Advanced Progamming Languages	**Lecture 29**
CS 6110 Spring 2015	**Wed. April 8, 2015**

Lecture 29

Topics

1. PS4 is out, I will add one more problem on IMP design – adding functions with state. We might also consider environments that give names to constants. There is an extra credit problem to explain Blum's Size Theorem. There will be a PS5 as well.
2. We will study the simple imperative programming language IMP from Winskel's book *The Formal Semantics of Programming Languages*, MIT Press, 1993. Chapter 2. PS4 includes material from Chapter 10.
3. Functional vs. Imperative languages, "imperative" is for commands, "do this to the state". We need *assignable* identifiers also called reference variables in the ML family.
4. Winskel gives a syntax, the easy bit, and a *large step* operational semantics (à la Khan) and a *small step* operational semantics (à la Plotkin).
5. We will add *functional procedures* to IMP (to get "FIMP"?). That will be part of PS4. This will be a collective "design" problem.

Defining IMP (with some CS6110 flourishes)

Abstract syntax

Arithmetic expressions (p.12)

$aexp ::= n \mid x \mid add(a_1; a_2) \mid sub(a_1; a_2) \mid mult(a_1; a_2)$

Syntactic conventions

n, m	range over $\mathbb{N}$	
x, y	range over Loc	(locations, variables)
a	range over $Aexp$	(arithmetic expressions)
b	range over $Bexp$	(Boolean expressions)
c	range over Cmd	(Commands – Com)

Some "Winskel ways" requiring parenthesis to make the expressions unique – a bit strange to me. Here are his syntactic classes: *Aexp*, *Bexp*, *Cmd* (he uses Com for Commands).

I show Algol68 syntax:

if $bexp$ **then** c_0 **else** c_1 **fi, while** $bexp$ **do** c **od**, and **until** $bexp$ **do** c **od**, etc.

Structured operational semantics (big step, à la Khan, also called *natural semantics*).

Evaluation requires states. Winskel writes σ, we write s. $s : Loc \to \mathbb{N}$ (recall Loc are *locations* in the memory or state).

The rules have the format

$$< exp, s > \to value$$

$$< cmd, s > \to state$$

On page 16 he suggests a *refinement* or top down style of reading the rules. Wirth suggested the same and called them *refinement style* rules. In lecture we will write them this way.

Evaluation of sums p.14

Rule instances p.15 Note: $s_0(Init) = 0$

Derivation tree p.15

$\vdash\ < (Init) + 5 + (7 + 9), s_0 > \to \underline{5} + \underline{16}$

$\qquad\qquad \vdash\ < (Init + 5), s_0 > \longrightarrow \_\_ + 5$

$\qquad \vdash\ < Init, s_0 > \to s_0(Init) = 0$

$\qquad\qquad \vdash\ < 5, s_0 > \to 5$

$\qquad\qquad \vdash\ < (7 + 9), s_0 > \to \_\_ + \_\_ = 16$

$\qquad\qquad\quad \vdash\ < 7, s_0 > \to 7$

$\qquad\qquad\quad \vdash\ < 9, s_0 > \to 9$

Evaluation of Commands 2.4, p.19

Using $< C, s > \to s'$

For example: $< X := 5, s > \to s[5/X]$

List of Commands

skip	$< skip, s > \to s$
sequencing	$< C_0; C_1, s > \to$
conditionals	$<$ if b then C_0 else $C_1, s > \to$
while loops	(page 20)

$\vdash <$ **while** b **do** C **od** $, s > \to s'$

$\quad \vdash < b, s > \to true$

$\quad \vdash < C, s > \to s''$

$\quad \vdash <$ **while** b **do** $C, s'' > \to s'$

$\vdash <$ **while** b **do** C **od** $, s > \to s$

$\quad \vdash < b, s > \to false$