CS611 Lecture 03 **Large-Step Operational Semantics I** 3 September 2003

Scribe: Dan Williams, Art Munson Lecturer: Radu Rugina

Lecture 2 covered the syntax of IMP and the relations defining evaluation of arithmetic and boolean expressions and commands. In order to give formal meaning to the syntax of IMP, we next need to define the inference rules governing the large step operational semantics of the language. After that, we will give an example of a proof tree for a simple IMP program.

1 Inference Rules

An *inference rule* is used to inductively define a set. Suppose we have n elements $x_1 \dots x_n$ that are *premises*, and from them it is valid to reach the *conclusion* x_{n+1}. Then we can write the following inference rule:

$$\frac{x_1 \dots x_n}{x_{n+1}}$$

If there are no premises, then the rule is an *axiom*. For example:

$$\frac{}{x}$$

Rules that contain *metavariables*, such as n in an expression, are called *rule schemas*, while instantiated versions of these rules are known as *rule instances*. Rule instances replace metavariables with concrete syntax from the language.

We use inference rules to define the operational semantics of IMP. The rules allow us to infer what are the elements of the evaluation relations. Also, the rules correspond to productions in the grammar of the language. When the syntax is defined inductively, the corresponding rule will usually follow the inductive structure of the syntax in its premises.

1.1 Arithmetic Expressions

The axioms for constants and variables are:

$$\frac{}{\langle n,\, \sigma\rangle \to n} \qquad \frac{}{\langle X,\, \sigma\rangle \to \sigma(X)}$$

Recall that σ is a state in Σ, the set of functions from **Loc** to $\mathbb{Z}$. Given X, $\sigma(X)$ returns the value stored in location X.

The inference rules governing addition, subtraction, and multiplication are:

$$\frac{\langle a_0,\, \sigma\rangle \to n_0 \quad \langle a_1,\, \sigma\rangle \to n_1}{\langle a_0 \oplus a_1,\, \sigma\rangle \to n} \text{ where } n = n_0 \oplus n_1, \text{ and } \oplus \in \{+, -, *\}$$

1.2 Boolean Expressions

The axioms for truth values are:

$$\frac{}{\langle \mathbf{true},\, \sigma\rangle \to \mathit{true}} \quad \frac{}{\langle \mathbf{false},\, \sigma\rangle \to \mathit{false}}$$

The rule for the comparison operations = and < is:

$$\frac{\langle a_0,\, \sigma\rangle \to n_0 \quad \langle a_1,\, \sigma\rangle \to n_1}{\langle a_0 \ominus a_1,\, \sigma\rangle \to t} \text{ where } t = n_0 \ominus n_1, \text{ and } \ominus \in \{=, <\}$$

The rule for the conjunction and disjunction operations $\wedge$ and $\vee$ is:

$$\frac{\langle b_0,\, \sigma\rangle \to t_0 \quad \langle b_1,\, \sigma\rangle \to t_1}{\langle b_0 \owedge b_1,\, \sigma\rangle \to t} \text{ where } t = t_0 \owedge t_1, \text{ and } \owedge \in \{\wedge, \vee\}$$

Note that this does not behave like C or Java where boolean evaluation of conjunction and disjunction short circuits. See Section 1.4 for this discussion.

The rule for the negation operation is:

$$\frac{\langle b_0,\, \sigma\rangle \to t_0}{\langle \neg b_0,\, \sigma\rangle \to t} \text{ where } t = \neg t_0$$

1.3 Commands

The only axiom for commands is **skip**:

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

The rule for assignment is:

$$\frac{\langle a,\, \sigma\rangle \to n}{\langle X := a,\, \sigma\rangle \to \sigma[X \mapsto n]}$$

The notation $\sigma[X \mapsto n]$ describes a new state (i.e., a map) identical to σ except for the value of the variable X, which now becomes n. Some texts (including Winskel) use an alternate notation $\sigma[n/X]$ to represent this.

The inference rule for a sequence of commands is:

$$\frac{\langle c_0,\, \sigma\rangle \to \sigma' \quad \langle c_1,\, \sigma'\rangle \to \sigma''}{\langle c_0; c_1,\, \sigma\rangle \to \sigma''}$$

The inference rules for **if** statements are:

$$\frac{\langle b,\, \sigma\rangle \to \mathit{true} \quad \langle c_0,\, \sigma\rangle \to \sigma'}{\langle \mathbf{if}\, b\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1,\, \sigma\rangle \to \sigma'} \quad \frac{\langle b,\, \sigma\rangle \to \mathit{false} \quad \langle c_1,\, \sigma\rangle \to \sigma'}{\langle \mathbf{if}\, b\, \mathbf{then}\, c_0\, \mathbf{else}\, c_1,\, \sigma\rangle \to \sigma'}$$

The inference rules for **while** loops are:

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

The more complex rule can be thought of as unrolling the loop once.

1.4 Alternative Semantics for Boolean Expressions

Notice that the inference rules for boolean expressions discussed above do not short circuit for $\wedge$ and $\vee$. *Ordered evaluation* for boolean conjunction is defined as:

$$\frac{\langle b_0,\, \sigma\rangle \to \mathit{false}}{\langle b_0 \wedge b_1,\, \sigma\rangle \to \mathit{false}} \quad \frac{\langle b_0,\, \sigma\rangle \to \mathit{true} \quad \langle b_1,\, \sigma\rangle \to t}{\langle b_0 \wedge b_1,\, \sigma\rangle \to t}$$

In other words, if b_0 evaluates to *false*, b_1 is never evaluated. The rules for disjunction are similar:

$$\frac{\langle b_0,\, \sigma\rangle \to \mathit{true}}{\langle b_0 \vee b_1,\, \sigma\rangle \to \mathit{true}} \quad \frac{\langle b_0,\, \sigma\rangle \to \mathit{false} \quad \langle b_1,\, \sigma\rangle \to t}{\langle b_0 \vee b_1,\, \sigma\rangle \to t}$$

These evaluation rules are not symmetric, however. In evaluating $b_0 \wedge b_1$, we can say the expression is *false* if b_1 is *false*, without regard for the value of b_0. We can capture these semantics using *parallel evaluation*:

$$\frac{\langle b_0,\,\sigma\rangle \to \mathit{false}}{\langle b_0 \wedge b_1,\,\sigma\rangle \to \mathit{false}} \qquad \frac{\langle b_1,\,\sigma\rangle \to \mathit{false}}{\langle b_0 \wedge b_1,\,\sigma\rangle \to \mathit{false}} \qquad \frac{\langle b_0,\,\sigma\rangle \to \mathit{true} \quad \langle b_1,\,\sigma\rangle \to \mathit{true}}{\langle b_0 \wedge b_1,\,\sigma\rangle \to \mathit{true}}$$

Both b_0 and b_1 are checked in parallel. If either is *false*, $b_0 \wedge b_1$ evaluates to *false*. Otherwise, the expression is *true*. Again, the rules for disjunction are similar:

$$\frac{\langle b_0,\,\sigma\rangle \to \mathit{true}}{\langle b_0 \vee b_1,\,\sigma\rangle \to \mathit{true}} \qquad \frac{\langle b_1,\,\sigma\rangle \to \mathit{true}}{\langle b_0 \vee b_1,\,\sigma\rangle \to \mathit{true}} \qquad \frac{\langle b_0,\,\sigma\rangle \to \mathit{false} \quad \langle b_1,\,\sigma\rangle \to \mathit{false}}{\langle b_0 \vee b_1,\,\sigma\rangle \to \mathit{false}}$$

Note that although there is nondeterminism in the execution of the parallel evaluation semantics,[1] the result of the evaluation is still deterministic. In a language that allows side effects, however, parallel evaluation would allow nondeterministic results. Using parallel evaluation semantics can be useful in shrinking the size of proof trees.

2 Proof Trees

A *proof tree* uses rule instances to prove a given fact or find the result of evaluating a program. The idea of the proof tree is to show how to reach a conclusion based on specific instances of the rules and axioms of the language. The conclusion is the root of the proof tree; axioms are the leaves of the tree; and internal nodes are rule instances. It is important to require that *proof trees must be finite*. Otherwise, the conclusion is not valid.

2.1 Example Proof Tree

Given: $c \equiv \textbf{if}\, X < Y \,\textbf{then}\, Z := Y - X \,\textbf{else}\, Z := X - Y$ with an initial state $\sigma = \{X \mapsto 1, Y \mapsto 2, Z \mapsto 3\}$. Find σ' such that $\langle c,\,\sigma\rangle \to \sigma'$.

$$\frac{\dfrac{\dfrac{}{\langle X,\,\sigma\rangle \to 1} \quad \dfrac{}{\langle Y,\,\sigma\rangle \to 2}}{\langle X < Y,\,\sigma\rangle \to \mathit{true}} \qquad \dfrac{\dfrac{\dfrac{}{\langle Y,\,\sigma\rangle \to 2} \quad \dfrac{}{\langle X,\,\sigma\rangle \to 1}}{\langle Y - X,\,\sigma\rangle \to 1}}{\langle Z := Y - X,\,\sigma\rangle \to \sigma[Z \mapsto 1]}}{\langle \textbf{if}\, X < Y \,\textbf{then}\, Z := Y - X \,\textbf{else}\, Z := X - Y,\,\sigma\rangle \to \sigma[Z \mapsto 1]}$$

So, $\sigma' = \sigma[Z \mapsto 1]$.

[1]To clarify, the nondeterminism is due to the fact that the rules do not explicitly state whether b_0 or b_1 is evaluated first.