Small step semantics

The semantics that we have seen so far (*eg.* $\langle c, \sigma \rangle \Downarrow \sigma'$) directly depicts, in a single step, the final state (σ') obtained when a command (c) is executed in the current state of the system (σ). This style of semantics is known as *large step semantics* (or *natural semantics*).

However, it is sometimes necessary to have a fine grained view of execution of a command as a series of smaller atomic substeps from one configuration to another. This is required, for example, to express parallel execution or for tracing the execution of a command.

Small step semantics is used to describe the execution of a command in terms of several smaller steps from one configuration to another.

$$eg.\langle c, \sigma \rangle \rightarrow \langle c', \sigma' \rangle \ldots \rightarrow \langle c'', \sigma'' \rangle$$

Consider, for example, the configuration:

$$\langle \textbf{if } X < Y \textbf{ then } X := 0 \textbf{ else skip}, [X \mapsto 1, Y \mapsto 2] \rangle$$

One way of expressing this as a series of incremental steps could be:

$$\begin{array}{ll} & \langle \textbf{if } X < Y \textbf{ then } X := 0 \textbf{ else skip}, [X \mapsto 1, Y \mapsto 2] \rangle \\ \rightarrow & \langle \textbf{if } 1 < y \textbf{ then } X := 0 \textbf{ else skip}, [X \mapsto 1, Y \mapsto 2] \rangle \\ \rightarrow & \langle \textbf{if } 1 < 2 \textbf{ then } X := 0 \textbf{ else skip}, [X \mapsto 1, Y \mapsto 2] \rangle \\ \rightarrow & \langle \textbf{if } \mathit{true} \textbf{ then } X := 0 \textbf{ else skip}, [X \mapsto 1, Y \mapsto 2] \rangle \\ \rightarrow & \langle X := 0, [X \mapsto 1, Y \mapsto 2] \rangle \\ \rightarrow & \langle \textbf{skip}, [X \mapsto 0, Y \mapsto 2] \rangle \end{array}$$

In small step semantics, any configuration of the form $\langle \textbf{skip}, \sigma \rangle$ is called a *final configuration.* Any configuration that represents a computation that terminates can be brought to the form $\langle \textbf{skip}, \sigma \rangle$.

Since booleans and arithmetic expressions do not alter the state (no side effects) in IMP, we can represent one small-step of a boolean or arithmetic expression as:

$$\langle a, \sigma \rangle \rightarrow \langle a', \sigma \rangle \quad (a = \text{Arithmetic expression})$$
$$\langle b, \sigma \rangle \rightarrow \langle b', \sigma \rangle \quad (b = \text{Boolean expression})$$

However, since commands can change state, a single small-step in case of a command may be represented, in general, as:

$$\langle c, \sigma \rangle \rightarrow \langle c', \sigma' \rangle \quad (c = \text{Command})$$

For the sake of generality, we shall represent the new state after a small-step computation of a boolean or arithmetic expression in state σ as σ' (as opposed to σ).

Tracing command executions via small step semantics

- $\langle \textbf{skip}, \sigma \rangle$: Since $\langle \textbf{skip}, \sigma \rangle$ represents a final configuration, we shall not have any small step rule for its evaluation.

- $\langle X := a, \sigma\rangle$: Here a is an arithmetic expression and must first be evaluated to a number, via (possibly repeated) applications of the inference rule:

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

Once 'a' has been reduced to a number, say n, we can then apply the following axiom to reach the final state:

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

- $\langle a_0 \oplus a_1, \sigma\rangle$: ($\oplus$ is a generic symbol for a mathematical operator).
The following rules enforce left to right evaluation:

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

$$\frac{\langle a_1, \sigma\rangle \rightarrow \langle a_1{}', \sigma'\rangle}{\langle n \oplus a_1, \sigma\rangle \rightarrow \langle n \oplus a_1{}', \sigma'\rangle}$$

$$\frac{}{\langle n_0 \oplus n_1, \sigma\rangle \rightarrow \langle n_2, \sigma\rangle}\ (n_2 = n_0 \oplus n_1)$$

- $\langle c_0; c_1, \sigma\rangle$: Presumably, the semi-colon operator is a left to right one:

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

$$\langle \textbf{skip}; c, \sigma\rangle \rightarrow \langle c, \sigma\rangle$$

- $\langle$ **if** b **then** c_0 **else** c_1, $\sigma\rangle$:

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

For the case where the boolean expression evaluates to $true$, we have the axiom:

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

For the $false$ case:

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

- $\langle \textbf{while } b \textbf{ do } c, \sigma\rangle$:

$$\frac{\langle b, \sigma\rangle \rightarrow \langle b', \sigma'\rangle}{\langle \textbf{while } b \textbf{ do } c, \sigma\rangle \rightarrow \langle \textbf{while } b' \textbf{ do } c, \sigma'\rangle}$$

$$\langle \textbf{while } false \textbf{ do } c, \sigma\rangle \rightarrow \langle \textbf{skip}, \sigma\rangle$$

For the case when the looping condition is true, one needs to keep track of the boolean test condition for the next iteration, and hence cannot discard it:

$$\langle \textbf{while } b \textbf{ do } c, \sigma\rangle \rightarrow \langle \textbf{if } b \textbf{ then } c; \textbf{while } b \textbf{ do } c \textbf{ else skip}, \sigma\rangle$$

Advantages of Small-step semantics over Natural semantics

Using natural semantics, we could not describe computations that didn't halt. Using small step semantics, not only can we capture the notion of a non terminating computation, but we can also distinguish between an error condition and a non terminating computation.
eg. In natural semantics, we cannot find a σ' satisfying $\langle X := 1/0, \sigma\rangle \Downarrow \sigma'$ – and this error condition was indistinguishable from a non terminating computation.

In small step semantics, we can distinguish between a non terminating computation and an error condition:
eg. In case of the divide by zero error condition, when we reach the axiom:

$$\overline{\langle n_0 \oplus n_1, \sigma\rangle \to \langle n_2, \sigma\rangle}(n_2 = n_0 \oplus n_1)$$

we get 'stuck' and cannot proceed further, since we are unable to find a number n_2 satisfying $n_2 = n_0 \oplus n_1$. However, in case of an infinite computation, we never get 'stuck' – we just have an infinite sequence of steps.

Small step semantics also helps us deal with the notion of parallelism. Consider a parallel computation of the form $\langle \textbf{cobegin}\ c_0\ c_1, \sigma\rangle$. Its execution can be tracked by the following small step inference rules:

$$\frac{\langle c_0, \sigma\rangle \to \langle c_0{}', \sigma'\rangle}{\langle \textbf{cobegin}\ c_0\ c_1, \sigma\rangle \to \langle \textbf{cobegin}\ c_0{}'\ c_1, \sigma'\rangle}$$

$$\frac{\langle c_1, \sigma\rangle \to \langle c_1{}', \sigma'\rangle}{\langle \textbf{cobegin}\ c_0\ c_1, \sigma\rangle \to \langle \textbf{cobegin}\ c_0\ c_1{}', \sigma'\rangle}$$

$$\langle \textbf{cobegin skip skip}, \sigma\rangle \to \langle \textbf{skip}, \sigma\rangle$$

Equivalence of natural and small-step semantics

We now need to show that the two semantics (natural and small-step) that we used to describe IMP are equivalent to each other, viz. both semantics give the same evaluation for all legal programs.

Notation: Let $\langle c, \sigma\rangle \to^* \langle c', \sigma'\rangle$ denote the statement that the configuration $\langle c', \sigma'\rangle$ can be reached from $\langle c, \sigma\rangle$ in 0 or more 'small-steps'.

To prove the equivalence of the two semantics, we need to show:

$$\langle c, \sigma\rangle \Downarrow \sigma' \Leftrightarrow\ \langle c, \sigma\rangle \to^* \langle skip, \sigma'\rangle$$

Let us first prove that at least for arithmetic expressions, the above holds, viz:

$$\langle a, \sigma\rangle \Downarrow n \Leftrightarrow \langle a, \sigma\rangle \to^* \langle n, \sigma\rangle$$

Let us prove this case by case for all possible types of arithmetic expressions:

- **Number** n: $\langle n, \sigma\rangle \Downarrow n \Leftrightarrow \langle n, \sigma\rangle \to^* \langle n, \sigma\rangle$ (Trivially true)
- **Variable** X: $\langle X, \sigma\rangle \Downarrow n \Leftrightarrow \langle X, \sigma\rangle \to^* \langle n, \sigma\rangle$
  $LHS \Rightarrow n = \sigma(X) \Rightarrow \langle X, \sigma\rangle \to \langle n, \sigma\rangle \Rightarrow \langle X, \sigma\rangle \to^* \langle n, \sigma\rangle$.
  Similar for converse.
- $\langle a_0 \oplus a_1, \sigma\rangle \Downarrow n \Leftrightarrow\ \langle a_0 \oplus a_1, \sigma\rangle \to^* \langle n, \sigma\rangle$

  Can show this using structural induction on "size" of the arithmetic expression. By size, we mean the height of the parse tree corresponding to the arithmetic expression.

  Let P(m) denote the hypothesis $\langle a, \sigma\rangle \Downarrow n \Leftrightarrow \langle a, \sigma\rangle \to^* \langle n, \sigma\rangle$, if height(a)=m

  Base case: Height of parse tree of expression is 1 – already proven.

  Induction step: Assume P(i) holds for all a whose parse tree has height $1 \leq i \leq m$. Consider $a_0 \oplus a_1$ of height $m + 1$. By induction hypothesis,

  $$\langle a_0, \sigma\rangle \Downarrow n_0 \Leftrightarrow \langle a_0, \sigma\rangle \to^* \langle n_0, \sigma\rangle \ \ldots (1)$$
  $$\langle a_1, \sigma\rangle \Downarrow n_1 \Leftrightarrow \langle a_1, \sigma\rangle \to^* \langle n_1, \sigma\rangle \ \ldots (2)$$

  Also, $\langle a_0 \oplus a_1, \sigma\rangle \Downarrow n \Rightarrow n_0 \oplus n_1 = n$

  $$(1) \Rightarrow \langle a_0 \oplus a_1, \sigma\rangle \to^* \langle n_0 \oplus a_1, \sigma\rangle$$
  $$(2) \Rightarrow \langle n_0 \oplus a_1, \sigma\rangle \to^* \langle n_0 \oplus n_1, \sigma\rangle$$

  Also, $\langle n_0 \oplus n_1, \sigma\rangle \to \langle n, \sigma\rangle$
  Thus $\langle a, \sigma\rangle \Downarrow n \Rightarrow \langle a, \sigma\rangle \to^* \langle n, \sigma\rangle$.
  Working backwards on similar lines, the converse follows.