Recursive Definitions

• Given a Scott-domain $D$, we can write equations of the form:
  \[ f = E_f \]
  where $E_f$ is an expression constructed from constants in $D$, operations (continuous functions) on $D$, and $f$.

• Example: let $D$ be the domain of Scheme unary functions on numbers. Then
  \[
  \text{fact} = \ 
  (\lambda (n) (\text{if} \ (\text{zero?} \ n) \ 1 \ (* \ n \ (\text{fact} \ (- \ n \ 1))))))
  \]
is such an equation.

• Such equations are called *recursive definitions*. 
Solutions to Recursion Equations

Given an equation:

\[ f = E_f \]

what is a solution? All of the constants and operations in \( E_f \) are given except \( f \).

A solution is any function \( f^* \) such that

\[ f^* = E_{f^*} \]

is a solution. But there may be more than one solution. We want to select the “best” solution. Note that \( f^* \) is an element of whatever domain \( D^* \) is the type of \( E_f \). In the most common case, it is \( D \rightarrow D \), for a domain of values \( D \), but it can be \( D, D^k \rightarrow D, \ldots \). The best solution (the one that always exists, is unique, and is \textit{computable}) is the \textit{least} solution under the approximation ordering in \( D^* \). This \textit{least} solution is roughly the “least defined” solution (the one that diverges most often).
Proving \( f^* \) is a fixed point of \( F \)

Must show: \( F(f^*) = f^* \) where \( F = \lambda f . E_f \).

Claim: By definition \( f^* = \bigcup F^k(\text{bot}_{D^*}) \). Since \( F \) is continuous

\[
F(f^*) = F(\bigcup F^k(\text{bot}_{D^*})) \\
= \bigcup F^{k+1}(\text{bot}_{D^*}) \quad \text{(by continuity)} \\
= \bigcup F^k(\text{bot}_{D^*}) \quad \text{(since } \text{bot}_{D^*} <= F(\text{bot}_{D^*})\text{)} \\
= f^*
\]

Q.E.D.
Examples

Look at factorial in detail using DrRacket or DrScala.
How Can We Compute \( f^* \) Given \( F \)?

Need to construct \( F^\infty(\bot) \) from \( F \) using only \( \lambda \)-abstraction and application. We need to define an operator \( Y \) such that:

\[
Y(F) = f^* = F^*(\bot).
\]

Idea: use syntactic trick in \( \Omega \) to build a potentially infinite stack of \( F \)s.

- Preliminary attempt:

\[
(\lambda x. F(x x)) (\lambda x. F(x x))
\]

- Reduces to (in one step):

\[
F ((\lambda x. F(x x)) (\lambda x. F(x x)))
\]

- Reduces to (in \( k \) steps):

\[
F^k ((\lambda x. F(x x)) (\lambda x. F(x x)))
\]
**What Is the Code for $\Upsilon$?**

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

- Does this work for Scala (or Java with an appropriate encoding of functions as anonymous inner classes)? No!

- Why not? What about divergence? Assume $G$ is a $\lambda$-expression defining a functional like $\text{FACT}$

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

$$= G((\lambda x. G(x\ x))(\lambda x. G(x\ x)))$$

$$= ... \ (\text{divergence forced by CBV})$$
What If We Use Call-by-name?

By assumption \( G \) must have the form \((\lambda f. (\lambda n. M))\)

\[
(\lambda F. (\lambda x. F(x\ x)) (\lambda x. F(x\ x))) \ G
\]

\[
\Rightarrow (\lambda x. G(x\ x)) (\lambda x. G(x\ x)) \quad \text{<**>}
\]

\[
\Rightarrow G \quad \text{<**>}
\]

\[
= (\lambda f. (\lambda n. M)) \quad \text{<**>}
\]

\[
\Rightarrow (\lambda n. M[f:=\text{<**>}] ) \quad \text{<*>}
\]

which is a value. If this value \( \text{<*>} \) is applied to a value \( k \) and \( M[f:=\text{<**>}][n:=k] \) does not require evaluating an occurrence of \( \text{<**>} \), then the computation returns a base answer determined by \( M \). Otherwise, \( \text{<**>} \) is unwound once, as in the computation above to produce \( \text{<*>} \) applied to its argument. If this argument is “simpler” than \( k \) (the previous argument) this process eventually terminates when the argument is a value that does not force the evaluation of \( \text{<**>} \). At this point, the subcomputation \( \text{<*>} \) returns a base value and the enclosing computation (not involving recursive calls \( \text{<**>} \)) is performed, returning a value. The notion of “simpler” corresponds to a well-formed

Exercise: how can we workaround the divergence problem to create a version of the \( Y \) operator that works for call-by-value Scheme and Jam? Hint: if \( N \) is a divergent term denoting a unary function, then \( \lambda x. N x \) is an “equivalent” term that is not divergent (assuming \( x \) does occur in \( N \)). Note that if \( N \) is a divergent term denoting a \( n \)-ary function where \( n>1 \), then \( \lambda x_1 \ldots x_n. N x_1 \ldots x_n \) is equivalent to \( N \).