Advanced Progamming Languages	**Lecture 8**
CS 6110 Spring 2015	**Fri. Feb. 6, 2015**

Lecture 8

Topics

1. Comments on proof style for homework.
2. There is a "one point basis" several different ones.

   $\mathbf{X} \equiv \overline{\lambda(x.x(x\mathbf{S}(\mathbf{K}^3\mathbf{I})\mathbf{K}))}$

   Also note, $\mathbf{Y} \equiv \mathbf{WS}(\mathbf{BWB})$, add to the dictionary.
3. How do we define the *partial recursive functions*, and what do we usually mean by them?
4. Development of the *primitive recursive functions*. The primitive recursion formalism is a very simple *subrecursive programming language* just as Coq is.

Comments on PS1:
The condition is "x is **not free** in b" in problem 2.

Partial Recursive Functions

Defining the partial recursive functions, $\mathbb{PR}$, is to first define the *primitive recursive functions* and then add the "least number operator". Kleene proceeds this way. The pattern of primitive recursion is typically used to define addition, multiplication, and exponentiation recursively.

$$
\begin{array}{lll}
 & & \underline{\text{Note}} \\
a_0(x,y) = x+1 \qquad S(x) = x+1 & & a_0(x,y) = S(x) \\
a_1(x,y) = add(x,y) & & \\
\quad add(0,y) = y & & \underline{\text{Note}} \\
\quad add(S(x),y) = S(add(x,y)) & & S(add(x,y)) = a_0(a_1(x,y),y) \\
a_2(x,y) = mult(x,y) & & \\
\quad mult(0,y) = 0 & & \underline{\text{Note}} \\
\quad mult(S(x),y) = add(mult(x,y),y) & & add(mult(x,y),y) = a_1(a_2(x,y),y)
\end{array}
$$

Etc. See notes on Examples of Primitive Recursion.

Why are these special?

Interesting facts about combinators (from Mark Bickford)

- The programming language Miranda (mentioned in our textbook because Simon Thompson was part of the Miranda project) was compiled to combinators.
- Burroughs computer corporation (for which Dykstra consulted) was building a combinator based computer. Its instruction set was like this:

$\mathbf{S}fgx$	x goes to both	$(fx)(fg)$
$\mathbf{C}fgx$	x goes to one (g)	$f(gx)$
$\mathbf{B}fgx$	x goes to the other (f)	$(fx)g$
$\mathbf{P}fgx$	x goes to neither	$\mathbf{P}g$

- Mayer Goldberg uses large bases, up to 12, and thinks of the combinators as points in high dimensional space with *computation as paths* in those "spaces".

Kleene distinguishes two instances of the basic primitive recursive scheme 5a and 5b. The first four cases of his definition are:

(1) $S(x) = x + 1$ successor

(2) $Const(x_1, ..., x_n) = c$ constant

(3) $Proj_i(x_1, ..., x_n) = x_i$ projection

(4) $f(x_1, .., x_n) = h(g_1(x_1, ..., x_n), ..., g_m(x_1, ..., x_n))$ composition

(5_a) $f(0) = c$
$f(x + 1) = ind(x, f(x))$

(5_b) $f(0, y_1, ..., y_n) = g(y_1, ..., y_n)$ primitive recursion
$f(x + 1, y_1, ..., y_n) = h(x, f(x, y_1, ..., y_n), y_1, ..., y_n)$

A function is *primitive recursive* iff it can be defined by a sequence of applications of these five operations. What is a good name for (5_a)?

Note, there is a natural way of extending the sequence

$a_0(x, y)$,	$a_1(x, y)$,	$a_2(x, y)$
Successor	addition	multiplication

What is next? How to define it?

How can we define these functions with just **S** and **K**?!

Defining primitive recursion with combinators.
We clearly need a way to accomplish the following tasks.

1. Define the natural numbers 0, $S(0)$, $S(S(0)), ...$

$$0, \quad 1, \quad 2....$$

2. Define a conditional, $\underline{if}\ \ b(x)\ \ \underline{then}\ \ t(x)\ \ \underline{else}\ \ f(x)$.
3. Define recursion.

One of these, the central operator, we have already defined. What is that?

Let $\bar{n}$ be the combinator for the number n, define it as follows

$$\bar{0} \equiv \mathbf{KI}$$
$$\overline{S(n)} \equiv \mathbf{SB}\bar{n}$$

Then we get the sequence of "numbers".

0,	1,	2,	3,	4,...
KI,	**I**,	**SBI**,	**SB(SBI)**,...	

What is the conditional? It is called **D** for decide.

$$\mathbf{D} \equiv [x, y, z].z(\mathbf{K}y)x$$

This has the key property

$$\mathbf{D}XY\bar{0} = X$$
$$\mathbf{D}XY\overline{S(n)} = Y \qquad \text{Can also write } \mathbf{D}XY\overline{n+1} = Y$$

Now we have all we need. See if you can define addition using the idea from primitive recursion.

For the philosophically inclined, how would *you* define the number 0, the number 1, etc. What is the "real meaning"? Is there such a thing? What is it in set theory? What is the computational meaning of 0? Of recursion?