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

Corky Cartwright
February 2, 2018
Key Intuitions

• Computation is incremental not monolithic
• Slogan: general computation is successive approximation (typically in response to successive demand for more information). In simple computations, only the standard output stream is repeatedly demanded until EOF (end-of-file) datum is encountered.
Key Mathematical Concepts

Semantic Domains

- A partial order (po) is a set $S$ with a reflexive, transitive, anti-symmetric binary relation $\leq$.
- A chain in a po is a countable totally ordered set $c_0 \leq c_1 \leq c_2 \ldots \leq c_k \leq \ldots$ (See Wikipedia for the definition of a countable set, which may be empty.)
- A po is chain-complete iff every chain has a least upper bound (LUB) in the po. Such a partial order is called a complete partial order (cpo). Since a chain can be empty, every cpo must have a least element, which we denote by the symbol $\perp$, called “bottom”. In the domain theory monograph, directed sets are used instead of chains; it is easy to prove the two notions are equivalent for domains with a countable basis (defined below). We are only interested in cpos with countable bases.
  - The home-plate cpo given in class is not chain-complete.
- A subset $S$ within a po is consistent iff it has an upper bound in the po.
- A po is finitely consistent if every
- A finitary basis is a countable po in which every finite consistent set has a LUB.
- Given a finitary basis $B$, the (Scott) domain determined by $B$ is the cpo created by adding LUBs for infinite chains in $B$. The elements of $B$ are called the finite elements of this domain. The monograph contains an explicit construction of this domain using ideals. The intuition is simple: the generated domain simply adds an element for each infinite chain of finite elements that is only above all elements in the downward closure of the chain.
- The topologically finite elements of the cpo determined by $B$ are precisely the elements of $B$. (Don’t worry about the definition of topologically finite; it is defined in the monograph.)
Key Mathematical Concepts

All (incrementally) computable functions $f$ mapping domain $A$ into domain $B$ are:

- **monotonic**: $x \leq y \Rightarrow f(x) \leq f(y)$
- **continuous**: given a chain $C = \{c_i \mid i \in \mathbb{N}\}$,
  \[ f(\bigsqcup C) = \bigsqcup \{ f(c) \mid c \in C \} \]
  Note that a continuous function may not be computable.

The domains, other than function domains, supported by most programming languages are *flat*: every element $d \in D$ except $\bot$ is *finite* and *maximal*. Some examples include integers, booleans, strings, structures, arrays of structures, *etc*. Consider some unary total function $g$ on the natural numbers that is not recursive (computable). In domain theory, there is a simple function corresponding to $g$ over the flat domain of natural numbers called the *natural extension* of $g$ where $g(\bot) = \bot$. This function is monotonic and continuous but it is not computable. In languages support the lazy construction of objects (structures), the data domains corresponding to lazy constructions are *not* flat.
Some Domain Examples

• flat domains
• strict function spaces on flat domains
• non-strict function spaces (call-by-name!)
• lazy trees of boolean
• lazy abstract syntax trees
• functions from domain $A$ into domain $B$
• What if domain $A^+$ contains $A$ and $B$ contains domain $B^-$? What is relationship between $A \rightarrow B$ and $A^+ \rightarrow B^-$?

• Contra-variance vs. co-variance
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` [recursive binding as in Java methods])?

- Key problem: must expand `letrec` into `lambda`

- No simple solution to this problem. We need to invoke syntactic magic or develop some sophisticated mathematical machinery.