Domain Theory and Denotational Semantics

Tanner Duve - Penn Graduate Logic Seminar 04/07/2025

Contents

Abstract

Domain theory was introduced by Dana Scott in the 1960s, motivated by the search for a denotational semantics of the lambda calculus. Denotational semantics is concerned with the mathematical meaning of programming languages, where programs are to be interpreted as morphisms in certain categories. This talk will introduce the basic theory of domains, viewed as both order-theoretic structures and topological spaces, and explore some of their friendly properties that make them uniquely suitable for modeling computer programs. We will explore the connections between computability and continuity, and discuss the role of domains in the semantics of programming languages - in particular interpreting recursive definitions as least fixpoints. Basic concepts in order theory, category theory, and topology will be used but no advanced understanding of these is required from the audience.

1 Motivation

1.1 Denotational Semantics

Programming language semantics is concerned with giving meaning to valid pieces of syntax in a programming language, such that one can reason about both programs written in the language and about the language itself (metatheory)

There are three primary approaches to PL semantics

- Operational Semantics: rules that define how programs are executed/evaluated, eg. β-reduction
- Denotational Semantics: assign each program to a mathematical object - a "model theory" of programming languages
- Axiomatic semantics - programs are defined by their effect on the theory of the program state, that is the collection of properties which hold of the program state - eg. Hoare logic

A useful analogy is that operational semantics correspond to an idealized interpreter, denotational semantics to an idealized compiler, and axiomatic semantics to an idealized verifier.

More precisely, denotational semantics takes each syntactic object t in a programming language to a mathematical object $[\![t]\!]$. The important property that any denotational semantics must have is *compositionality*: the denotation of a term is defined in terms of the denotation of its subterms, that is our denotation function

$$[\![\cdot]\!] : \text{Syntax} \to \text{Semantics}$$

must be a homomorphism.

1.2 Historical Motivations

As mentioned, domain theory was introduced as a way of providing a mathematical model of the untyped lambda calculus. Here we describe the problem which gave rise to domain theory.

To give a denotational semantics for the untyped λ-calculus requires defining a mathematical structure D such that, since there are no types, all terms can be interpreted as having type D. More specifically, we are looking to construct a category with one object (up to isomorphism) D. Now consider the self application term $\lambda x.xx$. If the entire term xx has type D, and the second occurrence of x has type D, then the first occurrence can only be sensibly thought of as having type $[D \to D]$. This means we require that

$$D \cong [D \to D]$$

We need a one object category that is closed under internal hom. If we view $[\cdot \to \cdot]$ as a functor $F : C^{op} \times C \to C$ over some category C, then what we are looking for is a fixpoint $D \cong F(D, D)$

Domain theory ends up being the solution here - terms were interpreted as continuous functions on a topological space D which is isomorphic to its own function space. These spaces are called domains.

1.3 Key Insights

There a few key insights from domain theory which speak to its utility in theoretical computer science:

1. Certain categories of domains are **Cartesian closed**, and thus give rise to models of typed λ-calculi.
2. Domains formalize the notion of partial information, and can be thought of as orderings on information content. A least element $\bot$ is interpreted as having "no information", and can represent undefined values, or diverging computations.

2 Domain Theory

In this section we define our main objects of interest, constructions on them such as products, sums, and exponentials, we provide a topological view of our structures of interest, and we give a toy example of a denotational semantics.

2.1 Dcpo's

We will provide definitions for directed completeness and continuity, and arrive at a fixpoint theorem that will be used in modeling loops and recursive functions.

Definition 1. ***dcpo****: Given a poset $(D, \leq)$ and a nonempty subset $\Delta \subseteq D$, we say D is directed iff*

$$\forall x, y \in \Delta, \exists z \in \Delta, x \leq z \wedge y \leq z$$

D is called directed complete if every directed $\Delta \subset D$ has a supremum, $\bigvee \Delta$. If D moreover has a least element $\bot$, we say it is a complete partial order (cpo)

Example 1. *Any non empty chain is directed*

Definition 2. ***continuous:*** *Given dcpo's D, D', a function $f : D \to D'$ is continuous iff it is monotone and preserves directed suprema, ie.*

$$\forall \Delta \subseteq_{dir} D, f(\bigvee \Delta) = \bigvee f(\Delta)$$

Definition 3. *The category* ***Dcpo*** *has as objects dcpos and as morphisms continuous functions, and the category* ***Cpo*** *with cpos as objects is a full subcategory*

In a comprehensive domain theory text, more subtle kinds of cpos are used as "domains", namely continuous domains and algebraic domains, but for the sake of this presentation our domains will be cpos.

Example 2. *Given a set X, we take the set $X_\perp = X \cup \{\perp\}$, with the ordering $x \leq y \iff x = y \vee x = \perp$. This is called the **flat domain** on X*

Domain theory revolves around the following thesis:

1. All semantic domains of computation are complete partial orders
2. Computable functions are continuous functions

Now the following theorem is the key to interpreting recursively defined functions or commands:

Theorem 1. ***Kleene's fixpoint**: Given a cpo D and an endomorphism $f : D \to D$, $\bigvee_{n\in\omega} f^n(\perp)$ is the least fixpoint of f*

2.2 Dcpo's Topologically

Any topological space $(X, \Omega X)$ has an associated preorder called the specialization preorder, defined as follows:

$$x \leq y \iff \forall U \in \Omega X,\ x \in U \implies y \in U$$

A *Kolmogorov space* or T_0 space is exactly a topology whose specialization preorder is a partial order, more precisely is antisymmetric in the following sense:

$$x \neq y \implies \exists U \in \Omega X, \text{ exactly one of } x \text{ or } y \text{ is in } U$$

Want to define a T_0 topology for dcpos such that

1. The specialization order is the dcpo
2. Functions which are continuous on the ordering are exactly those which are continuous on the topology

The opens of a topological space X are in bijective correspondence with the continuous functions $X \to \{0, 1\}$ (the Sierpinski space), this is easy to see, given an open set U take $\chi_U : X \to \{0, 1\}$ and given $f : \{0, 1\}$ take $f^{-1}(1)$

Definition 4. *(Scott topology). A subset $A \subseteq D$ of a dcpo is called Scott open if*

1. *$x \in A$ and $x \leq y \implies y \in A$ (upward closed)*
2. *Δ directed and $\bigvee \Delta \in A \implies \exists x \in \Delta, x \in A$ (If a directed set converges into the open, it must touch the open)*

The collection $\Omega_S(D)$ of Scott opens is called the Scott topology on D

The following facts say we found what we were looking for:

Theorem 2. *Ω_S is T_0 and the specialization order of (D, Ω_S) is $(D, \leq)$*

Theorem 3. *For any Dcpos D, D'*

$$\mathrm{Hom}_{\mathrm{Top}}(D, D') = \mathrm{Hom}_{\mathrm{Dcpo}}(D, D')$$

2.3 Constructions on Dcpo's

Here we talk about some properties of the category of domains, in particular it being Cartesian closed. Cartesian closedness tells us our category at least gives a model of the simply typed lambda calculus.

2.3.1 Products

We can define arbitrary products in our categories **Dcpo** and **Cpo** as follows:
Given a family of cpos $\{X_i\}_{i\in I}$, their product $\prod_{i\in I} X_i$ is their Cartesian product, with the component-wise ordering and component-wise supremum

2.3.2 Exponentials

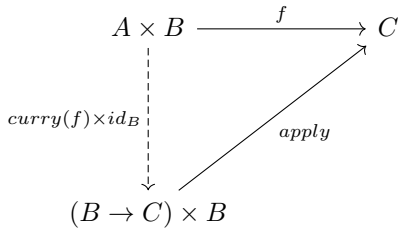
We may also define the Cpo $[X \to Y]$ as the set of continuous functions from X to Y, with the following ordering:

$$f \leq g \iff \forall x \in X, f(x) \leq_Y g(x)$$

We then have

1. The function apply : $(X \to Y) \times X \to Y$ defined as $\text{apply}(f, x) = f(x)$ is continuous
2. Given continuous $f : X \times Y \to Z$, the function $\text{curry}(f) : X \to (Y \to Z)$ defined as $\text{curry}(f)(a) = \lambda b f(a, b)$ is continuous

These satisfy the following universal property:



Telling us that the category Cpo has exponentials, and is thus Cartesian closed.

2.3.3 Limits

Remark 1. ***Dcpo*** *has limits and colimits of arbitrary diagrams, thus is Cartesian closed, complete, and cocomplete*

3 Recursion and Fixpoints

Now in the last part, we show how recursively defined programming constructs can be interpreted as least fixpoints in a cpo

3.1 Recursion in Typed Lambda Calculus via Y

We consider a simply-typed lambda calculus extended with a fixpoint combinator Y satisfying the reduction rule:

$$YM \to M(YM)$$

This allows recursive definitions without requiring a full language specification. We assume the presence of base types (such as $\mathbb{N}$, $\mathbb{B}$) and function types $\sigma \to \tau$.

Denotational Setting. Each type is interpreted as an object in the category **Cpo**:

- Base types are interpreted as cpos, for example $[\![\mathbb{N}]\!] = \mathbb{N}_\perp$
- Function types are interpreted as exponentials: $[\![\sigma \to \tau]\!] = [[\![\sigma]\!] \to [\![\tau]\!]]$

In particular, for YM to be well-typed, we assume $M : \sigma \to \sigma$, so that $YM : \sigma$.

Operational vs Denotational Behavior. The operational rule

$$YM \to M(YM)$$

unfolds the recursion. Semantically, this suggests that $[\![YM]\!]$ should be a fixed point of the function $[\![M]\!] : [\![\sigma]\!] \to [\![\sigma]\!]$.

Domain theory guarantees that every continuous function on a cpo has a least fixed point:

$$\text{Fix}(f) = \bigvee_{n \in \mathbb{N}} f^n(\perp)$$

This fixpoint is taken to be the meaning of YM:

$$[\![YM]\!] = \text{Fix}([\![M]\!])$$

Continuity and Soundness. Because M is a well-typed term, its interpretation $[\![M]\!]$ is continuous. Hence, the least fixed point exists and gives a well-defined denotation for YM.

This interpretation agrees with the operational semantics: the denotational meaning of YM is the limit of the unfolding

$$YM \to M(YM) \to M(M(YM)) \to \dots$$

and this corresponds precisely to the chain

$$\bot \leq f(\bot) \leq f^2(\bot) \leq \dots$$

in the domain $[\![\sigma]\!]$, whose supremum is $[\![YM]\!]$.

3.2 Recursive Types via Functorial Fixpoints

We have seen how recursive functions are interpreted as least fixpoints of continuous endofunctions on cpos:

$$f : D \to D, \qquad \mathrm{Fix}(f) = \bigvee_n f^n(\bot)$$

This idea extends beyond functions to recursive *types*. To capture types like lists or trees, we want to solve equations of the form:

$$D \cong F(D)$$

where F is no longer a function but a functor:

$$F : \mathbf{Cpo} \to \mathbf{Cpo}$$

Our goal is to give meaning to such domain equations by finding their least solutions, just like for functions. To do this, we need to generalize the notion of continuity from functions to functors.

The Category Cpo. Recall that **Cpo** is the category whose:

- objects are cpos (complete partial orders with least elements),
- morphisms are continuous functions (i.e., monotone maps preserving directed suprema).

This category is not just complete and cocomplete — it is also enriched over itself in a useful way: we can define an ordering on objects and use it to construct colimits.

Chains of CPOs. To define a notion of approximation between objects in **Cpo**, we look at chains:

$$D_0 \xrightarrow{f_0} D_1 \xrightarrow{f_1} D_2 \xrightarrow{f_2} \dots$$

where each f_n is a continuous embedding (e.g., an injective continuous function with dense image). This forms a diagram in **Cpo**, and we can take its colimit — the "limit" object that coherently includes all the D_n's.

This colimit plays the same role as the supremum of an increasing chain in ordinary domain theory: it is a kind of "limit of approximations."

Cocontinuous Functors. We now want to lift the idea of continuity to functors:

Definition 5. *A functor $F : \mathbf{Cpo} \to \mathbf{Cpo}$ is* cocontinuous *if it preserves colimits of ω-chains. That is, for any chain of cpos*

$$D_0 \to D_1 \to D_2 \to \dots$$

with colimit $\operatorname{colim} D_n$*, we have:*

$$F(\operatorname{colim} D_n) \cong \operatorname{colim} F(D_n)$$

This is the functorial analogue of Scott continuity: a function $f : D \to D$ is continuous if it preserves least upper bounds of directed sets; a functor is cocontinuous if it preserves colimits of directed diagrams of domains.

Fixpoints of Functors. If F is cocontinuous, we can build the least fixpoint of F by iterating it from the bottom object:

$$\bot \xrightarrow{f_0} F(\bot) \xrightarrow{f_1} F^2(\bot) \xrightarrow{f_2} \dots$$

and taking the colimit:

$$\mu F = \text{colim}_{n \in \mathbb{N}} F^n(\bot)$$

This solves the domain equation:

$$\mu F \cong F(\mu F)$$

in a canonical way, just like how $\text{Fix}(f)$ is the least fixpoint of a continuous function.

Example: Binary Numbers. Consider the inductive type of binary numerals defined by:

```
data B = Zero B | Empty | One B
```

This corresponds to the domain equation:

$$B \cong B + \mathbb{1} + B$$

Define a functor:

$$F(X) = X + \mathbb{1} + X$$

$$F(f) = f + id_{\mathbb{1}} + f$$

Then $F : \mathbf{Cpo} \to \mathbf{Cpo}$ is cocontinuous.
The semantics of **B** is then the least fixpoint of F, ie. the colimit of the diagram:

$$\mathbb{1} \xrightarrow{\lambda x.\bot} F(\mathbb{1}) \xrightarrow{F(\lambda x.\bot)} FF(\mathbb{1}) \xrightarrow{FF(\lambda x.\bot)} FFF(\mathbb{1}) \dots$$

If you recall from an earlier GLoS talk by Oualid, inductive types are **initial algebras** of a certain endofunctor - that's precisely what is being done here: a solution to this domain equation is an initial algebra, and initiality is exactly "leastness" in our ordering.