CS611 Lecture 21 | Denotational Semantics of REC | 18 October 2006
Lecturer: Dexter Kozen

1 A Metalanguage for Domain Constructions

Last time we did several constructions that required us to check that various domains were CPOs and that various associated operations were continuous. How can we avoid doing this kind of check over and over again? One solution is to create an abstract metalanguage consisting of some basic operations that will allow us to do domain constructions (like function spaces, direct products, etc.) and that will ensure that the domains that are constructed are CPOs and the associated functions are continuous. We can compose these constructions to create more complicated domains from simpler ones and always be assured that the desired mathematical properties hold.

The simplest objects will be the discrete CPOs $\mathbb{Z}$, $\mathbb{N}$ and $\mathbb{U}$ for the integers, the natural numbers, and the unit domain, respectively. The unit domain contains a single element unit. These all have the *discrete order*, meaning that if $x \sqsubseteq y$, then $x = y$.

For any domain A, we can construct a new domain $A_\bot$, which is A adjoined with a new element $\bot$ below all the previous elements. Note that $\bot$ is intended to be a new element, so we can actually iterate this operation. The associated operations are the natural embedding $D \to D_\bot$ and the lifting operation $(\cdot)^* : (D \to E_\bot) \to (D_\bot \to E_\bot)$ defined by

$$(d)^*(x) \quad \triangleq \quad \begin{cases} d(x), & \text{if } x \neq \bot, \\ \bot, & \text{if } x = \bot. \end{cases}$$

Both these operations are continuous.

Given CPOs D and E, we can form the product $D \times E$ consisting of all ordered pairs $\langle d, e\rangle$ with $d \in D$ and $e \in E$, ordered componentwise. This is the set-theoretic Cartesian product of D and E with $\langle d, e\rangle \sqsubseteq \langle d', e'\rangle$ iff $d \sqsubseteq d'$ and $e \sqsubseteq e'$. This is a CPO, and it is easy to check that $\bigsqcup_{d\in X,\ e\in Y}\langle d, e\rangle = \langle \bigsqcup X, \bigsqcup Y\rangle$. Along with the product constructor come the projections π_1 and π_2 defined by $\pi_1(\langle d, e\rangle) = d$ and $\pi_2(\langle d, e\rangle) = e$, which are continuous. If $f : C \to D$ and $g : C \to E$, then the function $\langle f, g\rangle : C \to D \times E$ defined by $\langle f, g\rangle \triangleq \lambda x.\, \langle f(x), g(x)\rangle$ is continuous if f and g are. This is the unique function satisfying the equations $f = \pi_1 \circ \langle f, g\rangle$ and $g = \pi_2 \circ \langle f, g\rangle$. The binary product can be generalized to an arbitrary product $\prod_{x\in X} D_x$ with associated projections $\pi_y : \prod_{x\in X} D_x \to D_y$.

Given CPOs D and E, we can form the *sum* (or *coproduct*) $D + E$, corresponding to the disjoint union of D and E. We would like to take the union of the sets D and E, but we need to mark the elements to make sure we can tell which set they originally came from in case D and E have a nonempty intersection. To do this, we define

$$D + E \quad \triangleq \quad \{\mathsf{in}_1(d) \mid d \in D\} \cup \{\mathsf{in}_2(e) \mid e \in E\},$$

where in_1 and in_2 are any one-to-one functions with disjoint ranges; for example, we could take $\mathsf{in}_1(x) = \langle 1, x\rangle$ and $\mathsf{in}_2(x) = \langle 2, x\rangle$. We define $\mathsf{in}_i(x) \sqsubseteq \mathsf{in}_j(y)$ iff $i = j$ and $x \sqsubseteq y$. Any chain in $D + E$ must be completely contained in $\{\mathsf{in}_1(x) \mid x \in D\}$ or $\{\mathsf{in}_2(x) \mid x \in E\}$, so $D + E$ is a CPO. The associated operations are the injections $\mathsf{in}_1 : D \to D + E$ and $\mathsf{in}_2 : E \to D + E$, which are continuous. If $f : D \to C$ and $g : E \to C$, then we can combine f and g into a function $f + g : D + E \to C$ using a case construct:

$$f + g \quad \triangleq \quad \lambda x.\, \mathsf{case}\ x\ \mathsf{of}\ \mathsf{in}_1(y) \to f(y) \mid \mathsf{in}_2(y) \to g(y).$$

This is continuous if f and g are, and it is the unique function satisfying the equations $f = (f + g) \circ \mathsf{in}_1$ and $g = (f + g) \circ \mathsf{in}_2$. As with products and projections, the binary coproduct can be generalized to an arbitrary coproduct $\sum_{x\in X} D_x$ with associated injections $\mathsf{in}_y : D_y \to \sum_{x\in X} D_x$.

Finally, given CPOs D and E, we can define the CPO $[D \to E]$ of all continuous functions from D to E with the pointwise ordering. The corresponding operations are:

1. $\mathsf{apply} : [D \to E] \times D \to E$ that applies a given function to a given argument;
2. $\mathsf{compose} : [E \to F] \times [D \to E] \to [D \to F]$;
3. $\mathsf{curry} : [D \times E \to F] \to [D \to [E \to F]]$;
4. $\mathsf{uncurry} : [D \to [E \to F]] \to [D \times E \to F]$; and most importantly,
5. $\mathsf{fix} : [D \to D] \to D$, defined by $\lambda g \in [D \to D].\ \bigsqcup g^n(\bot)$, that takes a function and returns its least fixpoint. To apply fix, D must have a bottom element $\bot$.

All these functions are continuous.

2 Denotational Semantics for REC

2.1 REC Syntax

$$\begin{array}{lcl}
p & ::= & \mathsf{let}\ d\ \mathsf{in}\ e \\
d & ::= & f_1(x_1, \ldots, x_{a_1}) = e_1 \\
 & & \vdots \\
 & & f_n(x_1, \ldots, x_{a_n}) = e_n \\
e & ::= & n \mid x \mid e_1 \oplus e_2 \mid \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 \mid \mathsf{ifp}\ e_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2 \mid f_i(e_1, \ldots, e_{a_i})
\end{array}$$

The functions in d are mutually recursive. It is reasonable to expect that under most semantics, $\mathsf{let}\ f_1(x_1) = f_1(x_1)\ \mathsf{in}\ f_1(0)$ will loop infinitely, but $\mathsf{let}\ f_1(x_1) = f_1(x_1)\ \mathsf{in}\ 0$ will halt and return 0.

For example,

$$\begin{array}{ll}
\mathsf{let} & \\
 & f_1(n, m) = \mathsf{ifp}\ m^2 - n\ \mathsf{then}\ 1\ \mathsf{else}\ (n - m(n\ \mathsf{div}\ m)) \cdot f_1(n, m+1) \\
 & f_2(n) = \mathsf{ifp}\ f_1(n, 2)\ \mathsf{then}\ n\ \mathsf{else}\ f_2(n+1) \\
\mathsf{in} & \\
 & f_2(1000)
\end{array}$$

In this REC program, $f_2(n)$ finds the first prime number $p \geq n$. The value of $n - m(n\ \mathsf{div}\ m$ is positive iff m does not divide n.

2.2 CBV Denotational Semantics for REC

The meaning function is $[\![e]\!] \in \mathbf{FEnv} \to \mathbf{Env} \to \mathbb{Z}_\bot$, where **FEnv** and **Env** denote the sets of variable environments and function environments, respectively, as used in REC.

$$\begin{array}{lcl}
\rho \in \mathbf{Env} & = & \mathbf{Var} \to \mathbb{Z} \\
\varphi \in \mathbf{FEnv} & = & (\mathbb{Z}^{a_1} \to \mathbb{Z}_\bot) \times \cdots \times (\mathbb{Z}^{a_n} \to \mathbb{Z}_\bot)
\end{array}$$

Here **Var** is a countable set of variables, $\mathbb{Z}$ is the set of integers, which are the values that can be bound to a variable in an environment, and $\mathbb{Z}^m = \underbrace{\mathbb{Z} \times \mathbb{Z} \times \cdots \times \mathbb{Z}}_{m \text{ times}}$.

$$
\begin{array}{rcl}
[\![n]\!]\,\varphi\,\rho & \stackrel{\triangle}{=} & n \\[1ex]
[\![x]\!]\,\varphi\,\rho & \stackrel{\triangle}{=} & \rho(x) \\[1ex]
[\![e_1 \ \oplus\ e_2]\!]\,\varphi\,\rho & \stackrel{\triangle}{=} & \mathsf{let}\ v_1 \in \mathbb{Z} = [\![e_1]\!]\,\varphi\,\rho\ \mathsf{in} \\
& & \quad \mathsf{let}\ v_2 \in \mathbb{Z} = [\![e_2]\!]\,\varphi\,\rho\ \mathsf{in} \\
& & \qquad v_1 \ \oplus\ v_2 \\
& = & [\![e_1]\!]\,\varphi\,\rho \quad \oplus_\perp \quad [\![e_2]\!]\,\varphi\,\rho \\[1ex]
[\![\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2]\!]\,\varphi\,\rho & \stackrel{\triangle}{=} & \mathsf{let}\ y \in \mathbb{Z} = [\![e_1]\!]\,\varphi\,\rho\ \mathsf{in} \\
& & \quad [\![e_2]\!]\,\varphi\,\rho[y/x] \\[1ex]
[\![\mathsf{ifp}\ e_0\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2]\!]\,\varphi\,\rho & \stackrel{\triangle}{=} & \mathsf{let}\ v_0 \in \mathbb{Z} = [\![e_0]\!]\,\varphi\,\rho\ \mathsf{in} \\
& & \quad \mathsf{if}\ v_0 > 0\ \mathsf{then}\ [\![e_1]\!]\,\varphi\,\rho\ \mathsf{else}\ [\![e_2]\!]\,\varphi\,\rho \\[1ex]
[\![f_i(e_1, \ldots, e_{a_i})]\!]\,\varphi\,\rho & \stackrel{\triangle}{=} & \mathsf{let}\ v_1 \in \mathbb{Z} = [\![e_1]\!]\,\varphi\,\rho\ \mathsf{in} \\
& & \vdots \\
& & \mathsf{let}\ v_{a_i} \in \mathbb{Z} = [\![e_{a_i}]\!]\,\varphi\,\rho\ \mathsf{in} \\
& & \quad (\pi_i\ \varphi)\langle v_1, \ldots, v_{a_i} \rangle
\end{array}
$$

The meaning of a program let d in e is

$$[\![\mathsf{let}\ d\ \mathsf{in}\ e]\!] \ \stackrel{\triangle}{=}\ [\![e]\!]\,\varphi\,\rho,$$

where ρ is some initial environment containing default values for the variables, and

$$
\begin{array}{rcl}
\varphi & = & \mathsf{fix}\ \lambda\psi \in \mathbf{FEnv}.\,\langle \lambda v_1 \in \mathbb{Z}, \ldots, v_{a_1} \in \mathbb{Z}.\,[\![e_1]\!]\,\psi\,\rho[v_1/x_1, \ldots, v_{a_1}/x_{a_1}], \\
& & \qquad\qquad\qquad \vdots \\
& & \qquad\qquad\qquad \lambda v_1 \in \mathbb{Z}, \ldots, v_{a_n} \in \mathbb{Z}.\,[\![e_n]\!]\,\psi\,\rho[v_1/x_1, \ldots, v_{a_n}/x_{a_n}] \rangle.
\end{array}
$$

For this fixpoint to exist, we need to know that **FEnv** a pointed CPO. But **FEnv** is a product, and a product is a pointed CPO when each factor is a pointed CPO. Each factor $\mathbb{Z}^{a_i} \to \mathbb{Z}_\perp$ is a pointed CPO, since a function is a pointed CPO when the codomain of that function is a pointed CPO, and $\mathbb{Z}_\perp$ is a pointed CPO. Therefore, **FEnv** is a pointed CPO.

We also need to know that the function $\mathbf{FEnv} \to \mathbf{FEnv}$ to which we are applying fix is continuous, but it is because is written using the metalanguage.

2.3 CBN Denotational Semantics

The denotational semantics for CBN is the same as for CBV with two exceptions:

$$
\begin{array}{rcl}
[\![\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2]\!]\,\varphi\,\rho & \stackrel{\triangle}{=} & [\![e_2]\!]\,\varphi\,\rho[[\![e_1]\!]\,\varphi\,\rho/x] \\[1ex]
[\![f_i(e_1, \ldots, e_{a_i})]\!]\,\varphi\,\rho & \stackrel{\triangle}{=} & (\pi_i\ \varphi)\langle [\![e_1]\!]\,\varphi\,\rho, \ldots, [\![e_{a_i}]\!]\,\varphi\,\rho \rangle.
\end{array}
$$