Lecture Notes on Domain Theory *

Robert Muller

August 31, 1999

Abstract

What follows is a collection of lecture notes for a series of three lectures introducing Scott's *domain theory*. The lecture notes are not intended to serve as a primary reference but rather as a supplement to a more comprehensive treatment such as [Sch86] or [Sto77].

1 Introduction

One of the basic principles of the Scott-Strachey approach to programming language theory is to define the meaning of a computer program as a mathematical entity that is independent of any implementation the language. It seems natural to model the data types of the language as *sets* and programs operating on the data types as *functions* between sets.

A fundamental problem which arises in this regard is inherent in the difference between the *specification* and *definition* of an object. For example, function specifications may not have an obvious function associated with them. Consider the specification:

$$g(n) \equiv \text{if } n = 0 \text{ then } 1 \text{ else } g(n + 1).$$

What is the mathematical function specified by g? Clearly,

$$g_1(x) = 1$$

gives the correct information on argument 0, but so does

$$g_2(x) = \begin{cases} 1 & x = 0 \\ 343 & \text{otherwise.} \end{cases}$$

The problem is that g is *undefined* for any argument > 0 but one usually works with total functions $f : D_1 \to D_2$ that are defined for every element of D_1.

Similarly, data types are often specified recursively. As we have seen, in any suitable value space for untyped λ-calculus one even has the specification:

$$D = D \to D,$$

and it is not obvious that any non-trivial set satisfies this equation.

The solution to these problems was first proposed in 1969 by D. Scott. As we will see, the basic idea of using sets and functions between sets holds up — provided that the sets have some internal structure (these will be the *complete partial orders*) and the functions preserve that structure (these will be the *continuous functions*).

In these notes we will develop the basic mathematics of domain theory. In Section 2 we define the basic structure of the complete partial orders and the properties of structure preserving functions. The payoff will be the Tarski-Scott Fixpoint theorem which ensures that the functions which arise in programming languages have well-defined meanings. In Section 3 we briefly take up the question of recursively defined domains showing that $P\omega$ (i.e., the powerset of natural numbers ordered by subset inclusion) serves as a universal domain.

Our primary reference is [Sch86] Chs. 3 and 6 but [Sto77, Hen88] and [Bar84] will also be useful.

*This paper appeared as Harvard University CRCT Technical Report TR-06-92.

2 The Theory of Fixpoints

We assume known elementary properties of set theory.

Definition 1 (Partial Order) $D = (U, \sqsubseteq_D)$ is a *partially ordered set* (poset) if U is a set and $\sqsubseteq_D$ is a binary relation on U such that

i) $\sqsubseteq_D$ is *reflexive*, i.e., $\forall x \in U,\ x \sqsubseteq_D x$.

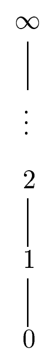
ii) $\sqsubseteq_D$ is *antisymmetric*, i.e., $\forall x,\ y \in U,\ x \sqsubseteq_D y$ and $y \sqsubseteq_D x \Rightarrow x = y$.

iii) $\sqsubseteq_D$ is *transitive*, i.e., $\forall x,\ y,\ z \in U,\ x \sqsubseteq_D y$, and $y \sqsubseteq_D z \Rightarrow x \sqsubseteq_D z$.

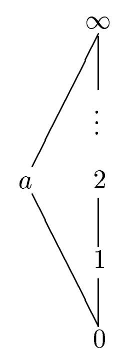
We will usually refer to $D = (U, \sqsubseteq_D)$ simply as D and treat it as though it were a set (e.g., $x \in D$). Whenever possible we will omit the subscript on the relation $\sqsubseteq_D$.

The following examples of partially ordered sets will be referenced throughout the notes.

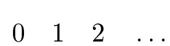
[1]



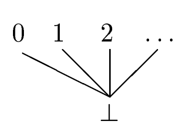
[2]



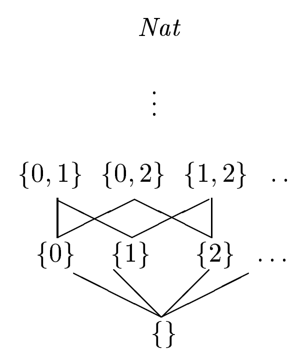
[3]



[4]



[5]



[6]

Definition 2 (Join) Let D be a poset and $x,\ y \in D$. The *join* of x and y in D, notation $x \sqcup y$, is the element of D (if it exists) such that,

i) $x \sqsubseteq x \sqcup y$ and $y \sqsubseteq x \sqcup y$,

ii) $\forall z \in D,\ x \sqsubseteq z$ and $y \sqsubseteq z \Rightarrow x \sqcup y \sqsubseteq z$.

In [1], $0 \sqcup 2 = 2$. In [4], $0 \sqcup 2$ does not exist.

Definition 3 (Least Upper Bound) Let D be a poset. For $X \subseteq D$, the *least upper bound* (lub) of X, notation $\bigsqcup_D X$, is the element of D (if it exists) such that

i) $\forall x \in X$, $x \sqsubseteq \bigsqcup_D X$,

ii) $\forall y \in D$, if $\forall x \in X$, $x \sqsubseteq y$ then $\bigsqcup_D X \sqsubseteq y$.

The element ∞ is the lub of the chain $\{0, 1, 2, \ldots\}$ in [2].
We will usually omit the subscript from $\bigsqcup_D$.
Our framework will require the existence of lubs of subsets of our value spaces. However, we will not require that *all* subsets have lubs.

Definition 4 (Chain) Let D be a poset. $C \subseteq D$ is a *chain* if $\forall x, y \in C$, either $x \sqsubseteq y$ or $y \sqsubseteq x$.

The elements in [1] form a chain but the chain does not have a least upper bound.

Definition 5 (Complete Partial Order) i) A poset D is a *complete partial order* (cpo) if for all chains $C \subseteq D$, $\bigsqcup C \in D$.

ii) A cpo D is a *pointed* cpo if it has a least element $x \in D$, such that $\forall y \in D$, $x \sqsubseteq y$.

Thus, [1] is not a cpo but [2] through [6] are. [4] is not a pointed cpo but [2], [3], [5] and [6] are.

2.1 Structure Preserving Functions

We now wish to consider functions which preserve the structure of partially ordered sets.

Definition 6 (Monotonicity) Let D_1, D_2 be posets. A function $f : D_1 \to D_2$ is *monotonic* if $\forall x$, $y \in D_1$,

$$x \sqsubseteq_{D_1} y \Rightarrow f(x) \sqsubseteq_{D_2} f(y).$$

For example, let

$$Bool_\perp = \begin{tikzcd} \mathrm{tt} \arrow[dr, dash] & & \mathrm{ff} \arrow[dl, dash] \\ & \perp & \end{tikzcd}$$

Then $f_1 : Bool_\perp \to Bool_\perp$, given by

$$f_1(x) = \begin{cases} \perp & x = \mathrm{tt} \\ \perp & x = \perp \\ \mathrm{ff} & x = \mathrm{ff}, \end{cases}$$

is monotonic, but $f_2 : Bool_\perp \to Bool_\perp$, given by

$$f_2(x) = \begin{cases} \perp & x = \mathrm{tt} \\ \mathrm{tt} & x = \perp \\ \mathrm{ff} & x = \mathrm{ff}, \end{cases}$$

is not since $\perp \sqsubseteq \mathrm{tt}$ but $f_2(\perp) \not\sqsubseteq f_2(\mathrm{tt})$.

Definition 7 (Continuity) Let D_1, D_2 be cpos. A function $f : D_1 \to D_2$ is *continuous* if for all chains $C \subseteq D_1$,

$$f(\bigsqcup_{D_1} C) = \bigsqcup_{D_2} \{f(c) \mid c \in C\}.$$

Proposition 1 *Any continuous function is monotonic.*

Proof: Let $f : D_1 \to D_2$ be continuous and $a_1, a_2 \in D_1$ such that $a_1 \sqsubseteq a_2$. Then $\{a_1, a_2\}$ is a chain.

$$\begin{array}{rcl} f(\bigsqcup\{a_1, a_2\}) & = & f(a_2) \\ & = & \bigsqcup\{f(a_1), f(a_2)\} \end{array}$$

That is, $f(a_1) \sqsubseteq f(a_2)$. □

The converse of proposition 1 is not true. Let $f : [2] \to \mathit{Bool}_\perp$, be given by,

$$f(x) = \begin{cases} \mathrm{tt} & x = \infty \\ \perp & \text{otherwise.} \end{cases}$$

Then f is monotonic but

$$f(\bigsqcup\{0, 1, \ldots\}) = f(\infty) = \mathrm{tt} \neq \perp = \bigsqcup\{\perp, \perp, \ldots\} = \bigsqcup f(\{0, 1, \ldots\}),$$

so f is discontinuous. Discontinuous functions do not preserve limits of chains.

Now the payoff.

Definition 8 (Fixpoint) Let D be a poset and f be a function from D to D.

i) $d \in D$ is a *fixpoint* of f if $f(d) = d$.

ii) d is the *least* fixpoint if for all $d' \in D$, $f(d') = d' \Rightarrow d \sqsubseteq d'$.

Theorem 1 (Tarski-Scott Fixpoint Theorem) *Let D be a pointed cpo. Any continuous function $F : D \to D$ has a least fixpoint given by*

$$\mathit{fix}(F) = \bigsqcup\{F^i(\perp_D) \mid i \geq 0\}.$$

Proof:

$$\begin{array}{rcll} F(\mathit{fix}(F)) & = & F(\bigsqcup\{F^i(\perp_D) \mid i \geq 0\}) & \\ & = & \bigsqcup\{F(F^i(\perp_D)) \mid i \geq 0\} & \text{by continuity} \\ & = & \bigsqcup\{F^i(\perp_D) \mid i \geq 1\} & \\ & = & \bigsqcup\{F^i(\perp_D) \mid i \geq 0\} & \\ & = & \mathit{fix}(F). & \end{array}$$

Thus, $\mathit{fix}(F)$, is a fixpoint of F. To see that $\mathit{fix}(F)$ is least, let d be such that $F(d) = d$. $\perp_D \sqsubseteq d$, and by monotonicity of F, $F^i(\perp_D) \sqsubseteq F^i(d)$. Thus,

$$\mathit{fix}(F) = \bigsqcup\{F^i(\perp_D) \mid i \geq 1\} \sqsubseteq d.$$

□

For example, let $D = \mathit{Nat} \to \mathit{Nat}_\perp$ and define,

$$F : D \to D \equiv \lambda f n.\text{if } n = 0 \text{ then } 1 \text{ else } n * f(n-1).$$

$$\begin{array}{rcl} \mathit{fix}(F) = \bigsqcup_{i \geq 0} F^i(\perp_D) & = & \{(0,1), (1,1), (2,2), \ldots, (i, i!), \ldots\} \\ & \vdots & \\ & \sqcup & \\ F^2(\perp_D) & = & \{(0,1), (1,1), (2,\perp), \ldots\} \\ & \sqcup & \\ F^1(\perp_D) & = & \{(0,1), (1,\perp), (2,\perp), \ldots\} \\ & \sqcup & \\ F^0(\perp_D) & = & \{(0,\perp), (1,\perp), (2,\perp), \ldots\}. \end{array}$$

3 Data Types

We now wish to establish that domains are closed under a rich enough variety of operations that they are suitable as value spaces for practical programming languages. The constructions we consider are continuous function space construction ($\to$), cartesian product ($\times$), and disjoint union ($+$).

First, a technical definition for adjoining a bottom element to a poset.

Definition 9 Let $D = (U, \sqsubseteq_D)$ be a poset. The *lifting* of D, notation $D_\perp$, is $D_\perp = (U \cup \{\perp_D\}, \sqsubseteq_{D_\perp})$, with $\perp_D \sqsubseteq_{D_\perp} x$, $\forall x \in U$.

Proposition 2 *If D is a cpo then $D_\perp$ is a pointed cpo.*

Definition 10 Let D_1 and D_2 be posets. Then the set of *continuous functions* from D_1 to D_2, is,

$$[D_1 \to D_2] = \{f \mid f \in D_1 \to D_2, f \text{ is continuous}\},$$

with the *pointwise* ordering, for f, $g \in [D_1 \to D_2]$, $f \sqsubseteq_{[D_1 \to D_2]} g$ iff $\forall x \in D_1$, $f(x) \sqsubseteq_{D_2} g(x)$.

Proposition 3 *Let D_1 and D_2 be cpos. Then $[D_1 \to D_2]$ is a cpo.*

Proof: Let $\{f_i : D_1 \to D_2 \mid i \in I\}$ be a chain in $[D_1 \to D_2]$. For each $a_j \in D_1$, let $A_j = \{f_i(a_j) \mid i \in I\}$. Each A_j is a chain in D_2, thus, $\bigsqcup A_j$ exists. Define $g : D_1 \to D_2$ by $g(a_j) = \bigsqcup A_j$, for all $a_j \in D_1$. We first show that g is continuous. Let $\{a_j \mid j \in J\}$ be a chain in D_1.

$$\begin{aligned}
\bigsqcup\{g(a_j) \mid j \in J\} &= \bigsqcup\{\bigsqcup\{f_i(a_j) \mid i \in I\} \mid j \in J\} \\
&= \bigsqcup\{\bigsqcup\{f_i(a_j) \mid j \in J\} \mid i \in I\} \quad \text{by associativity of } \bigsqcup \\
&= \bigsqcup\{f_i(\bigsqcup\{a_j \mid j \in J\}) \mid i \in I\} \quad \text{since } f_i \text{ is continuous} \\
&= g(\bigsqcup\{a_j \mid j \in J\}).
\end{aligned}$$

Thus, g is continuous. Moreover, g is the lub of $\{f_i : D_1 \to D_2 \mid i \in I\}$ by the ordering. □

Definition 11 Let D_1 and D_2 be posets. Then the *product* of D_1 and D_2 is,

$$D_1 \times D_2 = \{(d, d') \mid d \in D_1, d' \in D_2\},$$

with the *componentwise* ordering:

$$(d_1, d_2) \sqsubseteq_{D_1 \times D_2} (d_1', d_2') \text{ iff } d_1 \sqsubseteq_{D_1} d_1' \text{ and } d_2 \sqsubseteq_{D_2} d_2'.$$

Proposition 4 *Let D_1 and D_2 be (pointed) cpos. Then $D_1 \times D_2$ is a (pointed) cpo.*

Proof: Left as an exercise.

Definition 12 Let D_1 and D_2 be posets. Then the *disjoint union* of D_1 and D_2 is,

$$D_1 + D_2 = \{(i, d_i) \mid d_i \in D_i\},$$

with ordering:

$$(i, d_i) \sqsubseteq_{D_1 + D_2} (j, d_i') \text{ iff } i = j \text{ and } d_i \sqsubseteq_{D_i} d_i'.$$

Proposition 5 *Let D_1 and D_2 be cpos. Then $D_1 + D_2$ is a cpo.*

Proof: Left as an exercise.

3.1 Recursively Defined Datatypes

The problems inherent in recursively specified data types are in stark relief in the untyped λ-calculus. Remember the difficulty highlighted in previous lectures of finding a non-trivial model: since functions can be applied to themselves, any value space D must apparently include the set of functions from $(D \to D)$. By a simple cardinality argument, such an inclusion cannot hold.

We will overcome this problem by insisting that the set D and the class of functions $(D \to D)$ are sufficiently rich to provide meanings for all of our programs yet they are small enough that we can construct an *isomorphism* between D and $(D \to D)$. The isomorphism will be constructed via an *adjoined pair* of maps $\Phi : D \to (D \to D)$ and $\Psi : (D \to D) \to D$,

$$D \underset{\Psi}{\overset{\Phi}{\rightleftarrows}} (D \to D)$$

Adjoined pairs of this type have the essential property that it is possible to completely recover any function in $(D \to D)$ from its representation in D. That is,

$$\begin{array}{rcl} \Phi \circ \Psi & = & \mathrm{id}_{(D \to D)} \\ \Psi \circ \Phi & \sqsubseteq & \mathrm{id}_D. \end{array}$$

With this established we can then define the semantics of untyped λ-calculus as follows. Let $\mathrm{Var} = \{x_0, x_1, \ldots\}$. Define the set of terms *Term* by,

$$M ::= x \mid \lambda x.M \mid M\ M.$$

We will require an environment ρ mapping variables to values in D.

$$\rho \in \mathrm{Env} = \mathrm{Var} \to D.$$

The semantics is then be given by

$$[\![\cdot]\!] : \mathrm{Term} \to \mathrm{Env} \to D$$

.

$$\begin{array}{rcll} [\![x]\!]\rho & = & \rho(x) & \\ [\![\lambda x.M]\!]\rho & = & \Psi(f) & \text{where } f(d) = [\![M]\!]\rho[x \mapsto d] \\ [\![M\ N]\!]\rho & = & f([\![N]\!]\rho) & \text{where } f = \Phi([\![M]\!]\rho) \end{array}$$

3.2 $P\omega$ as a Reflexive Domain

What kind of set D and what class of functions $(D \to D)$ will admit such a construction? An *algebraic* cpo and continuous functions $[D \to D]$.

Definition 13 (Compact Element) i) Let D be a cpo. $x \in D$, is *compact* if for all chains $C \subseteq D$, $x \sqsubseteq \bigsqcup C \Rightarrow \exists c \in C$, such that $x \sqsubseteq c$.

ii) Let *Compact*(D) denote the set of compact elements of cpo D.

Elements ∞ in [2] and a and ∞ in [3] are not compact. The cpo in [6], called $P\omega$, is even a complete lattice. Its compact elements are the *finite sets*.

Obviously we will not require that all elements be compact, but we do insist that each element be the lub of a chain of compact elements.

Definition 14 (Algebraic cpo) i) Let D be a cpo. D is an *algebraic* cpo if $\forall x \in D$, $x = \bigsqcup C$, for some chain $C \subseteq$ *Compact*(D).

ii) An algebraic cpo is ω-*algebraic* if *Compact*(D) is countable.

[3] is not an algebraic cpo since element a is not the lub of a chain of compact elements. [2], [4], [5] and [6] are all ω-algebraic cpos.

Algebraic cpos have a number of important properties. Among them is the following:

Proposition 6 *Let D_1 and D_2 be algebraic cpos. A function $f : D_1 \to D_2$ is continuous iff $\forall x \in D_1$,*

$$f(x) = \bigsqcup\{f(d) \mid d \in \text{Compact}(D_1), d \sqsubseteq x\}.$$

Proof: $\Rightarrow$ Let f be continuous. Then

$$\begin{aligned} f(x) &= f(\bigsqcup\{d \sqsubseteq x \mid d \in \mathit{Compact}(D_1)\} \\ &= \bigsqcup\{f(d) \mid d \in \mathit{Compact}(D_1), d \sqsubseteq x\}. \end{aligned}$$

$\Leftarrow$ (See [Bar84], p. 17). □

Thus, continuous functions over algebraic cpos are determined by their values on compact elements.

We now turn our attention to the particular algebraic cpo $P\omega$. Recall that the compact elements of $P\omega$ are the finite sets. By proposition 6, we then have that for all $f \in [P\omega \to P\omega]$, $x \in P\omega$,

$$f(x) = \bigsqcup\{f(d) \mid d \text{ a finite set}, d \subseteq x\}.$$

Thus, any continuous function $f \in [P\omega \to P\omega]$ can be coded as an element of $P\omega$.

With $D = P\omega$ our adjunction (Ψ, Φ) will be of type:

$$\Phi : P\omega \to [P\omega \to P\omega]$$

$$\Psi : [P\omega \to P\omega] \to P\omega.$$

We will now define a particular coding of functions over sets as sets. The coding proceeds in two steps. First, we will require a coding of a finite set (i.e., a compact element of $P\omega$) to a value $n \in \mathit{Nat}$. For finite set $\{k_0, \ldots, k_{m-1}\}$, $k_0 < \ldots < k_{m-1}$, Define

$$\text{set-code}(\{k_0, \ldots, k_{m-1}\}) = \Sigma_{i<m} 2^{k_i}.$$

For example, set-code($\{0, 2, 3\}$) = 13.

Proposition 7 *set-code() is a bijection mapping $\{x \in P\omega \mid x \text{ is compact}\}$ onto* Nat.

Next we require a coding of *pairs* of natural numbers as a natural number. For n, $m \in \mathit{Nat}$, let

$$\text{pair-code}(n, m) = 1/2(n + m)(n + m + 1) + m.$$

pair-code(n, m) can be interpreted as cartesian coordinates in the following table:

$$\begin{array}{llllll} \vdots \\ 14 \\ 9 & 13 \\ 5 & 8 & 12 \\ 2 & 4 & 7 & 11 \\ 0 & 1 & 3 & 6 & 10 & \ldots \end{array}$$

For example, pair-code($1, 3$) = 13.

Proposition 8 *pair-code(,) is a bijection mapping* Nat × Nat *onto* Nat.

With these defined, the maps are now:

$$\begin{aligned} \Psi(f) &= \{\text{pair-code}(\text{set-code}(x), m) \mid m \in f(x)\} \\ \Phi(u)(x) &= \{m \mid \exists n = \text{set-code}(y), y \subseteq x, \text{ and pair-code}(n, m) \in u\}. \end{aligned}$$

For example, let

$$f \in [P\omega \rightarrow P\omega] = \{(\{0,3\},\{4,6\}),(\{8\},\{1,9\}),\ldots\}.$$

Then

$$\begin{aligned}\Psi(f) &= \{\text{pair-code}(9,4),\text{pair-code}(9,6),\text{pair-code}(256,1),\text{pair-code}(256,9),\ldots\}\\ &= \{95,126,33154,\ldots\}.\end{aligned}$$

Thus, we have shown that $P\omega$ together with the continuous functions over $P\omega$ provide a solution as outlined above. It should be noted that the domain $P\omega$ is rarely used today except as a pedogogical tool for constructing the isomorphism above. As a complete lattice it contains lubs (and greatest lower bounds) of *all* subsets, but one is really only interested in lubs of chains of approximations. The student is referred to [Sch86] Ch. 11 for a more in depth discussion of reflexive domains.

4 Summary

We have developed the basics of domain theory which underlies the Scott-Strachey approach to programming language theory. It should be noted that we have barely scratched the surface — domain theory is an active research area with new types of domains being proposed for various purposes.

References

[Bar84] H. Barendregt. *The Lambda Calculus — Its Syntax and Semantics.* North Holland Press, Amsterdam, 1984.

[Hen88] M. Hennessy. *Algebraic Theory of Processes.* MIT Press, 1988.

[Sch86] D. Schmidt. *Denotational Semantics: A Methodology for Language Development.* Allyn and Bacon, 1986.

[Sto77] J. Stoy. *Denotation Semantics: The Scott-Strachey Approach to Programming Language Theory.* MIT Press, 1977.