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 $()$ for the integers, the natural numbers, and the unit domain, respectively. The unit domain contains a single element. 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 $A_\perp$, which is A with a new element $\perp$ adjoined. The order is extended to put $\perp$ below all the previous elements. Note that $\perp$ is intended to be a new element, so we can actually iterate this operation. The associated operations are the natural embedding $\lfloor\cdot\rfloor : D \to D_\perp$ and the lifting operation $(\cdot)^\dagger : (D \to E_\perp) \to (D_\perp \to E_\perp)$ defined by

$$d^\dagger \triangleq \lambda x \in D_\perp . \begin{cases} d(x), & \text{if } x \neq \perp, \\ \perp, & \text{if } x = \perp. \end{cases}$$

Both these operations are continuous, and when $(\cdot)^\dagger$ is applied to a continuous function, the result is a continuous function. As a convenient syntactic sugar, we write *let* $x \in D = e_1$ *in* e_2 as a shorthand for $(\lambda x \in D . e_2)^\dagger\, e_1$. That is, let is *strict* in that if $e_1 = \perp$, the let is $\perp$ too.

1.1 Products

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

1.2 Sums

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

$$D + E \triangleq \{\iota_1(d) \mid d \in D\} \cup \{\iota_2(e) \mid e \in E\},$$

where ι_1 and ι_2 are any one-to-one functions with disjoint ranges; for example, we could take $\iota_1(x) = (1, x)$ and $\iota_2(x) = (2, x)$. We define $\iota_i(x) \sqsubseteq \iota_j(y)$ iff $i = j$ and $x \sqsubseteq y$. Any chain in $D + E$ must be completely

contained in $\{\iota_1(x) \mid x \in D\}$ or $\{\iota_2(x) \mid x \in E\}$, so $D + E$ is a CPO. The associated operations are the injections $\iota_1 : D \to D + E$ and $\iota_2 : E \to D + E$, 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 match construct:

$$f + g \triangleq \lambda x.\, \mathsf{match}\ x\ \mathsf{with}\ \iota_1(y) \to f(y) \mid \iota_2(y) \to g(y).$$

This is continuous if f and g are, and it is the unique function satisfying the equations $f = (f + g) \circ \iota_1$ and $g = (f + g) \circ \iota_2$. As with products and projections, the binary coproduct can be generalized to an arbitrary coproduct $\sum_{x \in X} D_x$ with associated injections $\iota_y : D_y \to \sum_{x \in X} D_x$.

1.3 Continuous Functions

We argued briefly in the last lecture that the space of continuous functions $[D \to E]$ from a CPO D to a CPO E under the pointwise order form a CPO. Let us look at this result a little more carefully.

Essentially, we need to show that if f_n is a chain in $[D \to E]$, then it has a supremum $\bigsqcup_n f_n \in [D \to E]$. There is an obvious candidate. For each $x \in D$, the set $f_n(x)$ forms a chain in E, because for all x, $f_n(x) \sqsubseteq f_{n+1}(x)$; and since E is a CPO, this chain has a supremum $\bigsqcup_n(f_n(x))$. Thus the function

$$f_\omega \triangleq \lambda x \in D.\, \bigsqcup_n (f_n(x))$$

seems a likely candidate. This function is clearly an upper bound of the chain f_n in the pointwise ordering, and for any other upper bound g, we have

$$\begin{aligned}
\forall n\ f_n \sqsubseteq g &\Rightarrow \forall n\ \forall x\ f_n(x) \sqsubseteq g(x) && \text{by definition of } \sqsubseteq \\
&\Rightarrow \forall x\ \forall n\ f_n(x) \sqsubseteq g(x) && \text{switching quantifiers} \\
&\Rightarrow \forall x\ \bigsqcup_n (f_n(x)) \sqsubseteq g(x) && \text{by definition of supremum} \\
&\Rightarrow \forall x\ f_\omega(x) \sqsubseteq g(x) && \text{by definition of } f_\omega \\
&\Rightarrow f_\omega \sqsubseteq g && \text{by definition of } \sqsubseteq,
\end{aligned}$$

thus f_ω is the supremum of the chain f_n among functions $D \to E$.

However, this is not quite enough. We know that $f_\omega : D \to E$, but we do not yet know that it is continuous. We must show that for any chain d_m in D, $f_\omega(\bigsqcup_m d_m) = \bigsqcup_m (f_\omega(d_m))$.

It would be nice to try to argue as follows:

$$\begin{aligned}
f_\omega(\bigsqcup_m d_m) &= \bigsqcup_n (f_n(\bigsqcup_m d_m)) && \text{by definition of } f_\omega \\
&= \bigsqcup_n \bigsqcup_m (f_n(d_m)) && \text{by the continuity of } f_n \\
&= \bigsqcup_m \bigsqcup_n (f_n(d_m)) && \text{by wishful thinking} \\
&= \bigsqcup_m f_\omega(d_m) && \text{by definition of } f_\omega.
\end{aligned}$$

Alas, the wishful thinking proves little. We need to show that joins ($\bigsqcup$) commute. This will follow from the following lemma.

Lemma 1. *If a_{nm} is a doubly-indexed collection of members of a partially ordered set such that*

(i) *for all* n, $\bigsqcup_m a_{nm}$ *exists,*

(ii) *for all* m, $\bigsqcup_n a_{nm}$ *exists, and*

(iii) $\bigsqcup_m \bigsqcup_n a_{nm}$ *exists,*

then $\bigsqcup_n \bigsqcup_m a_{nm}$ *exists and is equal to* $\bigsqcup_m \bigsqcup_n a_{nm}$.

Proof. Clearly $\bigsqcup_m \bigsqcup_n a_{nm}$ is an upper bound for all a_{nm}, therefore it is an upper bound for all $\bigsqcup_m a_{nm}$; and if b is any other upper bound for all $\bigsqcup_m a_{nm}$, then $a_{nm} \sqsubseteq b$ for all n, m, therefore $\bigsqcup_m \bigsqcup_n a_{nm} \sqsubseteq b$, so $\bigsqcup_m \bigsqcup_n a_{nm}$ is the least upper bound for all $\bigsqcup_m a_{nm}$; that is, $\bigsqcup_n \bigsqcup_m a_{nm} = \bigsqcup_m \bigsqcup_n a_{nm}$. □

To apply this lemma we let $a_{nm} = f_n(d_m)$. This justifies the "wishful thinking" step above. We have thus shown that $f_\omega = \bigsqcup_n f_n \in [D \to E]$.

1.4 Operations on Continuous Functions

There are several useful operations on continuous functions that we can use for domain constructions.

- *apply* : $[D \to E] \times D \to E$ that applies a given function to a given argument;
- *compose* : $[E \to F] \times [D \to E] \to [D \to F]$;
- *curry* : $[D \times E \to F] \to [D \to [E \to F]]$;
- *uncurry* : $[D \to [E \to F]] \to [D \times E \to F]$; and most importantly,
- *fix* : $[D \to D] \to D$, defined by $\lambda g \in [D \to D].\ \bigsqcup_n g^n(\bot)$, that takes a function and returns its least fixpoint. To apply *fix*, D must have a bottom element $\bot$.

All these functions are continuous. We argue that *compose* is continuous. The type of *compose* is

$$compose : [E \to F] \times [D \to E] \to [D \to F]$$

and we would like to know that

$$compose \in [[E \to F] \times [D \to E] \to [D \to F]].$$

We must show that

$$compose(\bigsqcup_{m,n}(f_m, g_n)) \;=\; \bigsqcup_{m,n} compose(f_m, g_n).$$

Since $\bigsqcup_{m,n}(f_m, g_n) = (\bigsqcup_m f_m, \bigsqcup_n g_n)$, this amounts to showing

$$(\bigsqcup_m f_m) \circ (\bigsqcup_n g_n) \;=\; \bigsqcup_{m,n}(f_m \circ g_n). \tag{1}$$

For all x,

$$\begin{aligned}
(\bigsqcup_m f_m)((\bigsqcup_n g_n)(x)) \;&=\; (\bigsqcup_m f_m)(\bigsqcup_n (g_n(x))) && \text{by definition of } \bigsqcup_n g_n \\
&= \bigsqcup_m (f_m(\bigsqcup_n (g_n(x)))) && \text{by definition of } \bigsqcup_m f_m \\
&= \bigsqcup_{m,n} (f_m(g_n(x))) && \text{by the continuity of } f_m.
\end{aligned}$$

This proves (1).

References

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