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 fixpoint over D.

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

We are looking for an isomorphism witnessed by continuous bijections up and $down = up^{-1}$:

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

These maps, being continuous, must also be monotone:

$$d \sqsubseteq d' \;\Rightarrow\; up(d) \sqsubseteq up(d') \qquad\qquad d \sqsubseteq d' \;\Rightarrow\; down(d) \sqsubseteq down(d')$$

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 $\mathcal{F}$ repeatedly to obtain a sequence of domains D_0, $\mathcal{F}(D_0)$, $\mathcal{F}^2(D_0)$, $\mathcal{F}^3(D_0)$, $\ldots$, 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

We need a way to relate domains in the sequence. We will define a relation $D \precsim E$ on CPOs that says roughly that E extends D while preserving its structure. Our goal is to have a sequence of better and better approximations

$$D_0 \;\precsim\; \mathcal{F}(D_0) \;\precsim\; \mathcal{F}^2(D_0) \;\precsim\; \mathcal{F}^3(D_0) \;\precsim\; \cdots,$$

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

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. Also, as depicted in Fig. 1, they must agree in the following sense: for all elements $x \in D$ and $y \in E$,

$$p(e(x)) \;=\; x \qquad\qquad e(p(y)) \;\sqsubseteq\; y.$$

Equivalently,

$$p \circ e \;=\; \mathsf{id}_D \qquad\qquad e \circ p \;\sqsubseteq\; \mathsf{id}_E.$$

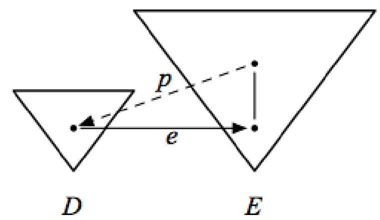


Figure 1: An embedding-projection pair

From the first equation it follows that e is injective (one-to-one) and p is surjective (onto). On elements of E of the form $e(x)$ for some $x \in D$, p acts as an inverse of e; and on elements in E not of that form, the projection function p maps them to an element of D whose corresponding E element is related. Together, these functions are called an *embedding-projection pair* (ep-pair). We write $D \precsim E$ when such an ep-pair exists.

If either D or E is pointed, then so is the other, and both e and p are strict. If $\bot_D$ exists, then for any $d \in E$, $\bot_D \sqsubseteq p(d)$, so by monotonicity, $e(\bot_D) \sqsubseteq e(p(d)) \sqsubseteq d$. As d was arbitrary, $\bot_E$ exists and equals $e(\bot_D)$. On the other hand, if $\bot_E$ exists, then $\bot_E \sqsubseteq d$ for every element $d \in E$. By monotonicity of p, $p(\bot_E) \sqsubseteq p(d)$. Since p is onto, $\bot_D$ exists and equals $p(\bot_E)$.

3 A Simple Domain Equation

For example, consider the domain equation $D \cong D_\bot$. This is essentially the domain equation for a lazy infinite stream of unit values. Assuming that the solution to the equation is a CPO (and it will be), we can use it to give meaning to expressions like $\mathsf{letrec}\ x.(\mathsf{null}, x)$, where we need to take a fixpoint over D.

Let $D_0 = \{\bot\}$ and let $D_{n+1} = (D_n)_\bot$ for $n \geq 0$. There is a simple way to define embedding-projection pairs $e_n : D_n \to D_{n+1}$ and $p_n : D_{n+1} \to D_n$ so that $D_n \precsim D_{n+1}$.

Recall that the map $\lfloor\cdot\rfloor : D \to D_\bot$ embeds D into $D_\bot$ by taking $d \in D$ to its copy $\lfloor d \rfloor \in D_\bot$ and adding a new bottom element $\bot$. Note that this cannot be e, since it is not strict.

We define e_n and p_n inductively in terms of $\lfloor\cdot\rfloor$:

$$\begin{aligned} e_n(\bot) &= \bot & p_n(\bot) &= p_0(\lfloor\bot\rfloor) = \bot \\ e_{n+1}(\lfloor d \rfloor) &= \lfloor e_n(d) \rfloor & p_{n+1}(\lfloor d \rfloor) &= \lfloor p_n(d) \rfloor \end{aligned}$$

The construction is illustrated in Fig. 2. In that figure, the solid left-to-right arrows represent e and the dashed right-to-left arrows represent p. Also, for every e arrow there is an implicit p arrow in the opposite direction. The vertical lines represent $\sqsubseteq$.

This may seem like a 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.

4 A Solution to the Domain Equation

We are now ready to define the solution domain. It is the *projective limit* (or *inverse limit*) $\lim_n D_n$ of the domains D_n: the set of infinite sequences $(d_n \mid n \geq 0) = (d_0, d_1, d_2, \ldots)$ such that for all $n \geq 0$, $d_n \in D_n$

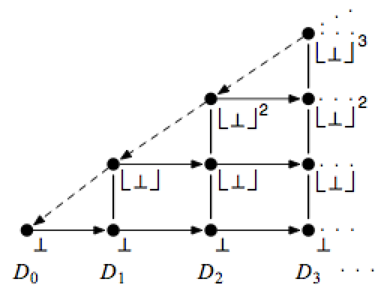


Figure 2: Successive approximations for $D \cong D_\perp$

and $d_n = p_n(d_{n+1})$, ordered componentwise. Given a component d_n of this tuple, 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 often write just (d_n) for $(d_n \mid n \geq 0)$.

The domain $\lim_n D_n$ forms a CPO under the componentwise order. To show it is complete, if A is a chain in $\lim_n D_n$, then for each index n, $\{\pi_n(d) \mid d \in A\}$ forms a chain in D_n, where π_n is the projection onto the nth component. Since D_n is complete, $\bigsqcup_{d \in A} \pi_n(d)$ exists, and it is easily shown that $\bigsqcup A = (\bigsqcup_{d \in A} \pi_n(d) \mid n \geq 0)$.

What are the elements of D? There is a least element $(\perp, \perp, \perp, \ldots)$ (call it x_0), and successive elements $x_1 = (\perp, \lfloor\perp\rfloor, \lfloor\perp\rfloor, \ldots)$, $x_2 = (\perp, \lfloor\perp\rfloor, \lfloor\lfloor\perp\rfloor\rfloor, \lfloor\lfloor\perp\rfloor\rfloor, \ldots)$, and so on. Finally, there is the supremum of all the other elements, $x_\infty = (\perp, \lfloor\perp\rfloor, \lfloor\lfloor\perp\rfloor\rfloor, \lfloor\lfloor\lfloor\perp\rfloor\rfloor\rfloor, \ldots)$, corresponding to the diagonal in Figure 2. This last element makes the partial order complete. Thus the domain is order-isomorphic to $\mathbb{N} \cup \{\infty\}$.

It remains to show that $\lim_n D_n$ is a solution to the domain equation $D \cong D_\perp$; that is, there is an isomorphism $up : D \to D_\perp$. Define

$$up(x_0) \triangleq \perp \qquad up(x_{n+1}) \triangleq \lfloor x_n \rfloor,\ n \geq 0 \qquad up(x_\infty) \triangleq \lfloor x_\infty \rfloor.$$

In other words,

$$up(\perp, \perp, \perp, \ldots) = \perp \qquad up(\perp, \lfloor d_0 \rfloor, \lfloor d_1 \rfloor, \lfloor d_2 \rfloor) = \lfloor (d_0, d_1, d_2, \ldots) \rfloor$$

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

$$down(\perp) = x_0 = (\perp, \perp, \perp, \ldots) \qquad down(\lfloor (d_n \mid n \geq 0) \rfloor) = (p_n(\lfloor d_n \rfloor) \mid n \geq 0).$$

5 A Related Example

Suppose we want to represent infinite lists of natural numbers. We might write the domain equation $D \cong (\mathbb{N} \times D)_\perp$. 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 would expect this code to return the infinite stream $(2, 3, 5, \ldots)$ (identifying $(a, (b, (c, \ldots)))$ with $(a, b, c, \ldots)$). To obtain this denotation, define

$$
\begin{aligned}
D_0 &= \{\bot\} & e_n(\bot) &= \bot & p_n(\bot) &= p_0(m,\bot) = \bot \\
D_{n+1} &= (\mathbb{N} \times D_n)_\bot & e_{n+1}(m,d) &= (m, e_n(d)) & p_{n+1}(m,d) &= (m, p_n(d))
\end{aligned}
$$

(we have omitted the lifting notation $\lfloor\cdot\rfloor$ in the definition of p_n and e_n for notational simplicity).

Then $D_{n+1} = \{(m,d) \mid d \in D_n\} \cup \{\bot\}$. One can prove inductively that D_n consists of all tuples of the form $(a_0, a_1, a_2, \ldots, a_k, \bot)$ for $k < n$. The functions e_n are identity functions and p_n takes $(a_0, a_1, \ldots, a_{m-1}, a_m, \bot)$ to itself if $m < n$ and to $(a_0, a_1, \ldots, a_{m-1}, \bot)$ if $m = n$.

The projective limit $\lim_n D_n$ consists of sequences that are constant for all but finitely many components, corresponding to elements of the D_n, and sequences of the form

$$(\bot, (a_0, \bot), (a_0, a_1, \bot), (a_0, a_1, a_2, \bot), \ldots)$$

whose nth component is of length n, corresponding to the infinite stream $(a_0, a_1, a_2, \ldots)$.

This example is more clearly understood by identifying the object $(a_0, a_1, a_2, \ldots, a_k, \bot)$ with the string $a_0 a_1 \cdots a_k \in \mathbb{N}^*$, where $\mathbb{N}^*$ denotes the set of finite length strings in $\mathbb{N}$. Under this correspondence, D_n consists of all strings in $\mathbb{N}^*$ of length at most n, and $\bot$ corresponds to the empty string. The projective limit consists of $\mathbb{N}^* \cup \mathbb{N}^\omega$, where $\mathbb{N}^\omega$ consists of all infinite-length strings $a_0 a_1 a_2 \cdots$. The ordering is $x \sqsubseteq y$ if x is a prefix of y.

Under this correspondence, it is easy to see how $\mathbb{N}^* \cup \mathbb{N}^\omega$ is a solution to the domain equation; that is, $\mathbb{N}^* \cup \mathbb{N}^\omega$ and $\mathbb{N} \times (\mathbb{N}^* \cup \mathbb{N}^\omega) \cup \{\varepsilon\}$ are isomorphic. For $a \in \mathbb{N}$ and $x \in \mathbb{N}^* \cup \mathbb{N}^\omega$, define

$$
\begin{aligned}
up(\varepsilon) &= \varepsilon & down(\varepsilon) &= \varepsilon \\
up(ax) &= (a, x) & down(a, x) &= ax
\end{aligned}
$$

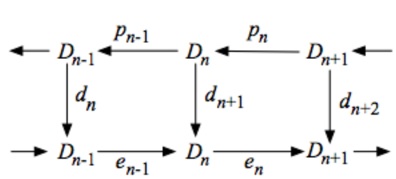
6 Scott's D_∞ Construction

Dana Scott showed that this general approach could be followed to obtain the first nontrivial solution to the equation $D \cong [D \to D]$, 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 $\bot \sqsubseteq *$. We 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. As before, we define $e_n : D_n \to D_{n+1}$ and $p_n : D_{n+1} \to D_n$ inductively:

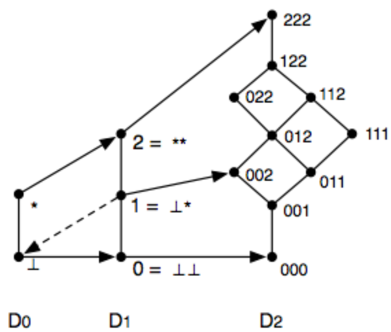
$$
\begin{aligned}
e_0(d_0) &= \lambda y \in D_0 \,.\, d_0 & p_0(d_1) &= d_1(\bot_{D_0}) \\
e_{n+1}(d_{n+1}) &= e_n \circ d_{n+1} \circ p_n & p_{n+1}(d_{n+2}) &= p_n \circ d_{n+2} \circ e_n
\end{aligned}
$$

where $d_n \in D_n$.

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



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 $D_2^{D_2}$!

Note that $D_1 = [D_0 \to D_0]$ contains only three elements. The function $\bot \mapsto *$, $* \mapsto \bot$ (which would be represented in the figure as $*\bot$) is not continuous; it is not even monotone. 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 the full function space $D_n \to D_n$ are not continuous.

As before, we define D_∞ as the projective limit $\lim_n D_n$. Thus an element of D_∞ is an infinite tuple of functions.

There is an embedding-projection pair $\widehat{e}_m, \widehat{p}_m$ between any D_m and $\lim_n D_n$. This is a general construction that holds for any inverse limit, not just D_∞. For any $d \in D_m$, define $\widehat{e}_m(d) = (d_0, d_1, d_2, \ldots)$ such that

$$d_n \;=\; \pi_n(\widehat{e}_m(d)) \;\triangleq\; \begin{cases} p_n(p_{n+1}(\cdots p_{m-1}(d)\cdots)) & \text{if } n \le m \\ e_{n-1}(e_{n-2}(\cdots e_m(d)\cdots)) & \text{if } n > m \end{cases}$$

and $\widehat{p}_m \triangleq \pi_m$. Using the properties of e_n and p_n, one can show the requisite properties of ep-pairs:

- $\widehat{e}_m : D_m \to D_\infty$ and $\widehat{p}_m : D_\infty \to D_m$ are continuous
- $\widehat{p}_m \circ \widehat{e}_m \;=\; \mathsf{id}_{D_m}$
- $\widehat{e}_m \circ \widehat{p}_m \;\sqsubseteq\; \mathsf{id}_{D_\infty}$.

Moreover, these maps commute with the e_n and p_n in the following sense:

- $\widehat{e}_{m+1} \circ e_m \;=\; \widehat{e}_m$
- $p_m \circ \widehat{p}_{m+1} \;=\; \widehat{p}_m$.

We define $up : D_\infty \to [D_\infty \to D_\infty]$ that maps an element $d \in D_\infty$ to a function $up(d) : [D_\infty \to D_\infty]$. The input to $up(d)$ is an infinite tuple $x = (x_n \mid n \ge 0)$ in which $x_n \in D_n$. Moreover, d itself is such a tuple, and we need a way to treat it as a function that operates on tuples. We define $y = (y_m \mid m \ge 0) = up(d)(x)$ by applying d_{n+1} to x_n for all n, then projecting all the results down to each y_m and joining them.

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

One must show that the elements to be joined form a chain in D_m.

Using *up*, we can define *down*, 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.

$$down(f) \;=\; (d_n \mid n \geq 0) \qquad\qquad d_0 \;=\; f(\perp_{D_0}) \qquad\qquad d_{n+1} \;=\; \widehat{p}_n \circ f \circ \widehat{e}_n.$$

7 Semantics of the Untyped λ-Calculus

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

$$\begin{aligned} [\![x]\!]\rho &= \rho(x) \\ [\![e_0\, e_1]\!]\rho &= up([\![e_0]\!]\rho)\,([\![e_1]\!]\rho) \\ [\![\lambda x.\, e]\!]\rho &= down(\lambda v \in D_\infty.\, [\![e]\!]\rho[v/x]) \end{aligned}$$

This semantics does not distinguish between nontermination and termination, which is a bit unsatisfactory. If we want to model the CBV λ-calculus more faithfully, we can use the domain equation $D \cong [D \to D_\perp]$ instead. For CBN, we would use $D \cong [D_\perp \to D_\perp]$. The equations are solved similarly to $D \cong [D \to D]$.

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 [Win93, Chp. 12] shows 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.

References

[Win93] Glynn Winskel. *The Formal Semantics of Programming Languages*. MIT Press, 1993.