Comp 411
Principles of Programming Languages
Lecture 10
The Semantics of Recursion II

Corky Cartwright

September 24, 2012

Recursive Definitions

- Given a Scott-domain D, we can write equations of the form:

  $$\mathbf{f} = \mathbf{E_f}$$

  where $\mathbf{E_f}$ 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

  ```
  fact =
   (lambda (n) (if (zero? n) 1 (* n (fact (- n 1)))))
  ```

  is such an equation.
- Such equations are called *recursive definitions*.

Solutions to Recursion Equations

Given an equation:

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

what is a solution? All of the constants and operations in $\mathbf{E}_{\mathbf{f}}$ are given except $\mathbf{f}$.

A solution is any function $\mathbf{f^*}$ such that

$$\mathbf{f^*} = \mathbf{E}_{\mathbf{f^*}}$$

is a solution. But there may be more than one solution. We want to select the "best" solution. Note that $\mathbf{f^*}$ is an element of whatever domain D* is the type of $\mathbf{E}_{\mathbf{f}}$. In the most common case, it is $D \rightarrow D$, for a domain of values D, but it can be D, $D^k \rightarrow D$, … The best solution (the one that always exists, is unique, and is *computable*) is the *least* solution under the approximation ordering in D*. This *least* solution is rougly the "least defined" solution (the one that diverges most often).

Proving $\mathbf{f^*}$ is a fixed point of $\mathbf{F}$

Must show: $\mathbf{F(f^*) = f^*}$ where $\mathbf{F = \lambda\, f\, .\, E_f}$.

Claim: By definition $\mathbf{f^* = \cup F^k(bot_{D^*})}$. Since $\mathbf{F}$ is continuous

$$\mathbf{F(f^*) = F(\cup F^k(bot_{D^*}))}$$

$$\mathbf{= \cup\ F^{k+1}(bot_{D^*})} \qquad \text{(by continuity)}$$

$$\mathbf{= \cup\ F^k(bot_{D^*})} \qquad \text{(since } \mathbf{bot_{D^*} <= F(bot_{D^*})}\text{)}$$

$$\mathbf{= f^*}$$

Q.E.D.

Examples

Look at factorial in detail using DrRacket or DrScala.

How Can We Compute **f*** Given **F**?

Need to construct $F^{\infty}(\perp)$ from **F** using only λ-abstraction and application. We need to define an operator **Y** such that:

$$Y(F) = f^* = F^*(\perp).$$

Idea: use syntactic trick in Ω to build a potentially infinite stack of **F**s.

- Preliminary attempt:

```
(λx. F(x x)) (λx. F(x x))
```

- Reduces to (in one step):

```
F ((λx. F(x x)) (λx. F(x x)))
```

- Reduces to (in k steps):

F^k `((λx. F(x x)) (λx. F(x x)))`

What Is the Code for **Y**?

λF. (λx. F(x x))(λx. 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 λ-expression defining a functional like **FACT**

(λF. (λx. F(x x))(λx. F(x x)))G
= G((λx. G(x x))(λx. G(x x)))
= … (divergence forced by CBV)

What If We Use Call-by-name?

By assumption `G` must have the form `(λf. (λn. M))`

```
    (λF. (λx. F(x x))(λx. F(x x))) G
 => (λx. G(x x))(λx. G(x x))              <**>
 => G <**>
 =  (λf. (λn. M)) <**>
 => (λn. M[f:=<**>])                      <*>
```

which is a value. If this value `<*>` is applied to a value `k` and `M[f:=<**>][n:=k]` does not require evaluating an occurrence of `<**>`, then the computation returns a base answer determined by `M`. Otherwise, `<**>` is unwound once, as in the computation above to produce `<*>` 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 `<**>`. At this point, the subcomputation `<*> b` returns a base value and the enclosing computation (not involving recursive calls `<**>`) 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 `λx.Nx` is an "equivalent" term that is not divergent (assuming `x` does occur in `N`). Note that if `N` is a diverergent 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` .