Call-by-name vs. Call-by-value Fixed-Points

Given a recursive definition $f \equiv E_f$ in a call-by-value language where $E_f$ is an expression constructed from constants in the base language and $f$. What does it mean?

Example: let $D$ be the domain of Scheme values. Then the base operations are continuous functions on $D$ and

$$\text{fact} \equiv \text{map} \ n \ \text{to} \ \text{if} \ n = 0 \ \text{then} \ 1 \ \text{else} \ n \ast \text{fact}(n - 1)$$

is a recursive definition of a function on $D$.

In a call-by-name language $\text{map} \ n \ \text{to} \ ...$ is interpreted using call-by-name, the meaning of $\text{fact}$ is

$$Y(\text{map} \ f \ \text{to} \ E_f)$$

What if $\text{map}$ ($\lambda$-abstraction) has call-by-value semantics? $Y$ does not quite work because evaluations of form $Y(\text{map} \ f \ \text{to} \ E_f)$ diverge.
Defining $Y$ in a Call-by-value Language

We want to define $Y_v$, a call-by-value variant of $Y$. Key trick: use $\eta$(eta)-conversion to delay the evaluation. In the mathematical literature on the $\lambda$-calculus, $\eta$-conversion is often assumed as an axiom. In models of the pure $\lambda$-calculus, it typically holds.

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

\[ M = \lambda x . \ M x \]

where $x$ is not free in $M$. If the $\lambda$-abstraction used in the definition of $Y$ has call-by-value semantics, then given the functional $F$ corresponding to recursive function definition, the computation $YF$ diverges. We can prevent this from happening by $\eta$-converting both occurrences of $F(x\ x)$ within $Y$. 
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 $\lambda f. \lambda n. M$, like FACT, for a recursive function definition. $G$ and $M$ are values ($\lambda$-abstractions). Then

\[ Y_v G = \lambda x. (\lambda y. (G(x \ x) \ y)) (\lambda y. (G(x \ x) \ y)) = \lambda y. (G((\lambda y. (G(x \ x) \ y)) (\lambda y. (G(x \ x) \ y))) \ y) \]

is a value.
- Hence, $G(Y_v G) = (\lambda f. \lambda n. M) (Y_v G) = \lambda n. M[f:=Y_v G]$, which is a value.
- It is straightforward to prove (using conversion rules) that

\[ Y_v G = G(Y_v G) \]

- Disadvantage of $Y_v$ vs. $Y$: $Y_v$ is arity-specific.
Loose Ends

- Meta-errors
- Read the notes!
- \texttt{letrec} (in notes)