

Comp 311  
Principles of Programming Languages  
Lecture 12  
The Semantics of Recursion III & Loose  
Ends

Corky Cartwright  
September 21, 2009

# Call-by-value Fixed-Point Operators

Given a recursive definition in a call-by-value language

$$\mathbf{f} = \mathbf{E}_f$$

where  $\mathbf{E}_f$  is an expression constructed from constants in the based data domain  $D$ , operations (continuous functions) on  $D$ , and  $\mathbf{f}$ , what does it mean?

- Example: let  $D$  be the domain of Scheme values. Then

**fact =**

**(lambda (n) (if (zero? n) 1 (\* n (fact (- n 1)))))**

is such a definition.

In a call-by-name language, the meaning is

**Y (lambda (f) E<sub>f</sub>)**

We need to define **Y<sub>v</sub>**

# Solutions to Recursion Equations

Key trick: use  $\eta$ -conversion to delay evaluation.

In the mathematical literature on the  $\lambda$ -calculus,  $\eta$ -conversion is often assumed as an axiom. In models of the  $\lambda$ -calculus, it is typically required to hold.

Definition:  $\eta$ -conversion is the following equation:

$$\mathbf{M} = \lambda \mathbf{x} . \mathbf{M}\mathbf{x}$$

where  $\mathbf{x}$  is not free in  $\mathbf{M}$

# What Is the Code for $Y_v$ ?

$\lambda F. (\lambda x. \lambda y. F(x x) y) (\lambda x. \lambda y. F(x x) y)$

- Does this work for Scheme (or Java with an appropriate encoding of functions as anonymous inner classes)? Yes!

- Let  $G$  be some functional  $G = \lambda f. \lambda n. M_f$  like **FACT** for a recursive *function* definition.  $G$  is a value. Then

$Y_v G \rightarrow (\lambda x. \lambda y. G(x x) y) (\lambda x. \lambda y. G(x x) y) \rightarrow$   
 $\lambda y. G ((\lambda x. \lambda z. G(x x) z) (\lambda x. \lambda z. G(x x) z)) y$   
is a value.

- Hence,  $G(Y_v G) \rightarrow (\lambda n. M_f) [f := Y_v G]$  is a value.

- Moreover,

$Y_v G = \lambda y. G ((\lambda x. \lambda z. G(x x) z) (\lambda x. \lambda z. G(x x) z)) y =$   
 $\lambda y. G(Y_v G) y$

which is the  $\eta$ -conversion of  $G(Y_v G)$

# Loose Ends

- Meta-errors
- Read the notes!
- rec-let (in notes)