Advanced Progamming Languages	**Lecture 30**
CS 6110 Spring 2015	**Fri. April 10, 2015**

Lecture 30

Topics

1. PS4 corrections
   - The $3x+1$ (Collatz) function is

     $f(x) = \textbf{if } x = 1 \textbf{ then } 0 \textbf{ else if } even(x) \textbf{ then } f(x/2) \textbf{ else } f(3x+1).$
   - Explain why the sentence on p.172 of Winskel hints at LCF failure.
   - Don't cite Coq Auto as an explanation. Abhishek and I have noted some interesting sociology around using Coq proofs – something to ponder. We are seeing advanced AI at work in unexplained ways. Can proof assistants get a "personality"?
2. Read Winskel, first part of Chapter 5, p.55-68, and Chapter 6.

The structured operational semantics is very precise and elementary. It simply presents a format for computation rules. The rules are readable and suitable for implementation. This approach is the foundations of functional languages as well. Indeed, once we have been precise in this way for functional languages, we can use them to give what is called a denotational semantics for imperative (or aka "procedural") languages. Here is how.

Denotational Semantics for IMP

Winskel Style – Chapter 5	Lecture style – partial functions
$A : Aexp \to (\Sigma \to \mathbb{N})$	$[\![aexp]\!] : states \to \overline{\mathbb{N}}$
$B : Bexp \to (\Sigma \to \mathbb{N})$	$[\![exp]\!] : states \to \mathbb{B}$
$\mathcal{C} : Cmd \to (\Sigma \to \Sigma)$	$[\![cmd]\!] : states \to \overline{states}$
See Winskel pages 56-58	

We only look at commands in lecture.

$[\![skip]\!](s) = s$

$[\![X := aexp]\!](s) = s[\,[\![aexp]\!](s)/X]$

$[\![\textbf{if } bexp \textbf{ then } \mathcal{C}_0 \textbf{ else } \mathcal{C}_1 \textbf{ fi}]\!](s) = \textbf{ if } [\![bexp]\!](s) \textbf{ then } [\![\mathcal{C}_0]\!](s) \textbf{ else } [\![\mathcal{C}_1]\!](s)$

$[\![\textbf{while } bexp \textbf{ do } \mathcal{C} \textbf{ od}]\!](s) = \textbf{ if } [\![bexp]\!](s) \textbf{ then } [\![\textbf{while } bexp \textbf{ do } \mathcal{C} \textbf{ od}]\!]([\![\mathcal{C}]\!](s)) \textbf{ else } s$