Comp 411
Principles of Programming Languages
Lecture 9
Semantics of Recursion: Overview

Corky Cartwright

September 24, 2012

Major Challenge

- LC does not include a recursive binding operation (like Scheme **letrec/local**) or Scala binding. How would we define **eval** for such a construct?
- Key problem: the closure structure for a recursive **lambda** must include an environment that refers to itself!
- In imperative Java, how would we construct such an environment. Hint: how did you build "circular" data structures in basic OO/imperative programming courses? Imperativity is *brute force*. But it works. We will use it in Project 3 and thereafter.

Minor Challenge

How could we define an environment that refers to itself in *functional* Scheme or Scala (without `lazy`)?

- Key problem: observe that in `let` and `lambda` the expression defining the value of a variable cannot refer to itself.
- Solution: does functional Scheme contain a recursive binding construct?
- How can we use this construct to define a recursive environment?
- What environment representation must we use?

A Bigger Challenge

Assume that we want to write LC in a purely functional language without a recursive binding construct (say functional Scheme without `define` and `letrec`)?

- Key problem: must expand `letrec` into `lambda`
- No simple solution to this problem. We need to invoke syntactic magic or (equivalently) develop some sophisticated mathematical machinery.

Key Intuitions

- Computation is incremental—not monolithic
- Slogan: general computation is successive approximation (typically in response to successive demands for more information).
- Familiar example: a program mapping a potentially infinite input stream of characters to a potentially infinite output stream of characters. Generalization: infinite trees mapped to infinite trees.

Key Mathematical Concepts

- A *partially ordered set* (**po**) is a pair (S, $\leq$) where S is a set of data values and $\leq$ is a reflexive, symmetric, and transitive binary relation on S.
- A *finitary basis* (set of finite approximations) is po that is
  - countable
  - closed under LUBs (least upper bounds) on finite bounded subsets
- Every *finitary basis* contains a least element denoted $\perp$ ("bottom").
- A **chain** in a po (S, $\leq$) is a countable sequence $s0$, $s1$, …, sk, ... that is totally ordered ($si \leq sj$ if i is less than j)
- A **po** (S, $\leq$) is *complete* iff every chain in S has a LUB (least upper bound) in S. (All limit points exist.) The LUB of a set C $\subseteq$ S is written $\cup$C
- A Scott-domain is a **po** which is the completion of a finitary basis (a finitary basis with limit points added where needed).
- Every Scott-domain is a **cpo**.
- The "home-plate" **cpo** is a simple example of a **cpo** that is not a Scott domain.
- Flat domain: monolithic set of values formulated as domain. All values other than $\perp$ are maximal.
  - integers, booleans, strings, conventional finite lists, ASTs

Key Mathematical Concepts cont.

Computable functions:

- A function f in the domain A $\rightarrow$ B is *monotonic* iff for all $a \leq a'$ in A, $f(a) \leq f(a')$.
- A function f in the domain A $\rightarrow$ B is *continuous* iff for every chain c in S, $f(\cup c) = \cup f(c)$
- A function f in the domain A $\rightarrow$ B is *strict* iff $f(\perp) = \perp$

Computable functions are necessarily continuous (which implies monotonic!) and typically are strict. (But not in Haskell or Scala [sort of])

Examples

Domains

- flat domains
- strict function spaces on flat domains
- lazy trees of boolean (of D where D is flat)
- factorial functional