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?

\[
\text{[while } E \text{ do } C \text{ od]} = \lambda s. \text{if } [E]s \text{ then } \text{[while } E \text{ do } C \text{ od]}([C]s) \text{ else } s
\]

⇒ fine for operational semantics
⇒ but what is the true meaning?

Idea

We define \( [\text{while } E \text{ do } C \text{ od}] = f \) where \( f : \text{Store}_\perp \rightarrow \text{Store}_\perp \) is an appropriate solution to the equation

\[
f = \text{if } [E]s \text{ then } f([C]s) \text{ else } s
\]
Examples and Desiderata

- Example (factorial):

  \[ f = \lambda n. \text{if } n = 0 \text{ then } 1 \text{ else } n \ast (f \ (n - 1)) \]

- Example:

  \[ g = \lambda n. \text{if } n = 0 \text{ then } 1 \text{ else } g \ (n + 1) \]

- Questions:

  1. Do recursive equations always have a solution?
  2. Do they have a *unique* solution?
     \[ \Rightarrow \text{ if not, how do we pick the *right* one?} \]
  3. Does such a solution correspond to the *operational definition*?
Solution Idea

Idea

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

⇒ successive iterations = better approximations

⇒ limit of the iterations = solution

Example

• Functional for the factorial function:

\[ F = \lambda f. \lambda n. \text{if } n = 0 \text{ then } 1 \text{ else } n \times (f(n - 1)) \]

• Approximations:

\[ f_0 = \{\} \quad f_1 = \{(0, 1)\} \quad f_2 = \{(0, 1), (1, 1)\} \]
\[ f_3 = \{(0, 1), (1, 1), (2, 2)\} \quad f_4 = \{(0, 1), (1, 1), (2, 2), (3, 6)\} \ldots \]

• 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
  
  $$a \leq a, \  a \leq b \land b \leq a \rightarrow a = b, \  a \leq b \land b \leq c \rightarrow a \leq c$$

- $\bot$ is a *least element* of $(D, \leq)$ if $\bot \leq a$ for all $a \in D$

- $C \subseteq D$ is a *chain* if $a \leq b \text{ or } b \leq a$ for all $a, b \in C$

- $\bigsqcup X$ is a *least upper bound* of $X \subseteq D$ if

  1. $x \leq \bigsqcup X$ for all $x \in X$, and
  2. for all $d \in D$, if $x \leq d$ for all $x \in X$ then $\bigsqcup X \leq d$
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 by their definedness)
programs = functions between CPOs

What functions do we consider?

Example

\[ \text{halts} = \lambda x. \text{if } x \neq \perp \text{ then true else false} \]

Let \( f : \text{int} \rightarrow \text{int} \) and \( n : \text{int} \). What does \( \text{halts}(f(n)) \) do?

Definition
A function \( f : D \rightarrow E \) is monotonic if

\[ a \leq_D b \rightarrow f(a) \leq_E f(b) \text{ for all } a, b \in D \]
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 $f : D \rightarrow E$ is continuous if

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

for all chains $X \subseteq D$. 
We are NOT approximating values but \textit{functions}.

\[ \implies \text{how do we take least upper bounds of functions?} \]

\textbf{Idea}

\textit{We arrange functions into a CPO ordered by their definedness!}

\textbf{Definition (Function Space)}

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

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

\[ \implies \text{we need to show that } A \rightarrow B \text{ is a CPO}, \]

\[ \implies \text{and that abstraction/application are continuous} \]

\[ \implies \text{this is important so that functionals are continuous!!} \]
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

$$\text{fix } F = \bigsqcup \{ F^i(\bot) \mid i \geq 0 \}$$

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

and we still need to show:

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

- Discrete domains = trivial CPOs
- We can *construct* complex CPOs from simple ones
  - the *lifting* $D_{\bot}$
  - 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 \(\lambda\)-calculus+\texttt{rec})

- so far only **stratified** types are allowed!!
  \(\Rightarrow\) this might be insufficient for
  
  1. recursively defined types, e.g., lists, trees, . . .:
     
     \[ \alpha \text{list} = \text{nil} + (\alpha \times \alpha \text{list}) \]
     
     \[ \alpha \text{tree} = \alpha + (\alpha \text{tree} \times \alpha \text{tree}) \]

  2. self-applications (procedures-as-parameters, untyped \(\lambda\)-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 \( e : D \rightarrow E \) and \( p : E \rightarrow D \) form an embedding-projection pair if

\[
p \circ e = id_D \quad \quad e \circ p \leq_E id_E
\]

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

\[
D_0 \overset{p_0}{\leftrightarrow} F(D_0) \overset{p_1}{\leftrightarrow} F^2(D_0) \overset{p_2}{\leftrightarrow} F^3(D_0) \overset{p_3}{\leftrightarrow} \ldots \overset{p_k}{\leftrightarrow} F^{k+1}(D_0) \overset{p_{k+1}}{\leftrightarrow} \ldots
\]

⇒ what is \( D_0 \)?  ⇒ the \((e_i, p_i)\) pairs?
Inverse Limit Construction

- Given a retraction sequence

\[
D_0 \xleftrightarrow{p_0} F(D_0) \xleftrightarrow{p_1} F^2(D_0) \xleftrightarrow{p_2} F^3(D_0) \xleftrightarrow{p_3} \ldots \xleftrightarrow{p_k} F^{k+1}(D_0) \xleftrightarrow{p_{k+1}} \ldots
\]

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:

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

⇒ 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.
  
  ⇒ for integer functions: subsets of graphs approach works
  ⇒ for higher-order functions it doesn’t

- The *approximation* approach works for recursive types (e.g., lists)
  ⇒ this needs *inverse limit* construction to get CPOs.