CS611 Lecture 05 **Small Step Operational Semantics** 8 September 2003

Scribe: David Martin, Pavel Dmitriev Lecturer: Radu Rugina

# 1 The One-Step Relation

Small Step Operational Semantics are used to explicitly define how computations are performed. Instead of describing just the result of a computation, they describe the full sequence of atomic operations that produce the computation.

## 1.1 Commands

For commands, configurations are of the form $\langle c, \sigma \rangle \in \mathbf{Com} \times \Sigma$. Here $c$ is a pending command to execute, and $\sigma$ is the current state. We want to gradualy evaluate the command in a number of steps, until there is nothing to execute. Therefore final configurations take the form $\langle \mathbf{skip}, \sigma \rangle$, for all $\sigma \in \Sigma$.

To show a step of the execution, we define $\rightarrow_1$ as a relation on $(\mathbf{Com} \times \Sigma) \times (\mathbf{Com} \times \Sigma)$. We will use the following notation: $\langle c, \sigma \rangle \rightarrow_1 \langle c', \sigma' \rangle$, with the meaning that $\langle c', \sigma' \rangle$ is the configuration after one step of execution of $c$ in state $\sigma$. The complete execution of the program is a sequence of one-step evaluations: $\langle c, \sigma \rangle \rightarrow_1 \langle c', \sigma' \rangle \rightarrow_1 \ldots \rightarrow_1 \langle \mathbf{skip}, \sigma_{final} \rangle$.

We will define the semantics of individual commands using inference rules, the same way as we did it for Large Step Operational Semantics.

### 1.1.1 Skip

$c \equiv \mathbf{skip}$

There are no inference rules for **skip** since every $\langle \mathbf{skip}, \sigma \rangle$ is a final configuration as described above.

### 1.1.2 Assignment

$c \equiv X := a$. There are two evaluation rules for assignments:

$$\frac{\langle a, \sigma \rangle \rightarrow_1 a'}{\langle X := a, \sigma \rangle \rightarrow_1 \langle X := a', \sigma \rangle}$$

$$\frac{}{\langle X := n, \sigma \rangle \rightarrow_1 \langle \mathbf{skip}, \sigma[X \mapsto n] \rangle}$$

The first rule is applied several times to gradually refine expression $a$. Then, when the evaluation reached a number, the second rule is applied. This rule updates the store $\sigma$ and leads the execution to the final configuration.

### 1.1.3 Sequence

$c \equiv c_0; c_1$

$$\frac{\langle c_0, \sigma \rangle \rightarrow_1 \langle c_0', \sigma' \rangle}{\langle c_0; c_1, \sigma \rangle \rightarrow_1 \langle c_0'; c_1, \sigma' \rangle}$$

$$\frac{}{\langle \mathbf{skip}; c_1, \sigma \rangle \rightarrow_1 \langle c_1, \sigma' \rangle}$$

### 1.1.4 Conditional

$c \equiv \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1$

$$\frac{\langle b, \sigma \rangle \to_1 b'}{\langle \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1, \sigma \rangle \to_1 \langle \textbf{if } b' \textbf{ then } c_0 \textbf{ else } c_1, \sigma \rangle}$$

$$\frac{}{\langle \textbf{if true then } c_0 \textbf{ else } c_1, \sigma \rangle \to_1 \langle c_0, \sigma \rangle}$$

$$\frac{}{\langle \textbf{if false then } c_0 \textbf{ else } c_1, \sigma \rangle \to_1 \langle c_1, \sigma \rangle}$$

### 1.1.5 Loop

$c \equiv \textbf{while } b \textbf{ do } c_0$

Suppose we tried to define rules of inference for loops similar to those for conditionals. Then we would obtain something that looks like the following:

$$\frac{\langle b, \sigma \rangle \to_1 b'}{\langle \textbf{while } b \textbf{ do } c_0, \sigma \rangle \to_1 \langle \textbf{while } b' \textbf{ do } c_0, \sigma \rangle}$$

$$\frac{}{\langle \textbf{while false do } c_0, \sigma \rangle \to_1 \langle \textbf{skip}, \sigma \rangle}$$

$$\frac{}{\langle \textbf{while true do } c_0, \sigma \rangle \to_1 \cdots}$$

Notice that any loop for which the $b$ initially evaluates to *true* will reduce to an infinite loop. Thus, *this set of rules is wrong*. The problem with is that these rules evaluate the condition $b$ once for all of the loop iterations; we should instead evaluate $b$ at each iteration.

The correct rule unrolls the loop once using an if command:

$$\frac{}{\langle \textbf{while } b \textbf{ do } c_0, \sigma \rangle \to_1 \langle \textbf{if } b \textbf{ then } (c_0; \textbf{while } b \textbf{ do } c_0) \textbf{ else skip}, \sigma \rangle}$$

## 1.2 Arithmetic Expressions

For arithmetic expressions, we define $\to_1$ as a relation on $(\mathbf{Aexp} \times \Sigma) \times \mathbf{Aexp}$. We will use the notation $\langle a, \sigma \rangle \to_1 a'$ to show that $a'$ is the expression left to evaluate after one step of evaluation of the expression $a$ in state $\sigma$. The evaluation terminates it reaches a number. Therefore, the final configuration for the evaluation of an arithmetic expression in a particular state is a number $n \in \mathbb{Z}$.

### 1.2.1 Constants

$a \equiv n$

There are no inference rules for constants since they are final configurations for arithmetic expressions.

### 1.2.2 Variables

$a \equiv X$

$$\frac{}{\langle X, \sigma \rangle \to_1 \sigma(X)}$$

### 1.2.3 Complex Expressions

$a \equiv a_0 \oplus a_1$, where $\oplus \in \{+, -, \times\}$

$$\frac{\langle a_0, \sigma\rangle \rightarrow_1 a_0'}{\langle a_0 \oplus a_1, \sigma\rangle \rightarrow_1 a_0' \oplus a_1}$$

$$\frac{\langle a_1, \sigma\rangle \rightarrow_1 a_1'}{\langle n_0 \oplus a_1, \sigma\rangle \rightarrow_1 n_0 \oplus a_1'}$$

$$\frac{}{\langle n_0 \oplus n_1, \sigma\rangle \rightarrow_1 n}\ n = n_0 \oplus n_1$$

Note the distinction between the $\oplus$ that occurs in the conclusion the rule, and the $\oplus$ that occurs as a part of the metalanguage inside the side condition.

### 1.3 Boolean Expressions

Boolean expressions are similar to Arithmetic expressions.

Notice that each inference rule has at most one premise. Therefore, the derivation for each step of Small Step Operational Semantics ends up being a chain.

## 2 Reflexive & Transitive Closure of the One-Step Relation

We define a relation $\rightarrow_1^*$ as the reflexive and transitive closure of $\rightarrow_1$:

$$\frac{}{\langle c, \sigma\rangle \rightarrow_1^* \langle c, \sigma\rangle}$$

$$\frac{\langle c, \sigma\rangle \rightarrow_1 \langle c', \sigma'\rangle}{\langle c, \sigma\rangle \rightarrow_1^* \langle c', \sigma'\rangle}$$

$$\frac{\langle c, \sigma\rangle \rightarrow_1^* \langle c', \sigma'\rangle \quad \langle c', \sigma'\rangle \rightarrow_1^* \langle c'', \sigma''\rangle}{\langle c, \sigma\rangle \rightarrow_1^* \langle c'', \sigma''\rangle}$$

This is a binary relation on two configurations, the second of which can be derived in 0 or more steps from the first.

## 3 Equivalence of Semantics

So far, we've seen two different ways of describing the meaning of our IMP language: Large Step Operational Semantics and Small Step Operational Semantics. Do they describe the same program meaning?

We say that the large step evaluation relation $\rightarrow$ is equivalent to the small step evaluation relation $\rightarrow_1^*$ if, starting with the same input state, they yield the same final state. Formally:

$$\forall c\ \forall \sigma, \sigma'\ .\ (\langle c, \sigma\rangle \rightarrow \sigma' \iff \langle c, \sigma\rangle \rightarrow_1^* \langle \mathbf{skip}, \sigma'\rangle)$$

To compare program evaluations under large step and small step semantics, consider the following program:

$$\mathbf{if}\ X < Y\ \mathbf{then}\ Z := Y - X\ \mathbf{else}\ Z := X - Y$$

Let's evaluate it in the state $\sigma = \{X \mapsto 1,\ Y \mapsto 2,\ Z \mapsto 3\}$.

The Large Step Operational Semantics proof tree is as follows:

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

The Small Step Operational Semantics execution sequence is as follows:

$$\begin{array}{ll}
 & \langle \mathbf{if}\ X < Y\ \mathbf{then}\ Z := Y - X\ \mathbf{else}\ Z := Y - X, \sigma\rangle \\
\to_1 & \langle \mathbf{if}\ 1 < Y\ \mathbf{then}\ Z := Y - X\ \mathbf{else}\ Z := Y - X, \sigma\rangle \\
\to_1 & \langle \mathbf{if}\ 1 < 2\ \mathbf{then}\ Z := Y - X\ \mathbf{else}\ Z := Y - X, \sigma\rangle \\
\to_1 & \langle \mathbf{if}\ \mathbf{true}\ \mathbf{then}\ Z := Y - X\ \mathbf{else}\ Z := Y - X, \sigma\rangle \\
\to_1 & \langle Z := Y - X, \sigma\rangle \\
\to_1 & \langle Z := 2 - X, \sigma\rangle \\
\to_1 & \langle Z := 2 - 1, \sigma\rangle \\
\to_1 & \langle Z := 1, \sigma\rangle \\
\to_1 & \langle \mathbf{skip}, \sigma[Z \mapsto 1]\rangle
\end{array}$$

The proof tree for each step in this sequence is a chain.

# 4 Program Properties

We can use the transitive, reflexive closure of the one step evaluation relation $\to_1^*$ to express the following program properties: equivalence, termination, and deterministic evaluation.

## 4.1 Equivalence

$c_0$ and $c_1$ are equivalent ($c_0 \sim c_1$) iff

$$\forall \sigma, \sigma' \ . \ (\langle c_0, \sigma\rangle \to_1^* \langle \mathbf{skip}, \sigma'\rangle \Leftrightarrow \langle c_1, \sigma\rangle \to_1^* \langle \mathbf{skip}, \sigma'\rangle)$$

### 4.1.1 Termination

The execution of a command $c$ in state $\sigma$ terminates iff

$$\exists \sigma' \ . \ (\langle c, \sigma\rangle \to_1^* \langle \mathbf{skip}, \sigma'\rangle)$$

### 4.1.2 Determinism

The execution of a command $c$ in state $\sigma\rangle$ is deterministic iff

$$\forall \sigma', \sigma'' \ . \ ((\langle c, \sigma\rangle \to_1^* \langle \mathbf{skip}, \sigma'\rangle \wedge \langle c, \sigma\rangle \to_1^* \langle \mathbf{skip}, \sigma''\rangle) \Rightarrow \sigma' = \sigma'')$$

# 5 Expressiveness of Large-Step Semantics

We next look at how easy is it to express non-termination, deal with errors, or support other language constructs, such as nondeterministic or parallel constructs.

## 5.1 Non-termination

Non-termination is characterized by a cycle in the execution sequence. For example,

$$\begin{array}{ll}
 & \langle \mathbf{while}\ \mathbf{true}\ \mathbf{do}\ \mathbf{skip}, \sigma\rangle \\
\to_1 & \langle \mathbf{if}\ \mathbf{true}\ \mathbf{then}\ (\mathbf{skip};\ \mathbf{while}\ \mathbf{true}\ \mathbf{do}\ \mathbf{skip})\ \mathbf{else}\ \mathbf{skip}, \sigma\rangle \\
\to_1 & \langle \mathbf{skip};\ \mathbf{while}\ \mathbf{true}\ \mathbf{do}\ \mathbf{skip}, \sigma\rangle \\
\to_1 & \langle \mathbf{while}\ \mathbf{true}\ \mathbf{do}\ \mathbf{skip}, \sigma\rangle \\
\to_1 & \ldots
\end{array}$$

### 5.2 Errors

To illustrate error handling, we extend arithmetic expressions with division. Then an attempt to divide by zero should cause an error. The extended syntax for arithmetic expressions is:

$$a ::= \cdots \mid a_0/a_1$$

We also need to add inference rules similar to the other operators. Add axioms as follows:

$$\frac{}{\langle n_0/n_1, \sigma\rangle \to_1 n}\ n_1 \neq 0, n = n_0/n_1$$

$$\frac{}{\langle n_0/0, \sigma\rangle \to_1 \mathit{Error}}$$

Note that there is no need to propagate errors, as was the case with Large Step Operational Semantics.

### 5.3 Parallelism

We extend commands to include the following parallel construct:

$$c ::= \cdots \mid c_0 || c_1$$

with the meaning that commands $c_0$ and $c_1$ are being executed in parallel. We express the semantics for this construct using three evaluation rules:

$$\frac{\langle c_0, \sigma\rangle \to_1 \langle c_0', \sigma'\rangle}{\langle c_0||c_1, \sigma\rangle \to_1 \langle c_0'||c_1, \sigma'\rangle}$$

$$\frac{\langle c_1, \sigma\rangle \to_1 \langle c_1', \sigma'\rangle}{\langle c_0||c_1, \sigma\rangle \to_1 \langle c_0||c_1', \sigma'\rangle}$$

$$\frac{}{\langle \mathbf{skip}||\mathbf{skip}, \sigma\rangle \to_1 \langle \mathbf{skip}, \sigma\rangle}$$

Note that we can easily express fine-grain parallelism with Small Step Operational Semantics, i.e., we can express the interleaving of individual commands from the parallel threads. This was not possible with Large Step Operational Semantics.