CS611 Lecture 1 Introduction to IMP 8/31/2001

Scribe: Manpreet Singh (manpreet@cs)
Rohit A (rohit@cs)
Lecturer: Andrew Myers

IMP is an "imperative" language because program execution involves carrying out a series of explicit commands to change state. Formally **IMP**'s behaviour is captured by the syntactic sets and the operational semantics.

Syntactic Sets associated with IMP

- numbers **Z** consisting of integers. n ranges over **Z**.
- truth values **T** = { **true, false** }. t ranges over **T**.
- locations/Variables **Loc/Var**. x ranges over **Loc**. Each location stores an integer.
- The set of all legal arithmetic expressions **AExp**. a ranges over **AExp**. a consists of integers and variables and the closure of the above set under addition, subtraction and multiplication.
- The set of all legal boolean expressions **BExp**. b ranges over **BExp**. b consists of **true**, **false**, and comparisons and the closure of the above set under negation, conjunction and disjunction.
- The set of all legal commands **Com**. c ranges over **Com**. A command can be a "do nothing"(skip), an assignment, a sequence of commands , if-then-else or while-do-loop.

We assume that the syntactic structure of numbers and locations is given. For example locations are named in the same way as in **C** and numbers are represented as decimal numerals. The rest are described using BNF [1].

- **AExp**
  $a ::= n \mid x \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 * a_1$
- **BExp**
  $b ::= \mathbf{true} \mid \mathbf{false} \mid a_0 = a_1 \mid a_0 < a_1 \mid \neg b \mid b_0 \vee b_1 \mid b_0 \wedge b_1$
- **Com**
  $c ::= \mathbf{skip} \mid x := a \mid c_0; c_1 \mid \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1 \mid \mathbf{while}\ b\ \mathbf{do}\ c$

Example :
while $x < 0$ do if $x < y$ then $(t := x\,;\, x := y\,;\, y := t)$ else skip; $x := x - y$

Note that there is more than one way of parsing the above string. A parse tree for this program is shown in *figure 1.* We assume that all elements of the syntactic sets are parse (or syntax) trees. We write parantheses only when it is needed to make it clear which abstract syntax tree do we mean.

Structural Operational Semantics (SOS)
Operational semantics describes the steps taken to execute a program. *Structural operational semantics* associates legal executions with proof. Thus it provides a compact and easy way for proving language properties.

Large step semantics
Recall that an **IMP** program is a command c. c is completely defined by the contents of the locations (starting with predefined values in locations) after the execution of c. Formally, it involves the definition of *states*, *evaluation of arithmetic/boolean expression* and *evaluation of commands.*

[1]Backus-Naur Form is a compact way of writing a context-free grammar. Different productions with the same left-hand side are combined, the right-hand sides separated by vertical lines

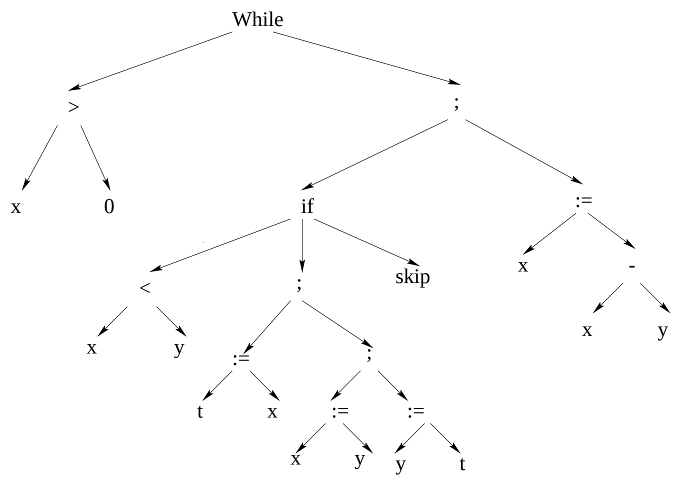

Figure 1: A syntax tree for the example **IMP** program

- *States* The set of states Σ consists of functions $\sigma : \mathbf{Loc} \rightarrow \mathbf{Z}$. Thus $\sigma(x)$ is the value/contents of location x in state σ.

- *Evaluation of commands* consists of relations $\Downarrow_c \subseteq \mathbf{Com} \times \Sigma \times \Sigma$. A triplet $\langle c, \sigma, \sigma' \rangle \in \Downarrow_c \iff c$ *when executed at initial state σ produces state σ' at completion.* The triplet is also written as

  $$\langle c, \sigma \rangle \ \Downarrow_c \ \sigma'$$

  Note that we don't allow commands which don't terminate. Also $\Downarrow_c$ is defined as a relation rather than a function from $\mathbf{Com} \times \Sigma$ to Σ because we don't want to rule out *non-deterministic commands.* (For example, $c_0 \Box c_1$ can result in more than one state)

- *Evaluation of arithmetic expressions* consists of relations $\Downarrow_a \subseteq \mathbf{AExp} \times \Sigma \times \mathbf{Z}$. A triplet $\langle a, \sigma, n \rangle \in \Downarrow_a \iff$ *evaluating a in state σ gives n.* The triplet is equivalent to $\langle a, \sigma \rangle \ \Downarrow_a \ n$

  By setting up the evaluation relation in this way, we are implicitly saying that there are no side effects.

- *Evaluation of boolean expressions* consists of relations $\Downarrow_b \subseteq \mathbf{BExp} \times \Sigma \times \mathbf{T}$. A triplet $\langle b, \sigma, t \rangle \in \Downarrow_b \iff$ *evaluating b in state σ gives t.* The triplet is equivalent to $\langle b, \sigma \rangle \ \Downarrow_b \ t$

  Due to the same reason as in AExp there are no side effects.

We will drop the subscripts c, a and b in $\Downarrow_c$, $\Downarrow_a$, and $\Downarrow_b$ respectively henceforth.

Inference Rules
An inference rule captures the notion that a set of statements imply another statement. We can start to define some inference rules. The simplest of these are the **axioms** which have no *premises* or *antecedents.* The following axioms hold for any arbitrary σ.

$\langle true, \sigma \rangle \Downarrow true$	i.e. $\forall \sigma \langle true, \sigma, true \rangle \ \in \ \Downarrow_b$
$\langle false, \sigma \rangle \Downarrow false$	i.e. $\forall \sigma \ \langle false, \sigma, false \rangle \ \in \ \Downarrow_b$
$\langle skip, \sigma \rangle \Downarrow \sigma$	i.e. $\forall \sigma \ \langle skip, \sigma, \sigma \rangle \ \in \ \Downarrow_c$
$\langle n, \sigma \rangle \Downarrow n$	i.e. $\forall \sigma, n \ \langle n, \sigma, n \rangle \ \in \ \Downarrow_a$
$\langle x, \sigma \rangle \Downarrow \sigma(x)$	i.e. the current value of x in the store.

However for more complex constructs we need inference rules with *premises* and *side-conditions.* For example the statement $(\langle a_0 + a_1, \sigma \rangle \Downarrow n)$ follows from the statements $(\langle a_0, \sigma \rangle \Downarrow n_0)$, $(\langle a_1, \sigma \rangle \Downarrow n_1)$ and $n = n_0 + n_1$. The statement $(\langle a_0 + a_1, \sigma \rangle \Downarrow n)$ is the *conclusion* and the statements $(\langle a_0, \sigma \rangle \Downarrow n_0)$ and $(\langle a_1, \sigma \rangle \Downarrow n_1)$ are the *premises* and $n = n_0 + n_1$ is the *side condition.* The inference rules are typically written with premises and conclusions separated by a horizontal line as illustrated below:

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