Lecturer: Andrew Myers

To develop a denotational semantics for a language with recursive types, or to give a denotational semantics for the untyped lambda calculus, it is necessary to find domains that are solutions to domain equations. Given some domain constructor $\mathcal{F}(\mathcal{D})$, we need to be able to solve for the domain D satisfying the isomorphism:

$$D \cong \mathcal{F}(D)$$

We have seen some strategies for solving such equations earlier. In particular, inductively defined sets also satisfy a similar the equation, with the rule operator taking the role of $\mathcal{F}$. However, inductively defined sets do not generate complete partial orders; they only produce the elements that can be constructed by some finite number of applications of $\mathcal{F}$. This means that we cannot use them in any semantics where it is necessary to take a fixed point over D.

While it would be nice to be able to solve this equation as an equality, an isomorphism between the domains is sufficient.

If we interpret a recursive type $\mu X.\tau$ as a domain D, then $\mathcal{F}(D)$ corresponds to the unfolding of the type, $\tau\{\mu X.\tau/X\}$. The isomorphism connecting D and $\mathcal{F}(D)$ therefore corresponds to the **fold** and **unfold** operations:

$$\mathbf{fold} : \tau\{\mu X.\tau/X\} \to \mu X.\tau$$
$$\mathbf{unfold} : \mu X.\tau \to \tau\{\mu X.\tau/X\}$$

We are looking for an isomorphism witnessed by a continuous bijection *up* and $\mathit{down} = \mathit{up}^{-1}$, so that we can use *up* to model **fold** and *down* to model **unfold**:

$$\mathit{up} : [\mathcal{F}(D) \to D]$$
$$\mathit{down} : [D \to \mathcal{F}(D)]$$

The isomorphism between the domains must preserve the ordering structure of the elements. That is, it should be homomorphic with respect to the ordering relation $\sqsubseteq$:

$$d \sqsubseteq d' \Rightarrow \mathit{up}(d) \sqsubseteq \mathit{up}(d')$$
$$d \sqsubseteq d' \Rightarrow \mathit{down}(d) \sqsubseteq \mathit{down}(d')$$

1 Approximating the solution

We have already seen that for other recursive definitions $x = f(x)$, we can find a solution by taking the limit of the sequence $f^n(\bot)$, where $\bot$ is some initial element. We can apply the same strategy to solving domain equations. We start from some initial domain D_0, and apply $\mathcal{F}$ to obtain a sequence of domains $\mathcal{F}(D_0), \mathcal{F}(\mathcal{F}(D_0)), \mathcal{F}(\mathcal{F}(\mathcal{F}(D_0))), \ldots$ where each domain in the sequence is a better approximation to the desired solution, yet preserves and extends the structure of the earlier approximations.

2 An ordering on domains

Therefore we need a way to relate two domains. We write $D \precsim E$ to indicate that D is a simplified version of E, to within some isomorphism. Our goal is to have

$$\mathcal{F}(D_0) \precsim \mathcal{F}(\mathcal{F}(D_0)) \precsim \mathcal{F}(\mathcal{F}(\mathcal{F}(D_0))) \precsim \ldots$$

and then to use these approximations to take a limit of the sequence, much as we did in previous fixed-point constructions.

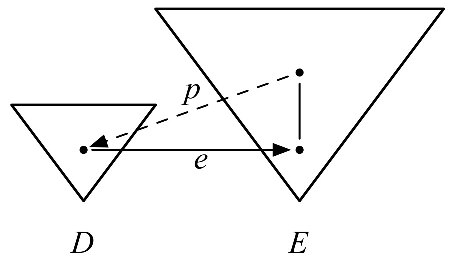


Figure 1: Embedding a domain D into a domain E

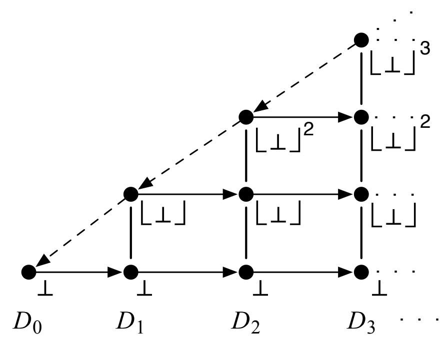


Figure 2: Successive approximations for $D = D_\perp$

Two domains D and E are related if there exists a way of embedding D into E while preserving its structure. We can characterize this embedding in terms of a pair of functions: an embedding function $e : [D \to E]$ and a projection function $p : [E \to D]$. These functions must be continuous, and as depicted in Figure 1, they must also agree in the following sense: for all elements $d \in D$ and $d' \in E$, $p(e(d)) = d$ and $e(p(d')) \sqsubseteq d'$. That is, on corresponding elements of D and E, the functions e and p act as inverses; on new elements in E, the projection function maps them to an element of D whose corresponding E element is related. Together, these functions are called an *embedding-projection pair* (ep-pair) (or just *projection pair*).

3 A simple domain equation

For example, consider the domain equation $D = D_\perp$. The function $\mathcal{F}(D)$ maps each element $d \in D$ to $\lfloor d \rfloor$, and introduces a new element $\perp$. This is essentially the domain equation for a lazy infinite stream of unit values, because $D_\perp \cong (\mathbb{U} \times D)_\perp$. So assuming that the solution to the equation is a CPO (and it is), we can use the solution to give meaning to expressions like $\textbf{rec}\ x.(\textbf{null}, x)$, where we need to take a fixed point over D.

There are two obvious ways to define an embedding-projection pair relating the domains D and $D_\perp$, leading to two different solutions to the domain equation. The one we'll explore is shown in Figure 2. In the figure, leftward arrows represent p. Rightward arrows represent e, and implicitly, a p arrow in the opposite direction.

Given a sequence of domains $D_0 \precsim D_1 \precsim D_2, \ldots,$ there is a corresponding sequence of embedding and projection functions $e_n : D_n \to D_{n+1}$ and $p_n : D_{n+1} \to D$. The diagram of Figure 2 corresponds to the following definition of these functions by induction on n:

$$
\begin{array}{rcll}
e_n(\bot) & = & \bot & \\
e_n(\lfloor d_{n-1} \rfloor) & = & \lfloor e_{n-1}(d_{n-1}) \rfloor & \quad \text{(where } n > 0\text{)} \\
p_n(\bot) & = & \bot & \\
p_0(\lfloor \bot \rfloor) & = & \bot & \\
p_n(\lfloor d_n \rfloor) & = & \lfloor p_{n-1}(d_n) \rfloor & \quad \text{(where } n > 0\text{)}
\end{array}
$$

This may seem like an needlessly complex way to define e_n and p_n, but it is done this way to show the approach that is used for more complex domain equations. Given these definitions, we easily show by induction that e_n and p_n form a valid ep-pair.

The definition is simplified if given a function $f : [D \to E]$, we define the notation $f^\bot : D_\bot \to E_\bot$ as follows:

$$
\begin{gathered}
f^\bot = \bot \\
f^\bot(\lfloor x \rfloor) = \lfloor f(x) \rfloor
\end{gathered}
$$

Then $e_{n+1} = e_n^\bot$ and $p_{n+1} = p_n^\bot$.

4 A solution to the domain equation

We are now ready to define the elements of the solution domain D. It is the *projective limit* (or *inverse limit*) of the domains D_n: the infinite *commuting tuples* $\langle d_0, d_1, d_2, \ldots \rangle$, where for all $n \geq 0$, $d_n \in D_n$, and further, $d_n = p_n(d_{n+1})$. Therefore, given an element d_n, it is possible to apply the projection functions $p_{n-1}, p_{n-2}, \ldots, p_0$ to obtain all the previous tuple elements. For brevity, we write these tuples in a comprehension form: $\langle d_n \rangle_{n \in \mathbb{N}}$ or even simply $\langle d_n \rangle$.

Since each of the D_n is a CPO, the the elements of D form a CPO when ordered pointwise: $\langle d_n \rangle \sqsubseteq \langle d'_n \rangle$ iff $\forall n.\ d_n \sqsubseteq_{D_n} d'_n$, and $\langle d_n \rangle \sqcup \langle d'_n \rangle = \langle d_n \sqcup d'_n \rangle$.

What are the elements of D? There is a lowest element $\langle \bot, \bot, \bot, \ldots \rangle$ (call it x_0), and successive elements $x_1 = \langle \bot, \lfloor \bot \rfloor, \lfloor \bot \rfloor, \ldots \rangle$, $x_2 = \langle \bot, \lfloor \bot \rfloor, \lfloor \lfloor \bot \rfloor \rfloor, \lfloor \lfloor \bot \rfloor \rfloor, \ldots \rangle$, and so on. Finally, there is the supremum of all the other elements, $x_\infty = \langle \bot, \lfloor \bot \rfloor, \lfloor \lfloor \bot \rfloor \rfloor, \lfloor \lfloor \lfloor \bot \rfloor \rfloor \rfloor, \ldots \rangle$, corresponding to the diagonal in Figure 2. This last element makes the partial order complete.

It remains to show that there is an homomorphism between D and $D_\bot$. The isomorphism is as follows, clearly preserving the relationship among mapped elements:

$$
\begin{array}{rcl}
x_0 & \longleftrightarrow & \bot \\
x_1 & \longleftrightarrow & \lfloor x_0 \rfloor \\
x_2 & \longleftrightarrow & \lfloor x_1 \rfloor \\
 & \cdots & \\
x_\infty & \longleftrightarrow & \lfloor x_\infty \rfloor
\end{array}
$$

We can define the isomorphism more formally in terms of the continuous function $up : D_\bot \to D$, which represents lifting of the entire tuple as lifting on each of its elements:

$$
\begin{array}{rcl}
up(\lfloor \langle d_n \rangle_{n \in \mathbb{N}} \rfloor) & = & \langle p_n(\lfloor d_n \rfloor) \rangle_{n \in \mathbb{N}} \\
up(\bot) & = & x_0 = \langle \bot, \bot, \bot, \ldots \rangle
\end{array}
$$

The inverse function is $down : D \to D_\bot$:

$$
\begin{aligned}
down(\langle \bot, \bot, \bot, \ldots \rangle) &= \bot \\
down(\langle \bot, \lfloor d_0 \rfloor, \lfloor d_1 \rfloor, \lfloor d_2 \rfloor \rangle) &= \lfloor \langle d_0, d_1, d_2, \ldots \rangle \rfloor
\end{aligned}
$$

These functions are clearly inverses and homomorphisms.

5 A related example

Suppose we want to represent infinite lists of natural numbers. We might write the domain equation $D = (\mathbb{N} \times D)_\bot$. This would allow us to give a semantics to the result of the following code, an infinite list of prime numbers, assuming that pairs in our language are lazy:

```
letrec primes_from = λn:nat. if is_prime(n)
                                 then (n, primes_from(n+1))
                                 else primes_from(n+1)
in
  primes_from(2)
```

Using the domain equation above, we'd expect this code to return the result **(2, (3, (5, . . .)))**, with the denotation $\lfloor \langle 2, \lfloor \langle 3, \lfloor \langle 5, \ldots \rangle \rfloor \rangle \rfloor \rangle \rfloor$. To obtain this denotation, we define p_n and e_n as follows (note $m \in \mathbb{N}$):

$$
\begin{aligned}
e_n(\bot) &= \bot \\
e_n(\lfloor \langle m, d_{n-1} \rangle \rfloor) &= \lfloor \langle m, e_{n-1}(d_{n-1}) \rangle \rfloor \qquad \text{(where } n > 0\text{)} \\
p_n(\bot) &= p_0(\lfloor m, \bot \rfloor) = \bot \\
p_n(\lfloor \langle m, d_n \rangle \rfloor) &= \lfloor \langle m, p_{n-1}(d_n) \rangle \rfloor
\end{aligned}
$$

Therefore, the representation of the list of primes as commuting tuples is:

$$
\langle \bot, \lfloor \langle 2, \bot \rangle \rfloor, \lfloor \langle 2, \lfloor \langle 3, \bot \rangle \rfloor \rfloor, \lfloor \langle 2, \lfloor \langle 3, \lfloor \langle 5, \bot \rfloor \rangle \rfloor \rfloor, \ldots \rangle
$$

The functions *up* and *down* are defined similarly to the previous example:

$$
\begin{aligned}
up(\bot) &= \langle \bot \rangle_{n \in \mathbb{N}} \\
up(\lfloor \langle m, d_n \rangle \rfloor) &= \langle p_n(\lfloor \langle m, d_n \rangle \rfloor) \rangle \\
\\
down(\langle \bot \rangle_{n \in \mathbb{N}}) &= \bot \\
down(\langle \bot, \lfloor \langle m, d_0 \rangle \rfloor, \lfloor \langle m, d_1 \rangle \rfloor, \ldots \rangle) &= \lfloor \langle m, \langle d_0, d_1, \ldots \rangle \rangle \rfloor
\end{aligned}
$$

6 Scott's D_∞ construction

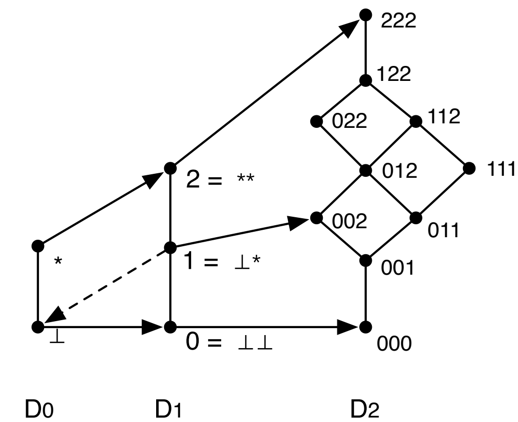
Scott showed that this general approach could be followed to obtain the first nontrivial solution to the equation $D = [D \to D]$, where $[D \to D]$ represents the set of all continuous functions from D to D. We start from some pointed domain D_0 containing at least two elements. For example, we could choose $D_0 = \{\bot, *\}$, with $\bot \sqsubseteq *$. Then apply $\mathcal{F}(D) = [D \to D]$ to obtain domains $D_1 = [D_0 \to D_0]$, $D_2 = [D_1 \to D_1]$, and so on. We define $e_n : D_n \to D_{n+1}$ and $p_n : D_{n+1} \to D_n$ inductively, as before:

$$
\begin{aligned}
e_0(d_0) &= \lambda y \in D_0 \,.\, d_0 \qquad \text{(where } d_0 \in D_0\text{)} \\
p_0(d_1) &= d_1(\bot_{D_0}) \qquad \text{(where } d_1 \in D_1\text{)} \\
\\
e_n(d_n) &= e_{n-1} \circ d_n \circ p_{n-1} \qquad \text{(where } d_n \in D_n,\ n > 0\text{)} \\
p_n(d_{n+1}) &= p_{n-1} \circ d_{n+1} \circ e_{n-1} \qquad \text{(where } d_{n+1} \in D_{n+1},\ n > 0\text{)}
\end{aligned}
$$

To understand the definition of e_n and p_n, it helps to consider the following diagram:

$$\begin{tikzcd}
{} & D_{n-1} \arrow[l] \arrow[d, "d_n"] & D_n \arrow[l, "p_{n-1}"'] \arrow[d, "d_{n+1}"] & D_{n+1} \arrow[l, "p_n"'] \arrow[d, "d_{n+2}"] & {} \arrow[l] \\
{} \arrow[r] & D_{n-1} \arrow[r, "e_{n-1}"'] & D_n \arrow[r, "e_n"'] & D_{n+1} \arrow[r] & {}
\end{tikzcd}$$

The first three domains constructed by this process (D_0, D_1, D_2) look like this:



The domains grow very rapidly after this point; D_3 contains 416416 elements, thought this is a small fraction of the 10^{10} elements of $D_2^{D_2}$!

Notice that in $D_1 = [D_0 \to D_0]$ there are only three possible elements. This is because the function $\{\perp \mapsto *, * \mapsto \perp\}$ (which would be represented in the figure as $*\perp$) is not monotonic (or continuous). This would be a function that terminates on a divergent argument and diverges on a value, which is clearly not computable. As we progress farther up the chain of domain approximations, more and more of the functions in $D_n \to D_n$ are not continuous, because they are not computable. This is why there is no cardinality paradox.

We define D_∞ as the projective limit of the D_n, as before, so an element of D_∞ is an infinite tuple of functions.

We define *down* : $D_\infty \to [D_\infty \to D_\infty]$ by mapping an element of $d \in D_\infty$ to a function f that works on each element of D_n. In other words, we need a way to treat a tuple of functions as a function that operates on tuples. Let $x = \langle x_n \rangle$ be an element of D_∞. We define $y = \langle y_m \rangle = f(x)$ by applying d_{n+1} to x_n for all n, then joining all the results and projecting them down to each y_m.

$$\begin{aligned}
y_0 &= d_1(x_0) \sqcup p_0(d_2(x_1)) \sqcup \cdots \sqcup (p_0 \circ p_1 \circ \cdots \circ p_n)(d_{n+2}(x_{n+1})) \sqcup \ldots \\
y_1 &= d_2(x_1) \sqcup p_1(d_3(x_2)) \sqcup \cdots \sqcup (p_1 \circ p_2 \circ \cdots \circ p_n)(d_{n+2}(x_{n+1})) \sqcup \ldots \\
&\ldots \\
y_m &= d_{m+1}(x_m) \sqcup p_m(d_{m+2}(x_{m+1})) \sqcup \cdots \sqcup (p_m \circ p_{m+1} \circ \cdots \circ p_{m+k})(d_{m+k+2}(x_{m+k+1})) \sqcup \ldots \\
&\ldots
\end{aligned}$$

Using *down*, we can define *up*, which constructs the tuple of approximations of $f \in D_\infty \to D_\infty$ at every D_n, by projecting the action of f down to D_n.

$$\begin{aligned}
up(f) &= \langle d_n \rangle \\
d_0 &= f(\perp_{D_0}) \\
d_{n+1} &= p_{\infty \to n} \circ f \circ e_{n \to \infty}
\end{aligned}$$

Here, $p_{\infty\to n}$ is a projection from D_∞ to D_n, and $e_{n\to\infty}$ is the inverse embedding, defined inductively on n as follows:

$$
\begin{array}{rcl}
e_{0\to\infty}(d_0) & = & \langle d_0, e_0(d_0), (e_1 \circ e_0)(d_0), \ldots\rangle \\
p_{\infty\to 0}(\langle d_n\rangle) & = & d_0
\end{array}
$$

$$
\begin{array}{rcl}
e_{n+1\to\infty}(d_{n+1}) & = & e_{n\to\infty} \circ d_{n+1} \circ p_{\infty\to n} \\
p_{\infty\to n+1}(d) & = & p_{\infty\to n} \circ \mathit{down}(d) \circ e_{n\to\infty}
\end{array}
$$

$$
\begin{tikzcd}
{} & D_{n-1} \arrow[l] & D_n \arrow[l, "p_{n-1}"'] & D_{n+1} \arrow[l, "p_n"'] & \cdots \arrow[l] & D_\infty \arrow[lll, bend right, "p_{\infty\to n}"'] \\
{} \arrow[r] & D_{n-1} \arrow[r, "e_{n-1}"'] & D_n \arrow[u, "d_{n+1}"] \arrow[r, "e_n"'] \arrow[rrr, bend right, "e_{n\to\infty}"'] & D_{n+1} \arrow[r] & \cdots & D_\infty \arrow[u, "down(d)"']
\end{tikzcd}
$$

7 Semantics of the untyped lambda calculus

With D_∞, we can give an extensional semantics for the untyped lambda calculus. It looks familiar except for the use of *up* and *down*. We have a naming environment $\rho \in \mathbf{Var} \to D_\infty$ and a semantic function such that $[\![e]\!]\rho \in D_\infty$:

$$
\begin{array}{rcl}
[\![x]\!]\rho & = & \rho(x) \\
[\![e_0\ e_1]\!]\rho & = & \mathit{down}([\![e_0]\!]\rho)\ [\![e_1]\!]\rho \\
[\![\lambda x.\, e]\!]\rho & = & \mathit{up}(\lambda y \in D_\infty\,.\, [\![e]\!]\rho[x \mapsto y])
\end{array}
$$

This semantics doesn't distinguish between nontermination and termination, which is a bit unsatisfactory. If we want to more faithfully model the CBV lambda calculus, we can use the domain equation $D \cong [D \to D_\perp]$ instead (for CBN, we'd use $D \cong [D_\perp \to D_\perp]$). The equations are solved similarly to $D \cong [D \to D]$. In the CBV case, we can start with $D_0 = \{*\}$ and modify the definitions for e_n and p_n as follows:

$$
\begin{array}{rcll}
e_0(*) & = & \lambda y \in D_0\,.\,\perp & \\
p_0(d_1) & = & \{*\} & \text{(where } d_1 \in D_1\text{)}
\end{array}
$$

$$
\begin{array}{rcll}
e_n(d_n) & = & e_{n-1}^{\perp} \circ d_n \circ p_{n-1} & \text{(where } d_n \in D_n,\ n > 0\text{)} \\
p_n(d_{n+1}) & = & p_{n-1}^{\perp} \circ d_{n+1} \circ e_{n-1} & \text{(where } d_{n+1} \in D_{n+1},\ n > 0\text{)}
\end{array}
$$

The first three approximations to the solution are shown in Figure 3.

The rest follows directly. The CBV semantics then have $[\![e]\!] : (\mathbf{Var} \rightharpoonup D) \to D_\perp$:

$$
\begin{array}{rcl}
[\![x]\!]\rho & = & \lfloor \rho(x) \rfloor \\
[\![e_0\ e_1]\!]\rho & = & \text{let } f \in D = [\![e_0]\!]\rho \text{ in let } v \in D = [\![e_1]\!]\rho \text{ in } \mathit{down}(f)(v) \\
[\![\lambda x.\, e]\!]\rho & = & \lfloor \mathit{up}(\lambda y \in D\,.\, [\![e]\!]\rho[x \mapsto y]) \rfloor
\end{array}
$$

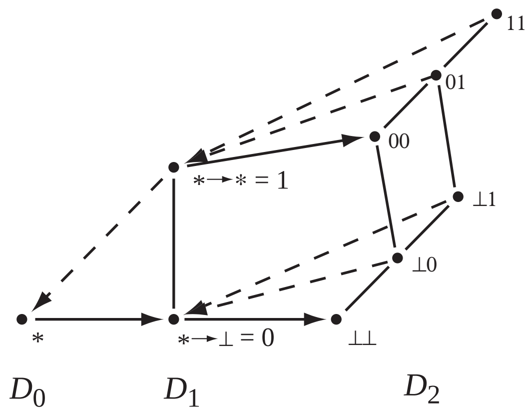


Figure 3: Approximations to a domain equation solution

8 Other equations

Can we find solutions to domain equations, in general? It turns out that a solution exists if we have a set of equations of the form $D_1 = \mathcal{F}_1(D_1, \ldots D_n), \ldots, D_n = \mathcal{F}_n(D_1, \ldots D_n)$, where each of the $\mathcal{F}_i$ is constructed using compositions of the following domain constructions: $D_\perp, D \times E, D + E, D \to E_\perp$. (This is a sufficient but not necessary condition). Winskel shows in Chapter 12 one way to build solutions using *information systems*. Thus, we can construct complex, recursive domain equations and be sure that we have a well-defined mathematical basis for denotational semantics.