1 IMP

IMP, a simple IMPerative language, was introduced at the end of last lecture to provide a simple, and arugably more familiar, framework for the study of language semantics.

1.1 Syntax

IMP contains several different language constructs:

x $\in$ **Var**
a $\in$ **Aexp** $::= n \mid x \mid a_1 \oplus a_2$ (where $\oplus$ is an arithmetic operation)
b $\in$ **Bexp** $::=$ **true** | **false** | $b_1 \wedge b_2 \mid b_1 \vee b_2 \mid a_1 \odot a_2$ (where $\odot$ is a comparison)
c $\in$ **Cmd** $::=$ **skip** | $x := e \mid c_1; c_2$ | **if** b **then** c_1 **else** c_2 | **while** b **do** c

Constants consist of integers $n \in \mathbf{Z}$ and truth values $\mathbf{T} = \{\mathbf{true}, \mathbf{false}\}$.

1.2 Configurations

Unlike in the λ-calculus, the storage of values in variables provides for the notion of state in IMP. We define a *configuration* to be the current state of evaluation:

$$\langle c, \sigma \rangle$$

where c is an IMP command at some stage of evaluation and σ is a *store*, a function $f : Var \rightarrow \mathbf{Z}$ mapping variables to integers.

We define the initial state to be σ_0, a function that maps all variables to 0, or in terms of the λ-calculus, $\lambda x.0$.

We define the final configuration, if one is reachable after zero or more steps of evaluation, to be:

$$\langle \mathbf{skip}, \sigma \rangle$$

where **skip** reflects that no further computation is possible and σ is some final state. Running programs in IMP, therefore, consists of stepping from an initial configuration to the final configuration in terminating programs:

$$\langle c, \sigma_0 \rangle \rightarrow \cdots \rightarrow \langle \mathbf{skip}, \sigma \rangle \Leftrightarrow \langle c, \sigma_0 \rangle \rightarrow^* \langle \mathbf{skip}, \sigma \rangle$$

1.3 Small-Step Operational Semantics

We now present the small-step semantics for evaluation of arithmetic and Boolean expressions and commands in IMP. Just as with the λ-calculus, the evaluation rules are presented as inference rules, which inductively define relations consisting of the acceptable computational steps in IMP.

1.3.1 Arithmetic Expressions

Integers: $\dfrac{}{\langle n, \sigma \rangle \longrightarrow n}$

Variables: $\dfrac{}{\langle x, \sigma \rangle \longrightarrow \sigma(x)}$

Arithmetic with values: $\dfrac{n_3 = n_1 \oplus n_2}{\langle n_1 \oplus n_2, \sigma \rangle \longrightarrow n_3}$

Arithmetic with expressions: $\dfrac{\langle a_1, \sigma \rangle \longrightarrow a_1'}{\langle a_1 \oplus a_2, \sigma \rangle \longrightarrow a_1' \oplus a_2}$

1.3.2 Boolean Expressions

Evaluation semantics for Boolean expressions are similar to those for arithmetic expressions, and we leave their construction as an exercise.

1.3.3 Commands

Skip: This command is always the final configuration as defined before, so there is no evaluation rule for it.

Assignment:

$$\frac{}{\langle x := n, \sigma\rangle \longrightarrow \langle \mathbf{skip}, \sigma[x \to n]\rangle}$$

$$\frac{\langle a, \sigma\rangle \longrightarrow \langle a', \sigma\rangle}{\langle x := a, \sigma\rangle \longrightarrow \langle x := a', \sigma\rangle}$$

Sequences:

$$\frac{}{\langle \mathbf{skip}; c_2, \sigma\rangle \longrightarrow \langle c_2, \sigma\rangle}$$

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

If-Else Conditional:

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

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

$$\frac{\langle b, \sigma\rangle \longrightarrow \langle b', \sigma\rangle}{\langle \mathbf{if}\, b \,\mathbf{then}\, c \,\mathbf{else}\, {}_1c_2, \sigma\rangle \longrightarrow \langle \mathbf{if}\, b' \,\mathbf{then}\, c_1 \,\mathbf{else}\, c_2, \sigma\rangle}$$

While-loops:
Evaluating while loops is tricky because the naïve evaluation rule would result in nontermination. The proper evaluation rule *unrolls* the while-loop, wrapping it within an if-else statement as follows:

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

We provide an example to clarify how this works. Consider the following simple loop:

$$\mathbf{while}\, b \leq 10 \,\mathbf{do}\, b := b + 1$$

Our evaluation rule stipulates the evaluation step:

$$\langle \mathbf{while}\, b \leq 10 \,\mathbf{do}\, b := b + 1, \sigma\rangle \to \langle \mathbf{if}\, b \leq 10 \,\mathbf{then}\, (b := b + 1; \mathbf{while}\, b \leq 10 \,\mathbf{do}\, b := b + 1) \,\mathbf{else\,skip}, \sigma\rangle$$

1.4 Inference Rules

Inference rules can be used to construct proof trees and derive conclusions about an expression or language.

1.4.1 Anatomy of an Inference Rule

$$\frac{\langle a, \sigma\rangle \ \longrightarrow\ \langle a', \sigma\rangle}{\langle x := a, \sigma\rangle \ \longrightarrow\ \langle x := a', \sigma\rangle} \qquad \frac{Premise(s)}{Conclusion}$$

$\uparrow \qquad \uparrow\uparrow$

meta-variables:

One can think of these rules as defining a *relation* R on four things:

$\mathrm{R} \subseteq Cmd \ \times State \times Cmd \times State$

1.4.2 Rule $\longrightarrow$ Rule Instances

A rule is any rule instance with consistent substitution of meta-variables.

Lets take the following rule:

$$\frac{a_1 \ \longrightarrow\ a_1'}{a_1 + a_2 \ \longrightarrow\ a_1' + a_2}$$

After meta-variables have been replaced, we obtain the following rule instance:

$$\frac{(3 * 4) \ \longrightarrow\ 12}{(3 * 4) + 5 \ \longrightarrow\ 12 + 5}$$

Another rule instance is the following, though it is useless:

$$\frac{(3 * 4) \ \longrightarrow\ 13}{(3 * 4) + 5 \ \longrightarrow\ 13 + 5}$$

1.4.3 Rule Instance Examples

Given the set {A, B, C, D}
And the following rule instances:

$$\frac{}{A} \quad \frac{A\ D}{C} \quad \frac{A}{D} \quad \frac{B}{C}$$

What elements of the set can we conclude, using derivations?
(derivation $\equiv$ finite height proof tree)

$$\text{A:} \quad \frac{}{A}, \qquad \text{D:} \quad \frac{\frac{}{A}}{D}, \qquad \text{and C:} \quad \frac{\frac{}{A} \quad \frac{\frac{}{A}}{D}}{C}$$

Example proof tree:

$$\frac{\frac{}{A} \quad \frac{\frac{}{A}}{D}}{C}$$

This is an example of an inductively defined set.

1.4.4 BNF proof system

Since a proof system is a set of rules, the BNF grammar can be viewed as a set of inference rules used to construct proof trees.

Rules:

$$\frac{}{n}\ \text{, and}\quad \frac{a_1 \quad a_2}{a_1\ +\ a_2} \qquad a ::= n \mid a_1 + a_2$$

Proof tree from rule instances:

$$\frac{\frac{}{4} \quad \frac{\frac{}{2} \quad \frac{}{3}}{2+3}}{4+(2+3)}$$

1.5 Big Step Operational Semantics

We now present the big-step semantics for evaluation of Commands, Arithmetic and Boolean expressions in IMP. As opposed to small-step SOS, which transistions from one IMP command at a time to the next step, big-step semantics defines the transition from a program and state to the final state. Big-step semantics is also know as "natural" semantics.

$$\begin{array}{ccc} \text{Small Step SOS} & \leftrightarrow & \text{Big Step SOS} \\ \langle c, \sigma_0 \rangle \ \longrightarrow^* \ \langle \mathbf{skip}, \sigma \rangle & \leftrightarrow & \langle c, \sigma \rangle \Downarrow \sigma \end{array}$$

1.5.1 Rules

Skip: $\quad \dfrac{}{\langle \mathbf{skip}, \sigma \rangle \Downarrow \sigma}$

Sequences: $\quad \dfrac{\langle c_1, \sigma \rangle \Downarrow \sigma'' \quad \langle c_2, \sigma'' \rangle \Downarrow \sigma'}{\langle c_1; c_2, \sigma \rangle \Downarrow \sigma'}$

Arithmetic: $\quad \dfrac{}{\langle a, \sigma \rangle \Downarrow \ n}$

Boolean: $\quad \dfrac{}{\langle b, \sigma \rangle \Downarrow t}$

Assignment: $\quad \dfrac{}{\langle x := a, \sigma \rangle \Downarrow \sigma[x \mapsto n]}$

If-Else Condition: $\quad \dfrac{\langle b, \sigma \rangle \Downarrow \mathbf{false} \quad \langle c_1, \sigma \rangle \Downarrow \sigma'}{\langle \mathbf{if}\ b\ \ \mathbf{then}\ c_1\ \mathbf{else}\ c_2, \sigma \rangle \Downarrow \sigma'}$

While Loops:

$$\frac{\langle b, \sigma \rangle \Downarrow \mathbf{false}}{\langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma \rangle \Downarrow \sigma}$$

$$\frac{\langle b, \sigma \rangle \Downarrow \mathbf{true} \quad \langle c, \sigma \rangle \Downarrow \sigma'' \quad \langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma'' \rangle \Downarrow \sigma'}{\mathbf{while}\ \ \mathbf{do}\ b\ c, \sigma \rangle \Downarrow \sigma}$$

1.6 Big-Step SOS vs. Small-Step SOS

Benefits of Big-step

- more extensional, relating initial and final states
- models a recursive interpreter
  (The proof tree exactly corresponds to the call tree of the interpreter.)

Benefits of Small-step

- can model more language features
- better for proving some properties of languages

Downside of Big-step:

- Non-terminating programs look the same as those with error(s).
  For both cases the solver will say there is no valid proof tree for the program.

Infinite Loop: $\langle \textbf{while true do skip}, \sigma \rangle \longrightarrow \langle \textbf{while true do skip}, \sigma \rangle \longrightarrow \cdots$

Arithmetic Error: $\langle x := \frac{2}{0}, \sigma \rangle \longrightarrow ?$