Today we introduce a very simple imperative language, IMP, along with two systems of rules for evaluation called *small-step* and *big-step* semantics. These both fall under the general style called *structural operational semantics* (SOS). We will also discuss why both the big-step and small-step approaches can be useful.

1 The IMP Language

1.1 Syntax of IMP

There are three distinct types of expressions in IMP:

- *arithmetic expressions* AExp with elements denoted by $a, a_0, a_1, \ldots$
- *Boolean expressions* BExp with elements denoted by $b, b_0, b_1, \ldots$
- *commands* Com with elements denoted by $c, c_0, c_1, \ldots$

A *program* in the IMP language is a command in Com.

Let $n, n_0, n_1, \ldots$ denote integers (elements of $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$). We will view n both as a number (a semantic object) and as a numeral (a syntactic object) representing the number n. This little bit of ambiguity should not cause any confusion; the numbers and the numerals are in one-to-one correspondence, so there is really no need to distinguish them. Similarly, there is no need to distinguish between the Boolean constants true and false (syntactic objects) and the Boolean values *true*, *false* $\in \mathbb{2}$ (semantic objects).

Let *Var* be a countable set of variables ranging over $\mathbb{Z}$. Elements of *Var* are denoted $x, x_0, x_1 \ldots$.

The BNF grammar for IMP is

$$\begin{array}{rrcl} \textsf{AExp}: & a & ::= & n \mid x \mid a_0 + a_1 \mid a_0 * a_1 \mid a_0 - a_1 \\ \textsf{BExp}: & b & ::= & \textsf{true} \mid \textsf{false} \mid a_0 = a_1 \mid a_0 \leq a_1 \mid b_0 \vee b_1 \mid b_0 \wedge b_1 \mid \neg b \\ \textsf{Com}: & c & ::= & \textsf{skip} \mid x := a \mid c_0 \,;\, c_1 \mid \textsf{if } b \textsf{ then } c_1 \textsf{ else } c_2 \mid \textsf{while } b \textsf{ do } c \end{array}$$

Note that in this definition, $n + m$ denotes the syntactic expression with three symbols n, $+$, and m, not to the number that is the sum of n and m.

1.2 Environments and Configurations

An *environment* (also known as a *valuation* or a *store*) is a function $\sigma : \mathit{Var} \to \mathbb{Z}$ that assigns an integer to each variable. Environments are denoted $\sigma, \sigma_1, \tau, \ldots$ and the set of all environments is denoted *Env*. In IMP semantics, environments are always total functions.

A *configuration* is a pair $\langle c, \sigma \rangle$, where $c \in$ Com is a command and σ is an environment. Intuitively, the configuration $\langle c, \sigma \rangle$ represents an instantaneous snapshot of reality during a computation, in which σ represents the current values of the variables and c represents the next command to be executed.

2 Small-Step Semantics

Small-step semantics specifies the operation of a program one step at a time. There is a set of rules that we continue to apply to configurations until reaching a final configuration $\langle \mathsf{skip}, \sigma \rangle$ (if ever). We write $\langle c, \sigma \rangle \to \langle c', \sigma' \rangle$ to indicate that the configuration $\langle c, \sigma \rangle$ reduces to $\langle c', \sigma' \rangle$ in one step, and we write $\langle c, \sigma \rangle \to^* \langle c', \sigma' \rangle$ to indicate that $\langle c, \sigma \rangle$ reduces to $\langle c', \sigma' \rangle$ in zero or more steps. Thus $\langle c, \sigma \rangle \to^* \langle c', \sigma' \rangle$ iff there exists $k \geq 0$ and configurations $\langle c_0, \sigma_0 \rangle, \ldots, \langle c_k, \sigma_k \rangle$ such that $\langle c, \sigma \rangle = \langle c_0, \sigma_0 \rangle$, $\langle c', \sigma' \rangle = \langle c_k, \sigma_k \rangle$, and $\langle c_i, \sigma_i \rangle \to \langle c_{i+1}, \sigma_{i+1} \rangle$ for $0 \leq i \leq k-1$.

To be completely proper, we will define auxiliary small-step operators $\to_a$ and $\to_b$ for arithmetic and Boolean expressions, respectively, as well as $\to$ for commands.[1] These operators are partial functions:

$$\begin{aligned} \to_a &: (\mathsf{AExp} \times \mathit{Env}) \rightharpoonup \mathsf{AExp} \\ \to_b &: (\mathsf{BExp} \times \mathit{Env}) \rightharpoonup \mathsf{BExp} \\ \to &: (\mathsf{Com} \times \mathit{Env}) \rightharpoonup (\mathsf{Com} \times \mathit{Env}) \end{aligned}$$

Intuitively, $\langle a, \sigma \rangle \to_a^* n$ if the expression a evaluates to the constant n in environment σ.

We now present the small-step rules for evaluation in IMP. Just as with the λ-calculus, evaluation is defined by a set of inference rules that inductively define relations consisting of acceptable computation steps.

2.1 Arithmetic and Boolean Expressions

- Variables:

$$\frac{}{\langle x, \sigma \rangle \to_a \sigma(x)}$$

- Arithmetic:

$$\frac{}{\langle n_1 + n_2, \sigma \rangle \to_a n_3} \text{ (where } n_3 \text{ is the sum of } n_1 \text{ and } n_2\text{)}$$

$$\frac{\langle a_1, \sigma \rangle \to_a a_1'}{\langle a_1 + a_2, \sigma \rangle \to_a a_1' + a_2} \qquad \frac{\langle a_2, \sigma \rangle \to_a a_2'}{\langle n_1 + a_2, \sigma \rangle \to_a n_1 + a_2'}$$

and similar rules for $*$ and $-$. These rules say: If the reduction above the line can be performed, then the reduction below the line can be performed. The rules are thus inductive on the structure of the expression to be evaluated.

Note that there is no rule that applies in the case $\langle n, \sigma \rangle$. This configuration is *irreducible*. In all other cases, there is exactly one rule that applies.

The rules for Booleans and comparison operators are similar. We leave them as exercises.

2.2 Commands

Let us denote by $\sigma[n/x]$ the environment that is identical to σ except possibly for the value of x, which is n. That is,

$$\sigma[n/x](y) \quad \triangleq \quad \begin{cases} \sigma(y) & \text{if } y \neq x, \\ n & \text{if } y = x. \end{cases}$$

[1] Winskel [1] uses $\to_1$ instead of $\to$.

The construct $[n/x]$ is called a *rebinding operator*. It is a meta-operator that is used to rebind a variable to a different value in the environment. Note that in the definition of $\sigma[n/x]$, the condition "$y \neq x$" does not mean that y and x are bound to different values, but that they are syntactically different variables.

- Assignment:

$$\frac{}{\langle x := n,\, \sigma\rangle \to \langle \mathsf{skip},\, \sigma[n/x]\rangle} \qquad \frac{\langle a,\, \sigma\rangle \to_a a'}{\langle x := a,\, \sigma\rangle \to \langle x := a',\, \sigma\rangle}$$

- Sequence:

$$\frac{\langle c_0,\, \sigma\rangle \to \langle c_0',\, \sigma'\rangle}{\langle c_0\ ;\ c_1,\, \sigma\rangle \to \langle c_0'\ ;\ c_1,\, \sigma'\rangle} \qquad \frac{}{\langle \mathsf{skip}\ ;\ c_1,\, \sigma\rangle \to \langle c_1,\, \sigma\rangle}$$

- Conditional:

$$\frac{\langle b,\, \sigma\rangle \to_b b'}{\langle \mathsf{if}\ b\ \mathsf{then}\ c_0\ \mathsf{else}\ c_1,\, \sigma\rangle \to \langle \mathsf{if}\ b'\ \mathsf{then}\ c_0\ \mathsf{else}\ c_1,\, \sigma\rangle}$$

$$\frac{}{\langle \mathsf{if\ true\ then}\ c_0\ \mathsf{else}\ c_1,\, \sigma\rangle \to \langle c_0,\, \sigma\rangle} \qquad \frac{}{\langle \mathsf{if\ false\ then}\ c_0\ \mathsf{else}\ c_1,\, \sigma\rangle \to \langle c_1,\, \sigma\rangle}$$

- While statement:

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

There is no rule for skip; the configuration $\langle \mathsf{skip},\, \sigma\rangle$ is irreducible. In all other cases, there is exactly one rule that applies. These rules tell us all we need to know to run IMP programs.

3 Big-Step Semantics

As an alternative to small-step structural operational semantics, which specifies the operation of the program one step at a time, we now consider *big-step operational semantics*, in which we specify the entire transition from a configuration (an ⟨expression, environment⟩ pair) to a final value. This relation is denoted $\Downarrow$. For arithmetic expressions, the final value is an integer $n \in \mathbb{Z}$; for Boolean expressions, it is a Boolean truth value $\mathit{true}, \mathit{false} \in \mathbb{2}$; and for commands, it is an environment $\sigma : \mathit{Var} \to \mathbb{Z}$. Thus

$$\Downarrow_a : (\mathsf{AExp} \times \mathit{Env}) \to \mathbb{Z} \qquad \Downarrow_b : (\mathsf{BExp} \times \mathit{Env}) \to \mathbb{2} \qquad \Downarrow : (\mathsf{Com} \times \mathit{Env}) \rightharpoonup \mathit{Env}$$

Here $\mathbb{2}$ represents the two-element Boolean algebra consisting of the two truth values $\{\mathit{true}, \mathit{false}\}$ with the usual Boolean operations. Then

- $\langle c,\, \sigma\rangle \Downarrow \sigma'$ says that σ' is the environment of the final configuration, starting in configuration $\langle c,\, \sigma\rangle$;
- $\langle a,\, \sigma\rangle \Downarrow_a n$ says that $n \in \mathbb{Z}$ is the integer value of arithmetic expression a evaluated in environment σ; and
- $\langle b,\, \sigma\rangle \Downarrow_b t$ says that $t \in \mathbb{2}$ is the truth value of Boolean expression b evaluated in environment σ.

3.1 Arithmetic and Boolean Expressions

The big-step rules for arithmetic and Boolean expressions are straightforward. The key when writing big-step rules is to think about how a recursive interpreter would evaluate the expression in question. The rules for arithmetic expressions are:

- Constants:

$$\frac{}{\langle n, \sigma\rangle \Downarrow_a n}$$

- Variables:

$$\frac{}{\langle x, \sigma\rangle \Downarrow_a \sigma(x)}$$

- Operations:

$$\frac{\langle a_0, \sigma\rangle \Downarrow_a n_0 \qquad \langle a_1, \sigma\rangle \Downarrow_a n_1}{\langle a_0 + a_1, \sigma\rangle \Downarrow_a n_2} \text{ (where } n_2 \text{ is the sum of } n_0 \text{ and } n_1\text{)}$$

  and similarly for $*$ and $-$.

The rules for evaluating Boolean expressions and comparison operators are similar.

3.2 Commands

- Skip:

$$\frac{}{\langle \mathsf{skip}, \sigma\rangle \Downarrow \sigma}$$

- Assignments:

$$\frac{\langle a, \sigma\rangle \Downarrow_a n}{\langle x := a, \sigma\rangle \Downarrow \sigma[n/x]}$$

- Sequences:

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

- Conditionals:

$$\frac{\langle b, \sigma\rangle \Downarrow_b \mathit{true} \quad \langle c_0, \sigma\rangle \Downarrow \sigma'}{\langle \mathsf{if}\ b\ \mathsf{then}\ c_0\ \mathsf{else}\ c_1, \sigma\rangle \Downarrow \sigma'} \qquad \frac{\langle b, \sigma\rangle \Downarrow_b \mathit{false} \quad \langle c_1, \sigma\rangle \Downarrow \sigma'}{\langle \mathsf{if}\ b\ \mathsf{then}\ c_0\ \mathsf{else}\ c_1, \sigma\rangle \Downarrow \sigma'}$$

- While statements:

$$\frac{\langle b, \sigma\rangle \Downarrow_b \mathit{false}}{\langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma\rangle \Downarrow \sigma} \qquad \frac{\langle b, \sigma\rangle \Downarrow_b \mathit{true} \quad \langle c, \sigma\rangle \Downarrow \sigma' \quad \langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma'\rangle \Downarrow \sigma''}{\langle \mathsf{while}\ b\ \mathsf{do}\ c, \sigma\rangle \Downarrow \sigma''}$$

4 Big-Step vs. Small-Step SOS

If the big-step and small-step semantics both describe the same language, we would expect them to agree. In particular, the relations $\to^*$ and $\Downarrow$ both capture the idea of a complete evaluation. We would expect that if $\langle c, \sigma\rangle$ is a configuration that evaluates in the small-step semantics to $\langle \mathsf{skip}, \sigma'\rangle$, then σ' should also be the result of the big-step evaluation, and vice-versa. Formally,

Theorem 1. *For all commands $c \in \mathsf{Com}$ and environments $\sigma, \tau \in Env$,*

$$\langle c, \sigma\rangle \to^* \langle \mathsf{skip}, \tau\rangle \;\Leftrightarrow\; \langle c, \sigma\rangle \Downarrow \tau.$$

One can prove this theorem by induction.

Note that this statement about the agreement of big-step and small-step semantics has nothing to say about the agreement of nonterminating computations. This is because big-step semantics cannot talk directly about nontermination. If $\langle c, \sigma \rangle$ does not terminate, then there is no τ such that $\langle c, \sigma \rangle \Downarrow \tau$.

Small-step semantics can model more complex features such as nonterminating programs and concurrency. However, in many cases it involves unnecessary extra work.

If we do not care about modeling nonterminating computations, it is often easier to reason in terms of big-step semantics. Moreover, big-step semantics more closely models an actual recursive interpreter. However, because evaluation skips over intermediate steps, all programs without final configurations are indistinguishable.

References

[1] Glynn Winskel. *The Formal Semantics of Programming Languages.* MIT Press, 1993.