CS6110 Lecture 12 February 17, 2012
CS 6116 Lecture 4

Primitive Recursive Definitions in Continuation Passing Style

$$f(0,y) = g(y)$$
$$f(S(x),y) = h(x,y,f(x,y))$$

normal style (Goodstein, Kleene, etc.)

$$f(x,y) = f'(x,y)\,k_1 \quad \text{for } k_1 = \lambda(x.x) \text{ where}$$
$$f'(0,y)\,k = k\,g(y) \quad \text{for any continuation } k$$
$$f'(S(x),y)\,k = ap(f'(x,y);\ \lambda(y'.\ k(h(x,y,y'))))\ \text{for any } k$$

This transformation, called CPS Conversion, rewrites the recursion into tail recursive or iterative form. This format saves "stack space" in the normal implementation used by compilers.

In Lecture 11 of CS6110 and in the CS6110 Lecture 14 notes from 2010 the factorial function is written in this style

$$fact(n) = \text{if } n=0 \text{ then } 1 \text{ else } n * fact(n-1)$$
$$fact(n) = fact'(n)\,(\lambda(x.x)), \text{ i.e. } k = \lambda(x.x)$$
$$fact'(n)\,k = \underline{\text{if}}\ n=0\ \underline{\text{then}}\ k1\ \underline{\text{else}}\ fact'(n-1)\,(\lambda v.\ k(n*v))$$

CS 6110 Continuations - more examples Lecture 12
CS 6116

$add(0,y) = y \qquad add(S(x),y) = S(add(x,y))$

$add(S(S(S(0))), S(0)) \downarrow$

$S(add(S(S(0)), S(0))) \downarrow$

$S(S(add(S(0), S(0)))) \downarrow$

$S(S(S(add(0, S(0))))) \downarrow$

$S(S(S(S(0)))) \downarrow$

$add(3,1) = 4$

not tail recursive, we stack up the S() operations, but this is the final answer, so it looks tail recursive. If we had to produce

$S(0) \downarrow 1$
$S(1) \downarrow 2$
$S(2) \downarrow 3$
$S(3) \downarrow 4$

it would look like a stack.

$$\begin{cases} mult(0,y) = 0 \\ mult(S(x),y) = add(mult(x,y), y) \end{cases}$$

$mult(0) = \lambda(y.0)$
$mult(S(x)) = \lambda(y.\ add(\ :mult(x)(y))(y))$

$mult(S(S(S(0))), S(S(0)))$ that is $mult(3,2)$

$add(mult(S(S(0))), S(S(0)), S(S(0))) \downarrow$

$add(add(mult(S(0), S(S(0))), S(S(0))), S(S(0)))$

$add(add(add(mult(0, S(S(0))), S(S(0))), S(S(0))), S(S(0)))$

$add(add(add(0, S(S(0))), S(S(0))), S(S(0)))$

How do continuations avoid stacking?
Consider this instance where

$$\begin{cases} mult(x,y) = mult'(x,y)\ I \quad \text{for } I = \lambda(x.x) \\ mult'(0,y)\ K = K0 \quad \text{for any } K \\ mult(S(x),y)\ K = mult'(x,y)\ \lambda(v.\ K(add(v,y))) \end{cases}$$

$mult(0,y) = mult'(0,y)\ I$, call I K_1, the 1st continuation

$mult'(0,y)\ I = I0 = 0$ the right result.

$mult(S(0),y) = mult'(S(0),y)\ I$ call I K_1

$mult'(S(0),y)\ I = mult'(0,y)\ \lambda(y'.\ I(add(y',y)))$ call this K_2

finally

$mult'(0,y)\ K_2 = K_2(0) = I(add(0,y)) = I(y) = y$

$mult(2,y) = mult'(2,y)\ I$ call I K_1

$mult'(2,y)I = mult(1,y)\ \underline{\lambda(v.\ I(add(v,y)))}$ — call this K_2

$mult(1,y)\ K_2 = mult(0,y)\ \underline{\lambda(v.\ K_2(add(v,y)))}$ — call this K_3

$$\begin{aligned} mult(0,y)\ K_3 &= K_3\ 0 \\ &= \lambda(v.\ K_2(add(v,y)))\ 0 \\ &= K_2(add(0,y)) \\ &= K_2(y) \\ &= \lambda(v.\ I(add(v,y)))\ (y) \\ &= I(add(y,y)) \\ &= I(2*y) \\ &= 2*y \end{aligned}$$

We see the general pattern for $mult(3,y)$

$K_4 = \lambda(v.\ \lambda(v.\ (\lambda(v.\ (\lambda(x.x)\ add(v,y)))\ add(v,y))\ add(v,y))\ add(v,y))$