Formal Programming Language Semantics note 12

Complete partial orders (CPOs)

In this note we re-examine some of the ideas in the previous two notes from a slightly more abstract perspective. In particular, we are going to concentrate on the ideas behind the denotational semantics of `while` commands. By isolating the aspects of the situation which really made the semantics work, we will arrive at the basic concepts of a subject known as *domain theory*, which provides a general mathematical framework for much of denotational semantics.

Complete partial orders. Let us write $\mathcal{D}$ for the set $\mathcal{D}^{\mathtt{Com}} = (\mathbb{S} \rightharpoonup \mathbb{S})$ from Note 9. Recall that we defined an element $h \in \mathcal{D}$ as the union of an increasing chain $h_0 \subseteq h_1 \subseteq \cdots$. We now ask: what are the essential features of $\mathcal{D}$ that made this construction work? One reasonable answer (though not the only one!) is given by the following definitions:

Definition 1 *A* partially ordered set*, or* poset*, is a set D equipped with a binary relation $\sqsubseteq$ (called an* order *relation) such that the following hold for all $x, y, z \in D$:*

- *$x \sqsubseteq x$ ($\sqsubseteq$ is* reflexive*);*
- *if $x \sqsubseteq y$ and $y \sqsubseteq z$ then $x \sqsubseteq z$ ($\sqsubseteq$ is* transitive*);*
- *if $x \sqsubseteq y$ and $y \sqsubseteq x$ then $x = y$ ($\sqsubseteq$ is* antisymmetric*).*

Definition 2 *A* complete partial order*, or* CPO*, is a poset with the following property: every sequence of elements $x_0 \sqsubseteq x_1 \sqsubseteq \cdots$ in D has a* limit *or* least upper bound *in D: that is, there is an element $x \in D$ (written as $\bigsqcup_i x_i$) such that*

- *$x_i \sqsubseteq x$ for all i (i.e. x is an upper bound for the sequence);*
- *if x' is any other upper bound for the x_i (i.e. $x_i \sqsubseteq x'$ for all i), then $x \sqsubseteq x'$.*

Posets arise ubiquitously in mathematics, and there are examples from everyday life: for example, the set of all people ordered by the relation "x is an ancestor of y" (where we count each person as an "ancestor" of him- or herself!) The notion of a CPO, however, is rather more special to domain theory.

It is easy to see that $\mathcal{D}$ equipped with the inclusion relation $\subseteq$ on partial functions is an example of a CPO: $\subseteq$ is clearly a partial ordering, and given any chain of partial functions $h_0 \subseteq h_1 \subseteq \cdots$, their least upper bound is simply their union $\bigcup h_i$. We used the existence of least upper bounds in $\mathcal{D}$ when defining the semantics of `while`.

Intuitively, we think of a CPO as a set of data values of some kind which can be partially ordered by *positive information content*: we interpret $x \sqsubseteq y$ as saying that any piece of positive information that can be obtained from x can also be obtained from y. For example, in the case of $\mathcal{D} = (\mathbb{S} \rightharpoonup \mathbb{S})$, every true statement of the form $d(\sigma) = \sigma'$ can be viewed as a piece of positive information about d — this corresponds to the idea that we can run a certain command in state σ and observe the resulting state σ'. Thus, $d \sqsubseteq d'$ means that every such true statement for d is also true for d'. Note that we do not consider statements of the form "$d(\sigma)$ is undefined" to be *positive* information, since we cannot in general observe them to be true: if the execution of a command in state σ does not terminate, all an observer can say at any stage is that it has not terminated *yet*.

Examples of CPOs. Usually, a particular CPO will be associated with some particular *type* of programs or data, so that it makes sense to compare values for information content. For example, the CPO $\mathcal{D}$ is associated with the class Com of commands in **IMP**. Another good example is the CPO $\mathcal{S}$ of finite and infinite sequences of integers, which we might use to model *streams* of data values along some channel. Here the relevant ordering $\sqsubseteq$ is the *prefix* ordering between sequences. [Exercise: satisfy yourself that $(\mathcal{S}, \sqsubseteq)$ is indeed a CPO.]

A couple more (much simpler) examples will be useful in what follows. Suppose we wish to model a language with a type `int` of integer expressions. If (as in **IMP**), the evaluation of integer expressions always terminates, we can simply use the set $\mathbb{Z}$, which can be regarded as a rather trivial CPO with the *discrete* ordering ($n \subseteq n$ for each n, and that's all). However, if non-terminating integer expressions are possible (as in the functional language we will consider in Note 12), we should use the CPO $\mathbb{Z}_\perp = \mathbb{Z} \sqcup \{\perp\}$, where $\perp$ ("bottom" or "undefined") represents non-termination. Clearly, the relevant information ordering on $\mathbb{Z}_\perp$ is given by $x \sqsubseteq y$ iff $x = y$ or $x = \perp$. [Exercise: draw pictures of the posets $\mathbb{Z}$ and $\mathbb{Z}_\perp$.] Similar remarks apply to boolean expressions, yielding a two-element CPO $\mathbb{T}$ and a three-element CPO $\mathbb{T}_\perp$. [Draw these.]

Note that any finite poset is automatically a CPO, since every chain $x_0 \sqsubseteq x_1 \sqsubseteq \cdots$ must eventually settle down to a constant value. In general though, most of the CPOs we encounter will contain strictly increasing chains $x_0 \sqsubset x_1 \sqsubset \cdots$ whose least upper bound is not itself one of the x_i. (Of course, our CPOs will also contain many non-strictly-increasing chains, e.g. *constant* chains of the form $x \sqsubseteq x \sqsubseteq \cdots$, whose least upper bound is (surprise!!) x.)

Monotone and continuous functions. Apart from the domain $\mathcal{D}$, the other ingredient in the definition of the semantics of `while` was the (implicit) operator $H : \mathcal{D} \to \mathcal{D}$ which took an "approximation" h_k to a "next approximation" $H(h_k) = h_{k+1}$. Specifically, H can be defined as follows:

$$H : d \mapsto \Lambda\sigma. \begin{cases} d(g(\sigma)) & \text{if } f(\sigma) = \textit{true} \\ \sigma & \text{if } f(\sigma) = \textit{false}. \end{cases}$$

where as in Note 9, f and g were the denotations of some boolean expression b and command c respectively. We may now ask: what are the essential properties of H that we were relying on? The following definition identifies two important properties of H, and of computable functions between CPOs generally.

Definition 3 *Let* $(D, \sqsubseteq_D)$ *and* $(E, \sqsubseteq_E)$ *be CPOs.*

(i) A function $f : D \to E$ *is* monotone *if it respects the ordering — that is, if for all* x, y *in* D *we have*

$$x \sqsubseteq_D y \implies f(x) \sqsubseteq_E f(y).$$

(ii) A monotone function $f : D \to E$ *is* continuous *if it preserves least upper bounds — that is, if for every chain* $x_0 \sqsubseteq_D x_1 \sqsubseteq_D \cdots$ *in* D *we have*

$$f(\bigsqcup_i x_i) \;=\; \bigsqcup_i f(x_i).$$

(Note that the right hand side here is defined since we have a chain $f(x_0) \sqsubseteq_E f(x_1) \sqsubseteq_E \cdots$*, since by assumption* f *is monotone.)*

Since we have defined continuity only for monotone functions, if we say a function f is continuous we may take it as read that f is also monotone.

For example, it is easy to check that the function H is monotone and continuous. For the sake of completeness we give a proof here — but it's BORING, so skip to the end of this paragraph if this fact is already obvious to you! To show H is monotone, suppose $d \sqsubseteq d' \in \mathcal{D}$; we wish to show that $H(d) \sqsubseteq H(d')$ (also in $\mathcal{D}$). For this, we need to show that for all $\sigma \in \mathbb{S}$, if $H(d)(\sigma)$ is defined then so is $H(d')(\sigma)$ and these are equal. There are two cases. If $f(\sigma) = $ *false* then $H(d)(\sigma) = H(d')(\sigma) = \sigma$. If $f(\sigma) = $ *true* then $H(d)(\sigma) = d(g(\sigma))$, and in particular the RHS here is defined; but $d \sqsubseteq d'$ so also $H(d')(\sigma) = d'(g(\sigma)) = d(g(\sigma)$. To show H is continuous, suppose $d_0 \sqsubseteq d_1 \sqsubseteq \cdots$ is a chain in $\mathcal{D}$, with $d = \bigsqcup_i d_i$. Since we already know H is monotone, we have $H(d_i) \sqsubseteq H(d)$ for each i, and so $\bigsqcup H(d_i) \sqsubseteq H(d)$ since $\bigsqcup H(d_i)$ is the *least* upper bound of the $H(d_i)$. We wish to show also that $H(d) \sqsubseteq \bigsqcup H(d_i)$ — that is, that for any $\sigma \in \mathbb{S}$, if $H(d)(\sigma)$ is defined then $H(d)(\sigma) = H(d_i)(\sigma)$ for some i. Again there are two cases. If $f(\sigma) = $ *false* then $H(d)(\sigma) = H(d_0)(\sigma) = \sigma$. If $f(\sigma) = $ *true* then in particular $d(g(\sigma)$ is defined; but $d = \bigsqcup d_i$ and so we must have $d(g(\sigma) = d_i(g(\sigma))$ for some i; thus $H(d)(\sigma) = H(d_i)(\sigma)$ for this i. We have now shown that $H(d) \sqsubseteq \bigsqcup H(d_i) \sqsubseteq H(d)$; it follows by antisymmetry that $H(d) = \bigsqcup H(d_i)$.

Computability implies monotonicity and continuity. In fact, the definitions of monotonicity and continuity reflect intrinsic properties that *any* computable operation must satisfy. For example, suppose we have a program P which reads data from an input channel and writes data to an output channel (it may perform read or write operations in any order). We may represent the behaviour of P by a function $f : \mathcal{S} \to \mathcal{S}$, where $\mathcal{S}$ is as defined on page 2. We may argue semi-formally that f is inevitably monotone. Suppose we have two input streams $s \sqsubseteq s' \in \mathcal{S}$; then clearly, any output produced by P with input s will also be produced by P with input s' (since the computations will proceed identically until, if ever, the former hangs up waiting for the next value from s), so we have $f(s) \sqsubseteq f(s')$.

A similar semi-formal argument shows that f is inevitably *continuous*. The key observation needed here is that any finite amount of output information can only depend on a finite amount of input information, since the output must be produced after some finite time, by which stage P has only had the opportunity to look at finitely many input values. [Good exercise: work through this semi-formal argument in detail, using this observation at the appropriate point.]

The fixed point property. We now come to the fundamental result about CPOs and continuous functions which makes them good for denotational semantics. We have already seen one example of this fact at work, in the construction of h as the least upper bound of $h_0, h_1, \ldots$, but the following theorem shows that the same idea works much more generally, and provides the key to modelling many iterative and recursive programming constructs.

Theorem 1 (Fixed point property) *Let $(D, \sqsubseteq)$ be any CPO with a least element $\perp$ (so that $\perp \sqsubseteq x$ for all $x \in D$), and let f be any continuous function from D to D. Then f has a* least fixed point x *— that is, $f(x) = x$, and if $f(x') = x'$ for some y then $x \sqsubseteq x'$. Moreover, x may be defined by $x = \bigsqcup_i f^i(\perp)$.*

PROOF Let us write x_i for $f^i(\perp)$. We first show that the elements x_i form a chain. Clearly $x_0 = \perp \sqsubseteq x_1$ because $\perp$ is the least element. And if $x_i \sqsubseteq x_{i+1}$ then $x_{i+1} = f(x_i) \sqsubseteq f(x_{i+1}) = x_{i+2}$, since f is monotone. Hence by induction $x_i \sqsubseteq x_{i+1}$ for all i. Thus, the x_i form a chain and so have a least upper bound $x = \bigsqcup_i x_i$.

To see that x is a fixed point of f, we use the continuity of f:

$$f(x) \;=\; f(\bigsqcup_i x_i) \;=\; \bigsqcup_i f(x_i) \;=\; \bigsqcup_i x_{i+1} \;=\; x.$$

The last step holds since the least upper bound of $x_1 \sqsubseteq x_2 \sqsubseteq \cdots$ is clearly the same as that of $x_0 \sqsubseteq x_1 \sqsubseteq x_2 \sqsubseteq \cdots$.

To see that x is the *least* fixed point of f, suppose x' is some other fixed point, i.e. $f(x') = x'$. Then clearly $x_0 \sqsubseteq x'$ since $\perp$ is the least element; and if $x_i \sqsubseteq x'$ then since f is monotone, $x_{i+1} = f(x_i) \sqsubseteq f(x') = x'$. So by induction, $x_i \sqsubseteq x'$ for all i; in other words, x' is an upper bound for the x_i. But this means $x \sqsubseteq x'$ since x was defined to be the *least* upper bound of the x_i. $\square$

As a first application, this shows that the element h constructed in Note 9 is a fixed point of H: that is, h does indeed satisfy the “recursive” equation given on page 3 of Note 9. This plugs a small gap in our earlier results.

More generally, the fixed point property gives us a way of making sense of “circular” definitions of this kind within the world of CPOs.