Scribe: Alexandru Niculescu-Mizil and Sabina Petride Lecturer: Andrew Myers

1 Review of IMP Syntax

The syntax in BNF form is:

$a ::= n \mid x \mid a_0 \ \mathbf{op} \ a_1$
$b ::= \mathbf{true} \mid \mathbf{false} \mid \neg b_0 \mid b_0 \wedge b_1 \mid b_0 \vee b_1$
$c ::= \mathbf{skip} \mid x := a \mid \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1 \mid \mathbf{while}\ b\ \mathbf{do}\ c_0 \mid c_0; c_1$

where $n \in \mathbf{Z}$, x is any variable, $\mathbf{op} \in \{+, -, *\}$, and a, a_0, a_1 are metavariables in $AExp$, b, b_0, b_1 metavariables in $BExp$ and c, c_0, c_1 metavariables in Com.

2 Rules

The evaluation relations are of three types:
$\langle a, \sigma \rangle \Downarrow n$, for Arithmetic Expressions
$\langle b, \sigma \rangle \Downarrow t$, for Boolean Expressions
$\langle c, \sigma \rangle \Downarrow \sigma'$, for Commands
where $t \in \{$ true, false $\}$, and σ, σ' are states.

2.1 Rules for Arithmetic Expressions

Axioms:

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

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

Inference rules:

$$\frac{\langle a_0, \sigma \rangle \Downarrow n_0 \quad \langle a_1, \sigma \rangle \Downarrow n_1}{\langle a_0 \ \mathbf{op} \ a_1, \sigma \rangle \Downarrow n} \ , \text{ where } \ n = n_0 \ op \ n_1.$$

Obs. **op** is part of IMP syntax, while op is arithmetic operation, not necessarily the same.

2.2 Rules for Boolean Expressions

Axioms:

$$\frac{}{\langle \mathbf{true}, \sigma \rangle \Downarrow \text{true}}$$

$$\frac{}{\langle \mathbf{false}, \sigma \rangle \Downarrow \text{false}}$$

Obs. **true, false** are part of IMP syntax, while true and false are truth values.
Inference rules:
Negation:

$$\frac{\langle b, \sigma \rangle \Downarrow \text{false}}{\langle \neg b, \sigma \rangle \Downarrow \text{true}} \qquad \frac{\langle b, \sigma \rangle \Downarrow \text{true}}{\langle \neg b, \sigma \rangle \Downarrow \text{false}}$$

Conjunction:

$$\frac{\langle b_0, \sigma \rangle \Downarrow \text{true} \quad \langle b_1, \sigma \rangle \Downarrow \text{true}}{\langle b_0 \wedge b_1, \sigma \rangle \Downarrow \text{true}}$$

$$\frac{\langle b_0, \sigma \rangle \Downarrow \text{false}}{\langle b_0 \wedge b_1, \sigma \rangle \Downarrow \text{false}} \qquad \frac{\langle b_1, \sigma \rangle \Downarrow \text{false}}{\langle b_0 \wedge b_1, \sigma \rangle \Downarrow \text{false}}$$

The above rules do not specify an order of evaluation. Since in IMP boolean expressions do not have side effects, the order of evaluation isn't important.We may impose an order of evaluation and have the same semantics. The left-first sequential rules are:

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

Disjunction:

$$\frac{\langle b_0, \sigma\rangle \Downarrow \mathsf{false} \quad \langle b_1, \sigma\rangle \Downarrow \mathsf{false}}{\langle b_0 \vee b_1, \sigma\rangle \Downarrow \mathsf{false}}$$

$$\frac{\langle b_0, \sigma\rangle \Downarrow \mathsf{true}}{\langle b_0 \vee b_1, \sigma\rangle \Downarrow \mathsf{true}} \qquad \frac{\langle b_1, \sigma\rangle \Downarrow \mathsf{true}}{\langle b_0 \wedge b_1, \sigma\rangle \Downarrow \mathsf{true}}$$

The same order imposed by conjunction can be achieved by:

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

2.3 Evaluation Rules for **IMP** Commands

Skip:

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

Semicollon:

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

Attribution:

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

where $\sigma[x \mapsto n](y)$ is $\sigma(y)$ if $y \neq x$ and n if $y = x$

while do:

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

if then else:

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

3 Proof Trees

A program in **IMP** is actually a command. The question is when does the semantics permit a particular execution result? The approach is that, given a program c and an initial state σ, $\langle c, \sigma\rangle \Downarrow \sigma'$ is legal if there is a derivation starting with $\langle c, \sigma\rangle \Downarrow \sigma'$ and built in an upward fashion:

- find a rule with the conclusion $\langle c, \sigma\rangle \Downarrow \sigma'$
- if the rule is an axiom, then the derivation is complete, otherwise recursively construct a *finite height* derivation for each of the hypotheses.

Such a derivation is also called a **proof tree**. Notice that each step in a proof tree is a **rule instance**: a rule with all metavariables substituted consistently with values from the corresponding sets.

For example the proof tree for $\langle \textbf{if } x < y \textbf{ then } x := 0 \textbf{ else skip}, \sigma_1 \rangle \Downarrow \sigma_2$ is:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\langle 1, \sigma_1 \rangle \Downarrow 1}\quad \dfrac{}{\langle 2, \sigma_1 \rangle \Downarrow 2}}{\langle x, \sigma_1 \rangle \Downarrow 1}\quad \dfrac{}{\langle y, \sigma_1 \rangle \Downarrow 2}}{\langle x < y, \sigma_1 \rangle \Downarrow \mathsf{true}}\quad \dfrac{\dfrac{}{\langle 0, \sigma_1 \rangle \Downarrow 0}}{\langle x := 0, \sigma_1 \rangle \Downarrow \sigma_2}}{\langle \textbf{if } x < y \textbf{ then } x := 0 \textbf{ else skip}, \sigma_1 \rangle \Downarrow \sigma_2}$$

where we use the notation σ_0 for the state in which all variables have value 0, and $\sigma_1 = \sigma_0[x \mapsto 1, y \mapsto 2]$, $\sigma_2 = \sigma_0[y \mapsto 2]$. Note that $\sigma_1(x) = 1$, $\sigma_1(y) = 2$, and $\sigma_2(x) = 0$.

3.1 A Rule Instance Not Part of Any Proof

Rule instances are formed by mechanically substituting for metavariables appearing in rules. This substitution can produce rule instances that will never appear in any proof. For example, the instance $\dfrac{\langle 2, \sigma \rangle \Downarrow 2 \quad \langle 2, \sigma \rangle \Downarrow 3}{\langle 2 + 2, \sigma \rangle \Downarrow 5}$ where we applied the rule for $\oplus \to +,\ a_0 \to 2,\ a_1 \to 2,\ n_0 \to 2,\ n_1 \to 3,\ n \to 5$ is a rule instance. However it will never appear in any proof because $\langle 2, \sigma \rangle \Downarrow 3$ is not an axiom instance. Note that the only proper axiom is $\langle n, \sigma \rangle \Downarrow n$ (the same integer n).

4 Problems with Large Step Semantics

4.1 Infinite Loop

Let's consider the configuration $\langle \textbf{while true do skip}, \sigma_0 \rangle$. Checking if there is a proof tree comes to writing:

$$\dfrac{\dfrac{}{\langle \textbf{true}, \sigma_0 \rangle \Downarrow \mathsf{true}}\quad \dfrac{}{\langle \textbf{skip}, \sigma_0 \rangle \Downarrow \sigma_0}\quad \dfrac{\cdots}{\langle \textbf{while true do skip}, \sigma_0 \rangle \Downarrow \sigma}}{\langle \textbf{while true do skip}, \sigma_0 \rangle \Downarrow \sigma}$$

Note that the height of the proof tree for $\langle \textbf{while true do skip}, \sigma_0 \rangle \Downarrow \sigma$ in the premise must be as large as the height of the whole tree since it is the same relation as in the conclusion. Therefore the height of the proof tree cannot be finite.

Since the tree has infinite height, it follows that there is no proof tree for this case, and the interpreter will never stop. This is a weakness of the large step semantics for **IMP**, since it says nothing about the correctness of programs that have to run forever, as servers.

4.2 Handling Errors

Suppose that we extend the syntax of arithmetic expressions by allowing division:

$a ::= \ldots \mid a_0 \div a_1$

The corresponding large step rule is: $\dfrac{\langle a_0, \sigma \rangle \Downarrow n_0 \quad \langle a_1, \sigma \rangle \Downarrow n_1}{\langle a_0 \div a_1, \sigma \rangle \Downarrow n}$ where $n = \lfloor n_0 / n_1 \rfloor$

Then consider the following example: $\langle 2 \div 0, \sigma \rangle \Downarrow n$. If we try ro construct a proof tree we will have:

$\dfrac{\langle 2, \sigma \rangle \Downarrow 2 \quad \langle 0, \sigma \rangle \Downarrow 0}{\langle 2 \div 0, \sigma \rangle \Downarrow n}$ where $n = \lfloor 2/0 \rfloor$.

Since there is no n such that $n = \lfloor 2/0 \rfloor$, it means that there is no proof tree for $\langle 2 \div 0, \sigma \rangle \Downarrow n$. This is again a weakness of the large step semantics for **IMP**, since there is no explicit difference between nontermination, as in the previous section, and error.

4.3 Parallel Composition

Suppose we extend the syntax by allowing parallel composition of commands:

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

where the execution of $c_0|c_1$ consists of simultaneously executing c_0 and c_1, allowing that each command to have side effects on the other. If we try to find the associated rules we discover it is impossible, since there is no way to capture the situation when the execution of, for example, c_1 has side effects on the execution of c_0. So the large step semantics does not cover parallel composition.

4.4 Nondeterministic Choice

Finally, we extend the syntax with the deterministic choice of commands:

$c ::= \ldots \mid c_0 \square c_1$

where the execution of $c_0 \square c_1$ stands for the nondeterministic execution of either c_0 or c_1. The obvious corresponding rules are:

$$\frac{\langle c_0, \sigma\rangle \Downarrow \sigma'}{\langle c_0 \square c_1, \sigma\rangle \Downarrow \sigma'} \qquad \frac{\langle c_1, \sigma\rangle \Downarrow \sigma'}{\langle c_0 \square c_1, \sigma\rangle \Downarrow \sigma'}$$

But what if the execution of the command first chosen by the interpreter never ends? It won't agree with the semantics, which say that there is a legal execution. The interpreter must choose correctly to avoid non-termination or errors. This resumes to supposing that it should be the case that, if at least one of the commands is correct, then this is the first chosen (the so called "angelic" choice, implementable as a fork to create two parallel threads; first one to complete wins).