A configuration is a tuple of the form \( \langle c, \sigma \rangle \), where \( c \) is the command to be executed, and \( \sigma \) is the current store. The program terminates when we reach a configuration of the form \( \langle \text{skip}, \sigma \rangle \). The idea behind small-step semantics is that we make one small step at a time. A small step is the evaluation of some part of the expression. The notation is as follows: if \( a \) denotes some arithmetic expression, then \( a' \) denotes \( a \) after one small step was made. Similarly, if \( b \) is a boolean expression, then \( b' \) is \( b \) after one small step. These notes cover the small-step semantics.

Rules are of the form \( \langle c, \sigma \rangle \mapsto \langle c', \sigma' \rangle \), where \( \mapsto \subseteq (\text{Com} \times \text{Store}) \times (\text{Com} \times \text{Store}) \)

1 Commands

1.1 Skip
\( \langle \text{skip}, \sigma \rangle - \) we are at the final step. No rule is needed.

1.2 Assignment

\[
\begin{align*}
\langle a, \sigma \rangle &\mapsto \langle a', \sigma \rangle \\
\langle x := a, \sigma \rangle &\mapsto \langle x := a', \sigma \rangle \\
\langle x := n, \sigma \rangle &\mapsto \langle \text{skip}[x \mapsto n], \sigma \rangle
\end{align*}
\]

1.3 ; (Semicolon)

\[
\begin{align*}
\langle c_0, \sigma \rangle &\mapsto \langle c'_0, \sigma' \rangle \\
\langle c_0; c_1, \sigma \rangle &\mapsto \langle c'_0; c_1, \sigma' \rangle \\
\langle \text{skip}; c_1, \sigma \rangle &\mapsto \langle \text{skip}[x \mapsto n], \sigma \rangle
\end{align*}
\]

1.4 If

\[
\begin{align*}
\langle b, \sigma \rangle &\mapsto \langle b', \sigma \rangle \\
\langle \text{if } b \text{ then } c_0 \text{ else } c_1, \sigma \rangle &\mapsto \langle \text{if } b' \text{ then } c_0 \text{ else } c_1, \sigma \rangle \\
\langle \text{if } \text{true} \text{ then } c_0 \text{ else } c_1, \sigma \rangle &\mapsto \langle c_0, \sigma \rangle \\
\langle \text{if } \text{false} \text{ then } c_0 \text{ else } c_1, \sigma \rangle &\mapsto \langle c_1, \sigma \rangle
\end{align*}
\]

1.5 While

\[
\langle \text{while } b \text{ do } c, \sigma \rangle \mapsto \langle \text{if } b \text{ then } (c; \text{ while } b \text{ do } c) \text{ else } \text{skip}, \sigma \rangle
\]

1.6 Variable evaluation

\[
\langle x, \sigma \rangle \mapsto \langle \sigma(x), \sigma \rangle
\]
2 Order of evaluation

Some rules enforce the order of evaluation and other rules actually evaluate. For instance, simply having the rule on the left will force left-to-right evaluation, while having both allows evaluation of the right side before the evaluation of the left side has been completed:

\[
\begin{align*}
\langle a_0, \sigma \rangle &\mapsto \langle a_0', \sigma \rangle \\
\langle a_0 + a_1, \sigma \rangle &\mapsto \langle a_0' + a_1, \sigma \rangle \\
\langle a_1, \sigma \rangle &\mapsto \langle a_1', \sigma \rangle \\
\langle a_0 + a_1, \sigma \rangle &\mapsto \langle a_0 + a_1', \sigma \rangle
\end{align*}
\]

3 Arithmetic Expressions

Evaluation of arithmetic expressions proceeds as normal with addition (left rule):

\[
\begin{align*}
\langle n = n_0 + n_1, \sigma \rangle &\mapsto \langle n, \sigma \rangle \\
\langle n_0 + n_1, \sigma \rangle &\mapsto \langle n, \sigma \rangle
\end{align*}
\]

The rule for division (right rule above), however, cannot be accepted since it may result in a runtime error on \( (2 \div 0, \sigma) \mapsto ? \), resulting in a stuck configuration.

4 Parallelism

Since the commands in the language \texttt{IMP} may lack interdependency on each other, we may allow command evaluation to proceed in parallel as given by these inference rules:

\[
\begin{align*}
\langle c_0, \sigma \rangle &\mapsto \langle c_0', \sigma' \rangle \\
\langle c_0 | c_1, \sigma \rangle &\mapsto \langle c_0', c_1, \sigma \rangle \\
\langle c_1, \sigma \rangle &\mapsto \langle c_1', \sigma' \rangle \\
\langle c_0 | c_1, \sigma \rangle &\mapsto \langle c_0 | c_1', \sigma \rangle
\end{align*}
\]

This allows evaluation of either commands in a pair of parallel commands proceed before completion of evaluation of the other.

5 Non-determinism

Non-determinism allows us to specify that either of two commands will be executed at run time:

\[
\langle c_0 \bigtriangleup c_1, \sigma \rangle \mapsto \langle c_0, \sigma \rangle \quad \langle c_0 \bigtriangleup c_1, \sigma \rangle \mapsto \langle c_1, \sigma \rangle
\]

It is interesting to note that we could not specify either parallelism or non-determinism using large-step semantics, but small-step semantics allow us to express both succinctly.

6 Equivalence of Large- and Small-Step Semantics

In addition, it turns out that large-step semantics are equivalent to small-step semantics. Define the relation \( \mapsto^* \) as follows:

\[
\begin{align*}
\langle c, \sigma \rangle &\mapsto^* \langle c, \sigma \rangle \\
\langle c, \sigma \rangle &\mapsto^* \langle c', \sigma' \rangle \\
\langle c', \sigma' \rangle &\mapsto^* \langle c'', \sigma'' \rangle
\end{align*}
\]

The idea is to prove that

\[
\langle c, \sigma \rangle \Downarrow \sigma' \iff \langle a, \sigma \rangle \mapsto^* \langle \text{skip}, \sigma' \rangle
\]

Proof (this lecture covered only arithmetic expressions): by induction on the depth of the abstract syntax tree of the expression. For arithmetic expressions we need

\[
\bullet \langle a, \sigma \rangle \Downarrow n \iff \langle a, \sigma \rangle \mapsto^* \langle n, \sigma \rangle
\]
• \langle x, \sigma \rangle \Downarrow \sigma(x) \iff \langle x, \sigma \rangle \mapsto^* \langle \sigma(x), \sigma \rangle \\
• \langle a_0 \oplus a_1, \sigma \rangle \Downarrow n \iff \langle a_0 \oplus a_1, \sigma \rangle \mapsto^* \langle n, \sigma \rangle \\

The first two cases are trivial.

Now assume \langle a_0 \oplus a_1, \sigma \rangle \Downarrow n. Then \langle a_0, \sigma \rangle \Downarrow n_0 and \langle a_1, \sigma \rangle \Downarrow n_1, where n_0 \oplus n_1 = n. By induction hypothesis, \langle a_0, \sigma \rangle \mapsto^* \langle n_0, \sigma \rangle and \langle a_0, \sigma \rangle \mapsto^* \langle n_0, \sigma \rangle, since the tree associated with a_0 and the tree associated with a_1 are both less deep than the tree associated with a_0 \oplus a_1. Therefore, by induction, we have \langle a_0, \sigma \rangle \mapsto^* \langle n_0, \sigma \rangle, and \langle a_0, \sigma \rangle \mapsto^* \langle n_0, \sigma \rangle. Thus, \langle a_0 \oplus a_1, \sigma \rangle \mapsto^* \langle n_0 \oplus a_1, \sigma \rangle \mapsto^* \langle n_0 \oplus n_1, \sigma \rangle \mapsto \langle n, \sigma \rangle.

The other direction was not covered in lecture.