Introduction to Domain Theory and Denotational Semantics

Programming Languages CS442

David Toman

School of Computer Science
University of Waterloo

What is the Problem?

- What should be the semantics of the **while** loop?

$$[\![\textbf{while}\, E\, \textbf{do}\, C\, \textbf{od}]\!] = \\ \underline{\lambda s}.\mathit{if}\ [\![E]\!]s\ \mathit{then}\ [\![\textbf{while}\, E\, \textbf{do}\, C\, \textbf{od}]\!]([\![C]\!]s)\ \mathit{else}\ s$$

$\Rightarrow$ fine for *operational semantics*

$\Rightarrow$ but what is the true meaning?

Idea

We define $[\![\textbf{while}\, E\, \textbf{do}\, C\, \textbf{od}\,]\!] = f$ *where* $f : Store_\perp \to Store_\perp$ *is an appropriate solution to the equation*

$$f = \mathit{if}\ [\![E]\!]s\ \mathit{then}\ f([\![C]\!]s)\ \mathit{else}\ s$$

Examples and Desiderata

- Example (factorial):

$$f = \lambda n.\,\textbf{if}\, n = 0\,\textbf{then}\,1\,\textbf{else}\, n * (f\ (n-1))$$

- Example:

$$g = \lambda n.\,\textbf{if}\, n = 0\,\textbf{then}\,1\,\textbf{else}\, g\ (n+1)$$

- Questions:
  1. Do recursive equations always have a solution?
  2. Do they have a *unique* solution?

     $\Rightarrow$ if not, how do we pick the *right* one?
  3. Does such a solution correspond to the *operational definition*?

Solution Idea

Idea

Define the graph of the function by iterating the associated functional.

$\Rightarrow$ *successive iterations = better approximations*

$\Rightarrow$ *limit of the iterations = solution*

Example

- Functional for the *factorial* function:

$$F = \lambda f.\lambda n.\,\textbf{if}\, n = 0\,\textbf{then}\,1\,\textbf{else}\, n * (f\ (n-1))$$

- Approximations:

$$\begin{array}{l} f_0 = \{\}\ \ f_1 = \{(0,1)\}\ \ f_2 = \{(0,1),(1,1)\} \\ f_3 = \{(0,1),(1,1),(2,2)\}\ \ f_4 = \{(0,1),(1,1),(2,2),(3,6)\}\ldots \end{array}$$

- Solution: $f = \bigcup_{i \geq 0} f_i$

Partial Orders and CPOs

How do we guarantee that the *iteration-limit* trick works?

Idea

Define structures where the existence of solutions is guaranteed.

- $(D, \leq)$ is a *partial order* if for all $a, b, c \in D$ we have

$$a \leq a, \quad a \leq b \wedge b \leq a \rightarrow a = b, \quad a \leq b \wedge b \leq c \rightarrow a \leq c$$

- $\perp$ is a *least element* of $(D, \leq)$ if $\perp \leq a$ *for all* $a \in D$
- $C \subseteq D$ is a *chain* if $a \leq b$ *or* $b \leq a$ *for all* $a, b \in C$
- $\bigsqcup X$ is a *least upper bound* of $X \subseteq D$ if

$$\begin{array}{l} (1)\ x \leq \bigsqcup X \ \ \textit{for all} \ \ x \in X, \ \textit{and} \\ (2)\ \textit{for all} \ \ d \in D, \textit{if} \ x \leq d \ \ \textit{for all} \ \ x \in X \ \textit{then} \ \bigsqcup X \leq d \end{array}$$

Partial Orders and CPOs (cont.)

Definition (Complete Partial Order)

A partial order $(D, \leq)$ is a (pointed) CPO if (it has a least element and) each chain $C \subseteq D$ there is a $\bigsqcup C \in D$.

Examples:

1. *flat* domains are CPOs (with discrete order):

   $\Rightarrow$ booleans, integers (note the *different* order!), ...

2. the *powerset* is a pointed CPO

   $\Rightarrow$ ordered by *set inclusion*

Functions on CPOs

Idea

values = elements of CPOs (ordered the their definedness)

programs = functions between CPOs

What functions do we consider?

Example

$$halts = \lambda x.\,\textbf{if}\, x \neq \bot\, \textbf{then}\, true\, \textbf{else}\, false$$

Let $f : int \to int$ and $n : int$. What does $halts(f(n))$ do?

Definition

A function $f : D \to E$ is monotonic if

$$a \leq_D b \to f(a) \leq_E f(b) \textit{ for all } a, b \in D$$

Functions on CPOs (cont.)

Is monotonicity quite enough? NO!

Idea

We want to be able to define the result of an application of a function on a limit of approximations by a limit of applying the function on the approximations.

Definition (Continuity)

A function $f : D \to E$ is continuous if

$$f\left(\bigsqcup X\right) = \bigsqcup\{f(x) \mid x \in X\}$$

for all chains $X \subseteq D$.

The Function Space CPO

We are NOT approximating values but *functions*

$\Rightarrow$ how do we take *least upper bounds of functions*?

Idea

We arrange functions into a CPO ordered by their definedness!

Definition (Function Space)

$A \rightarrow B$ is the set of all continuous functions between CPOs A and B; partially ordered as follows:

$$f \leq_{A \rightarrow B} g \iff \forall a \in A. f(a) \leq_B g(a)$$

$\Rightarrow$ we need to show that $A \rightarrow B$ is a CPO,

$\Rightarrow$ and that abstraction/application are continuous

this is important so that *functionals* are continuous!!

Solution to Recursive Functions

Theorem

Let D be a pointed CPO and $F : D \to D$ a continuous function. Then the least fixed point of F exists and is defined as

$$\mathit{fix}\ F = \bigsqcup \{F^i(\perp) \mid i \geq 0\}$$

The meaning of a definition of the form $f = F(f)$ is $\mathit{fix}\ F$.

and we still need to show:

1. $\mathit{fix}\ F$ is indeed the least fixed point, i.e., $\mathit{fix}\ F = F(\mathit{fix}\ F)$
2. $\mathit{fix}\ F$ matches the operational semantics

More Compound Domains

- Discrete domains = trivial CPOs
- We can *construct* complex CPOs from simple ones
  - the *lifting* $D_\perp$
  - the *product* $D \times E$
  - the *sum* $D + E$
  - the *function space* $D \rightarrow E$

    $\Rightarrow$ the set of all *continuous functions*
- the associated operations are continuous functions

Theorem

Any operation built using functional notation is a continuous function.

Can Domains be Defined Recursively?

- *domain constructors* $\Rightarrow$ more complex domains

  $\Rightarrow$ sufficient to interpret PCF (=simply typed λ-calculus+**rec**)

- so far only stratified types are allowed!!

  $\Rightarrow$ this might be insufficient for

  1. recursively defined types, e.g., *lists, trees, ...*:

$$\alpha \mathit{list} = \mathit{nil} + (\alpha \times \alpha \mathit{list})$$
$$\alpha \mathit{tree} = \alpha + (\alpha \mathit{tree} \times \alpha \mathit{tree})$$

  2. self-applications (procedures-as-parameters, untyped λ-calculus)

$$D = D \rightarrow D$$

- we try the *approximation-n-limits* approach again...

How do we Order Domains?

Idea

Smaller domains can be embedded into larger ones.

Definition (Embedding-projection pair)

Continuous functions $e : D \to E$ and $p : E \to D$ form an embedding-projection pair if

$$p \circ e = id_D \qquad e \circ p \leq_E id_E$$

We use the body of recursive type equation, F, to construct:

$$D_0 \overset{p_0}{\underset{e_0}{\leftrightarrows}} F(D_0) \overset{p_1}{\underset{e_1}{\leftrightarrows}} F^2(D_0) \overset{p_2}{\underset{e_2}{\leftrightarrows}} F^3(D_0) \overset{p_3}{\underset{e_3}{\leftrightarrows}} \ldots \overset{p_k}{\underset{e_k}{\leftrightarrows}} F^{k+1}(D_0) \overset{p_{k+1}}{\underset{e_{k+1}}{\leftrightarrows}} \ldots$$

$\Rightarrow$ what is D_0? $\quad\Rightarrow$ the (e_i, p_i) pairs?

Inverse Limit Construction

- Given a *retraction sequence*

$$D_0 \underset{e_0}{\overset{p_0}{\leftrightarrows}} F(D_0) \underset{e_1}{\overset{p_1}{\leftrightarrows}} F^2(D_0) \underset{e_2}{\overset{p_2}{\leftrightarrows}} F^3(D_0) \underset{e_3}{\overset{p_3}{\leftrightarrows}} \ldots \underset{e_k}{\overset{p_k}{\leftrightarrows}} F^{k+1}(D_0) \underset{e_{k+1}}{\overset{p_{k+1}}{\leftrightarrows}} \ldots$$

  how do we construct the *limit*? It better be a CPO!

  and taking a directed limit (essentially union) doesn't work

- *Inverse limit* (co-limit) construction:

$$D_\infty = \{(a_0, a_1, \ldots) \mid a_i \in F^i(D_0), x_i = p_i(x_{i+1})\}$$

$\Rightarrow$ does $D_\infty = F(D_\infty)$ hold? No, but they're isomorphic!

Summary

- *Fixpoint semantics* gives a precise *mathematical understanding* of loops and recursion.
- *Continuity* is essential to understand how infinite objects can be approximated by finite programs.
- The CPO machinery pays off when we look on higher-order programming languages.

  $\Rightarrow$ for integer functions: subsets of graphs approach works

  $\Rightarrow$ for higher-order functions it doesn't
- The *approximation* approach works for recursive types (e.g., lists)

  $\Rightarrow$ this needs *inverse limit* construction to get CPOs.