Scribe: Vicky Weissman and Daniel Marques Lecturer: Andrew Myers

General course information was given at the beginning of lecture. For details, please refer to the web site at **courses.cs.cornell.edu/cs611** (or, from outside of the department, **www.cs.cornell.edu/courses/cs611**.)

IMP

- discussed in Chap. 2 of Winskel
- a simple imperative language (commands cause side-effects)
- all variables are pre-defined (cannot introduce new variables)
- variables can only take integer values

Syntax

The following abstract syntax defines a legal program in IMP.
IMP has 3 syntactic sets:

1) **AExp**

- The set of legal arithmetic expressions, a
- In English, a is constructed by initially including all variables and integers.
  The set is then closed under addition, subtraction, and multiplication.
- In Backus-Naur Form (BNF),
  $\mathrm{a} ::= \mathrm{n} \mid \mathrm{X} \mid \mathrm{a}_0 + \mathrm{a}_1 \mid \mathrm{a}_0 - \mathrm{a}_1 \mid \mathrm{a}_0 * \mathrm{a}_1$
  where $\mathrm{n} \in$ integers, $\mathrm{X} \in$ LOC (location/variable), and $\mathrm{a}_0, \mathrm{a}_1 \in$ AExp

2) **BExp**

- The set of legal boolean expressions, b
- In English, b contains comparisons and is closed under conjunction, disjunction and negation.
- In BNF,
  $\mathrm{b} ::= \mathrm{a}_0 = \mathrm{a}_1 \mid \mathrm{a}_0 \leq \mathrm{a}_1 \mid \mathrm{b}_0 \vee \mathrm{b}_1 \mid \mathrm{b}_0 \wedge \mathrm{b}_1 \mid \neg b_0$
  where $\mathrm{a}_0, \mathrm{a}_1 \in$ AExp, $\mathrm{b}_0, \mathrm{b}_1 \in$ BExp
- true and false are encoded as integers

3) **Com**

- The set of legal commands, c
- In English, a command can be 'do nothing' (skip), an assignment, an if/else clause, a while loop or a sequence of commands.
- In BNF,
  $\mathrm{c} ::= \mathrm{skip} \mid \mathrm{X} := \mathrm{a} \mid \mathrm{if\ b\ then\ c}_0 \mathrm{\ else\ c}_1 \mid \mathrm{while\ b\ do\ c} \mid \mathrm{c}_0; \mathrm{c}_1$
  where $\mathrm{X} \in$ LOC (location/variable), $\mathrm{a} \in$ AExp, $\mathrm{b} \in$ BExp, and $\mathrm{c}, \mathrm{c}_0, \mathrm{c}_1 \in$ Com
- A legal program consists of a single command (usually $\mathrm{c}_0; \mathrm{c}_1$)

Example:
while $x \neq y$ **do if** $x \leq y$ **then** $y := y - x$ **else** $x = x - y$

We use parentheses to clarify how to parse an expression. However, parentheses are not part of the syntax of the language — we assume that all elements of the syntactic set are expression (*parse*) trees.

Operational Semantics

- operational semantics define a program's execution
- structural operational semantics associate legal executions with proofs
- structural operational semantics is a compact and convenient method for proving language properties.

A configuration is the information needed to determine how a program will behave. It consists of the command to be executed and the current state of the system. The notation used to represent configurations is $\langle c, \sigma \rangle$, where c is the command about to be executed and σ (a mapping from all variable names to their integer values) is the current state of the system. We will write $\langle c, \sigma \rangle \Downarrow \sigma'$ to mean "the command c can execute in state σ to produce state σ'".

We will have corresponding statements for arithmetic and boolean expressions:
$\langle a, \sigma \rangle \Downarrow n$, for some integer n
$\langle b, \sigma \rangle \Downarrow t$, for some truth value t

We can start to define rules that capture when these statements are true. For "skip"' and "X" we have:
$\langle \mathbf{skip}, \sigma \rangle \Downarrow \sigma$
$\langle X, \sigma \rangle \Downarrow \sigma(X)$, i.e. the current value of X in the store.

For more complex constructs we need inference rules. An inference rule captures the notion that a set of statements imply another statement. For example, the statement $\langle c_0; c_1, \sigma \rangle \Downarrow \sigma'$ follows from the statements $\langle c_0, \sigma \rangle \Downarrow \sigma''$ and $\langle c_1, \sigma'' \rangle \Downarrow \sigma'$. The implied statement ($\langle c_1, \sigma'' \rangle \Downarrow \sigma'$) is called the conclusion and the other statements are called premises. Unspecified pieces of abstract syntax in the premises, such as σ'', are called meta-variables. Inference rules are typically written with premises and conclusions separated by a horizontal line as illustrated in the following examples.

Sample Inference Rules

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

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

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

Recall that we are representing state as a function from variable names to integers. $\sigma[X \mapsto n]$ is a new function where X is mapped to n, and all other variables are mapped to the values they had in σ.

The conclusion of an inference rule that does not have any premises is called an axiom. The rule for "skip" is an axiom:

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

IMP Properties

- Turing complete (barely)
- a program will never 'crash'
- evaluation is deterministic
- all expressions terminate, but commands may not
- no functions or data structures