SEMANTIC DOMAINS

by

C. A. Gunter
Dept. of Computer and Information Sciences, Univ. Pennsylvania, Philadelphia, PA 19104, USA

and

D. S. Scott
Dept. of Computer Science, Carnegie-Mellon Univ., Pittsburgh PA 15213, USA

In: **Handbook of Theoretical Computer Science, Volume B,** edited by J. van Leeuwen, North Holland, 1990, pp. 633–674.

Contents

1 Introduction.

The theory of domains was established in order to have appropriate spaces on which to define semantic functions for the denotational approach to programming-language semantics. There were two needs: first, there had to be spaces of several different types available to mirror both the type distinctions in the languages and also to allow for different kinds of semantical constructs—especially in dealing with languages with side effects; and second, the theory had to account for computability properties of functions—if the theory was going to be realistic. The first need is complicated by the fact that types can be both compound (or made up from other types) and recursive (or self-referential), and that a high-level language of types and a suitable semantics of types is required to explain what is going on. The second need is complicated by these complications of the semantical definitions and the fact that it has to be checked that the level of abstraction reached still allows a precise definition of computability.

This degree of abstraction had only partly been served by the state of recursion theory in 1969 when the senior author of this report started working on denotational semantics in collaboration with Christopher Strachey. In order to fix some mathematical precision, he took over some definitions of recursion theorists such as Kleene, Nerode, Davis, and Platek and gave an approach to a simple type theory of higher-type functionals. It was only after giving an abstract characterization of the spaces obtained (through the construction of bases) that he realized that recursive definitions of types could be accommodated as well—and that the recursive definitions could incorporate function spaces as well. Though it was not the original intention to find semantics of the so-called untyped λ-calculus, such a semantics emerged along with many ways of interpreting a very large variety of languages.

A large number of people have made essential contributions to the subsequent developments, and they have shown in particular that domain theory is not one monolithic theory, but that there are several different kinds of constructions giving classes of domains appropriate for different mixtures of constructs. The story is, in fact, far from finished even today. In this report we will only be able to touch on a few of the possibilities, but we give pointers to the literature. Also, we have attempted to explain the foundations in an elementary way—avoiding heavy prerequisites (such as category theory) but still maintaining some level of abstraction—with the hope that such an introduction will aid the reader in going further into the theory.

The chapter is divided into seven sections. In the second section we introduce a simple class of ordered structures and discuss the idea of fixed points of continuous functions as meanings for recursive programs. In the third section we discuss computable functions and effective presentations. The fourth section defines some of the operators and functions which are used in semantic definitions and describes their distinguishing characteristics. A special collection of such operators called *powerdomains* are discussed in the fifth section. Closure problems with respect to the *convex* powerdomain motivate the introduction of the class of *bifinite* domains which we describe in the

sixth section. The seventh section deals with the important issue of obtaining fixed points for (certain) operators on *domains.* We illustrate the method by showing how to find domains D satisfying isomorphisms such as $D \cong D \times D \cong D \to D$ and $D \cong \mathsf{N} + (D \to D)$. (Such domains are models of the above-mentioned untyped λ-calculus.)

Many of the proofs for results presented below are sketched or omitted. With a few exceptions, the enthusiastic reader should be able to fill in proofs without great difficulty. For the exceptions we provide a warning and a pointer to the literature.

2 Recursive definitions of functions.

It is the essential purpose of the theory of domains to study classes of spaces which may be used to give semantics for recursive definitions. In this section we discuss spaces having certain kinds of limits in which a useful fixed point existence theorem holds. We will briefly indicate how this theorem can be used in semantic specification.

2.1 Cpo's and the Fixed Point Theorem.

A *partially ordered set* is a set D together with a binary relation $\sqsubseteq$ which is reflexive, anti-symmetric and transitive. We will usually write D for the pair $\langle D, \sqsubseteq \rangle$ and abbreviate the phrase "partially ordered set" with the term "poset". A subset $M \subseteq D$ is *directed* if, for every finite set $u \subseteq M$, there is an upper bound $x \in M$ for u. A poset D is *complete* (and hence a *cpo*) if every directed subset $M \subseteq D$ has a least upper bound $\bigsqcup M$ and there is a least element $\bot_D$ in D. When D is understood from context, the subscript on $\bot_D$ will usually be dropped.

It is not hard to see that *any* finite poset that has a least element is a cpo. The easiest such example is the one point poset I. Another easy example which will come up later is the poset O which has two distinct elements $\top$ and $\bot$ with $\bot \sqsubseteq \top$. The *truth value cpo* T is the poset which has three distinct points, $\bot, \mathsf{true}, \mathsf{false}$, where $\bot \sqsubseteq \mathsf{true}$ and $\bot \sqsubseteq \mathsf{false}$ (see Figure 1). To get an example of an infinite cpo, consider the set N of natural numbers with the discrete ordering (*i.e.* $n \sqsubseteq m$ if and only if $n = m$). To get a cpo, we need to add a "bottom" element to N. The result is a cpo $\mathsf{N}_\bot$ which is pictured in Figure 1. This is a rather simple example because it does not have any interesting directed subsets. Consider the ordinal ω; it is not a cpo because it has a directed subset (namely ω itself) which has no least upper bound. To get a cpo, one needs to add a top element to get the cpo $\omega^\top$ pictured in Figure 1. For a more subtle class of examples of cpo's, let $\mathcal{P}S$ be the set of (all) subsets of a set S. Ordered by ordinary set inclusion, $\mathcal{P}S$ forms a cpo whose least upper bound operation is just set inclusion. As a last example, consider the set Q of rational numbers with their usual ordering. Of course, Q lacks the bottom and top elements, but there is another problem which causes Q to fail to be a cpo: Q lacks, for example, the square root of 2! However, the unit interval $[0, 1]$ of real numbers *does* form a cpo.

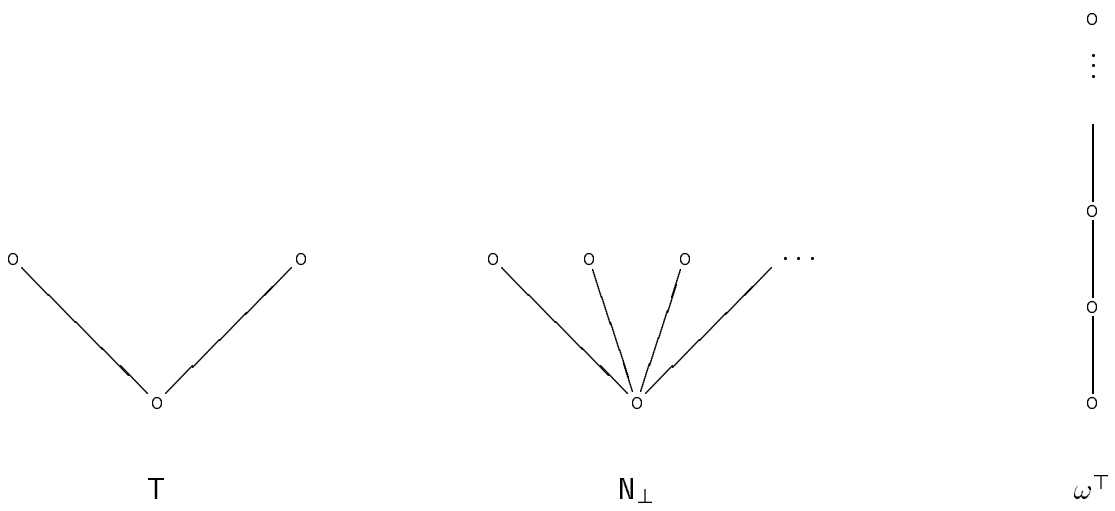


Figure 1: Examples of cpo's.

Given cpo's D and E, a function $f : D \to E$ is monotone if $f(x) \sqsubseteq f(y)$ whenever $x \sqsubseteq y$. If f is monotone and $f(\bigsqcup M) = \bigsqcup f(M)$ for every directed M, then f is said to be *continuous.* A function $f : D \to E$ is said to be *strict* if $f(\perp) = \perp$. We will usually write $f : D \multimap E$ to indicate that f is strict. If $f, g : D \to E$, then we say that $f \sqsubseteq g$ if and only if $f(x) \sqsubseteq g(x)$ for every $x \in D$. With this ordering, the poset of continuous functions $D \to E$ is itself a cpo. Similarly, the poset of strict continuous functions $D \multimap E$ is also a cpo. (Warning: we use the notation $f : D \to E$ to indicate that f is a function with domain D and codomain E in the usual set-theoretic sense. On the other hand, $f \in D \to E$ means that $f : D \to E$ is continuous. A similar convention applies to $D \multimap E$.)

To get a few examples of continuous functions, note that when $f : D \to E$ is monotone and D is finite, then f is continuous. In fact, this is true whenever D has no infinite ascending chains. For example, any monotone function $f : \mathsf{N}_\perp \to E$ is continuous. On the other hand, the function $f : \omega^\top \to \mathsf{O}$ which sends the elements of ω to $\perp$ and sends $\top$ to $\top$ is monotone, but it is not continuous. Given sets S, T and function $f : S \to T$ we define the *extension* of f to be the function $f^* : \mathcal{P}S \to \mathcal{P}T$ given by taking

$$f^*(X) = \{f(x) \mid x \in X\}$$

for each subset $X \subseteq S$. The function f^* is monotone and, for any collection X_i of subsets of S, we have

$$f^*(\bigcup_i X_i) = \bigcup_i f^*(X_i).$$

In particular, f^* is continuous. For readers who know a bit about functions on the real numbers, it is worth noting that a function $f : [0,1] \to [0,1]$ on the unit interval may be continuous in the cpo sense without being continuous in the usual sense.

Now, the central theorem may be stated as follows:

Theorem 1 (Fixed Point) *If D is a cpo and $f : D \to D$ is continuous, then there is a point $\mathsf{fix}(f) \in D$ such that $\mathsf{fix}(f) = f(\mathsf{fix}(f))$ and $\mathsf{fix}(f) \sqsubseteq x$ for any $x \in D$ such that $x = f(x)$. In other words, $\mathsf{fix}(f)$ is the least fixed point of f.*

Proof: Note that $\bot \sqsubseteq f(\bot)$. By an induction on n using the monotonicity of f, it is easy to see that $f^n(\bot) \sqsubseteq f^{n+1}(\bot)$ for every n. Set $\mathsf{fix}(f) = \bigsqcup_n f^n(\bot)$. By the continuity of f, it is easy to see that $\mathsf{fix}(f)$ is a fixed point of f. To see that it is the least such, note that if x is a fixed point of f, then, for each n, $f^n(\bot) \sqsubseteq f^n(x) = x$. ∎

2.2 Some applications of the Fixed Point Theorem.

The factorial function. As a first illustration of the use of the Fixed Point Theorem, let us consider how one might define the *factorial function* $\mathsf{fact} : \mathsf{N}_\bot \to \mathsf{N}_\bot$. The usual approach is to say that the factorial function is a strict function which satisfies the following recursive equation for each number n:

$$\mathsf{fact}(n) = \begin{cases} 1 & \text{if } n = 0 \\ n * \mathsf{fact}(n \dot{-} 1) & \text{if } n > 0. \end{cases}$$

where $*, \dot{-} : \mathsf{N} \times \mathsf{N} \to \mathsf{N}$ are multiplication and subtraction respectively. But how do we know that there *is* a function fact which satisfies this equation? Define a function

$$F : (\mathsf{N}_\bot \multimap \mathsf{N}_\bot) \to (\mathsf{N}_\bot \multimap \mathsf{N}_\bot)$$

by setting:

$$F(f)(n) = \begin{cases} 1 & \text{if } n = 0 \\ n * f(n \dot{-} 1) & \text{if } n > 0 \\ \bot & \text{if } n = \bot \end{cases}$$

for each $f : \mathsf{N}_\bot \multimap \mathsf{N}_\bot$. The definition of F is *not* recursive (F appears only on the left side of the equation) so F certainly exists. Moreover, it is easy to check that F is continuous (but not strict). Hence, by the Fixed Point Theorem, F has a least fixed point $\mathsf{fix}(F)$ and this solution will satisfy the equation for fact.

Context Free Grammars. One familiar kind of recursion equation is a context free grammar. Let Σ be an alphabet. One uses context free grammars to specify subsets of the collection Σ^* of finite sequences of letters from Σ.[1] Here are some easy examples:

[1] The superscripted asterisk will be used in three entirely different ways in this chapter. Unfortunately, all of these usages are standard. Fortunately, however, it is usually easy to tell which meaning is correct from context.

1. $$E \ ::= \ \epsilon \mid Ea$$
   defines the strings of a's (including the empty string ϵ).

2. $$E \ ::= \ a \mid bEb$$
   defines strings consisting either of the letter a alone or a string of n b's followed by an a followed by n more b's.

3. $$E \ ::= \ \epsilon \mid aa \mid EE$$
   defines strings of a's of even length.

We may use the Fixed Point Theorem to provide a precise explanation of the semantics of these grammars. Since the operations $X \mapsto \{\epsilon\} \cup X\{a\}$, $X \mapsto \{a\} \cup \{b\}X\{b\}$, and $X \mapsto \{\epsilon\} \cup \{a\}\{a\} \cup XX$ are all continuous in the variable X, it follows from the Fixed Point Theorem that equations such as

1. $X = \{\epsilon\} \cup X\{a\}$
2. $X = \{a\} \cup \{b\}X\{b\}$
3. $X = \{\epsilon\} \cup \{a\}\{a\} \cup XX$

corresponding to the three grammars mentioned above all have least solutions. These solutions are the languages defined by the grammars.

The Schroder-Bernstein Theorem. As a set-theoretic application of the Fixed Point Theorem we offer the proof of the following:

Theorem 2 (Schroder-Bernstein) *Let S and T be sets. If $f : S \to T$ and $g : T \to S$ are injections, then there is a bijection $h : S \to T$.*

Proof: The function $Y \mapsto (T \perp f^*(S)) \cup f^*(g^*(Y))$ from $\mathcal{P}T$ to $\mathcal{P}T$ is easily seen to be continuous with respect to the inclusion ordering. Hence, by the Fixed Point Theorem, there is a subset

$$Y = (T \perp f^*(S)) \cup f^*(g^*(Y)).$$

In particular, $T \perp Y = f^*(S \perp g^*(Y))$ since

$$\begin{aligned} T \perp Y &= T \perp ((T \perp f^*(S)) \cup f^*(g^*(Y))) \\ &= (T \perp (T \perp f^*(S))) \cap (T \perp (f^*(g^*(Y)))) \\ &= f^*(S) \cap (T \perp (f^*(g^*(Y)))) \\ &= f^*(S \perp g^*(Y)) \end{aligned}$$

Now define $h : S \to T$ by

$$h(x) = \begin{cases} y & \text{if } x = g(y) \text{ for some } y \in Y \\ f(x) & \text{otherwise} \end{cases}$$

This makes sense because g is an injection. Moreover, h itself is an injection since f and g are injections. To see that it is a surjection, suppose $y \in T$. If $y \in Y$, then $h(g(y)) = y$. If $y \notin Y$, then $y \in f^*(S \perp g^*(Y))$, so $y = f(x) = h(x)$ for some x. Thus h is a bijection. ∎

2.3 Uniformity.

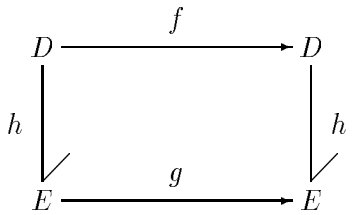
The question naturally arises as to why we take the *least* fixed point in order to get the meaning. In most instances there will be other choices. There are several answers to this question. First of all, it seems intuitively reasonable to take the least defined function satisfying a given recursive equation. But more importantly, taking the least fixed point yields a *canonical* solution. Indeed, it is possible to show that, given a cpo D, the function $\mathsf{fix}_D : (D \to D) \to D$ given by $\mathsf{fix}_D(f) = \bigsqcup_n f^n(\perp)$ is actually *continuous.* But are there other operators like fix that could be used? A definition is helpful:

Definition: A *fixed point operator* F is a class of continuous functions

$$F_D : (D \to D) \to D$$

such that, for each cpo D and continuous function $f : D \to D$, we have $F_D(f) = f(F_D(f))$. ∎

Let us say that a fixed point operator F is *uniform* if, for any pair of continuous functions $f : D \to D$ and $g : E \to E$ and strict continuous function $h : D \multimap E$ which makes the following diagram commute



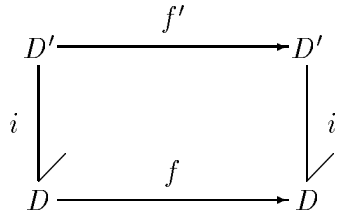
we have $h(F_D(f)) = F_E(g)$. We leave it to the reader to show that fix is a uniform fixed point operator. What is less obvious, and somewhat more surprising, is the following:

Theorem 3 fix *is the* unique *uniform fixed point operator.*

Proof: To see why this must be the case, let D be a cpo and suppose $f : D \to D$ is continuous. Then the set

$$D' = \{x \in D \mid x \sqsubseteq \mathsf{fix}(f)\}$$

is a cpo under the order that it inherits from the order on D. In particular, the restriction f' of f to D' has $\mathsf{fix}_D(f)$ as its *unique* fixed point. Now, if $i : D' \to D$ is the inclusion map then the following diagram commutes



Thus, if F is a uniform fixed point operator, we must have $F_D(f) = F_{D'}(f')$. But $F_{D'}(f')$ is a fixed point of f' and must therefore be equal to $\mathsf{fix}_D(f)$. ∎

We hope that these results go some distance toward convincing the reader that fix is a reasonable operator to use for the semantics of recursively defined functions.

3 Effectively presented domains.

There is a significant problem with the full class of cpo's as far as the theory of computation goes. There does not seem to be any reasonable way to define a general notion of *computable function* between cpo's. It is easy to see that these ideas make perfectly good sense for a noteworthy collection of examples. Consider a strict function $f : \mathsf{N}_\perp \multimap \mathsf{N}_\perp$. If we take $f(n) = \perp$ to mean that f is *undefined* at n, then f can be viewed as a partial function on N. We wish to have a concept of computability for functions on (some class of) cpo's so that f is computable just in case it corresponds to the usual notion of a partial recursive function. But we must also have a definition that applies to *functionals*, that is, functions which may take functions as arguments or return functions as values. We already encountered a functional earlier when we defined the factorial. To illustrate the point that there is a concept of computability that applies to such operators, consider, for example, a functional $F : (\mathsf{N}_\perp \multimap \mathsf{N}_\perp) \multimap \mathsf{N}_\perp$ which takes a function $f : \mathsf{N}_\perp \multimap \mathsf{N}_\perp$ and computes the value of f on the number 3. The functional F is continuous and it is *intuitively* computable. This intuition comes from the fact that, to compute $F(f)$ on an argument one needs only know how to compute f on an argument.

Our goal is to define a class of cpo's for which a notion of "finite approximation" makes sense. Let D be a cpo. An element $x \in D$ is *compact* if, whenever M is a directed subset of D and $x \sqsubseteq \bigsqcup M$, there is a point $y \in M$ such that $x \sqsubseteq y$. We let $K(D)$ denote the set of compact elements of D. The cpo D is said to be *algebraic* if, for every $x \in D$, the set $M = \{x_0 \in K(D) \mid x_0 \sqsubseteq x\}$ is directed and $\bigsqcup M = x$. In other words, in an algebraic cpo, each element is a directed limit of its "finite" (compact) approximations. If D is algebraic and $K(D)$ is countable, then we will say that D is a *domain.*

With the exception of the unit interval of real numbers, all of the cpo's we have mentioned so far are domains. The compact elements of the domain $\mathsf{N}_\perp \multimap \mathsf{N}_\perp$ are the functions with finite domain of definition, *i.e.* those continuous functions $f : \mathsf{N}_\perp \multimap \mathsf{N}_\perp$ such that $\{n \mid f(n) \neq \perp\}$ is finite. As another example, the collection $\mathcal{P}\mathsf{N}$ of subsets of N, ordered by subset inclusion is a domain whose compact elements are just the finite subsets of N.

One thing which makes domains particularly nice to work with is the way one may describe a continuous function $f : D \to E$ between domains D and E using the compact elements. Let G_f be the set of pairs (x_0, y_0) such that $x_0 \in K(D)$ and $y_0 \in K(E)$ and $y_0 \sqsubseteq f(x_0)$. If $x \in D$, then one may recover from G_f the value of f on x as

$$f(x) = \bigsqcup\{y_0 \mid (x_0, y_0) \in G_f \text{ and } x_0 \sqsubseteq x\}.$$

This allows us to characterize, for example, a continuous function $f : \mathcal{P}\mathsf{N} \to \mathcal{P}\mathsf{N}$ between *uncountable* cpo's with a *countable* set G_f. The significance of this fact for the theory of computability is not hard to see; we will say that the function f is computable just in case G_f is computable (in a sense to be made precise below).

3.1 Normal subposets and projections.

Before we give the formal definition of computability for domains and continuous functions, we digress briefly to introduce a useful relation on subposets. Given a poset $\langle A, \sqsubseteq\rangle$ and $x \in A$, let $\downarrow x = \{y \in A \mid y \sqsubseteq x\}$.

Definition: Let A be a poset and suppose $N \subseteq A$. Then N is said to be *normal* in A (and we write $N \lhd A$) if, for every $x \in A$, the set $N \cap \downarrow x$ is directed. ∎

The following lemma lists some useful properties of the relation $\lhd$.

Lemma 4 *Let C be a poset with a least element and suppose A and B are subsets of C.*

1. *If $A \lhd B \lhd C$ then $A \lhd C$.*
2. *If $A \subseteq B \subseteq C$ and $A \lhd C$ then $A \lhd B$.*
3. *If $A \lhd C$, then $\perp \in A$.*
4. *$\langle \mathcal{P}(C), \lhd\rangle$ is a cpo with $\{\perp\}$ as its least element.* ∎

Intuitively, a normal subposet $N \lhd A$ is an "approximation" to A. The notion of normal subposet is closely related to one of the central concepts in the theory of domains. A pair of continuous

functions $g : D \to E$ and $f : E \to D$ is said to be an *embedding-projection* pair (g is the embedding and f is the projection) if they satisfy the following

$$f \circ g = \mathsf{id}_D$$
$$g \circ f \sqsubseteq \mathsf{id}_E$$

where id_D and id_E are the identity functions on D and E respectively (in future, we drop the subscripts when D and E are clear from context) and composition of functions is defined by $(f \circ g)(x) = f(g(x))$. One can show that each of f and g uniquely determines the other. Hence it makes sense to refer to f as the projection *determined by* g and refer to g as the embedding *determined by* f. There is quite a lot to be said about properties of projections and embeddings and we cannot begin to provide, in the space of this chapter, the full discussion that these concepts deserve (the reader may consult Chapter 0 of [GHK+80] for this). However, a few observations will be essential to what follows. We first provide a simple example:

Example: If $f : D \to E$ is a continuous function then there is a strict continuous function $\mathsf{strict} : (D \to E) \to (D \multimap E)$ given by:

$$\mathsf{strict}(f)(x) = \begin{cases} f(x) & \text{if } x \neq \bot \\ \bot & \text{if } x = \bot \end{cases}$$

The function strict is a projection whose corresponding embedding is the inclusion map $\mathsf{incl} : (D \multimap E) \hookrightarrow (D \to E)$. ∎

In our discussion below we will not try to make much of the distinction between $f : D \multimap E$ and $\mathsf{incl}(f) : D \to E$ (for example, we may write $\mathsf{id} : D \multimap D$ as well as $\mathsf{id} : D \to D$ or even $\mathsf{incl}(\mathsf{id}) : D \to D$). From the two equations that define the relationship between a projection and embedding, it is easy to see that a projection is a surjection (*i.e.* onto) and an embedding is an injection (*i.e.* one-to-one). Thus one may well think of the image of an embedding $g : D \to E$ as a special kind of sub-cpo of E. We shall be especially interested in the case where an embedding is an *inclusion* as in the case of $D \multimap E$ and $D \to E$. Let D be a cpo. We say that a continuous function $p : D \to D$ is a *finitary projection* if $p \circ p = p \sqsubseteq \mathsf{id}$ and $\mathsf{im}(p) = \{p(x) \mid x \in D\}$ is a domain. Note, in particular, that the inclusion map from $\mathsf{im}(p)$ into D is an embedding (which has the corestriction of p to its image as the corresponding projection). It is possible to characterize the basis of $\mathsf{im}(p)$ as follows:

Lemma 5 *If D is a domain and $p : D \to D$ is a finitary projection, then the set of compact elements of $\mathsf{im}(p)$ is just $\mathsf{im}(p) \cap K(D)$. Moreover, $\mathsf{im}(p) \cap K(D) \triangleleft K(D)$.* ∎

Suppose, on the other hand, that $N \triangleleft K(D)$. Then it is easy to check that the function $p_N : D \to D$ given by

$$p_N(x) = \bigsqcup\{y \in N \mid y \sqsubseteq x\}$$

is a finitary projection. Indeed, the correspondence $N \mapsto p_N$ is inverse to the correspondence $p \mapsto \mathsf{im}(p) \cap K(D)$ and we have the following:

Theorem 6 *For any domain D there is an isomorphism between the cpo of normal substructures of $K(D)$ and the poset $\mathsf{Fp}(D)$ of finitary projections on D.* ∎

In particular, if $M \subseteq \mathsf{Fp}(D)$ is directed then $\mathsf{im}(\bigsqcup M)$ is a domain. This is a fact which will be significant later. Indeed, the notions of projection and normal subposet will come up again and again throughout the rest of our discussion.

3.2 Effectively presented domains.

Returning now to the topic of computability, we will say that a domain is effectively presented if the ordering on its basis is decidable and it is possible to effectively recognize the finite normal subposets of the basis:

Definition: Let D be a domain and suppose $d : \mathsf{N} \to K(D)$ is a surjection. Then d is an *effective presentation* of D if

1. the set $\{(m, n) \mid d_m \sqsubseteq d_n\}$ is effectively decidable, and
2. for any finite set $u \subseteq \mathsf{N}$, it is decidable whether $\{d_n \mid n \in u\} \triangleleft K(D)$.

If $\langle D, d\rangle$ and $\langle E, e\rangle$ are effectively presented domains, then a continuous function $f : D \to E$ is said to be *computable* (with respect to d and e) if and only if, for every $n \in \mathsf{N}$, the set $\{m \mid e_m \sqsubseteq f(d_n)\}$ is recursively enumerable. ∎

Unfortunately, the full class of domains has a serious problem. It is this: there are domains D, E such that the cpo $D \to E$ is *not* a domain (we will return to this topic in Section 6). Since we wish to use $D \to E$ in defining computability at higher types, we need some restriction on domains D and E which will insure that $D \to E$ is a domain. There are several restrictions which will work. We begin by presenting one which is relatively simple. Another will be discussed later.

Definition: A poset A is said to be *bounded complete* if A has a least element and every bounded subset of A has a least upper bound. ∎

The bounded complete domains are closely related to a more familiar class of cpo's which arise in many places in classical mathematics. A domain D is a (countably based) *algebraic lattice* if every subset of D has a least upper bound. It is not hard to see that a domain D is bounded complete if and only if the cpo $D^\top$ which results from adding a new top element to D is an algebraic lattice. The poset $\mathcal{P}\mathsf{N}$ is an example of an algebraic lattice. On the other hand, the bounded complete domain $\mathsf{N}_\perp \multimap \mathsf{N}_\perp$ lacks a top element and therefore fails to be an algebraic lattice. All of the domains we have discussed so far are bounded complete. In particular, we have the following:

Theorem 7 *If D and E are bounded complete domains, then $D \to E$ is also a bounded complete domain. Moreover, if D and E have effective presentations, then $D \to E$ has an effective presentation as well. Similar facts hold for $D \circ\!\!\to E$.*

Proof: (Sketch) It is not hard to see that $D \to E$ is a bounded complete cpo whenever E is. To prove that $D \to E$ is a domain we must demonstrate its basis. Suppose $N \lhd K(D)$ is finite and $s : N \to K(E)$ is monotone. Then the function $\mathsf{step}(s) : D \to E$ given by taking $\mathsf{step}(s)(x) = \bigsqcup\{f(y) \mid y \in N \cap {\downarrow} x\}$ is continuous and compact in the ordering on $D \to E$. These are called *step functions* and it is possible to show that they form a basis for $D \to E$. The proof that the poset of step functions has decidable ordering and finite normal subposets is tedious, but not difficult, using the effective presentations of D and E. The proof of these facts for $D \circ\!\!\to E$ is essentially the same since the strict step functions form a basis. ∎

In the remaining sections of the chapter we will discuss a great many operators like $\cdot \to \cdot$ and $\cdot \circ\!\!\to \cdot$. We will leave it to the reader to convince himself that all of these operators preserve the property of having an effective presentation. Further discussion of computability on domains may be found in [Smy77] and [KT84]. It is hoped that future research in the theory of domains will provide a general technique which will incorporate computability into the *logic* whereby we reason about the existence of our operators. This will eliminate the need to provide demonstrations of effective presentations. This is a central idea in current investigations but it is beyond our scope to discuss it further.

4 Operators and functions.

There are a host of operators on domains which are needed for the purposes of semantic definitions. In this section we mention a few of them. An essential technique for building new operators from those which we present here will be introduced below when we discuss solutions of recursive equations.

4.1 Products.

Given posets D and E, the *product* $D \times E$ is the set of pairs (x, y), where $x \in D$ and $y \in E$. The ordering is coordinatewise, *i.e.* $(x, y) \sqsubseteq (x', y')$ if and only if $x \sqsubseteq x'$ and $y \sqsubseteq y'$. We define functions $\mathsf{fst} : D \times E \to D$ and $\mathsf{snd} : D \times E \to E$ given by $\mathsf{fst}(x, y) = x$ and $\mathsf{snd}(x, y) = y$. If a subset $L \subseteq D \times E$ is directed, then

$$M = \mathsf{fst}^*(L) = \{x \mid \exists y \in E.\, (x, y) \in L\}$$
$$N = \mathsf{snd}^*(L) = \{y \mid \exists x \in D.\, (x, y) \in L\}$$

are directed. In particular, if D and E are cpo's, then $\bigsqcup L = (\bigsqcup M, \bigsqcup N)$ and, of course, $\bot_{D\times E} = (\bot_D, \bot_E)$, so $D \times E$ is a cpo. Indeed, if D and E are domains, then $D \times E$ is also a domain with $K(D \times E) = K(D) \times K(E)$. The property of bounded completeness is also preserved by $\times$.

Given cpos D, E, F, one can show that a function $f : D \times E \to F$ is continuous if and only if it is continuous in each of its arguments individually. In other words, f is continuous iff each of the following conditions holds:

1. The function $f_1 : D \to F$ given by $x \mapsto f(x, e)$ is continuous.
2. The function $f_2 : E \to F$ given by $y \mapsto f(d, y)$ is continuous.

We leave the proof of this equivalence as an exercise for the reader.

It is easy to see that each of the functions fst and snd is continuous. Moreover, given any cpo F and continuous functions $f : F \to D$ and $g : F \to E$, there is a continuous function $\langle f, g\rangle : F \to D \times E$ such that

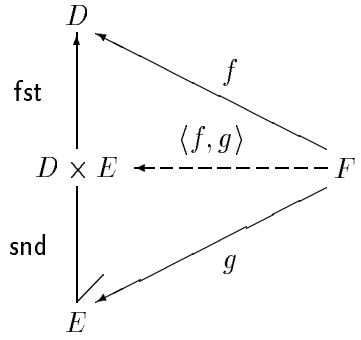
$$\mathsf{fst} \circ \langle f, g\rangle = f$$
$$\mathsf{snd} \circ \langle f, g\rangle = g$$

and, for any continuous function $h : F \to D \times E$,

$$\langle \mathsf{fst} \circ h, \mathsf{snd} \circ h\rangle = h.$$

The function $\langle f, g\rangle$ is given by $\langle f, g\rangle(x) = (f(x), g(x))$.

There is another, more pictorial, way of stating these equational properties of the operator $\langle \cdot, \cdot \rangle$ using a commutative diagram. The desired property can be stated in the following manner: given any cpo F and continuous functions $f : F \to D$ and $g : F \to E$, there is a *unique* continuous function $\langle f, g\rangle$ which completes the following diagram:



This is referred to as the *universal property* of the operator $\times$. As operators are given below we will describe the universal properties that they satisfy and these will form the basis of a system of equational reasoning about continuous functions. Virtually all of the functions needed to describe the semantics of (a wide variety of) programming languages may be built from those which are used in expressing these universal properties!

Given continuous functions $f : D \to D'$ and $g : E \to E'$, we may define a continuous function $f \times g$ which takes (x, y) to $(f(x), g(y))$ by setting

$$f \times g = \langle f \circ \mathsf{fst}, g \circ \mathsf{snd} \rangle : D \times E \to D' \times E'.$$

It is easy to show that $\mathsf{id}_D \times \mathsf{id}_E = \mathsf{id}_{D \times E}$ and

$$(f \times g) \circ (f' \times g') = (f \circ f') \times (g \circ g').$$

Note that we have "overloaded" the symbol $\times$ so that it works both on pairs of *domains* and pairs of *functions.* This sort of overloading is quite common in mathematics and we will use it often below. In this case (and others to follow) we have an example of what mathematicians call a *functor.*

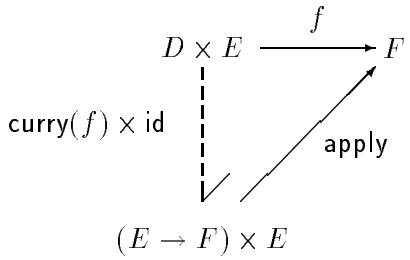
There is a very important relationship between the operators $\to$ and $\times$. Let D, E and F be cpo's. Then there is a function

$$\mathsf{apply} : ((E \to F) \times E) \to F$$

given by taking $\mathsf{apply}(f, x)$ to be $f(x)$ for any function $f : E \to F$ and element $x \in E$. Indeed, the function apply is *continuous.* Also, given a function $f : D \times E \to F$, there is a continuous function

$$\mathsf{curry}(f) : D \to (E \to F)$$

given by taking $\mathsf{curry}(f)(x)(y)$ to be $f(x, y)$. Moreover, $\mathsf{curry}(f)$ is the *unique* continuous function which makes the following diagram commute:



This uniqueness condition is equivalent to the following equation:

$$\mathsf{curry}(\mathsf{apply} \circ (h \times \mathsf{id}_E)) = h \tag{1}$$

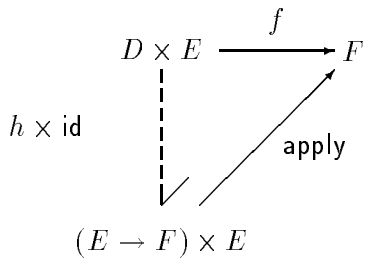
To see this, suppose equation (1) holds and h satisfies

$$f = \mathsf{apply} \circ (h \times \mathsf{id})$$

then

$$\mathsf{curry}(f) = \mathsf{curry}(\mathsf{apply} \circ (h \times \mathsf{id})) = h$$

so the uniqueness condition is satisfied. On the other hand, if $\mathsf{curry}(f)$ is uniquely determined by the diagram above, then equation (1) follows immediately from the commutativity of the following diagram:



for $f = \mathsf{apply} \circ (h \times \mathsf{id}_E)$.

It is often useful to have a multiary notation for products. We write

$$\begin{aligned} \times() &= \mathsf{I} \\ \times(D_1, \ldots, D_n) &= \times(D_1, \ldots, D_{n\perp 1}) \times D_n \end{aligned}$$

and define projections

$$\mathsf{on}_i : \times(D_1, \ldots, D_n) \to D_i$$

by

$$\mathsf{on}_i = \mathsf{snd} \circ \mathsf{fst}^{n\perp i}$$

Similarly, one defines a multiary version of the pairing operation by taking $\langle\,\rangle$ to be the identity on the one point domain and defining

$$\langle f_1, \ldots, f_n \rangle = \langle \langle f_1, \ldots, f_{n\perp 1} \rangle, f_n \rangle.$$

These multiary versions of projection and pairing satisfy a universal property similar to the one for the binary product.

4.2 Church's λ-notation.

If we wish to define a function from, say, natural numbers to natural numbers, we typically do so by describing the action of that function on a generic number x (a *variable*) using previously defined functions. For example, the squaring function f has the action $x \mapsto x * x$ where $*$ is the multiplication function. We may now use f to define other functions: for example, a function g which takes a function $h : \mathsf{N} \to \mathsf{N}$ to $f \circ h$. Continuing in this way we may construct increasingly complex function definitions. However, it is sometimes useful to have a notation for functions which alleviates the necessity of introducing intermediate names. This purpose is served by a terminology known as λ-notation which is originally due to Church.

The idea is this. Instead of introducing a term such as f and describing its action as a function, one simply gives the function a name which is basically a description of what it does with its argument. In the above case one writes $\lambda x.\ x * x$ for f and $\lambda h.\ f \circ h$ for g. One can use this notation to define g without introducing f by defining g to be the function $\lambda h.\ (\lambda x.\ x * x) \circ h$. The

λh at the beginning of this expression says that g is a function which is computed by taking its argument and *substituting* it for the variable h in the expression $(\lambda x.\ x * x) \circ h$.

The use of the Greek letter λ for the operator which binds variables is primarily an historical accident. Various programming languages incorporate something essentially equivalent to λ-notation using other names. In mathematics textbooks it is common to avoid the use of such notation by assuming conventions about variable names. For example, one may write

$$x^2 \perp 2 * x$$

for the function which takes a real number as an argument and produces as result the square of that number less its double. An expression such as

$$x^2 + x * y + y^2$$

would denote a function which takes two numbers as arguments—that is, the values of x and y—and produces the square of the one number plus the square of the other plus the product of the two. One might therefore provide a name for this function by writing something like:

$$f(x, y) = x^2 + x * y + y^2.$$

So f is a function which takes a pair of numbers and produces a number. But what notation should we use for the function g that takes a number n as argument and produces the *function* $n \mapsto x^2 + x * n + n^2$? For example, $g(2)$ is the function $x^2 + 2 * x + 4$. It is not hard to see that this is closely related to the function curry which we discussed above. Modulo the fact that we defined curry for domains above, we might have written $g = \mathsf{curry}(f)$. Or, to define g directly, we would write

$$g = \lambda y.\ \lambda x.\ x^2 + x * y + y^2.$$

The definition of f would need to be given differently since f takes a *pair* as an argument. We therefore write:

$$f = \lambda(x, y).\ x^2 + x * y + y^2.$$

There is no impediment to using this notation to describe higher-order functions as well. For example,

$$\lambda f.\ f(3)$$

takes a function f and evaluates it on the number 3 and

$$\lambda f.\ f \circ f$$

takes a function and composes it with itself. But these definitions highlight a very critical issue. Note that both definitions are *ambiguous* as they stand. Does the function $\lambda f.\ f(3)$ take, for example, functions from numbers to reals as argument or does it take a function from numbers to

sets of numbers as argument? Either of these would, by itself, make sense. What we need to do is indicate somewhere in the expression the *types* of the variables (and constants if their types are not already understood). So we might write

$$\lambda f : \mathsf{N} \to \mathsf{R}.\ f(3)$$

for the operator taking a real valued function as argument and

$$\lambda f : \mathsf{N} \to \mathcal{P}\mathsf{N}.\ f(3)$$

for the operator taking a $\mathcal{P}\mathsf{N}$ valued function.

So far, what we have said applies to almost any class of spaces and functions where products and an operator like curry are defined. But for the purposes of programming semantics, we need a semantic theory that includes the concept of a fixed point. Such fixed points are guaranteed if we stay within the realm of cpo's and continuous functions. But the crucial fact is this: *the process of λ-abstraction preserves continuity.* This is because $\mathsf{curry}(f)$ is continuous whenever f is. We may therefore use the notational tools we have described above with complete freedom and still be sure that recursive definitions using this notation make sense.

Demonstrating that the typed λ-calculus (*i.e.* the system of notations that we have been describing informally here) is really useful in explaining the semantics of programming languages is not the objective of this chapter. However, one can already see that it provides a considerable latitude for writing function definitions in a simple and mathematically perspicuous manner.

4.3 Smash products.

In the product $D \times E$ of cpo's D and E, there are elements of the form $(x, \perp)$ and $(\perp, y)$. If $x \neq \perp$ or $y \neq \perp$, then these will be *distinct* members of $D \times E$. In programming semantics, there are occasions when it is desirable to *identify* the pairs $(x, \perp)$ and $(\perp, y)$. For this purpose, there is a collapsed version of the product called the *smash product.* For cpo's D and E, the smash product $D \otimes E$ is the set

$$\{(x, y) \in D \times E \mid x \neq \perp \text{ and } y \neq \perp\} \cup \{\perp_{D \otimes E}\}$$

where $\perp_{D \otimes E}$ is some new element which is not a pair. The ordering on pairs is coordinatewise and we stipulate that $\perp_{D \otimes E} \sqsubseteq z$ for every $z \in D \otimes E$. There is a continuous surjection

$$\mathsf{smash} : D \times E \to D \otimes E$$

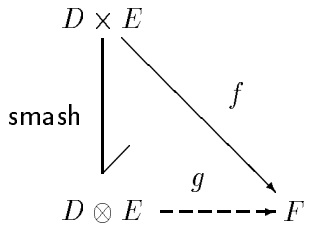
given by taking

$$\mathsf{smash}(x, y) = \begin{cases} (x, y) & x \neq \perp \text{ and } x \neq \perp \\ \perp_{D \otimes E} & \text{otherwise} \end{cases}$$

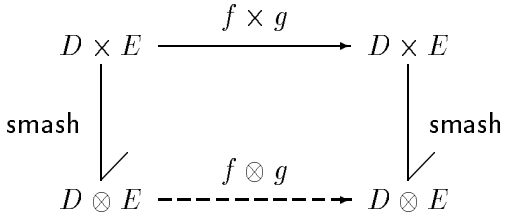
This function establishes a useful relationship between $D \times E$ and $D \otimes E$. In fact, it is a projection whose corresponding embedding is the function $\mathsf{unsmash} : D \otimes E \to D \times E$ given by

$$\mathsf{unsmash}(z) = \begin{cases} z & \text{if } z = (x, y) \text{ is a pair} \\ (\bot, \bot) & \text{if } z = \bot_{D \otimes E} \end{cases}$$

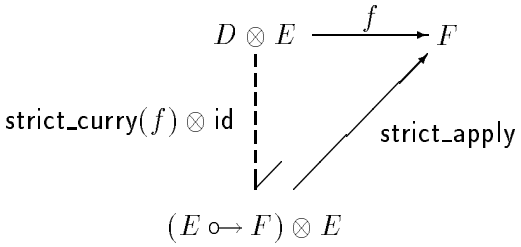
Let us say that a function $f : D \times E \to F$ is *bistrict* if $f(x, y) = \bot$ whenever $x = \bot$ or $y = \bot$. If $f : D \times E \to F$ is bistrict and continuous, then $g = f \circ \mathsf{unsmash}$ is the unique strict, continuous function which completes the following diagram:



If $f : D \to D'$ and $g : E \to E'$ are strict continuous functions, then $f \otimes g = \mathsf{smash} \circ (f \times g) \circ \mathsf{unsmash}$ is the unique strict, continuous function which completes the following diagram:



As with the product $\times$ and function space $\to$, there is a relationship between the smash product $\otimes$ and the strict function space $\multimap$. In particular, there is a strict continuous function $\mathsf{strict\_apply}$ such that for any strict function f, there is a unique strict function $\mathsf{strict\_curry}$ such that the following diagram commutes:



4.4 Sums and lifts.

Given cpo's D and E, we define the *coalesced sum* $D \oplus E$ to be the set

$$\Big((D \perp \{\bot_D\}) \times \{0\}\Big) \cup \Big((E \perp \{\bot_E\}) \times \{1\}\Big) \cup \{\bot_{D \oplus E}\}$$

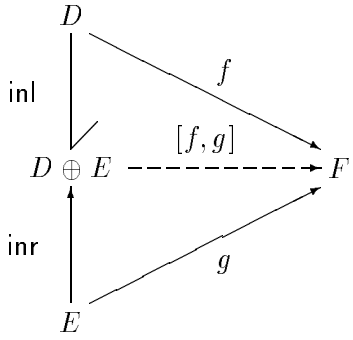
where $D \perp \{\perp_D\}$ and $E \perp \{\perp_E\}$ are the sets D and E with their respective bottom elements removed and $\perp_{D\oplus E}$ is a new element which is not a pair. It is ordered by taking $\perp_{D\oplus E} \sqsubseteq z$ for all $z \in D \oplus E$ and taking $(x, m) \sqsubseteq (y, n)$ if and only if $m = n$ and $x \sqsubseteq y$. There are strict continuous functions $\mathsf{inl} : D \circ\!\!\to (D \oplus E)$ and $\mathsf{inr} : E \circ\!\!\to (D \oplus E)$ given by taking

$$\mathsf{inl}(x) = \begin{cases} (x, 0) & \text{if } x \neq \perp \\ \perp_{D\oplus E} & \text{if } x = \perp \end{cases}$$

and

$$\mathsf{inr}(x) = \begin{cases} (x, 1) & \text{if } x \neq \perp \\ \perp_{D\oplus E} & \text{if } x = \perp \end{cases}$$

Moreover, if $f : D \circ\!\!\to F$ and $g : E \circ\!\!\to F$ are strict continuous functions, then there is a unique strict continuous function $[f, g]$ which completes the following diagram:



The function $[f, g]$ is given by

$$[f, g](z) = \begin{cases} f(x) & \text{if } z = (x, 0) \\ g(y) & \text{if } z = (y, 1) \\ \perp & \text{if } z = \perp. \end{cases}$$

Given continuous functions $f : D \circ\!\!\to D'$ and $g : E \circ\!\!\to E'$, we define

$$f \oplus g = [\mathsf{inl} \circ f, \mathsf{inr} \circ g] : D \oplus E \circ\!\!\to D' \oplus E'.$$

As with the product, it is useful to have a multiary notation for the coalesced sum. We define

$$\begin{aligned} \oplus() &= \mathsf{I} \\ \oplus(D_1, \ldots, D_n) &= \oplus(D_1, \ldots, D_{n\perp 1}) \oplus D_n \end{aligned}$$

and

$$\mathsf{in}_i = \mathsf{inr} \circ \mathsf{inl}^{n\perp i}.$$

One may also define $[f_1, \ldots, f_n]$ and prove a universal property.

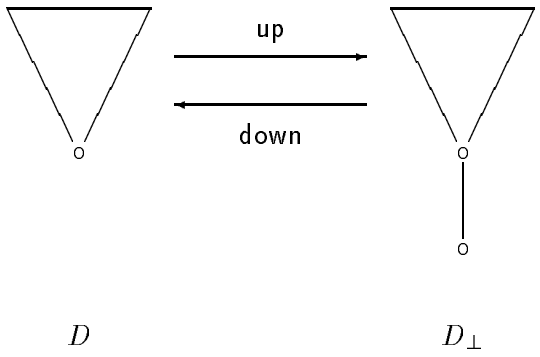


Figure 2: The lift of a cpo.

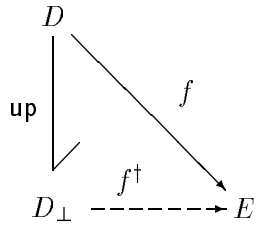
Given a cpo D, we define the *lift* of D to be the set $D_\perp = (D \times \{0\}) \cup \{\perp\}$, where $\perp$ is a new element which is not a pair, together with an partial ordering $\sqsubseteq$ which is given by stipulating that $(x,0) \sqsubseteq (y,0)$ when $x \sqsubseteq y$ and $\perp \sqsubseteq z$ for every $z \in D_\perp$. In short, $D_\perp$ is the poset obtained by adding a new bottom to D—see Figure 2. It is easy to show that $D_\perp$ is a cpo if D is. We define a strict continuous function $\mathsf{down} : D_\perp \circ\!\!\to D$ by

$$\mathsf{down}(z) = \begin{cases} x & \text{if } z = (x,0) \\ \perp_D & \text{otherwise} \end{cases}$$

and a (non-strict) continuous function $\mathsf{up} : D \to D_\perp$ given by $\mathsf{up} : x \mapsto (x,0)$. These functions are related by

$$\mathsf{down} \circ \mathsf{up} = \mathsf{id}_D$$
$$\mathsf{up} \circ \mathsf{down} \sqsupseteq \mathsf{id}_{D_\perp}$$

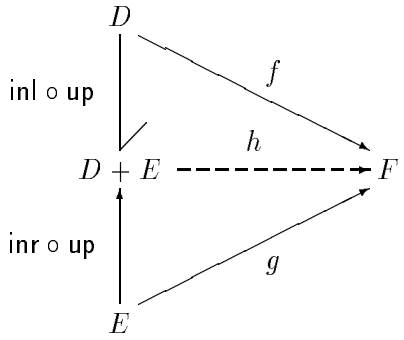
These inequations are reminiscent of those which we gave for embedding-projection pairs, but the second inequation has $\sqsupseteq$ rather than $\sqsubseteq$. We will discuss such pairs of functions later. Given cpo's D and E and continuous function $f : D \to E$, there is a unique strict continuous function $f^\dagger$ which completes the following diagram:



Given a continuous function $f : D \to E$, we define a strict continuous function

$$f_\perp = (\mathsf{up} \circ f)^\dagger : D_\perp \circ\!\!\to E_\perp.$$

Given cpo's D and E, we define the *separated sum* $D + E$ to be the cpo $D_\perp \oplus E_\perp$. By the universal properties for $\oplus$ and $(\cdot)_\perp$, we know that $h = [f^\dagger, g^\dagger]$ is the unique *strict* continuous function which completes the following diagram:



However, h may not be the only *continuous* function which completes the diagram. Given continuous functions $f : D \to D'$ and $g : E \to E'$, we define

$$f + g = f_\perp \oplus g_\perp : D + E \to D' + E'.$$

4.5 Isomorphisms and closure properties.

There are quite a few interesting relationships between the operators above which are implied by the definitions and commutative diagrams. We list a few of these in the following lemmas.

Lemma 8 *Let D, E and F be cpo's, then*

1. $D \times E \cong E \times D$,
2. $(D \times E) \times F \cong D \times (E \times F)$,
3. $D \to (E \times F) \cong (D \to E) \times (D \to F)$,
4. $D \to (E \to F) \cong (D \times E) \to F$. ∎

Lemma 9 *Let D, E and F be cpo's, then*

1. $D \otimes E \cong E \otimes D$,
2. $(D \otimes E) \otimes F \cong D \otimes (E \otimes F)$,
3. $(E \oplus F) \multimap D \cong (E \multimap D) \times (E \multimap F)$,
4. $D \multimap (E \multimap F) \cong (D \otimes E) \multimap F$,
5. $D \otimes (E \oplus F) \cong (D \otimes E) \oplus (D \otimes E)$

6. $D_\perp \circ\!\!\to E \cong D \to E$. ∎

We remarked already that $D \to E$ and $D \circ\!\!\to E$ are bounded complete domains whenever D and E are. It is not difficult to see that similar closure properties will hold for the other operators we have defined in this section:

Lemma 10 *If D and E are bounded complete domains then so are the cpo's $D \to E$, $D \circ\!\!\to E$, $D \times E$, $D \otimes E$, $D + E$, $D \oplus E$, $D_\perp$.* ∎

Further discussion of the operators defined in this section and others may be found in [Sco82a] and [Sco82b].

5 Powerdomains.

We now turn our attention to another collection of operators on domains. Just as we have defined a computable analog to the *function space,* we will now define a computable analog to the *powerset operation.* Actually, we will produce three such operators. In the domain theory literature these are called *powerdomains.* If D is a domain we write

- $D^\sharp$ for the *upper* powerdomain of D,
- $D^\natural$ for the *convex* powerdomain of D, and
- $D^\flat$ for the *lower* powerdomain of D.

The names we use for these operators come from the concepts of upper and lower semi-continuity and the interested reader can consult [Smy83b] for a detailed explanation. They commonly appear under other names as well. The convex powerdomain $D^\natural$ was introduced by Gordon Plotkin [Plo76] and is therefore sometimes referred to as the *Plotkin powerdomain.* The upper powerdomain $D^\sharp$ was introduced by Mike Smyth [Smy78] and is sometimes called the *Smyth powerdomain.* For reasons that we will discuss briefly below, this latter powerdomain corresponds to the *total correctness* interpretation of programs. Since Tony Hoare has done much to popularize the study of *partial correctness* properties of programs, the remaining powerdomain $D^\flat$—which corresponds to the partial correctness interpretation—sometimes bears his name.

5.1 Intuition.

There is a basic intuition underlying the powerdomain concept which can be explained through the concept of *partial information.* To keep things simple, let us assume that we are given a finite poset A and asked to form the *poset* of finite non-empty subsets of A. As a first guess, one might take the non-empty subsets and order them by subset inclusion. However, this operation ignores

the order structure on A! Think of A as a collection of partial descriptions of data elements: $x \sqsubseteq y$ just in case x is a partial description of y. What should it mean for one non-empty subset of A to be a "partial description" of another? The are at least three reasonable philosophies that one might adopt in attempting to answer this question.

Suppose, for example, that I hold a bag of fruit and I wish to give you information about what is in the bag. One such description might be

A fruit in the bag is a yellow fruit or a red fruit.

This description is based on two basic pieces of data: "is a yellow fruit" and "is a red fruit". These are used to *restrict* the kinds of fruit which are in the bag. A more informative description of this kind would provide further restrictions. Consider the following example:

A fruit in the bag is a yellow fruit or a cherry or a strawberry.

It is based on three pieces of data: "is a yellow fruit", "is a cherry" and "is a strawberry". Since these three data provide further restrictions on the contents of the bag (by ruling out the possibility of an apple, for example) it is a more informative statement about the bag's contents. On the other hand,

A fruit in the bag is a yellow fruit or a red fruit or a purple fruit.

is a less informative description because it is more permissive; for instance, it does not rule out the possibility that the bag holds a grape. Now suppose that u, v are subsets of the poset A from the previous paragraph. With this way of seeing things, we should say that u is below v if the restrictions imposed by v are refinements of the restrictions imposed by u: that is, for each $y \in v$, there is an $x \in u$ such that $x \sqsubseteq y$. This is the basic idea behind the *upper powerdomain* of A.

Returning to the bag of fruit analogy, we might view the following as a piece of information about the contents of the bag

There is some yellow fruit and some red fruit in the bag.

This information is based on two pieces of data: "is a yellow fruit" and "is a red fruit". However, these data are not being used as before. They do not restrict possibilities; instead they offer a *positive* assertion about the contents of the bag. A more informative description of this kind would provide a further enumeration and refinement of the contents:

There is a banana, a cherry and some purple fruit in the bag.

This refined description does not rule out the possibility that the bag holds a apple, but it does insure that there is an cherry. A statment such as

There is some yellow fruit in the bag.

is less informative since it does not mention the presence of red fruit. Now suppose that u, v are subsets of the poset A. With this way of seeing things, we should say that u is below v if the positive assertions provided by u are extended and refined by v: that is, for each $x \in u$, there is a $y \in v$ such that $x \sqsubseteq y$. This is the basic idea behind the *lower powerdomain* of A.

Now, the *convex powerdomain* combines these two forms of information. For example, the assertion

> *If you pull a fruit from the bag, then it must be yellow or a cherry, and you can pull a yellow fruit from the bag and you can pull a cherry from the bag.*

is this combined kind of information. The pair of assertions means that the bag holds some yellow fruit and at least one cherry, but nothing else. A more refined description might be

> *If you pull a fruit from the bag, then it must be a banana or a cherry, and you can pull a banana from the bag and you can pull a cherry from the bag.*

A less refined description might be

> *If you pull a fruit from the bag, then it must be yellow or red, and you can pull a yellow fruit from the bag and you can pull a red fruit from the bag.*

The reader may be curious about what bags of fruit have to do with programming semantics. The powerdomains are used to model non-deterministic computations where one wishes to speak about the set of outcomes of a computation. How one wishes to describe such outcomes will determine which of the three powerdomains is used. We will attempt to illustrate this idea later in this section—when we have given some formal definitions.

5.2 Formal definitions.

In order to give the definitions of the powerdomains, it is helpful to have a little information about the representation of domains using the concept of a pre-order:

Definition: A *pre-order* is a set A together with a binary relation $\vdash$ which is reflexive and transitive.

It is conventional to think of the relation $a \vdash b$ as indicating that a is "larger" than b (as in mathematical logic, where $\phi \vdash \psi$ means that the formula ψ follows from the hypothesis ϕ). Of course, any poset is also a pre-order. On the other hand, a pre-order may fail to be a poset by not satisfying the anti-symmetry axiom. In other words, we may have $x \vdash y$ and $y \vdash x$ but $x \neq y$. By identifying elements x, y which satisfy $x \vdash y$ and $y \vdash x$, we obtain an induced partially ordered set from a pre-order (and this why they are called *pre*-orders). We shall be particularly interested in a special kind of subset of a pre-order:

Definition: An *ideal* over a pre-order $\langle A, \vdash \rangle$ is a subset $s \subseteq A$ such that

1. if $u \subseteq s$ is finite, then there is an $x \in s$ such that $x \vdash y$ for each $y \in u$, and
2. if $x \in s$ and $x \vdash y$, then $y \in s$. ∎

In short, an ideal is a subset which is directed and downward closed. If $x \in A$ for a pre-order A, then the set

$$\downarrow x = \{y \in A \mid x \vdash y\}$$

is an ideal called the *principal ideal generated by* x. To induce a poset from a pre-order, one can take the poset of principal ideals under set inclusion. The poset of all ideals on a pre-order is somewhat more interesting:

Theorem 11 *Given a countable pre-order* $\langle A, \vdash \rangle$*, let* D *be the poset consisting of the ideals over* A*, ordered by set inclusion. If there is an element* $\perp \in A$ *such that* $x \vdash \perp$ *for each* $x \in A$*, then* D *is a domain and* $K(D)$ *is the set of principal ideals over* A*.*

Proof: Clearly, the ideals of A form a poset under set inclusion and the principal ideal $\downarrow \perp$ is the least element. To see that this poset is complete, suppose that $M \subseteq D$ and let $x = \bigcup M$. If we can show that x is an ideal, then it is certainly the least upper bound of M in D. To this end, suppose $u \subseteq x$ is finite. Since each element of u must be contained in some element of M, there is a finite collection of ideals $s \subseteq M$ such that $u \subseteq \bigcup s$. Since M is directed, there is an element $y \in M$ such that $z \subseteq y$ for each $z \in s$. Thus $u \subseteq y$ and since y is ideal, there is an element $a \in y$ such that $b \sqsubseteq a$ for each $b \in u$. But $a \in y \subseteq x$, so it follows that x is an ideal.

To see that D is a domain, we show that the set of principal ideals is a basis. Suppose $M \subseteq D$ is directed and $\downarrow a \subseteq \bigcup M$ for some $a \in A$. Then $a \in x$ for some $x \in M$, so $\downarrow a \subseteq x$. Hence $\downarrow a$ is compact in D. Now suppose $x \in D$ and $u \subseteq A$ is a finite collection of elements of A such that $\downarrow a \subseteq x$ for each $a \in u$. Then $u \subseteq x$ and since x is an ideal, there is an element $b \in x$ with $b \vdash a$ for each $a \in u$. Thus $\downarrow a \subseteq \downarrow b$ for each $a \in u$ and it follows that the principal ideals below x form a directed collection. It is obvious that the least upper bound (*i.e.* union) of that collection is x. Since x was arbitrary, it follows that D is an algebraic cpo with principal ideals of A as its basis. Since A is countable, there are only countably many principal ideals, so D is a domain. ∎

For any set S, we let $\mathcal{P}_f^*(S)$ be the set of finite non-empty subsets of S. We write $\mathcal{P}_f(S)$ for the set of all finite subsets (including the empty set). Given a poset $\langle A, \sqsubseteq \rangle$, define a pre-ordering $\vdash^\sharp$ on $\mathcal{P}_f^*(A)$ as follows,

$$u \vdash^\sharp v \text{ if and only if } (\forall x \in u)(\exists y \in v).\ x \sqsupseteq y.$$

Dually, define a pre-ordering $\vdash^\flat$ on $\mathcal{P}_f^*(A)$ by

$$u \vdash^\flat v \text{ if and only if } (\forall y \in v)(\exists x \in u).\ x \sqsupseteq y.$$

And define $\vdash^\natural$ on $\mathcal{P}^*_f(A)$ by

$$u \vdash^\natural v \text{ if and only if } u \vdash^\sharp v \text{ and } u \vdash^\flat v.$$

If D is a domain, then let $D^\natural$ be the domain of ideals over $\langle \mathcal{P}^*_f(K(D)), \vdash^\natural \rangle$. We call $D^\natural$ the *convex powerdomain* of D. Similarly, define $D^\sharp$ and $D^\flat$ to be the domains of ideals over $\langle \mathcal{P}^*_f(K(D)), \vdash^\sharp \rangle$ and $\langle \mathcal{P}^*_f(K(D)), \vdash^\flat \rangle$ respectively. We call $D^\sharp$ the *upper powerdomain* of D and $D^\flat$ the *lower powerdomain* of D.

As an example, we compute the lower powerdomain of $\mathsf{N}_\perp$. Since $K(\mathsf{N}_\perp) = \mathsf{N}_\perp$, the lower powerdomain of $\mathsf{N}_\perp$ is the set of ideals over the pre-order $\langle \mathcal{P}^*_f(\mathsf{N}_\perp), \vdash^\flat \rangle$. To see what such an ideal must look like, note first that $u \vdash^\flat u \cup \{\perp\}$ and $u \cup \{\perp\} \vdash^\flat u$ for any $u \in \mathcal{P}^*_f(\mathsf{N}_\perp)$. From this fact it is already possible to see why $\vdash^\flat$ is usually only a *pre-order* and not a poset. Now, if u and v both contain $\perp$, then $u \vdash^\flat v$ iff $u \supseteq v$. Hence we may identify an ideal $x \in (\mathsf{N}_\perp)^\flat$ with the union $\bigcup x$ of all the elements in x. Thus $(\mathsf{N}_\perp)^\flat$ is isomorphic to the domain $\mathcal{P}\mathsf{N}$ of all subsets of N under subset inclusion.

Now let us compute the upper powerdomain of $\mathsf{N}_\perp$. Note that if u and v are finite non-empty subsets of $\mathsf{N}_\perp$ and $\perp \in v$, then $u \vdash^\sharp v$. In particular, any ideal x in $(\mathsf{N}_\perp)^\sharp$ contains all of the finite subsets v of $\mathsf{N}_\perp$ with $\perp \in v$. So, let us say that a set $u \in \mathcal{P}^*_f(\mathsf{N}_\perp)$ is *non-trivial* if it does not contain $\perp$ and an ideal $x \in (\mathsf{N}_\perp)^\sharp$ is non-trivial if there is a non-trivial $u \in x$. Now, if u and v are non-trivial, then $u \vdash^\sharp v$ iff $u \subseteq v$. Therefore, if an ideal x is non-trivial, then it is the principal ideal generated by the intersection of its non-trivial elements! The smaller this set is, the larger is the ideal x. Hence, the non-trivial ideals in the powerdomain (ordered by subset inclusion) correspond to finite subsets of N (ordered by superset inclusion). If we now throw in the unique trivial ideal, we can see that $(\mathsf{N}_\perp)^\sharp$ is isomorphic to the domain of sets $\{\mathsf{N}\} \cup \mathcal{P}^*_f(\mathsf{N})$ ordered by superset inclusion.

Finally, let us look at the convex powerdomain of $\mathsf{N}_\perp$. If $u, v \in \mathcal{P}^*_f(\mathsf{N}_\perp)$, then $u \vdash^\natural v$ iff

1. $\perp \in v$ and $u \supseteq v$ or
2. $u = v$

Hence, if x is an ideal and there is a set $u \in x$ with $\perp \notin u$, then x is the principal ideal generated by u. No two distinct principal ideals like this will be comparable. On the other hand, if x is an ideal with $\perp \in u$ for each $u \in x$, then $x \subseteq y$ for an arbitrary ideal y iff $\bigcup x \subseteq \bigcup y$. Thus the convex powerdomain of $\mathsf{N}_\perp$ corresponds to the set of finite, non-empty subsets of N unioned with the set of arbitrary subsets of $\mathsf{N}_\perp$ that contain $\perp$. The ordering on these sets is like the pre-ordering $\vdash^\natural$ but extended to include infinite sets.

5.3 Universal and closure properties.

If $s, t \in D^\natural$ then we define a binary operation

$$s \uplus t = \{w \mid u \cup v \vdash^\natural w \text{ for some } u \in s \text{ and } v \in t\}.$$

This set is an ideal and the function $\uplus : D^\natural \times D^\natural \to D^\natural$ is continuous. Similar facts apply when $\uplus$ is defined in this way for $D^\sharp$ and $D^\flat$. Now, if $x \in D$, define

$$\{\!|x|\!\} = \{u \in \mathcal{P}^*_f(K(D)) \mid \{x_0\} \vdash^\natural u \text{ for some compact } x_0 \sqsubseteq x\}.$$

This forms an ideal and $\{\!|\cdot|\!\} : D \to D^\natural$ is a continuous function. When one replaces $\vdash^\natural$ in this definition by $\vdash^\sharp$ or $\vdash^\flat$, then similar facts apply. Strictly speaking, we should decorate the symbols $\uplus$ and $\{\!|\cdot|\!\}$ with indices to indicate their types, but this clutters the notation somewhat. Context will determine what is intended.

These three operators $(\cdot)^\sharp$, $(\cdot)^\flat$ and $(\cdot)^\natural$ may not seem to be the most obvious choices for the computable analog of the powerset operator. We will attempt to provide some motivation for choosing them in the remainder of this section. Given the operators $\uplus$ and $\{\!|\cdot|\!\}$, we may say that a point $x \in D$ for a domain D is an "element" of a set s in a powerdomain of D if $\{\!|x|\!\} \uplus s = s$. If s and t lie in a powerdomain of D, then s is a "subset" of t if $s \uplus t = t$. Care must be taken, however, not to confuse "sets" in a powerdomain with sets in the usual sense. The relations of "element" and "subset" described above will have different properties in the three different powerdomains. Moreover, it may be the case that s is a "subset" of t without it being the case that $s \subseteq t$!

To get some idea how the powerdomains are related to the semantics of non-deterministic programs, let us discuss non-deterministic partial functions from N to N. As we have noted before, there is a correspondence between partial functions from N to N and strict functions $f : \mathsf{N}_\perp \multimap \mathsf{N}_\perp$. These may be thought of as the meanings of "deterministic" programs, because the output of a program is uniquely determined by its input (*i.e.* the meaning is a partial *function*). Suppose, however, that we are dealing with programs which permit some *finite non-determinism* as discussed in the section on non-determinism in the chapter of Peter Mosses. Then we may wish to think of a program as having as its meaning a function $f : \mathsf{N}_\perp \to P(\mathsf{N}_\perp)$ where P is one of the powerdomains. For example, if a program may give a 1 or a 2 as an output when given a 0 as input, then we will want the meaning f of this program to satisfy $f(0) = \{\!|1|\!\} \uplus \{\!|2|\!\} = \{\!|1, 2|\!\}$. The three different powerdomains reflect three different views of how to relate the various possible program behaviors in the case of divergence. The upper powerdomain identifies program behaviors which *may* diverge. For example, if program P_1 can give output 1 or diverge on any of its inputs, then it will be identified with the program Q which diverges everywhere, since $\{\!|1, \perp|\!\} = \perp = \{\!|\perp|\!\}$ in $(\mathsf{N}_\perp)^\sharp$. However, program a P_2 which always gives 1 as its output (on inputs other than $\perp$) will *not* have the same meaning as P_1 and $\lambda x.\ \perp$. On the other hand, if the lower powerdomain is used in the interpretation of these programs, then P_1 and P_2 will be given the same meaning since $\{\!|1, \perp|\!\} = \{\!|1|\!\}$ in $(\mathsf{N}_\perp)^\flat$. However, P_1 and P_2 will not have the same meaning as the always divergent program Q since $\{\!|1, \perp|\!\} \neq \perp$ in the lower powerdomain. Finally, in the convex powerdomain, *none* of the programs P_1, P_2, Q have the same meaning since $\{\!|1, \perp|\!\}$, $\{\!|1|\!\}$ and $\{\!|\perp|\!\}$ are all distinct in $(\mathsf{N}_\perp)^\natural$.

To derive properties of the powerdomains like those that we discussed in the previous section for the other operators, we need to introduce the concept of a domain with binary operator.

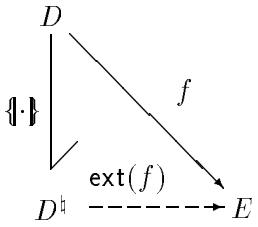
Definition: A *continuous algebra (of signature (2))* is a cpo E together with a continuous binary function $* : E \times E \to E$. We refer to the following collection of axioms on $*$ as theory $T^\natural$:

1. associativity: $(r * s) * t = r * (s * t)$
2. commutativity: $r * s = s * r$
3. idempotence: $s * s = s$.

(These are the well-known semi-lattice axioms.) A *homomorphism* between continuous algebras D and E is a continuous function $f : D \to E$ such that $f(s * t) = f(s) * f(t)$ for all $s, t \in D$. ∎

It is easy to check that, for any domain D, each of the algebras $D^\natural$, $D^\sharp$ and $D^\flat$ satisfies $T^\natural$. However, $D^\natural$ is the "free" continuous algebra over D which satisfies $T^\natural$:

Theorem 12 *Let D be a domain. Suppose $\langle E, * \rangle$ is a continuous algebra which satisfies $T^\natural$. For any continuous $f : D \to E$, there is a unique homomorphism $\mathsf{ext}(f) : D^\natural \to E$ which completes the following diagram:*



Proof: (Hint) If $u = \{x_1, \ldots, x_n\} \in s \in D^\natural$, and $\hat{u}$ is the principal ideal generated by u, then define $\mathsf{ext}(f)(\hat{u}) = f(x_1) * \cdots * f(x_n)$. This function has a unique continuous extension to all of $D^\natural$ given by $\mathsf{ext}(f)(s) = \bigsqcup\{\mathsf{ext}(f)(\hat{u}) \mid u \in s\}$. ∎

Now, consider the following axiom:

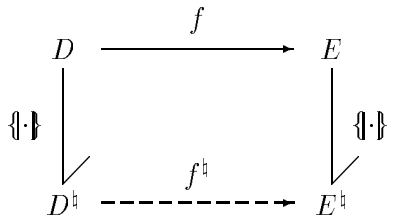
$4^\sharp$. $s \uplus t \sqsubseteq s$.

Let $T^\sharp$ be the set of axioms obtained by adding axiom $4^\sharp$ to the axioms in $T^\natural$. Similarly, let $T^\flat$ be obtained by adding the axiom

$4^\flat$. $s \sqsubseteq s \uplus t$

to the axioms in $T^\natural$. The point is this: *Theorem 12 still holds when $D^\natural$ and $T^\natural$ are replaced by $D^\sharp$ and $T^\sharp$ respectively,* or *by $D^\flat$ and $T^\flat$ respectively.*

As was the case with the smash product and lift operators, a diagram like the one in Theorem 12 gives rise to another important operation on functions. If $f : D \to E$ is a continuous function, then there is a unique homomorphism $f^\natural$ which completes the following diagram:



Namely, one defines $f^\natural = \mathsf{ext}(\{|\cdot|\} \circ f)$. Of course, there are functions $f^\sharp$ and $f^\flat$ with similar definitions.

Two of the powerdomains preserve the property of bounded completeness:

Lemma 13 *If D is a bounded complete domain then so are $D^\natural$ and $D^\flat$.*

Proof: We leave for the reader the exercise of showing that a domain D is bounded complete if and only if every finite bounded subset of its basis has a least upper bound. To see that $D^\flat$ is bounded complete, just note that, for any pair of sets $u, v \in \mathcal{P}_f^*(K(D))$, the ideal generated by their union $u \cup v$ is the least upper bound in $D^\flat$ for the ideals generated by u and v. To see that $D^\sharp$ is bounded complete, suppose $u, v, w \in \mathcal{P}_f^*(K(D))$ with $w \vdash^\sharp u$ and $w \vdash^\sharp v$. Let w' be the set of elements $z \in K(D)$ such that there are elements $x \in u$ and $y \in v$ and z is the least upper bound of $\{x, y\}$. The set w' is non-empty because $\{u, v\}$ is bounded. Moreover, it is not hard to see that $w \vdash^\sharp w'$ and $w' \vdash^\sharp u$ and $w' \vdash^\sharp v$. Hence the ideal generated by w' is the least upper bound of the ideals generated by u and v. ∎

6 Bifinite domains.

Of the operators that we have discussed so far, only the convex powerdomain $(\cdot)^\natural$ does not take bounded complete domains to bounded complete domains. To see this in a simple example, consider the finite poset $\mathsf{T} \times \mathsf{T}$ and the following elements of $\mathcal{P}_f^*(\mathsf{T} \times \mathsf{T})$:

$$u = \{\langle \bot, \mathsf{true}\rangle,\ \langle \bot, \mathsf{false}\rangle\}$$
$$v = \{\langle \mathsf{true}, \bot\rangle,\ \langle \mathsf{false}, \bot\rangle\}$$
$$u' = \{\langle \mathsf{true}, \mathsf{true}\rangle,\ \langle \mathsf{false}, \mathsf{false}\rangle\}$$
$$v' = \{\langle \mathsf{true}, \mathsf{false}\rangle,\ \langle \mathsf{false}, \mathsf{true}\rangle\}$$

It is not hard to see that u' and v' are *minimal* upper bounds for $\{u, v\}$ with respect to the ordering $\vdash^\natural$. Hence no *least* upper bound for $\{u, u'\}$ exists and $(\mathsf{T} \times \mathsf{T})^\natural$ is therefore not bounded complete. In this section we introduce a natural class of domains on which *all* of the operators we have discussed above (including the convex powerdomain) are closed. This class is defined as follows:

Definition: Let D be a cpo. Let $\mathcal{M}$ be the set of finitary projections with finite image. Then D is said to be *bifinite* if $\mathcal{M}$ is countable, directed and $\bigsqcup \mathcal{M} = \mathsf{id}$. ∎

The bifinite cpo's are motivated, in part, by considerations from category theory and the definition above is a restatement of their categorical definition. They were first defined by Plotkin [Plo76] (where they are called "**SFP**-objects") and the term "bifinite" is due to Paul Taylor. Bifinite domains (and various closely related classes of cpo's) have also been discussed under other names such as "strongly algebraic" [Smy83a, Gun86] and "profinite" [Gun87] domains.

6.1 Plotkin orders.

As we suggested earlier, the image of a finitary projection $p : D \to D$ on a domain D can be viewed as an approximation to D. A bifinite domain is one which is a directed limit of its finite approximations. But what is this really saying about the structure of D? First of all, it follows from properties of finitary projections that we mentioned earlier that whenever $p : D \to D$ is a finitary projection and $\mathsf{im}(p)$ is finite, then $\mathsf{im}(p) \subseteq K(D)$. From this, together with the fact that the set $\mathcal{M}$ is directed and $\bigsqcup \mathcal{M} = \mathsf{id}$, it is possible to show D is a domain with $\bigcup\{\mathsf{im}(p) \mid p \in \mathcal{M}\}$ as its basis. We may now use the correspondence which we noted in Theorem 6 to provide a condition on the basis of a domain which characterizes the domain as being bifinite. Recall that $N \lhd A$ for posets N and A if $N \cap {\downarrow} x$ is directed for every $x \in A$.

Definition: A poset A is a *Plotkin order* if, for every finite subset $u \subseteq A$, there is a finite set $N \lhd A$ with $u \subseteq N$. ∎

Theorem 14 *The following are equivalent for any cpo D.*

1. *D is bifinite.*
2. *D is a domain and $K(D)$ is a Plotkin order.* ∎

To get some idea what a Plotkin order looks like, it helps to have a definition. Given a poset A and a finite set $u \subseteq A$, an upper bound x for u is *minimal* if, for any upper bound y for u, $y \sqsubseteq x$ implies $y = x$. A set v of minimal upper bounds for u is said to be *complete* if, for every upper bound x for u, there is a $y \in v$ with $y \sqsubseteq x$. Now, let A be a Plotkin order and suppose $u \subseteq A$ is finite. Then there is a finite $N \lhd A$ with $u \subseteq N$. The set N must contain a complete set of minimal upper bounds for u (why?). This shows the first fact about Plotkin orders: every finite subset has a complete set of minimal upper bounds. This rules out configurations like the one pictured in Figure 3a where the pair of points indicated by closed circles do not have such a complete set of minimal upper bounds. But the set N is *finite* so we have our second fact: every finite subset must have a *finite* complete set of minimal upper bounds. This rules out configurations like the one pictured in Figure 3b where the pair of points indicated by closed circles has a complete set of minimal upper bounds but not a finite one. However, having finite complete sets of minimal upper bounds for finite subsets is not a sufficient condition for characterizing the Plotkin orders. To see

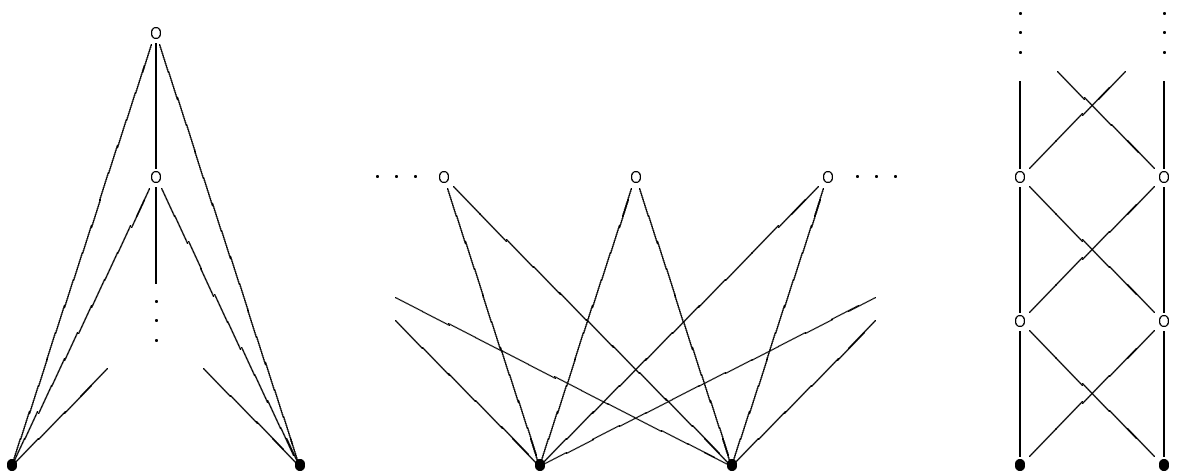


Figure 3: Posets that are not Plotkin orders.

why, let A be a poset which has finite complete sets of minimal upper bounds for finite subsets. If $u \subseteq A$ is finite, let

$$\mathcal{U}(u) = \{x \mid x \text{ is the minimal upper bound for some } v \subseteq u\}.$$

Now, if $u \subseteq N \triangleleft A$, then $\mathcal{U}(u) \subseteq N$. Hence, $\mathcal{U}^n(u) \subseteq N$ for each n. If N is finite, then there must be an n for which $\mathcal{U}^n(u) = \mathcal{U}^{n+1}(u)$. This is a third fact about Plotkin orders: for each finite $u \subseteq A$, $\mathcal{U}^\infty(u) = \bigcup_n \mathcal{U}^n(u)$ is finite. To see what can go wrong, note that $\mathcal{U}^\infty(u)$ is infinite when u is the pair of points indicated by closed circles in Figure 3c.

6.2 Closure properties.

Proposition 15 *A bounded complete domain is bifinite.*

Proof: Suppose D is bounded complete and $u \subseteq K(D)$ is a finite subset of the basis of D. Let

$$N = \{x \mid \ x \text{ is the least upper bound of a finite subset of } u\ \}.$$

Note that N is finite; we claim that $N \triangleleft K(D)$. Suppose x is the least upper bound of a finite set $v \subseteq K(D)$. Since D is algebraic, there is a directed subset $M \subseteq K(D)$ such that $x = \bigsqcup M$. But the elements of v are compact. Hence, for every $y \in v$, there is a $y' \in M$ with $y \sqsubseteq y'$. Since M is directed, there is some $z \in M$ which is an upper bound for v. Now, $z \sqsubseteq x$ so $x = z$ and x is therefore compact. This shows that $N \subseteq K(D)$. Suppose $v \subseteq N$ is bounded, then the least upper bound of v is the same as the least upper bound of the set $\{x \in u \mid x \sqsubseteq y \text{ for some } y \in v\}$ so the least upper bound of v is in N. Now, if $x \in K(D)$, then $S = (\downarrow x) \cap N$ is bounded. Since S has a least upper bound which, apparently, lies in S, we conclude that S is directed. ∎

Theorem 16 *If D is bifinite, then the poset $\mathsf{Fp}(D)$ of finitary projections on D is an algebraic lattice and the inclusion map $i : \mathsf{Fp}(D) \hookrightarrow (D \to D)$ is an embedding.*

Proof: (Sketch) One uses Theorem 6 to show that $\mathsf{Fp}(D)$ is an algebraic lattice. Suppose $f : D \to D$ is continuous. Let

$$S_f = \{x \in K(D) \mid x \sqsubseteq f(x)\}.$$

One can show that there is a least set N_f such that $S_f \subseteq N_f \triangleleft K(D)$. This set determines a finitary projection p_{N_f} as in the discussion before Theorem 6. On the other hand, if $f : D \to D$ is a finitary projection then $N_f = \mathsf{im}(f) \cap K(D)$ and $f = p_{N_f}$. The remaining steps required to verify that $f \mapsto N_f$ is a projection are straight-forward. ∎

Lemma 17 *If D and E are bifinite domains, then so are the cpo's $D \to E$, $D \circ\!\!\to E$, $D \times E$, $D \otimes E$, $D + E$, $D \oplus E$, $D_\perp$, $D^\natural$, $D^\sharp$ and $D^\flat$.*

Proof: We will outline proofs for two sample cases. We begin with the function space operator. Suppose $p : D \to D$ and $q : E \to E$ are finitary projections. Given a continuous function $f : D \to E$, define $\Theta(q,p)(f) = q \circ f \circ p$. The function $\Theta(q,p)$ defines a finitary projection on $D \to E$. Moreover, if p and q have finite images, then so does $\Theta(q,p)$. If we let $\mathcal{M}$ be the set of functions $\Theta(q,p)$ such that p and q are finitary projections with finite image, then it is easy to see that $\bigsqcup \mathcal{M} = \mathsf{id}$. Hence $D \to E$ is bifinite. We will encounter the function Θ again in the next section.

To see that $D^\natural$ is bifinite, one shows that the set

$$\mathcal{M} = \{p^\natural \mid p \in \mathsf{Fp}(D) \text{ and } \mathsf{im}(p) \text{ is finite}\}$$

is directed and has the identity as its least upper bound. The functions in $\mathcal{M}$ are themselves finitary projections with finite images so $D^\natural$ is bifinite. ∎

One may conclude from this lemma that the bifinite domains have rather robust closure properties. But there is something else about bifinite domains which makes them special. They are the *largest* class of domains which are closed under the operators listed in the Lemma. In fact, there is the following:

Theorem 18 *If D and $D \to D$ are domains, then D is bifinite.* ∎

The theorem is due to Smyth and its proof may be found in [Smy83a]. It is carried out by analyzing each of the cases pictured in Figure 3 and showing that if $D \to D$ is not a domain, then D cannot be bifinite. A similar result for the bounded complete domains can be found in [Gun86].

7 Recursive definitions of domains.

Many of the data types that arise in the semantics of computer programming languages may be seen as solutions of *recursive domain equations.* Consider, for example, the equation $T \cong T + T$ (of course, this is an *isomorphism* rather than an *equality*, but let us not make much of this distinction for the moment). How would we go about finding a domain which solves this equation? Suppose we start with the one point domain $T_0 = \mathsf{I}$ as the first approximation to the desired solution. Taking the proof of the Fixed Point Theorem as our guide, we build the domain $T_1 = T_0 + T_0 = \mathsf{I} + \mathsf{I}$ as the second approximation. Now, there is a unique embedding $e_0 : T_0 \to T_1$ so this gives a precise sense in which T_0 approximates T_1. The next approximation to our solution is the domain $T_2 = T_1 + T_1$ and again there is an embedding $e_1 = e_0 + e_0 : T_1 \to T_2$. If we continue along this path we build a sequence

$$T_0 \stackrel{e_0}{\perp\!\!\!\to} T_1 \stackrel{e_1}{\perp\!\!\!\to} T_2 \stackrel{e_2}{\perp\!\!\!\to} \cdots$$

of approximations to the full simple binary tree. To get a domain, we must add limits for each of the branches. The resulting domain (*i.e.* the full simple binary tree with the limit points added) is, indeed, a "solution" of $T \cong T + T$. This is all very informal, however; how are we to make this idea mathematically *precise* and, at the same time, sufficiently *general?*

7.1 Solving domain equations with closures.

In this section we discuss a technique for solving recursive domain equations by relating domains to functions by the "image" map (im) and then using the ideas of the previous section to solve equations. There are two (closely related) ways of doing this which we will illustrate. The first of these is based on the following concept:

Definition: Let D and E be cpo's. A continuous function $r : D \to E$ is a *closure* if there is a continuous function $s : E \to D$ such that $r \circ s = \mathsf{id}$ and $s \circ r \sqsupseteq \mathsf{id}$. ∎

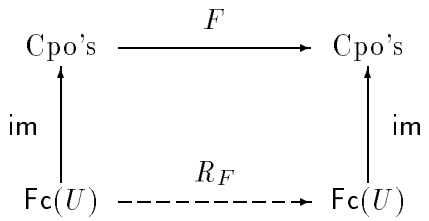
By analogy with the notion of a finitary projection, we will say that a function $r : D \to D$ is a *finitary closure* if $r \circ r = r \sqsupseteq \mathsf{id}$ and $\mathsf{im}(r)$ is a domain. In the event that D is a domain, the requirement that $\mathsf{im}(r)$ be a domain is unnecessary because we have the following:

Lemma 19 *If D is a domain and $r : D \to D$ satisfies the equation $r \circ r = r \sqsupseteq \mathsf{id}$, then $\mathsf{im}(r)$ is a domain.* ∎

The Lemma is proved by showing that $\{r(x) \mid x \in K(D)\}$ forms a basis for $\mathsf{im}(r)$. We will say that a domain E is a *closure of* D if it is isomorphic to $\mathsf{im}(r)$ for some finitary closure r on D. We let $\mathsf{Fc}(D)$ be the poset of finitary closures $r : D \to D$.

Lemma 20 *If D is a domain, then $\mathsf{Fc}(D)$ is a cpo.* ∎

Definition: Let us say that an operator F on cpo's is *representable* over a cpo U if and only if there is a continuous function R_F which completes the following diagram (up to isomorphism):



i.e. $\mathsf{im}(R_F(r)) \cong F(\mathsf{im}(r))$ for every closure r. ∎

This idea extends to multiary operators as well. For example, the function space operator $\cdot \to \cdot$ is representable over a cpo U if there is a continuous function

$$R : \mathsf{Fc}(U) \times \mathsf{Fc}(U) \to \mathsf{Fc}(U)$$

such that, for any $r, s \in \mathsf{Fc}(U)$,

$$\mathsf{im}(R(r,s)) \cong \mathsf{im}(r) \to \mathsf{im}(s)$$

A operator $\langle F_1, \ldots, F_n \rangle$ is defined to be representable if each of the operators F_i is. Note that a composition of representable operators is representable.

Theorem 21 *If an operator F is representable over a cpo U, then there is a domain D such that $D \cong F(D)$.*

Proof: Suppose R_F represents F. By the Fixed Point Theorem, there is an $r \in \mathsf{Fc}(U)$ such that $r = R_F(r)$. Thus $\mathsf{im}(r) = \mathsf{im}(R_F(r)) \cong F(\mathsf{im}(r))$ so $\mathsf{im}(r)$ is the desired domain. ∎

Now we know how to solve domain equations. For example, to solve $T \cong T + T$ we need to find a domain U and continuous function $f : U \to U$ which represents the operator $F(X) = X + X$. But we are still left with the problem of finding a domain over which such operations may be represented! The next step is to look at a simple structure which can be used to represent several of the operations in which we are interested.

Given sets S and T, let T^S be the set of (all) functions from S into T. If T is a cpo, then T^S is also a cpo under the pointwise ordering. Now, it is not hard to see that the domain equation $X \cong X \times \mathsf{I}^\mathsf{T}$ (where I^T is the two point lattice) has, as one of its solutions, the cpo $(\mathsf{I}^\mathsf{T})^\mathsf{N}$. In fact, this cpo is isomorphic to the algebraic cpo $\mathcal{P}\mathsf{N}$ of subsets of N which we discussed in the first section. It is particularly interesting because of the following:

Theorem 22 *For any (countably based) algebraic lattice L, there is a closure $r : \mathcal{P}\mathsf{N} \to L$.*

Proof: Let $l_0, l_1, l_2, \ldots$ be an enumeration of the basis of L. Given $S \subseteq \mathsf{N}$, let $r(S) = \bigsqcup\{l_n \mid n \in S\}$. If $l \in L$, let $s(l) = \{n \mid l_n \sqsubseteq l\}$. We leave for the reader the (easy) demonstration that r, s are continuous with $r \circ s = \mathsf{id}$ and $s \circ r \sqsupseteq \mathsf{id}$. ∎

Structures such as $\mathcal{P}\mathsf{N}$ are often referred to as *universal domains* because they have a rich collection of domains as retracts. In the remainder of this section we will discuss two more similar constructions and show how they may be used to provide representations for operators.

Unfortunately, there is no representation for the operator $F(X) = X + X$ over $\mathcal{P}\mathsf{N}$. However, there are some much more interesting operators which *are* representable over $\mathcal{P}\mathsf{N}$. In particular,

Lemma 23 *The function space operator is representable over* $\mathcal{P}\mathsf{N}$.

Proof: Consider the algebraic lattice of functions $\mathcal{P}\mathsf{N} \to \mathcal{P}\mathsf{N}$. By Theorem 22, we know that there are continuous functions

$$\Phi_{\to} : \mathcal{P}\mathsf{N} \to (\mathcal{P}\mathsf{N} \to \mathcal{P}\mathsf{N})$$
$$\Psi_{\to} : (\mathcal{P}\mathsf{N} \to \mathcal{P}\mathsf{N}) \to \mathcal{P}\mathsf{N}$$

such that $\Phi_{\to} \circ \Psi_{\to} = \mathsf{id}$ and $\Psi_{\to} \circ \Phi_{\to} \sqsupseteq \mathsf{id}$. Now, suppose $r, s \in \mathsf{Fc}(\mathcal{P}\mathsf{N})$ (that is, $r \circ r = r \sqsupseteq \mathsf{id}$ and $s \circ s = s \sqsupseteq \mathsf{id}$). Given a continuous function $f : \mathcal{P}\mathsf{N} \to \mathcal{P}\mathsf{N}$, let $\Theta(s, r)(f) = s \circ f \circ r$ and define

$$R_{\to}(r, s) = \Psi_{\to} \circ \Theta(s, r) \circ \Phi_{\to}.$$

To see that this function is a finitary closure, we take $x \in \mathcal{P}\mathsf{N}$ and compute

$$\begin{aligned}
& (R_{\to}(r,s) \circ R_{\to}(r,s))(x) \\
= \;& (\Psi_{\to} \circ \Theta(s,r) \circ \Phi_{\to})(\Psi_{\to}(s \circ (\Phi_{\to}(x)) \circ r) \\
= \;& (\Psi_{\to} \circ \Theta(s,r) \circ \Phi_{\to} \circ \Psi_{\to})(s \circ (\Phi_{\to}(x)) \circ r) \\
= \;& (\Phi_{\to} \circ \Theta(s,r))(s \circ (\Phi_{\to}(x)) \circ r) \\
= \;& \Psi_{\to}((s \circ s) \circ (\Phi_{\to}(x)) \circ (r \circ r)) \\
= \;& \Psi_{\to}(s \circ (\Phi_{\to}(x)) \circ r) \\
= \;& R_{\to}(r,s)(x)
\end{aligned}$$

and

$$R_{\to}(r, s)(x) = \Psi_{\to}(s \circ (\Phi_{\to}(x)) \circ r) \sqsupseteq \Psi_{\to}(\Phi_{\to}(x)) \sqsupseteq x.$$

Thus we have defined a function,

$$R_{\to} : \mathsf{Fc}(\mathcal{P}\mathsf{N}) \times \mathsf{Fc}(\mathcal{P}\mathsf{N}) \to \mathsf{Fc}(\mathcal{P}\mathsf{N})$$

which we now demonstrate to be a representation of the function space operator.

Given $r, s \in \mathsf{Fc}(\mathcal{P}\mathsf{N})$, we must show that there is an isomorphism

$$\mathsf{im}(R(r, s)) \cong \mathsf{im}(r) \to \mathsf{im}(s)$$

for each $r, s \in \mathsf{Fc}(\mathcal{P}\mathsf{N})$. Now, there is an evident isomorphism between continuous functions $f : \mathsf{im}(r) \to \mathsf{im}(s)$ and continuous functions $g : \mathcal{P}\mathsf{N} \to \mathcal{P}\mathsf{N}$ such that $g = s \circ g \circ r$. We claim that $\Psi_{\to}$ cuts down to an isomorphism between such functions and the sets in the image of $R_{\to}(r, s)$. Since $\Phi_{\to} \circ \Psi_{\to} = \mathsf{id}$, we need only show that $(\Psi_{\to} \circ \Phi_{\to})(x) = x$ for each $x = R_{\to}(r, s)(x)$. But if

$$x = \Psi_{\to}(s \circ (\Phi_{\to}(x)) \circ r)$$

then

$$\begin{aligned}(\Psi_{\to} \circ \Phi_{\to})(x) &= (\Psi_{\to} \circ \Phi_{\to} \circ \Psi_{\to})(s \circ (\Phi_{\to}(x)) \circ r) \\ &= \Psi_{\to}(s \circ (\Phi_{\to}(x)) \circ r) \\ &= x\end{aligned}$$

Hence $\mathsf{im}(R_{\to}(r, s)) \cong \mathsf{im}(r) \to \mathsf{im}(s)$ and we may conclude that $R_{\to}$ represents $\to$ over $\mathcal{P}\mathsf{N}$. ∎

A similar construction can be carried out for the product operator. Suppose

$$\Phi_{\mathsf{x}} : \mathcal{P}\mathsf{N} \to (\mathcal{P}\mathsf{N} \times \mathcal{P}\mathsf{N})$$
$$\Psi_{\mathsf{x}} : (\mathcal{P}\mathsf{N} \times \mathcal{P}\mathsf{N}) \to \mathcal{P}\mathsf{N}$$

such that $\Phi_{\mathsf{x}} \circ \Psi_{\mathsf{x}} = \mathsf{id}$ and $\Psi_{\mathsf{x}} \circ \Phi_{\mathsf{x}} \sqsupseteq \mathsf{id}$. For $r, s \in \mathsf{Fp}(\mathcal{P}\mathsf{N})$ define

$$R_{\mathsf{x}}(r, s) = \Psi_{\mathsf{x}} \circ (r \times s) \circ \Phi_{\mathsf{x}}$$

We leave for the reader the demonstration that this makes sense and R_{x} represents the product operator.

Suppose that L is an algebraic lattice. Then there are continuous functions

$$\Phi_L : \mathcal{P}\mathsf{N} \to \mathcal{P}\mathsf{N}$$
$$\Psi_L : \mathcal{P}\mathsf{N} \to \mathcal{P}\mathsf{N}$$

such that $\Phi_L \circ \Psi_L = \mathsf{id}$ and $\Psi_L \circ \Phi_L \sqsupseteq \mathsf{id}$. Then the function

$$R_L(r, s) = \Psi_L \circ \Phi_L$$

represents the constant operator $X \mapsto L$ because $\mathsf{im}(\Psi_L \circ \Phi_L) \cong L$. A similar argument can be used to show that a constant operator $X \mapsto D$ is representable over a domain U if and only if D is a closure of U.

7.2 Modelling the untyped λ-calculus.

It is tempting to try to solve the domain equation $D \cong D \to D$ by the methods just discussed. Unfortunately, the equation $\mathsf{I} \cong \mathsf{I} \to \mathsf{I}$ (corresponding to the fact that on a one-point set there is

only one possible self-map) shows that there is no guarantee that the result will be at all interesting. There has to be a way to build in some nontrival structure that is not wiped out by the fixed-point process. Methods are described in [Sco76a, Sco80a], but the following, from [Sco76b, Sco80b], is more direct and more general.

Lemma 24 *Let U be a non-trivial cpo. If the product and function space operators can be represented over U, then there are non-trivial domains D and E such that $E \cong E \times E$ and $D \cong D \to E$.*

Proof: We can represent $F(X) = U \times X \times X$ over U, so there is a closure A of U such that $A \cong U \times A \times A$. Thus $U \times A \cong U \times (U \times A \times A) \cong (U \times A) \times (U \times A)$. So $E = U \times A$ is non-trivial and $E \cong E \times E$. Now, E is a closure of U so $G(X) = X \to E$ is representable over U. Hence there is a cpo $D \cong D \to E$. This cpo is non-trivial because E is. ∎

Theorem 25 *If U is a non-trivial domain which represents products and function spaces, then there is a non-trivial domain D such that $D \cong D \times D \cong D \to D$ and D is the image of a closure on U.*

Proof: Let D and E be the domains given by Lemma 24. Then

$$D \times D \cong (D \to E) \times (D \to E) \cong D \to (E \times E) \cong D \to E \cong D$$

and

$$D \to D \cong D \to (D \to E) \cong (D \times D) \to E \cong D \to E \cong D. \ ∎$$

We note, in fact, that D will have $\mathcal{P}\mathsf{N}$ itself represented by a closure on U. Hence, to get a non-trivial solution for $D \cong D \to D \cong D \times D$, take U in the theorem to be $\mathcal{P}\mathsf{N}$. What good is such a domain? The answer is that a D satisfying these isomorphisms is a model for a very strong λ-calculus. If we expand the syntax of λ-calculus given in Section 5.3 of the chapter by Mosses to allow pairings, we would have:

$$E \ ::= \ (\lambda x.\ E) \mid E_1(E_2) \mid x \mid \mathsf{pair} \mid \mathsf{fst} \mid \mathsf{snd}$$

Now, Mosses points out that under the semantic function he defines, many *different* expressions are mapped into the *same* values. We can say that the model *satisfies* certain equations. In particular, under the isomorphisms obtained in our theorems above, the following equations will be satisfied:

1. $(\lambda x.\ E) = (\lambda y.\ [y/x]E)$ (provided y is not free in E)
2. $(\lambda x.\ E)(E') = [E'/x]E$
3. $(\lambda x.\ E(x)) = E$ (provided x is not free in E)

4. $\mathsf{fst}(\mathsf{pair}(E)(E')) = E$

5. $\mathsf{snd}(\mathsf{pair}(E)(E')) = E'$

6. $\mathsf{pair}(\mathsf{fst}(E))(\mathsf{snd}(E)) = E$

In these equations, the third and sixth especially emphasize the isomorphisms $D = D \to D$ and $D = D \times D$. There are models where $D \to D$ is represented by a closure on D (as is $D \times D$) but where this is not an isomorphism. It follows that the special equations are independent of the others.

In [R87] the question is brought up whether we can add to the above equations one relating functional abstraction with pairing. In particular, the following would be interesting:

$$\mathsf{pair}(x)(y) = (\lambda z.\ \mathsf{pair}(x(z))(y(z))).$$

This equation identifies the primitive pairing with what could be called *pointwise pairing*. This equation is independent from the others, but a model for it can be obtained from the first model by introducing a new pairing and application operation that does things pointwise in a suitable sense. There must be many other kinds of models that relate the functional structure to other constructs as well.

Suppose we have domains that satisfy just the six equations. Then from the primitive operations given, many others can be defined. The operation of λ-abstraction is, to be sure, a variable-binding operator (somewhat like a quantifier), but the others are algebraic in nature. As stated, application is a binary operation, and pair, fst and snd are constants. But we can define binary, ternary, and unary operations such as: $\mathsf{pair}(x)(y)$, $\mathsf{pair}(x)(\mathsf{pair}(y)(z))$, $\mathsf{fst}(x)$, $\mathsf{snd}(y)$, $\mathsf{pair}(\mathsf{snd}(z))(\mathsf{fst}(z))$, and many, many more. In other words, the domain D will become a model of many kinds of algebras.

In general, an *algebra* is a set together with several operations defined on it, taking values in the same set. The simplest situation is to consider finitary operations (*i.e.*, operations taking a fixed finite number of arguments). When giving an algebra, the sequence of arities of the fundamental operations is called the *signature* of the algebra. Thus, a *ring* is often given with just two binary operations (*addition* and *multiplication*) making a signature (2,2). Now, subtraction is definable in first-order logic from addition, but the definition is not equational. Therefore, it may be better to consider a ring as an algebra of signature (2,2,2) with subtraction being taken as primitive. Of, course it is enough to have the minus operation, which is unary. So, a signature (2,1,2) is also popular. Strictly speaking, however, different signatures correspond to algebras of different types. Not every algebra of signature (2,2,2) is "equivalent" to one of signature (2,1,2); rings as algebras have very special properties.

By a *continuous algebra* we mean a domain with various continuous operations singled out. In particular, our λ-calculus model can be considered as a continuous algebra of signature (2,0,0,0,0,0). The binary operation is the operation of functional application. Here, 0 indicates a 0-ary operation,

which is just a *constant*. We already know the constants pair, fst, snd. The other two popular constants from the literature on λ-calculus are called S and K. In terms of λ-abstraction they can be defined as follows:

$$S = (\lambda x.\ (\lambda y.\ (\lambda z.\ x(z)(y(z)))))$$
$$K = (\lambda x.\ (\lambda y.\ x))$$

They enjoy many, many equations in the algebra (see, for example, [Bar84]) and, in fact, any equation involving the λ-operator can be rewritten purely algebraically in terms of S and K and application.

We will call an expression in the notation of applicative algebra which has no variables a *combination*. Any combination F defines an n-ary operation:

$$F(x_1)(x_2)\cdots(x_n).$$

What we have been remarking is that the algebras so obtained from combinations can be very rich. In a series of papers [Eng81, Eng] Engeler discussed just how rich these algebras can be. A representative result, following Engeler, will be exhibited here.

Theorem 26 *Given a signature $(s_1, s_2, \ldots, s_n)$, there are combinations $F_1, F_2, \ldots, F_n$ defining operations on D of these arities such that whenever a continuous algebra of this signature is given on a domain A that is a retract of D, then A can be made isomorphic to a subalgebra of this fixed algebra structure on D.*

Proof: If A is a retract of D, then A can be regarded as a subset of D, and all the continuous operations on A can be naturally extended to continuous operations on D of the same arities. (This does not solve the problem, since the operations on D depend on the choice of A. That is to say, at the start A is a subalgebra of the wrong algebra on D.) We can call these operations $o_1, o_2, \ldots, o_n$.

We are going to define the representation of A as a subalgebra of D by means of a continuous function $\rho : A \to D$ defined by means of a fixed-point equation:

$$\begin{aligned}
\rho(a) \quad = \quad & \mathsf{pair}(a) \\
& (\mathsf{pair}(\lambda x_2 \ldots \lambda x_{s_1}.\ \rho(o_1(a, \mathsf{fst}(x_2), \ldots, \mathsf{fst}(x_{s_1}))))) \\
& (\mathsf{pair}(\lambda x_2 \ldots \lambda x_{s_2}.\ \rho(o_2(a, \mathsf{fst}(x_2), \ldots, \mathsf{fst}(x_{s_2}))))) \\
& \quad \vdots \\
& (\mathsf{pair}(\lambda x_2 \ldots \lambda x_{s_n}.\ \rho(o_n(a, \mathsf{fst}(x_2), \ldots, \mathsf{fst}(x_{s_n}))))) \\
& (K)\ )\cdots)
\end{aligned}$$

In this way, we build into ρ the elements from A and the operations as well. The question is how to read off the coded information.

Consider the following combinations:

$$\begin{aligned} F_1 &= \lambda x.\, \mathsf{fst}(\mathsf{snd}(x)) \\ F_2 &= \lambda x.\, \mathsf{fst}(\mathsf{snd}(\mathsf{snd}(x))) \\ &\vdots \\ F_n &= \lambda x.\, \mathsf{fst}(\mathsf{snd}(\mathsf{snd}(\cdots\mathsf{snd}(x)))), \end{aligned}$$

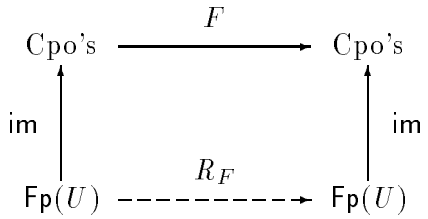
which have to be rewritten in terms of S, K, fst, and snd. We then calculate that

$$F_i(\rho(a_1))(\rho(a_2))\cdots(\rho(a_{s_i})) = \rho(o_i(a_1, a_2, \ldots a_{s_i})).$$

This means if we consider the algebra $\langle D, F_1, F_2, \ldots, F_n\rangle$, then we can find by means of the definition of ρ any algebra $\langle A, o_1, o_2, \ldots, o_n\rangle$, isomorphic to a subalgebra of the first algebra. ∎

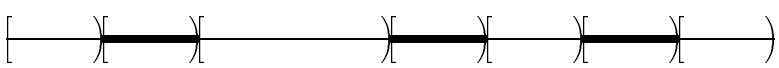
7.3 Solving domain equations with projections.

As we mentioned earlier, one slightly bothersome drawback to $\mathcal{P}\mathsf{N}$ as a domain for solving recursive domain equations is the fact that it cannot represent the sum operator $+$. One might try to overcome this problem by using the operator $(\,\cdot\, + \,\cdot\,)^\top$ as a substitute since this *is* representable over $\mathcal{P}\mathsf{N}$. However, the added top element seems unmotivated and gets in the way. It is probably possible to find a cpo which will represent the operators $\times, \to, +$. However, for the sake of variety, we will discuss a slightly different method for solving domain equations. Let us say that an operator F on cpo's is *p-representable* over a cpo U if and only if there is a continuous function R_F which completes the following diagram (up to isomorphism):



Since there will be no chance of confusion, let us just use the term "representable" for "p-representable" for the remainder of this section. Since $\mathsf{Fp}(U)$ is a cpo we can solve domain equations in the same way we did before *provided we can find domains over which the necessary operators can be represented.*

The construction of a suitable domain is somewhat more involved than was the case for $\mathcal{P}\mathsf{N}$. We begin by describing the basis of a domain U. Let S be the set of rational numbers of the form $n/2^m$ where $0 \le n < 2^m$ and $0 < m$. As the basis U_0 of our domain we take finite (non-empty) unions of half open intervals $[r, t) = \{s \in S \mid r \le s < t\}$. A typical element would look like



We order these sets by superset so that the interval $[0,1)$ is the *least* element. There is no top element under this ordering. If we adjoin the emptyset, say $B = \mathsf{U}_0 \cup \{\emptyset\}$, then we get a *Boolean algebra.* (Note that the complement of a finite union of intervals is again one such—unless it is empty.) In particular, any interval contains a proper sub-interval so, as a Boolean algebra, B is *atomless.* But B is countable, and—up to isomorphism—the only countable atomless Boolean algebra is the free one on countably many generators. But this Boolean algebra has the property that every countable Boolean algebra is isomorphic to a subalgebra. Now, suppose A is a countable bounded complete poset. Let B' be the boolean algebra of subsets of A generated by those subsets of the form $\uparrow x = \{y \in A \mid x \sqsubseteq y\}$ and order this collection by superset so that $\emptyset$ will be its largest element. The map $i : x \mapsto \uparrow x$ is a monotone injection which preserves existing least upper bounds. Moreover, a subset $u \subseteq A$ is bounded just in case $\bigcap_{x \in u} \uparrow x$ is non-empty. Now, if $j : B' \to B$ maps B' isomorphically onto a subalgebra of B, then the composition $j \circ i$ cuts down to an isomorphism between A and a normal subposet $A' \triangleleft \mathsf{U}_0$. Letting U be the domain of ideals over U_0 we may now conclude the following:

Theorem 27 *For any bounded complete domain D, there is a projection*

$$p : \mathsf{U} \to D. \ \blacksquare$$

We can now use this to see that an equation like $X \cong \mathsf{N}_\perp + (X \to X)$ has a solution. The proof that $\to$ is representable over U is almost identical to the proof we gave above that it is representable over $\mathcal{P}\mathsf{N}$. To get a representation for $+$, take a pair of continuous functions

$$\Phi_+ : \mathsf{U} \to (\mathsf{U} + \mathsf{U})$$
$$\Psi_+ : (\mathsf{U} + \mathsf{U}) \to \mathsf{U}$$

such that $\Phi_+ \circ \Psi_+ = \mathsf{id}$ and $\Psi_+ \circ \Phi_+ \sqsubseteq \mathsf{id}$. Then take

$$R_+(r,s) = \Psi_+ \circ (r + s) \circ \Phi_+.$$

Also, there is a representation $R_{\mathsf{N}_\perp}$ for constant operator $X \mapsto \mathsf{N}_\perp$. Hence the operator $X \mapsto \mathsf{N}_\perp + (X \to X)$ is represented over U by the function

$$p \mapsto R_+(R_{\mathsf{N}_\perp}(p), R_\to(p,p)).$$

We have, in fact, the following:

Lemma 28 *The following operators are representable over U: $\to$, $\circ\!\!\to$, $\times$, $\otimes$, $+$, $\oplus$, $(\cdot)_\perp$, $(\cdot)^\sharp$, $(\cdot)^\flat$.* $\blacksquare$

This means that we have solutions over the bounded complete domains for a quite substantial class of recursive equations. More discussion of U may be found in [Sco81], [Sco82a] and [Sco82b].

7.4 Representing operators on bifinite domains.

The convex powerdomain $(\cdot)^\natural$ cannot be representable over U because it does not preserve bounded completeness. We construct a domain over which this operator can be represented as follows. Given a poset A, define $M(A)$ to be the of pairs $(x,u) \in A \times \mathcal{P}_f(A)$ such that $x \sqsubseteq z$ for every $z \in u$. Define a pre-ordering on $M(A)$ by setting $(x,u) \vdash (y,v)$ if and only if there is a $z \in u$ such that $z \sqsubseteq y$. Now, given a domain D, we define D^+ to be the domain of ideals over $\langle M(A), \vdash \rangle$.

Theorem 29 *If D is bifinite, then so is D^+. Moreover, if $D \cong D^+$ and E is any bifinite domain, then there is a projection $p : D \to E$.* ∎

A full proof of the theorem may be found in [Gun87]. We will attempt to offer some hint about how the desired fixed point is obtained. At the first step we take the domain $\mathsf{I} = \{\bot\}$ containing only the single point $\bot$. At the second step, I^+, there are elements $a = (\bot, \{\bot\})$ and $b = (\bot, \emptyset)$ with $b \vdash a$. At the third step there are five elements

$$(a,\{a\}),\ (a,\{b\}),\ (b,\{b\}),\ (b,\emptyset),\ (a,\emptyset)$$

which form the partially ordered set I^{++} pictured in Figure 4. Note that there is another element $(a,\{a,b\}) \in M(\mathsf{I}^+)$ but this satisfies $(a,\{a\}) \vdash (a,\{a,b\})$ and $(a,\{a,b\}) \vdash (a,\{a\})$ so we have identified these elements in the picture. The next step I^{+++} has 20 elements (up to equivalence in the sense just mentioned) and it is also pictured in Figure 4. We leave the task of drawing a picture of I^{++++} as an exercise for the (zealous) reader. It should be noted that each stage of the construction is *embedded* in the next one by the map $x \mapsto (x,\{x\})$. The closed circles in the figure are intended to give a hint of how this embedding looks.

The technique which we have used to build this domain can be generalized and used for other classes as well [GJ90].

We have the following:

Lemma 30 *The following operators are p-representable over V: $\to$, $\multimap$, $\times$, $\otimes, +$, $\oplus, (\cdot)_\bot$, $(\cdot)^\sharp$, $(\cdot)^\flat$, $(\cdot)^\natural$.* ∎

As with most of the other operators, to get a representation for $(\cdot)^\natural$, take a pair of continuous functions

$$\Phi_\natural : \mathsf{V} \to \mathsf{V}^\natural$$
$$\Psi_\natural : \mathsf{V}^\natural \to \mathsf{V}$$

such that $\Phi_\natural \circ \Psi_\natural = \mathsf{id}$ and $\Psi_\natural \circ \Phi_\natural \sqsubseteq \mathsf{id}$. Then

$$R_\natural(p) = \Psi_\natural \circ (p^\natural) \circ \Phi_\natural$$

is a representation for the convex powerdomain operator.

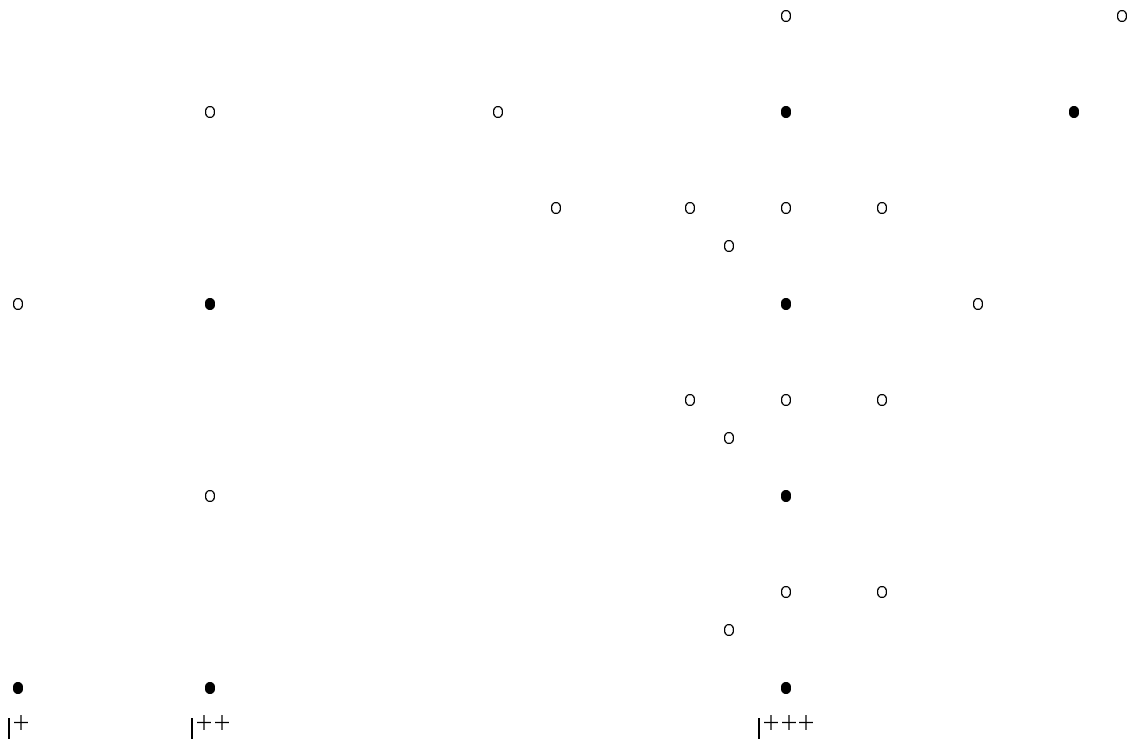


Figure 4: A domain for representing operators on bifinites.

We hope that the reader has begun to note a pattern in the way operators are represented. Most of the operators $(\times, \otimes, +, \oplus, (\cdot)_\perp, (\cdot)^\sharp, (\cdot)^\flat, (\cdot)^\natural)$ may be handled rather straight-forwardly using the corresponding action of these operators on functions. Slightly more care must be taken in dealing with the function space and strict function space operators where one must use a function like Θ. The stock of operators that we have defined in this chapter is quite powerful and it can be used for a wide range of denotational specifications. However, the methods that we have used to show facts such as representability (using finitary closures or finitary projections) will apply to a very large class of operators which satisfy certain sufficient conditions.

To understand this phenomenon, one must pass to a more general theory in which such operators are a basic topic of study. This is the theory of *categories.* Many people find it difficult to gain access to the theory of domains when it is described with categorical terminology. On the other hand, it is difficult to explain basic concepts of domain theory without the extremely useful general language of category theory. A good exposition of the relevance of category theory to the theory of semantic domains may be found in [SP82].

Only a small number of categories of spaces having the properties which we have described above are known to exist. What are the special traits that these categories possess? First of all, they have product and function space functors which satisfy the relationship we described at the beginning of section 4. This property, known as *cartesian closure* is a well-known characteristic of categories such as that of sets and functions. But our cartesian closed categories have not only fixed points for (all) morphisms but fixed points for many functors as well. It is this latter feature which makes them well adapted to the task of acting as classes of semantic domains. One additional property which makes these categories special is the existence of domains for representing functors.

This is not to say that there are not other categories which will have the desired properties. One particularly interesting example are the stable structures of Berry [Ber78] which we have not had the space to discuss here. Interesting new examples of such categories are being uncovered by researchers at the time of the writing of this chapter. The reader will find a few leads to such examples in the published literature listed below, and we expect that many quite different approaches will be put forward in future years.

References

[Bar84] H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, revised edition, 1984.

[Ber78] G. Berry. Stable models of typed λ-calculus. In *International Colloquium on Automata, Languages and Programs*, pages 72–89. *Lecture Notes in Computer Science vol. 62,* Springer, 1978.

[Eng] E. Engeler. A combinatory representation of varieties and universal classes. *Algebra Universalis*, ??:??–??, ?? To appear.

[Eng81] E. Engeler. Algebra and combinators. *Algebra Universalis*, 13:389–392, 1981.

[GHK+80] G. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. Mislove, and D. S. Scott. *A Compendium of Continuous Lattices*. Springer, 1980.

[GJ90] C. A. Gunter and A. Jung. Coherence and consistency in domains. *Journal of Pure and Applied Algebra*, 63:49–66, 1990.

[Gun86] C. A. Gunter. The largest first-order axiomatizable cartesian closed category of domains. In A. Meyer, editor, *Logic in Computer Science*, pages 142–148. IEEE Computer Society, June 1986.

[Gun87] C. A. Gunter. Sets and the semantics of bounded nondeterminism. Manuscript, 1987.

[KT84] T. Kamimura and A. Tang. Effectively given spaces. *Theoretical Computer Science*, 29:155–166, 1984.

[Plo76] G. D. Plotkin. A powerdomain construction. *SIAM Journal of Computing*, 5:452–487, 1976.

[Ré87] G. Révész. Rule-based semantics for an extended lambda-calculus. In M. Main, A. Melton, M. Mislove, and D. Schmidt, editors, *Mathematical Foundations of Programming Language Semantics*, pages 43–56. *Lecture Notes in Computer Science vol. 298,* Springer, April 1987.

[Sco76a] D. S. Scott. Data types as lattices. *SIAM Journal of Computing*, 5:522–587, 1976.

[Sco76b] D. S. Scott. Logic and programming languages. *Communications of the ACM*, 20:634–641, 1976.

[Sco80a] D. S. Scott. The lambda calculus: some models, some philosophy. In J. Barwise, editor, *The Kleene Symposium*, pages 381–421. North-Holland, 1980.

[Sco80b] D. S. Scott. Relating theories of the lambda calculus. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 403–450. Academic Press, 1980.

[Sco81] D. S. Scott. Some ordered sets in computer science. In I. Rival, editor, *Ordered Sets*, pages 677–718. D. Reidel, 1981.

[Sco82a] D. S. Scott. Domains for denotational semantics. In M. Nielsen and E. M. Schmidt, editors, *International Colloquium on Automata, Languages and Programs*, pages 577–613. *Lecture Notes in Computer Science vol. 140,* Springer, 1982.

[Sco82b] D. S. Scott. Lectures on a mathematical theory of computation. In M. Broy and G. Schmidt, editors, *Theoretical Foundations of Programming Methodology*, pages 145–292. *NATO Advanced Study Institutes Series,* D. Reidel, 1982.

[Smy77] M. Smyth. Effectively given domains. *Theoretical Computer Science*, 5:257–274, 1977.

[Smy78] M. Smyth. Power domains. *Journal of Computer System Sciences*, 16:23–36, 1978.

[Smy83a] M. Smyth. The largest cartesian closed category of domains. *Theoretical Computer Science*, 27:109–119, 1983.

[Smy83b] M. Smyth. Power domains and predicate transformers: a topological view. In J. Diaz, editor, *International Colloquium on Automata, Languages and Programs*, pages 662–676. *Lecture Notes in Computer Science vol. 154,* Springer, 1983.

[SP82] M. Smyth and G. D. Plotkin. The category-theoretic solution of recursive domain equations. *SIAM Journal of Computing*, 11:761–783, 1982.