, expression e with store σ evaluates in one big step directly to integer n, and final store σ'.

The large step semantics boils down to defining the relation $\Downarrow$. We use inference rules to inductively define the relation $\Downarrow$, similar to how we specified the small-step operational semantics $\longrightarrow$.

$$\textsc{Int}_{\textsc{Large}}\ \frac{}{\langle n, \sigma\rangle \Downarrow \langle n, \sigma\rangle} \qquad \textsc{Var}_{\textsc{Large}}\ \frac{}{\langle x, \sigma\rangle \Downarrow \langle n, \sigma\rangle} \text{ where } \sigma(x) = n$$

$$\textsc{Add}_{\textsc{Large}}\ \frac{\langle e_1, \sigma\rangle \Downarrow \langle n_1, \sigma''\rangle \qquad \langle e_2, \sigma''\rangle \Downarrow \langle n_2, \sigma'\rangle}{\langle e_1 + e_2, \sigma\rangle \Downarrow \langle n, \sigma'\rangle} \text{ where } n \text{ is the sum of } n_1 \text{ and } n_2$$

$$\textsc{Mul}_{\textsc{Large}}\ \frac{\langle e_1, \sigma\rangle \Downarrow \langle n_1, \sigma''\rangle \qquad \langle e_2, \sigma''\rangle \Downarrow \langle n_2, \sigma'\rangle}{\langle e_1 \times e_2, \sigma\rangle \Downarrow \langle n, \sigma'\rangle} \text{ where } n \text{ is the product of } n_1 \text{ and } n_2$$

$$\textsc{Asg}_{\textsc{Large}}\ \frac{\langle e_1, \sigma\rangle \Downarrow \langle n_1, \sigma''\rangle \qquad \langle e_2, \sigma''[x \mapsto n_1]\rangle \Downarrow \langle n_2, \sigma'\rangle}{\langle x := e_1; e_2, \sigma\rangle \Downarrow \langle n_2, \sigma'\rangle}$$

To see how we use these rules, here is a proof tree that shows that $\langle \mathsf{foo} := 3; \mathsf{foo} \times \mathsf{bar}, \sigma\rangle \Downarrow \langle 21, \sigma'\rangle$ for a store σ such that $\sigma(\mathsf{bar}) = 7$, and $\sigma' = \sigma[\mathsf{foo} \mapsto 3]$.

$$\textsc{Asg}_{\textsc{Large}}\ \frac{\textsc{Int}_{\textsc{Large}}\ \dfrac{}{\langle 3, \sigma\rangle \Downarrow \langle 3, \sigma\rangle} \qquad \textsc{Mul}_{\textsc{Large}}\ \dfrac{\textsc{Var}_{\textsc{Large}}\ \dfrac{}{\langle \mathsf{foo}, \sigma'\rangle \Downarrow \langle 3, \sigma'\rangle} \quad \textsc{Var}_{\textsc{Large}}\ \dfrac{}{\langle \mathsf{bar}, \sigma'\rangle \Downarrow \langle 7, \sigma'\rangle}}{\langle \mathsf{foo} \times \mathsf{bar}, \sigma'\rangle \Downarrow \langle 21, \sigma'\rangle}}{\langle \mathsf{foo} := 3; \mathsf{foo} \times \mathsf{bar}, \sigma\rangle \Downarrow \langle 21, \sigma'\rangle}$$

A closer look to this structure reveals the relation between small step and large-step evaluation: a depth-first traversal of the large-step proof tree yields the sequence of one-step transitions in small-step evaluation.

2 Equivalence of semantics

So far, we have specified the semantics of our language of arithmetic expressions using two different sets of rules: small-step and large-step. Are they expressing the same meaning of arithmetic expressions? Can we show that they express the same thing?

Theorem (Equivalence of semantics)**.** *For all expressions e, stores σ, and integers n, we have:*

$$\langle e, \sigma \rangle \Downarrow \langle n, \sigma' \rangle \iff \langle e, \sigma \rangle \longrightarrow^* \langle n, \sigma' \rangle.$$

Proof sketch.

- $\Longrightarrow$. We want to prove that the following property P holds for all expressions $e \in$ **Exp**.

  $$P(e) = \forall \sigma, \sigma' \in \textbf{Store}. , \forall n \in \textbf{Int}. \ \langle e, \sigma \rangle \Downarrow \langle n, \sigma' \rangle \Longrightarrow \langle e, \sigma \rangle \longrightarrow^* \langle n, \sigma' \rangle$$

  We proceed by structural induction on expressions e. We have to consider each of the possible axioms and inference rules for constructing an expression.

  - **Case** $e = x$**.**
    Here, we are considering the case where the expression e is equal to some variable x. Assume that$\langle x, \sigma \rangle \Downarrow \langle n, \sigma \rangle$. That means that there is some derivation using the axioms and inference rules of the large-step operational semantics, whose conclusion is $\langle x, \sigma \rangle \Downarrow \langle n, \sigma \rangle$. There is only one rule whose conclusion could look like this, the rule $\text{Var}_{\text{Large}}$. That rule requires that $n = \sigma(x)$.
    Since $n = \sigma(x)$ we know that $\langle x, \sigma \rangle \longrightarrow \langle n, \sigma \rangle$ also holds, by using the small-step axiom VAR. So we can conclude that $\langle x, \sigma \rangle \longrightarrow^* \langle n, \sigma \rangle$ holds, which is what we needed to show.
  - **Case** $e = n$**.**
    Here, we are consider the case where expression e is equal to some integer n. But then $\langle n, \sigma \rangle \longrightarrow^* \langle n, \sigma \rangle$ holds trivially because of reflexivity of $\longrightarrow^*$.
  - **Case** $e = e_1 + e_2$**.**
    This is an inductive case. We want to prove that if $P(e_1)$ and $P(e_2)$ hold, then $P(e)$ holds too. Let's write out $P(e_1)$, $P(e_2)$, and $P(e)$ explicitly.

    $$\begin{aligned} P(e_1) &= \forall n, \sigma, \sigma' : \langle e_1, \sigma \rangle \Downarrow \langle n, \sigma' \rangle \Longrightarrow \langle e_1, \sigma \rangle \longrightarrow^* \langle n, \sigma' \rangle \\ P(e_2) &= \forall n, \sigma, \sigma' : \langle e_2, \sigma \rangle \Downarrow \langle n, \sigma' \rangle \Longrightarrow \langle e_2, \sigma \rangle \longrightarrow^* \langle n, \sigma' \rangle \\ P(e) &= \forall n, \sigma, \sigma' : \langle e_1 + e_2, \sigma \rangle \Downarrow \langle n, \sigma' \rangle \Longrightarrow \langle e_1 + e_2, \sigma \rangle \longrightarrow^* \langle n, \sigma' \rangle \end{aligned}$$

    Let's assume that $P(e_1)$ and $P(e_2)$ hold. Assume that we have values for σ, σ' and n such that $\langle e_1 + e_2, \sigma \rangle \Downarrow \langle n, \sigma' \rangle$. We need to show that $\langle e_1 + e_2, \sigma \rangle \longrightarrow^* \langle n, \sigma' \rangle$.
    We assumed that $\langle e_1 + e_2, \sigma \rangle \Downarrow \langle n, \sigma' \rangle$. This means that there is some derivation whose conclusion is $\langle e_1 + e_2, \sigma \rangle \Downarrow \langle n, \sigma' \rangle$. By looking at the large-step semantic rules, we see that only one rule could possible have a conclusion of this form: the rule $\text{Add}_{\text{Large}}$. So that means that the last rule use in the derivation was $\text{Add}_{\text{Large}}$. But in order to use the rule $\text{Add}_{\text{Large}}$, it must be the case that $\langle e_1, \sigma \rangle \Downarrow \langle n_1, \sigma'' \rangle$ and $\langle e_2, \sigma'' \rangle \Downarrow \langle n_2, \sigma' \rangle$ hold for some n_1 and n_2 such that $n = n_1 + n_2$ (i.e., there is a derivation whose conclusion is $\langle e_1, \sigma \rangle \Downarrow \langle n_1, \sigma'' \rangle$ and a derication whose conclusion is $\langle e_2, \sigma'' \rangle \Downarrow \langle n_2, \sigma' \rangle$).
    Using the inductive hypothesis $P(e_1)$, since $\langle e_1, \sigma \rangle \Downarrow \langle n_1, \sigma'' \rangle$, we must have $\langle e_1, \sigma \rangle \longrightarrow^* \langle n_1, \sigma'' \rangle$. Similarly, by $P(e_2)$, we have $\langle e_2, \sigma'' \rangle \longrightarrow^* \langle n_2, \sigma \rangle$. By Lemma 1 below, we have

    $$\langle e_1 + e_2, \sigma \rangle \longrightarrow^* \langle n_1 + e_2, \sigma'' \rangle$$

    and by another application of Lemma 1 we have

    $$\langle n_1 + e_2, \sigma'' \rangle \longrightarrow^* \langle n_1 + n_2, \sigma' \rangle$$

and by the rule ADD we have

$$\langle n_1 + n_2, \sigma' \rangle \longrightarrow \langle n, \sigma' \rangle.$$

Thus, we have $\langle e_1 + e_2, \sigma \rangle \longrightarrow^* \langle n, \sigma' \rangle$, which proves this case.

- **Case** $e = e_1 \times e_2$**.** Similar to the case $e = e_1 + e_2$ above.
- **Case** $e = x := e_1; e_2$**.** Omitted. Try it as an exercise.

- $\Longleftarrow$. We proceed by mathematical induction on the number of steps $\langle e, \sigma \rangle \longrightarrow^* \langle n, \sigma' \rangle$.
  - **Base case.** If $\langle e, \sigma \rangle \longrightarrow^* \langle n, \sigma' \rangle$ in zero steps, then we must have $e = n$ and $\sigma' = \sigma$. Then, $\langle n, \sigma \rangle \Downarrow \langle n, \sigma \rangle$ by the large-step operational semantics rule $\text{INT}_{\text{LARGE}}$.
  - **Inductive case.** Assume that $\langle e, \sigma \rangle \longrightarrow \langle e'', \sigma'' \rangle \longrightarrow^* \langle n, \sigma' \rangle$, and that (the inductive hypothesis) $\langle e'', \sigma'' \rangle \Downarrow \langle n, \sigma' \rangle$. That is, $\langle e'', \sigma'' \rangle \longrightarrow^* \langle n, \sigma' \rangle$ takes m steps, and we assume that the property holds for it ($\langle e'', \sigma'' \rangle \Downarrow \langle n, \sigma' \rangle$), and we are considering $\langle e, \sigma \rangle \longrightarrow^* \langle n, \sigma' \rangle$, which takes $m+1$ steps. We need to show that $\langle e, \sigma \rangle \Downarrow \langle n, \sigma' \rangle$. This follows immediately from Lemma 2 below.

□

Lemma 1. *If $\langle e, \sigma \rangle \longrightarrow^* \langle n, \sigma' \rangle$ then for all n_1, e_2 the following hold.*

- $\langle e + e_2, \sigma \rangle \longrightarrow^* \langle n + e_2, \sigma' \rangle$
- $\langle e \times e_2, \sigma \rangle \longrightarrow^* \langle n \times e_2, \sigma' \rangle$
- $\langle n_1 + e, \sigma \rangle \longrightarrow^* \langle n_1 + n, \sigma' \rangle$
- $\langle n_1 \times e, \sigma \rangle \longrightarrow^* \langle n_1 \times n, \sigma' \rangle$

Proof. By (mathematical) induction on the number of evaluation steps in $\longrightarrow^*$. □

Lemma 2. *For all e, e', σ, and n, if $\langle e, \sigma \rangle \longrightarrow \langle e', \sigma'' \rangle$ and $\langle e', \sigma'' \rangle \Downarrow \langle n, \sigma' \rangle$, then $\langle e, \sigma \rangle \Downarrow \langle n, \sigma' \rangle$.*

3 IMP: a simple imperative language

We shall now consider a more realistic programming language, one where we can assign values to variables and execute control constructs such as if and while. The syntax for this simple imperative language, called IMP, is as follows:

arithmetic expressions	$a \in \mathbf{Aexp}$	$a ::= x \mid n \mid a_1 + a_2 \mid a_1 \times a_2$
boolean expressions	$b \in \mathbf{Bexp}$	$b ::= \textbf{true} \mid \textbf{false} \mid a_1 < a_2$
commands	$c \in \mathbf{Com}$	$c ::= \textbf{skip} \mid x := a \mid c_1; c_2$
		$\mid \textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2$
		$\mid \textbf{while } b \textbf{ do } c$

3.1 Small-step operational semantics

We'll first give a small-step operational semantics for IMP. The configurations in this language are of the form $\langle c, \sigma \rangle$, $\langle b, \sigma \rangle$, and $\langle a, \sigma \rangle$, where σ is a store. The final configurations are of the form $\langle \textbf{skip}, \sigma \rangle$, $\langle \textbf{true}, \sigma \rangle$, $\langle \textbf{false}, \sigma \rangle$, and $\langle n, \sigma \rangle$. There are three different small-step operational semantics relations, one each for commands, boolean expressions, and arithmetic expressions.

$$\longrightarrow_{\mathbf{Com}} \subseteq \mathbf{Com} \times \mathbf{Store} \times \mathbf{Com} \times \mathbf{Store}$$
$$\longrightarrow_{\mathbf{Bexp}} \subseteq \mathbf{Bexp} \times \mathbf{Store} \times \mathbf{Bexp} \times \mathbf{Store}$$
$$\longrightarrow_{\mathbf{Aexp}} \subseteq \mathbf{Aexp} \times \mathbf{Store} \times \mathbf{Aexp} \times \mathbf{Store}$$

For brevity, we will overload the symbol $\longrightarrow$ and use it to refer to all of these relations. Which relation is being used will be clear from context.

The evaluation rules for arithmetic and boolean expressions are similar to the ones we've seen before. However, note that since the arithmetic expressions no longer contain assignment, arithmetic and boolean expressions can not update the store.

Arithmetic expressions

$$\frac{}{\langle x, \sigma\rangle \longrightarrow \langle n, \sigma\rangle} \text{ where } n = \sigma(x)$$

$$\frac{\langle e_1, \sigma\rangle \longrightarrow \langle e_1', \sigma\rangle}{\langle e_1 + e_2, \sigma\rangle \longrightarrow \langle e_1' + e_2, \sigma\rangle} \qquad \frac{\langle e_2, \sigma\rangle \longrightarrow \langle e_2', \sigma\rangle}{\langle n + e_2, \sigma\rangle \longrightarrow \langle n + e_2', \sigma\rangle} \qquad \frac{}{\langle n + m, \sigma\rangle \longrightarrow \langle p, \sigma\rangle} \text{ where } p = n + m$$

$$\frac{\langle e_1, \sigma\rangle \longrightarrow \langle e_1', \sigma\rangle}{\langle e_1 \times e_2, \sigma\rangle \longrightarrow \langle e_1' \times e_2, \sigma\rangle} \qquad \frac{\langle e_2, \sigma\rangle \longrightarrow \langle e_2', \sigma\rangle}{\langle n \times e_2, \sigma\rangle \longrightarrow \langle n \times e_2', \sigma\rangle} \qquad \frac{}{\langle n \times m, \sigma\rangle \longrightarrow \langle p, \sigma\rangle} \text{ where } p = n \times m$$

Boolean expressions

$$\frac{\langle a_1, \sigma\rangle \longrightarrow \langle a_1', \sigma\rangle}{\langle a_1 < a_2, \sigma\rangle \longrightarrow \langle a_1' < a_2, \sigma\rangle} \qquad \frac{\langle a_2, \sigma\rangle \longrightarrow \langle a_2', \sigma\rangle}{\langle n < a_2, \sigma\rangle \longrightarrow \langle n < a_2', \sigma\rangle}$$

$$\frac{}{\langle n < m, \sigma\rangle \longrightarrow \langle \textbf{true}, \sigma\rangle} \text{ where } n < m \qquad \frac{}{\langle n < m, \sigma\rangle \longrightarrow \langle \textbf{false}, \sigma\rangle} \text{ where } n \geq m$$

Commands

$$\frac{\langle e, \sigma\rangle \longrightarrow \langle e', \sigma\rangle}{\langle x := e, \sigma\rangle \longrightarrow \langle x := e', \sigma\rangle} \qquad \frac{}{\langle x := n, \sigma\rangle \longrightarrow \langle \textbf{skip}, \sigma[x \mapsto n]\rangle}$$

$$\frac{\langle c_1, \sigma\rangle \longrightarrow \langle c_1', \sigma'\rangle}{\langle c_1; c_2, \sigma\rangle \longrightarrow \langle c_1'; c_2, \sigma'\rangle} \qquad \frac{}{\langle \textbf{skip}; c_2, \sigma\rangle \longrightarrow \langle c_2, \sigma\rangle}$$

For if commands, we gradually reduce the test until we get either **true** or **false**; then, we execute the appropriate branch:

$$\frac{\langle b, \sigma\rangle \longrightarrow \langle b', \sigma\rangle}{\langle \textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2, \sigma\rangle \longrightarrow \langle \textbf{if } b' \textbf{ then } c_1 \textbf{ else } c_2, \sigma\rangle}$$

$$\frac{}{\langle \textbf{if true then } c_1 \textbf{ else } c_2, \sigma\rangle \longrightarrow \langle c_1, \sigma\rangle} \qquad \frac{}{\langle \textbf{if false then } c_1 \textbf{ else } c_2, \sigma\rangle \longrightarrow \langle c_2, \sigma\rangle}$$

For while loops, the above strategy doesn't work (why?). Instead, we use the following rule, which can be thought of as "unrolling" the loop, one iteration at a time.

$$\frac{}{\langle \textbf{while } b \textbf{ do } c, \sigma\rangle \longrightarrow \langle \textbf{if } b \textbf{ then } (c; \textbf{while } b \textbf{ do } c) \textbf{ else skip}, \sigma\rangle}$$

We can now take a concrete program and see how it executes under the above rules. Consider we start with state σ where $\sigma(\mathsf{foo}) = 0$ and we execute the program

$$\mathsf{foo} := 3; \textbf{while}\ \mathsf{foo} < 4\ \textbf{do}\ \mathsf{foo} := \mathsf{foo} + 5$$

The execution works as follows:

$$\begin{array}{lll}
 & \langle \mathsf{foo} := 3; \textbf{while}\ \mathsf{foo} < 4\ \textbf{do}\ \mathsf{foo} := \mathsf{foo} + 5, \sigma\rangle & \\
\longrightarrow & \langle \textbf{skip}; \textbf{while}\ \mathsf{foo} < 4\ \textbf{do}\ \mathsf{foo} := \mathsf{foo} + 5, \sigma'\rangle & \text{where } \sigma' = \sigma[\mathsf{foo} \mapsto 3] \\
\longrightarrow & \langle \textbf{while}\ \mathsf{foo} < 4\ \textbf{do}\ \mathsf{foo} := \mathsf{foo} + 5, \sigma'\rangle & \\
\longrightarrow & \langle \textbf{if}\ \mathsf{foo} < 4\ \textbf{then}\ (\mathsf{foo} := \mathsf{foo} + 5; W)\ \textbf{else skip}, \sigma'\rangle & \\
\longrightarrow & \langle \textbf{if}\ 3 < 4\ \textbf{then}\ (\mathsf{foo} := \mathsf{foo} + 5; W)\ \textbf{else skip}, \sigma'\rangle & \\
\longrightarrow & \langle \textbf{if true then}\ (\mathsf{foo} := \mathsf{foo} + 5; W)\ \textbf{else skip}, \sigma'\rangle & \\
\longrightarrow & \langle \mathsf{foo} := \mathsf{foo} + 5; \textbf{while}\ \mathsf{foo} < 4\ \textbf{do}\ \mathsf{foo} := \mathsf{foo} + 5, \sigma'\rangle & \\
\longrightarrow & \langle \mathsf{foo} := 3 + 5; \textbf{while}\ \mathsf{foo} < 4\ \textbf{do}\ \mathsf{foo} := \mathsf{foo} + 5, \sigma'\rangle & \\
\longrightarrow & \langle \mathsf{foo} := 8; \textbf{while}\ \mathsf{foo} < 4\ \textbf{do}\ \mathsf{foo} := \mathsf{foo} + 5, \sigma'\rangle & \\
\longrightarrow & \langle \textbf{while}\ \mathsf{foo} < 4\ \textbf{do}\ \mathsf{foo} := \mathsf{foo} + 5, \sigma''\rangle & \text{where } \sigma'' = \sigma'[\mathsf{foo} \mapsto 8] \\
\longrightarrow & \langle \textbf{if}\ \mathsf{foo} < 4\ \textbf{then}\ (\mathsf{foo} := \mathsf{foo} + 5; W)\ \textbf{else skip}, \sigma''\rangle & \\
\longrightarrow & \langle \textbf{if}\ 8 < 4\ \textbf{then}\ (\mathsf{foo} := \mathsf{foo} + 5; W)\ \textbf{else skip}, \sigma''\rangle & \\
\longrightarrow & \langle \textbf{if false then}\ (\mathsf{foo} := \mathsf{foo} + 5; W)\ \textbf{else skip}, \sigma''\rangle & \\
\longrightarrow & \langle \textbf{skip}, \sigma''\rangle &
\end{array}$$

(where W is an abbreviation for the while loop $\textbf{while}\ \mathsf{foo} < 4\ \textbf{do}\ \mathsf{foo} := \mathsf{foo} + 5$).