

1 Semantics of while

Let's return to the denotational semantics of the **while** loop. We previously defined the function

$$\begin{aligned} F & : (\Sigma \rightarrow \Sigma_{\perp}) \rightarrow (\Sigma \rightarrow \Sigma_{\perp}) \\ F & \triangleq \lambda w \in \Sigma \rightarrow \Sigma_{\perp}. \lambda \sigma \in \Sigma. \text{if } \mathcal{B}[[b]]\sigma \text{ then } (w)^*(\mathcal{C}[[c]]\sigma) \text{ else } \sigma. \end{aligned}$$

Any function $\Sigma \rightarrow \Sigma_{\perp}$ is continuous, since chains in the discrete space Σ contain at most one element, thus the space of functions $\Sigma \rightarrow \Sigma_{\perp}$ is the same as the space of continuous functions $[\Sigma \rightarrow \Sigma_{\perp}]$. Moreover, the lift $(w)^* : \Sigma_{\perp} \rightarrow \Sigma_{\perp}$ of any function $w : \Sigma \rightarrow \Sigma_{\perp}$ is continuous.

By previous arguments, the function space $[\Sigma \rightarrow \Sigma_{\perp}]$ is a pointed CPO, and F maps this space to itself. To obtain a least fixpoint by the Fixed-Point Theorem, we need to know that F is continuous.

Let's first check that it is monotonic. This will ensure that, when trying to check the definition of continuity, when C is a chain, $\{F(d) \mid d \in C\}$ is also a chain, so that $\bigsqcup_{d \in C} F(d)$ exists. Suppose $d \sqsubseteq d'$. We want to show that $F(d) \sqsubseteq F(d')$. But for all σ ,

$$\begin{aligned} F(d)(\sigma) & = \text{if } \mathcal{B}[[b]]\sigma \text{ then } (d)^*(\mathcal{C}[[c]]\sigma) \text{ else } \sigma \\ & \sqsubseteq \text{if } \mathcal{B}[[b]]\sigma \text{ then } (d')^*(\mathcal{C}[[c]]\sigma) \text{ else } \sigma \\ & = F(d')(\sigma). \end{aligned}$$

Here we have used the fact that the operator $(\cdot)^*$ is monotonic, which is easy to check.

Now let's check that F is continuous. Let C be an arbitrary chain. We want to show that $\bigsqcup_{d \in C} F(d) = F(\bigsqcup C)$. We have

$$\begin{aligned} \bigsqcup_{d \in C} F(d) & = \bigsqcup_{d \in C} \lambda \sigma. \text{if } \mathcal{B}[[b]]\sigma \text{ then } (d)^*(\mathcal{C}[[c]]\sigma) \text{ else } \sigma \\ & = \lambda \sigma. \bigsqcup_{d \in C} \text{if } \mathcal{B}[[b]]\sigma \text{ then } (d)^*(\mathcal{C}[[c]]\sigma) \text{ else } \sigma \\ & = \lambda \sigma. \text{if } \mathcal{B}[[b]]\sigma \text{ then } \bigsqcup_{d \in C} (d)^*(\mathcal{C}[[c]]\sigma) \text{ else } \sigma \\ & = \lambda \sigma. \text{if } \mathcal{B}[[b]]\sigma \text{ then } (\bigsqcup C)^*(\mathcal{C}[[c]]\sigma) \text{ else } \sigma = F(\bigsqcup C), \end{aligned}$$

since $\mathcal{B}[[b]]\sigma$ does not depend on d and since the lift operator $(\cdot)^*$ is continuous.

2 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*. Operations on discrete CPOs are automatically monotonic and continuous because any chain on a discrete CPO can only contain one element. This is why we can use operations like $+$ and $(\text{if } \cdot \text{ then } \cdot \text{ else } \cdot)$ in writing denotational semantics.

For any domain A , we can construct a new domain A_\perp , which is A adjoined with a new element \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 $[\cdot] : D \rightarrow D_\perp$ and the lifting operation $(\cdot)^* : (D \rightarrow E_\perp) \rightarrow (D_\perp \rightarrow E_\perp)$ defined by

$$\begin{aligned} (d)^*([\cdot]) &\triangleq d(x) \\ (d)^*(\perp) &\triangleq \perp \end{aligned}$$

Both these operations are continuous, and when $(\cdot)^*$ 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 $(\lambda x \in D. e_2)^* e_1$. That is, let is *strict* in that if $e_1 = \perp$, the let is \perp too.

2.1 Products

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 $f : C \rightarrow D$ and $g : C \rightarrow E$, then the function $\langle f, g \rangle : C \rightarrow D \times E$ 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 $\pi_y : \prod_{x \in X} D_x \rightarrow D_y$.

2.2 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

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

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

$$f + g \triangleq \lambda x. \text{case } x \text{ of } \text{in}_1(y) \rightarrow f(y) \mid \text{in}_2(y) \rightarrow g(y)$$

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

2.3 Continuous functions

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

It is not obvious that continuous functions themselves form a CPO. Essentially, we want to show that $\bigsqcup_n f_n$ is continuous (i.e. given chains f_n, d_m that $(\bigsqcup_n f_n)(\bigsqcup_m d_m) = \bigsqcup_m ((\bigsqcup_n f_n)d_m)$.)

The case with reversed indices is symmetric, so the lemma is proved. □

To complete the proof of the theorem, we let $e_{nm} = f_n(d_m)$, and note that for $n \geq n', m \geq m'$ we have $e_{nm} \geq e_{n'm'}$ by d_m and f_n being chains, and the f_n being monotonic, so we may apply the lemma. □

The operations on continuous functions are:

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

All these functions are continuous, which can be shown without too much effort.