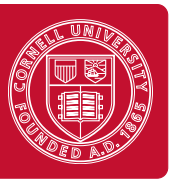
CS 4110 – Programming Languages and Logics
Lecture #6: The IMP Language



1 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 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	$b \in \mathbf{Bexp}$	$b ::= \textsf{true} \mid \textsf{false} \mid a_1 < a_2$
commands	$c \in \mathbf{Com}$	$c ::= \textsf{skip} \mid x := a \mid c_1; c_2 \mid \textsf{if } b \textsf{ then } c_1 \textsf{ else } c_2 \mid \textsf{while } b \textsf{ do } c$

1.1 Small-step operational semantics

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

$$\begin{aligned}
\rightarrow_{\mathbf{Com}} &\subseteq \mathbf{Com} \times \mathbf{Store} \times \mathbf{Com} \times \mathbf{Store} \\
\rightarrow_{\mathbf{Bexp}} &\subseteq \mathbf{Bexp} \times \mathbf{Store} \times \mathbf{Bexp} \times \mathbf{Store} \\
\rightarrow_{\mathbf{Aexp}} &\subseteq \mathbf{Aexp} \times \mathbf{Store} \times \mathbf{Aexp} \times \mathbf{Store}
\end{aligned}$$

For brevity, we will overload the symbol $\rightarrow$ 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

$$\frac{n = \sigma(x)}{\langle \sigma, x\rangle \rightarrow \langle \sigma, n\rangle}$$

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

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

Boolean expressions

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

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

Commands

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

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

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

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

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

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.

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

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:

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

where W is an abbreviation for the while loop **while** foo < 4 **do** foo $:=$ foo $+ 5$.

2 Large-step operational semantics for IMP

We define large-step evaluation relations for arithmetic expressions, boolean expressions, and commands. The relation for arithmetic expressions relates an arithmetic expression and store to the integer value that the expression evaluates to. For boolean expressions, the final value is in $\mathbf{Bool} = \{\textbf{true}, \textbf{false}\}$. For commands, the final value is a store.

$$
\begin{aligned}
\Downarrow_{\mathbf{Aexp}} &\subseteq \mathbf{Aexp} \times \mathbf{Store} \times \mathbf{Int} \\
\Downarrow_{\mathbf{Bexp}} &\subseteq \mathbf{Bexp} \times \mathbf{Store} \times \mathbf{Bool} \\
\Downarrow_{\mathbf{Com}} &\subseteq \mathbf{Com} \times \mathbf{Store} \times \mathbf{Store}
\end{aligned}
$$

Again, we overload the symbol $\Downarrow$ and use it for any of these three relations; which relation is intended will be clear from context. We also use infix notation, for example writing $\langle \sigma, c \rangle \Downarrow \sigma'$ if $(c, \sigma, \sigma') \in \Downarrow_{\mathbf{Com}}$.

Arithmetic expressions.

$$
\frac{}{\langle \sigma, n \rangle \Downarrow n} \qquad \frac{\sigma(x) = n}{\langle \sigma, x \rangle \Downarrow n}
$$

$$
\frac{\langle \sigma, e_1 \rangle \Downarrow n_1 \qquad \langle \sigma, e_2 \rangle \Downarrow n_2 \qquad n = n_1 + n_2}{\langle \sigma, e_1 + e_2 \rangle \Downarrow n} \qquad \frac{\langle \sigma, e_1 \rangle \Downarrow n_1 \qquad \langle \sigma, e_2 \rangle \Downarrow n_2 \qquad n = n_1 \times n_2}{\langle \sigma, e_1 \times e_2 \rangle \Downarrow n}
$$

Boolean expressions.

$$\frac{}{\langle \sigma, \textsf{true} \rangle \Downarrow \textsf{true}} \qquad \frac{}{\langle \sigma, \textsf{false} \rangle \Downarrow \textsf{false}}$$

$$\frac{\langle \sigma, a_1 \rangle \Downarrow n_1 \qquad \langle \sigma, a_2 \rangle \Downarrow n_2 \qquad n_1 < n_2}{\langle \sigma, a_1 < a_2 \rangle \Downarrow \textsf{true}} \qquad \frac{\langle \sigma, a_1 \rangle \Downarrow n_1 \qquad \langle \sigma, a_2 \rangle \Downarrow n_2 \qquad n_1 \geq n_2}{\langle \sigma, a_1 < a_2 \rangle \Downarrow \textsf{false}}$$

Commands.

$$\textsc{Skip}\ \frac{}{\langle \sigma, \textsf{skip} \rangle \Downarrow \sigma} \qquad \textsc{Assgn}\ \frac{\langle \sigma, e \rangle \Downarrow n}{\langle \sigma, x := e \rangle \Downarrow \sigma[x \mapsto n]} \qquad \textsc{Seq}\ \frac{\langle \sigma, c_1 \rangle \Downarrow \sigma' \qquad \langle \sigma', c_2 \rangle \Downarrow \sigma''}{\langle \sigma, c_1; c_2 \rangle \Downarrow \sigma''}$$

$$\textsc{If-T}\ \frac{\langle \sigma, b \rangle \Downarrow \textsf{true} \qquad \langle \sigma, c_1 \rangle \Downarrow \sigma'}{\langle \sigma, \textsf{if } b \textsf{ then } c_1 \textsf{ else } c_2 \rangle \Downarrow \sigma'} \qquad \textsc{If-F}\ \frac{\langle \sigma, b \rangle \Downarrow \textsf{false} \qquad \langle \sigma, c_2 \rangle \Downarrow \sigma'}{\langle \sigma, \textsf{if } b \textsf{ then } c_1 \textsf{ else } c_2 \rangle \Downarrow \sigma'}$$

$$\textsc{While-F}\ \frac{\langle \sigma, b \rangle \Downarrow \textsf{false}}{\langle \sigma, \textsf{while } b \textsf{ do } c \rangle \Downarrow \sigma} \qquad \textsc{While-T}\ \frac{\langle \sigma, b \rangle \Downarrow \textsf{true} \qquad \langle \sigma, c \rangle \Downarrow \sigma' \qquad \langle \sigma', \textsf{while } b \textsf{ do } c \rangle \Downarrow \sigma''}{\langle \sigma, \textsf{while } b \textsf{ do } c \rangle \Downarrow \sigma''}$$

It's interesting to see that the rule for while loops does not rely on using an if command (as we needed in the case of small-step semantics). Why does this rule work?

2.1 Command equivalence

The small-step operational semantics suggests that the loop **while** b **do** c should be equivalent to the command **if** b **then** $(c;$ **while** b **do** $c)$ **else skip**. Can we show that this indeed the case that the language is defined using the above large-step evaluation?

First, we need to to be more precise about what "equivalent commands" mean. Our formal model allows us to define this concept using large-step evaluations as follows. (One can write a similar definition using $\rightarrow^*$ in small-step semantics.)

Definition (Equivalence of commands)**.** Two commands c and c' are equivalent (written $c \sim c'$) if, for any stores σ and σ', we have

$$\langle \sigma, c \rangle \Downarrow \sigma' \iff \langle \sigma, c' \rangle \Downarrow \sigma'.$$

We can now state and prove the claim that **while** b **do** c and **if** b **then** $(c;$ **while** b **do** $c)$ **else skip** are equivalent.

Theorem. *For all* $b \in \mathbf{Bexp}$ *and* $c \in \mathbf{Com}$ *we have*

$$\textbf{\textit{while}}\ b\ \textbf{\textit{do}}\ c \sim \textbf{\textit{if}}\ b\ \textbf{\textit{then}}\ (c;\ \textbf{\textit{while}}\ b\ \textbf{\textit{do}}\ c)\ \textbf{\textit{else skip}}.$$

Proof. Let W be an abbreviation for **while** b **do** c. We want to show that for all stores σ, σ', we have:

$$\langle \sigma, W \rangle \Downarrow \sigma' \text{ if and only if } \textsf{if } b \textsf{ then } (c; W) \textsf{ else skip} \Downarrow \sigma'$$

For this, we must show that both directions ($\Longrightarrow$ and $\Longleftarrow$) hold. We'll show only direction $\Longrightarrow$; the other is similar.

Assume that σ and σ' are stores such that $\langle \sigma, W \rangle \Downarrow \sigma'$. It means that there is some derivation that proves for this fact. Inspecting the evaluation rules, we see that there are two possible rules whose conclusions match this fact: While-F and While-T. We analyze each of them in turn.

- WHILE-F. The derivation must look like the following.

$$\textsc{While-F}\ \frac{\dfrac{\vdots^{1}}{\langle \sigma, b\rangle \Downarrow \textbf{false}}}{\langle \sigma, W\rangle \Downarrow \sigma}$$

  Here, we use $\vdots^{1}$ to refer to the derivation of $\langle \sigma, b\rangle \Downarrow \textbf{false}$. Note that in this case, $\sigma' = \sigma$.

  We can use $\vdots^{1}$ to derive a proof tree showing that the evaluation of **if** b **then** $(c; W)$ **else skip** yields the same final state σ:

$$\textsc{If-F}\ \frac{\dfrac{\vdots^{1}}{\langle \sigma, b\rangle \Downarrow \textbf{false}} \qquad \textsc{Skip}\ \dfrac{}{\langle \sigma, \textbf{skip}\rangle \Downarrow \sigma}}{\langle \sigma, \textbf{if}\ b\ \textbf{then}\ (c; W)\ \textbf{else skip}\rangle \Downarrow \sigma}$$

- WHILE-T. In this case, the derivation has the following form.

$$\textsc{While-T}\ \frac{\dfrac{\vdots^{2}}{\langle \sigma, b\rangle \Downarrow \textbf{true}} \qquad \dfrac{\vdots^{3}}{\langle \sigma, c\rangle \Downarrow \sigma''} \qquad \dfrac{\vdots^{4}}{\langle \sigma'', W\rangle \Downarrow \sigma'}}{\langle \sigma, W\rangle \Downarrow \sigma'}$$

  We can use subderivations $\vdots^{2}$, $\vdots^{3}$, and $\vdots^{4}$ to show that the evaluation of **if** b **then** $(c; W)$ **else skip** yields the same final state σ.

$$\textsc{If-T}\ \frac{\dfrac{\vdots^{2}}{\langle \sigma, b\rangle \Downarrow \textbf{true}} \qquad \textsc{Seq}\ \dfrac{\dfrac{\vdots^{3}}{\langle \sigma, c\rangle \Downarrow \sigma''} \qquad \dfrac{\vdots^{4}}{\langle \sigma'', W\rangle \Downarrow \sigma'}}{\langle \sigma, c; W\rangle \Downarrow \sigma'}}{\langle \sigma, \textbf{if}\ b\ \textbf{then}\ (c; W)\ \textbf{else skip}\rangle \Downarrow \sigma'}$$

Hence, we showed that in each of the two possible cases, the command **if** b **then** $(c; W)$ **else skip** evaluates to the same final state as the command W. □