Scribe: Kevin Markman and Jon Guarino Lecturer: Radu Rugina

Today's lecture presented a simple imperative language, IMP, and gave the small-step and big-step rules for evaluation. These notes provide the following:

- The IMP language syntax
- A small-step semantics for IMP
- A big-step semantics for IMP
- Some notes on why both can be useful

1 Syntax

There are three types of statements in IMP:

- Arithemetic expressions: *AExp*
- Boolean expressions: *BExp*
- Commands: *Com*

A program in the IMP language is a command, $c \in Com$. Suppose n denotes an integer literal, x denotes a variable name, a_i denotes an arithmetic expression, b_i denotes a boolean expressions, and c_i represents a command. The grammar follows:

$$
\begin{array}{rcl}
\mathit{AExp} & ::= & n \mid x \mid (\mathsf{a}_0 \oplus \mathsf{a}_1) \\
\mathit{BExp} & ::= & \textbf{true} \mid \textbf{false} \mid (\mathsf{a}_0 \odot \mathsf{a}_1) \mid (\mathsf{b}_0 \oslash \mathsf{b}_1) \\
\mathit{Com} & ::= & \textbf{skip} \mid x := a \mid (\mathsf{c}_0 \,;\, \mathsf{c}_1) \mid (\textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2) \mid (\textbf{while } b \textbf{ do } c\ ) \\
\oplus & ::= & + \mid * \mid - \\
\odot & ::= & \leq \mid = \\
\oslash & ::= & \vee \mid \wedge
\end{array}
$$

1.1 Configurations

Configurations are pairs, $\langle c, \sigma \rangle$, where:

- $c \in Com$ is a command
- σ is a store (also known as a state or environment)
  $\sigma \in (\sum = Var \rightarrow \mathbb{Z})$ where $\sum$ is all possible stores

2 Structural Operational Semantics: Small-Step Semantics

We can evaluate a configuration "one step at a time" until we reach the final configuration, $\langle \textbf{skip}, \sigma \rangle$:

$$\langle c, \sigma \rangle \rightarrow \langle c', \sigma' \rangle$$

$$\langle c, \sigma_0 \rangle \rightarrow \langle c_1, \sigma_1 \rangle \rightarrow \ldots \rightarrow \langle c_k, \sigma_k \rangle \rightarrow \langle \textbf{skip}, \sigma \rangle$$

We can represent this more tersly using closure:

$$\langle c, \sigma_0 \rangle \to^* \langle \mathbf{skip}, \sigma \rangle$$

To be proper, we should define the $\to$ operator for commands[1], arithmetic expressions, and boolean expressions:

$$\begin{aligned}
\to &\in (Com \times \sum) \to (Com \times \sum) \\
\to_a &\in (AExp \times \sum) \to AExp \\
\to_b &\in (BExp, \times \sum) \to BExp
\end{aligned}$$

We will use $\to$ as shorthand for all three operations.

2.1 Arithmetic & Boolean Expressions

- Constants: no rule (already fully reduced)
- Variables: $\dfrac{}{\langle x, \sigma \rangle \to \sigma(x)}$
- Operations: $\dfrac{\langle a_0, \sigma \rangle \to a_0'}{\langle a_0 \oplus a_1, \sigma \rangle \to a_0' \oplus a_1}$ $\quad \dfrac{\langle a_1, \sigma \rangle \to a_1'}{\langle n \oplus a_1, \sigma \rangle \to n \oplus a_1'}$ $\quad \dfrac{}{\langle n_0 \oplus n_1, \sigma \rangle \to n}$ where $n = n_0 \oplus n_1$
- Corresponding rules of the above form exist for boolean expressions using $\odot$ and $\oslash$ operators

2.2 Commands

- Skip: no rule ($\langle \mathbf{skip}, \sigma \rangle$ is a final configuration)
- Assignments: $\dfrac{\langle a, \sigma \rangle \to a'}{\langle x := a, \sigma \rangle \to \langle x := a', \sigma \rangle}$ $\quad \dfrac{}{\langle x := n, \sigma \rangle \to \langle \mathbf{skip}, \sigma[x \mapsto n] \rangle}$
- Sequences: $\dfrac{\langle c_0, \sigma \rangle \to \langle c_0', \sigma' \rangle}{\langle c_0; c_1, \sigma \rangle \to \langle c_0'; c_1, \sigma' \rangle}$ $\quad \dfrac{}{\langle \mathbf{skip}; c_1, \sigma \rangle \to \langle c_1, \sigma \rangle}$
- If Statements: $$\dfrac{\langle b, \sigma \rangle \to b'}{\langle \mathbf{if}\ \text{b}\ \mathbf{then}\ \text{c}\mathit{0}\ \mathbf{else}\ c1, \sigma \rangle \to \langle \mathbf{if}\ \text{b'}\ \mathbf{then}\ \text{c}\mathit{0}\ \mathbf{else}\ c1, \sigma \rangle}$$ $$\dfrac{}{\langle \mathbf{if\ true\ then}\ c_0\ \mathbf{else}\ c_1, \sigma \rangle \to \langle c_0, \sigma \rangle}$$ $$\dfrac{}{\langle \mathbf{if\ false\ then}\ c_0\ \mathbf{else}\ c_1, \sigma \rangle \to \langle c_1, \sigma \rangle}$$
- While Statements: $\dfrac{}{\langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma \rangle \to \langle \mathbf{if}\ b\ \mathbf{then}\ (c; \mathbf{while}\ b\ \mathbf{do}\ c) \mathbf{else\ skip}, \sigma \rangle}$

3 Structural Operational Semantics: Big-Step Semantics

As an alternative to “one step at a time” evaluation, we now consider a step in evaluation to mean the entire transition from an expression and state to a final value. For example:

$\langle c, \sigma \rangle \Downarrow \sigma'$	(steps to the final store of its final configuration)
$\langle a, \sigma \rangle \Downarrow n$	(steps to its final integer value)
$\langle b, \sigma \rangle \Downarrow t$	(steps to t where $t \in \{\mathbf{true}, \mathbf{false}\}$)

[1]Winskel uses $\to_1$ instead of $\to$ to emphasize that only a single step is performed

The rules for specific commands are as follows.

- Skip: $\overline{\langle \mathbf{skip}, \sigma \rangle \Downarrow \sigma}$
- Assignments: $\dfrac{\langle a, \sigma \rangle \Downarrow n}{\langle x := a, \sigma \rangle \Downarrow \sigma[x \mapsto n]}$
- Sequences: $\dfrac{\langle c_0, \sigma \rangle \Downarrow \sigma'' \quad \langle c_1, \sigma'' \rangle \Downarrow \sigma'}{\langle c_0; c_1, \sigma \rangle \Downarrow \sigma'}$
- If Statements: $\dfrac{\langle b, \sigma \rangle \Downarrow \mathbf{true} \quad \langle c_0, \sigma \rangle \Downarrow \sigma'}{\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, \sigma \rangle \Downarrow \sigma'} \quad \dfrac{\langle b, \sigma \rangle \Downarrow \mathbf{false} \quad \langle c_1, \sigma \rangle \Downarrow \sigma'}{\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, \sigma \rangle \Downarrow \sigma'}$
- While Statements: $\dfrac{\langle b, \sigma \rangle \Downarrow \mathbf{false}}{\langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma \rangle \Downarrow \sigma} \quad \dfrac{\langle b, \sigma \rangle \Downarrow \mathbf{true} \quad \langle c, \sigma \rangle \Downarrow \sigma'' \quad \langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma'' \rangle \Downarrow \sigma'}{\langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma \rangle \Downarrow \sigma'}$

4 Big-Step vs. Small-Step SOS

4.1 Small-Step

- can model more complex features, like programs which run forever and concurrency
- Although "one step at a time" evaluation is useful for proving certain properties, in many cases it is unnecessary extra work.

4.2 Big-Step

- large steps in reasoning make it easier to prove things
- more closely models an actual recursive interpreter
- Because evaluation skips over intermediate steps, all programs without final configurations (infinite loops, errors/stuck configs) look the same.