Comp 311
Principles of Programming Languages
Lecture 11
The Semantics of Recursion II

Corky Cartwright

September 17, 2010

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 values. 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:

$$f \quad = \quad E_f$$

what is a solution? All of the constants and operations in E_f are known except f.

A solution is any function f^* such that

$$f^* \quad = \quad E_{f^*}$$

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 → D, but it can be D, D^k → D, … The best solution (the one that always exists, is unique, and is *computable* is the *least* solution under the approximation ordering in D*.

Constructing the Least Solution

How do we know that any solution exists to the equation $f = E_f$?

We will construct the least solution and prove it is a solution!

Since the domain D* for f is a Scott-Domain, it has a least element bot_{D*}. Hence, bot_{D*} approximates every solution to the equation

$$f = E_f .$$

Now form the function F: D* → D* defined by

$$F(f) = E_f$$

or equivalently,

$$F = \lambda f . E_f$$

Consider the sequence S: bot_{D*}, $F(bot_{D*})$, $F(F(bot_{D*}))$, ..., $F^k(bot_{D*})$, ...

Claim: S is an ascending chain (chain for short) in D* → D*.

Proof. $bot_{D*} <= F(bot_{D*})$ by the definition of Bot_{D*} . If $M <= N$, then $F(M) <= F(N)$ by monotonicity. Hence, $F^k(bot_D) <= F^{k+1}(bot_D)$ for all k. Q.E.D.

Claim: S has a least upper bound f^*

Proof. Trivial. S is a chain in D* and hence must have a least upper bound because D* is a Scott-Domain.

Proving f^* is a fixed point of F

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

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

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

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

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

$= f^*$

Q.E.D.

Note: all of the steps in the preceding proof are trivial except for the step justified by continuity.

Examples

Look at factorial in detail using DrScheme.

How Can We Compute f* Given F?

Need to construct $F^{\infty}(\bot)$ from F. Let

$Y(F) = f^* = F^{\infty}(\bot)$.

Can we write code for Y?

Idea: use syntactic trick in Ω to build a potentially infinite stack of Fs.

- 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 Scheme (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)))
= …  (diverging)
```

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
= G ((λx. G(x x))(λx. G(x x)))
= (λf. (λn. M)) ((λx. G(x x))(λx. G(x x)))
= (λn.M[x:=(λx.G(x x))(λx.G(x x))]))
```

If the evaluation **M** of does not require evaluating an occurrence of **f**, then **x** is not evaluated. Otherwise, the binding of **x** is unwound only as many times as required to get to the base case in the definition **f = λn. M**.

Exercise: how can we workaround this problem to create a version of the **Y** operator that works for call-by-value Scheme and Jam? Hint: if **M** is a divergent term denoting a unary function **λx.Mx** is an "equivalent" term that is not divergent! (As a concrete example, assume that **M** is the term Ω.)