Domain Theory

Mentee: Tanner Duve, Mentor: Oualid Merzouga

Spring 2024 Directed Reading Project

Michael Mislove. *An Introduction to Domain Theory Notes for a Short Course*, 2003

Introduction

This talk is about the field of domain theory and its applications in the theory of programming languages

Domain theory: Studies partially ordered sets equipped with additional structure giving rise to formal notions of continuity and approximation - closely related to topology

Introduction

This talk is about the field of domain theory and its applications in the theory of programming languages

Domain theory: Studies partially ordered sets equipped with additional structure giving rise to formal notions of continuity and approximation - closely related to topology

Motivation

Why domain theory?

- In PL theory we are interested in *denotational semantics* of programming languages
- Denotational semantics: maps syntactic terms to mathematical objects - essentially a model theory of programming languages
- Models computation in a natural way and thus provides a semantics for the λ-calculus

Motivation

Why domain theory?

- In PL theory we are interested in *denotational semantics* of programming languages
- Denotational semantics: maps syntactic terms to mathematical objects - essentially a model theory of programming languages
- Models computation in a natural way and thus provides a semantics for the λ-calculus

Motivation

Why domain theory?

- In PL theory we are interested in *denotational semantics* of programming languages
- Denotational semantics: maps syntactic terms to mathematical objects - essentially a model theory of programming languages
- Models computation in a natural way and thus provides a semantics for the λ-calculus

Motivation

Why domain theory?

- In PL theory we are interested in *denotational semantics* of programming languages
- Denotational semantics: maps syntactic terms to mathematical objects - essentially a model theory of programming languages
- Models computation in a natural way and thus provides a semantics for the λ-calculus

Outline

This talk will proceed in the following parts

- Background
  - The Untyped λ-Calculus
  - Order Theory
  - Basic Category Theory
- Fixed Points and the Category of ω-Complete Partial Orders
- Least Fixed Points for Denotational Semantics

Outline

This talk will proceed in the following parts

- Background
  - The Untyped λ-Calculus
  - Order Theory
  - Basic Category Theory
- Fixed Points and the Category of ω-Complete Partial Orders
- Least Fixed Points for Denotational Semantics

The Untyped λ-Calculus

The λ-calculus is a formal system to denote function application and abstraction developed by Alonzo Church in the 1930s

Gives rise to theory of functions as rules of computation rather than as sets of ordered pairs

Forms a minimal Turing-complete programming language, ie. any computable function (algorithm) can be expressed as a λ-term

The Untyped λ-Calculus

The λ-calculus is a formal system to denote function application and abstraction developed by Alonzo Church in the 1930s

Gives rise to theory of functions as rules of computation rather than as sets of ordered pairs

Forms a minimal Turing-complete programming language, ie. any computable function (algorithm) can be expressed as a λ-term

The Untyped λ-Calculus

The λ-calculus is a formal system to denote function application and abstraction developed by Alonzo Church in the 1930s

Gives rise to theory of functions as rules of computation rather than as sets of ordered pairs

Forms a minimal Turing-complete programming language, ie. any computable function (algorithm) can be expressed as a λ-term

The λ-Calculus

The syntax of the lambda calculus is defined inductively as follows

Definition (Syntax)

$$t := c \mid x \mid t_1 t_2 \mid \lambda x.t_1$$

Where:

- c ranges over a set of constants C and v a set of variables V
- $t_1 t_2$ represents functional application of the left term to the right term
- $\lambda x.t_1$ abstracts the variable x on the term t_1

Need to define a way to evaluate λ-terms to have notion of computation. Most important rule is β-reduction:

$$(\beta) \quad (\lambda x.M)N \equiv M[N/x]$$

The λ-Calculus

The syntax of the lambda calculus is defined inductively as follows

Definition (Syntax)

$$t := c \mid x \mid t_1 t_2 \mid \lambda x.t_1$$

Where:

- c ranges over a set of constants C and v a set of variables V
- $t_1 t_2$ represents functional application of the left term to the right term
- $\lambda x.t_1$ abstracts the variable x on the term t_1

Need to define a way to evaluate λ-terms to have notion of computation. Most important rule is β-reduction:

$(\beta) \quad (\lambda x.M)N \equiv M[N/x]$

The Y-Combinator

Consider the lambda term $\lambda f.(\lambda v.f(vv))(\lambda v.f(vv))$ (don't worry about parsing this). We denote this as simply Y.
It turns out this term is a fixed point combinator, that is:

Theorem

For any term f, $f(Yf) = Yf$

This property allows us to define recursive functions, a crucial property of any programming language

Order Theory

Order theory studies sets equipped with ordering on their elements

Definition (Partially Ordered Set)

$(P, \preceq)$ is a partially ordered set (poset) iff P is a set and $\preceq$ is a reflexive, transitive, and antisymmetric binary relation over P.
Example - $(\mathbb{N}, \leq)$

We will usually just write $(P, \preceq)$ as P

Definition (Least Upper Bound)

Let $(P, \leq)$ be a poset. For a subset $X \subseteq P$, the least upper bound (supremum) of X, denoted $\bigsqcup X$, is an element $x \in P$ such that:

- $\forall y \in X,\ y \leq x$
- $\forall z \in P$, if $\forall y \in X,\ y \leq z$, then $x \leq z$

Order Theory

Order theory studies sets equipped with ordering on their elements

Definition (Partially Ordered Set)

$(P, \preceq)$ is a partially ordered set (poset) iff P is a set and $\preceq$ is a reflexive, transitive, and antisymmetric binary relation over P.
Example - $(\mathbb{N}, \leq)$

We will usually just write $(P, \preceq)$ as P

Definition (Least Upper Bound)

Let $(P, \leq)$ be a poset. For a subset $X \subseteq P$, the least upper bound (supremum) of X, denoted $\bigsqcup X$, is an element $x \in P$ such that:

- $\forall y \in X,\ y \leq x$
- $\forall z \in P$, if $\forall y \in X,\ y \leq z$, then $x \leq z$

Order Theory

Order theory studies sets equipped with ordering on their elements

Definition (Partially Ordered Set)

$(P, \preceq)$ is a partially ordered set (poset) iff P is a set and $\preceq$ is a reflexive, transitive, and antisymmetric binary relation over P.
Example - $(\mathbb{N}, \leq)$

We will usually just write $(P, \preceq)$ as P

Definition (Least Upper Bound)

Let $(P, \leq)$ be a poset. For a subset $X \subseteq P$, the least upper bound (supremum) of X, denoted $\bigsqcup X$, is an element $x \in P$ such that:

- $\forall y \in X,\ y \leq x$
- $\forall z \in P$, if $\forall y \in X,\ y \leq z$, then $x \leq z$

Order Theory

We now define our structure preserving maps

Definition (Monotonicity)

Let P, Q be posets. A map $f : P \to Q$ is monotone iff $x \leq_P y \implies f(x) \leq_Q f(y)$ for all $x, y \in P$

Monotone = Order-Preserving

Order Theory

We now define our structure preserving maps

Definition (Monotonicity)

Let P, Q be posets. A map $f : P \to Q$ is monotone iff $x \leq_P y \implies f(x) \leq_Q f(y)$ for all $x, y \in P$

Monotone = Order-Preserving

Order Theory

We now define our structure preserving maps

Definition (Monotonicity)

Let P, Q be posets. A map $f : P \to Q$ is monotone iff $x \leq_P y \implies f(x) \leq_Q f(y)$ for all $x, y \in P$

Monotone = Order-Preserving

Category Theory

To move forward in building a model of the λ-calculus we introduce some category theory

Definition (Category)

A category $\mathfrak{C}$ consists of a collection of objects $\mathrm{Ob}(\mathfrak{C})$, and for any $A, B \in \mathrm{Ob}(\mathfrak{C})$, a set $\mathfrak{C}(A, B)$ of *morphisms* from A to B, satisfying the following axioms:

- For each $C \in \mathrm{Ob}(\mathfrak{C})$, there is a morphism $Id_C \in \mathfrak{C}(C, C)$
- For each $A, B, C \in \mathrm{Ob}(\mathfrak{C})$, there is an *associative* composition map $\circ : \mathfrak{C}(A, B) \times \mathfrak{C}(B, C) \to \mathfrak{C}(A, C)$
- For any $A, B \in \mathfrak{C}, f \in \mathfrak{C}(A, B)$,

$$Id_C \circ f = f = f \circ Id_A$$

Category Theory

To move forward in building a model of the λ-calculus we introduce some category theory

Definition (Category)

A category $\mathfrak{C}$ consists of a collection of objects $\mathrm{Ob}(\mathfrak{C})$, and for any $A, B \in \mathrm{Ob}(\mathfrak{C})$, a set $\mathfrak{C}(A, B)$ of *morphisms* from A to B, satisfying the following axioms:

- For each $C \in \mathrm{Ob}(\mathfrak{C})$, there is a morphism $Id_C \in \mathfrak{C}(C, C)$
- For each $A, B, C \in \mathrm{Ob}(\mathfrak{C})$, there is an *associative* composition map $\circ : \mathfrak{C}(A, B) \times \mathfrak{C}(B, C) \to \mathfrak{C}(A, C)$
- For any $A, B \in \mathfrak{C}, f \in \mathfrak{C}(A, B)$,

$$Id_C \circ f = f = f \circ Id_A$$

Examples of Categories

- Set is a category in which objects are sets and morphisms are functions between sets
- Pos is a category in which objects are partially ordered sets and morphisms are monotone maps.
- Grp is a category in which objects are groups and morphisms are group homomorphisms

Examples of Categories

- Set is a category in which objects are sets and morphisms are functions between sets
- Pos is a category in which objects are partially ordered sets and morphisms are monotone maps.
- Grp is a category in which objects are groups and morphisms are group homomorphisms

Examples of Categories

- Set is a category in which objects are sets and morphisms are functions between sets
- Pos is a category in which objects are partially ordered sets and morphisms are monotone maps.
- Grp is a category in which objects are groups and morphisms are group homomorphisms

Examples of Categories

- Set is a category in which objects are sets and morphisms are functions between sets
- Pos is a category in which objects are partially ordered sets and morphisms are monotone maps.
- Grp is a category in which objects are groups and morphisms are group homomorphisms

ω-Complete Partial Orders

We now introduce a category in which we can construct a model of the untyped λ-calculus.

Definition (ω-Completeness)

A partially ordered set P is ω-complete if every increasing chain $C \subseteq P$ has a least upper bound in P.

Definition (Continuous Maps)

A map $f : P \to Q$ is continuous iff it is monotone and preserves suprema of increasing chains, ie. $f(\bigsqcup C) = \bigsqcup f(C)$

Definition (Ω-Pos)

The category of ω-complete partial orders and continous maps is denoted Ω-Pos

ω-Complete Partial Orders

We now introduce a category in which we can construct a model of the untyped λ-calculus.

Definition (ω-Completeness)

A partially ordered set P is ω-complete if every increasing chain $C \subseteq P$ has a least upper bound in P.

Definition (Continuous Maps)

A map $f : P \to Q$ is continuous iff it is monotone and preserves suprema of increasing chains, ie. $f(\bigsqcup C) = \bigsqcup f(C)$

Definition (Ω-Pos)

The category of ω-complete partial orders and continous maps is denoted Ω-Pos

ω-Complete Partial Orders

We now introduce a category in which we can construct a model of the untyped λ-calculus.

Definition (ω-Completeness)

A partially ordered set P is ω-complete if every increasing chain $C \subseteq P$ has a least upper bound in P.

Definition (Continuous Maps)

A map $f : P \to Q$ is continuous iff it is monotone and preserves suprema of increasing chains, ie. $f(\bigsqcup C) = \bigsqcup f(C)$

Definition (Ω-Pos)

The category of ω-complete partial orders and continous maps is denoted Ω-Pos

Closure Properties of Ω-Pos

The category Ω-Pos is closed under just the right operations to make it a suitable denotational semantics

Theorem

If P, Q are ω-complete posets, then the set of all continuous maps $f : P \to Q$, denoted $P \to Q$, is a ω-complete poset

In categorical terms, the function space $P \to Q$ is called an **exponential object**, *and this theorem states that Ω-Pos* **has exponentials**

Theorem

If P, Q are ω-complete posets, their cartesian product $P \times Q$ ordered pairwise is an ω-complete poset

In categorical terms, Ω-Pos **has products**

A category with exponentials and products is called *cartesian closed*

Closure Properties of Ω-Pos

The category Ω-Pos is closed under just the right operations to make it a suitable denotational semantics

Theorem

If P, Q are ω-complete posets, then the set of all continuous maps $f : P \to Q$, denoted $P \to Q$, is a ω-complete poset

In categorical terms, the function space $P \to Q$ is called an **exponential object**, *and this theorem states that Ω-Pos* **has exponentials**

Theorem

If P, Q are ω-complete posets, their cartesian product $P \times Q$ ordered pairwise is an ω-complete poset

In categorical terms, Ω-Pos **has products**

A category with exponentials and products is called *cartesian closed*

Closure Properties of Ω-Pos

The category Ω-Pos is closed under just the right operations to make it a suitable denotational semantics

Theorem

If P, Q are ω-complete posets, then the set of all continuous maps $f : P \to Q$, denoted $P \to Q$, is a ω-complete poset

In categorical terms, the function space $P \to Q$ is called an **exponential object**, *and this theorem states that Ω-Pos* **has exponentials**

Theorem

If P, Q are ω-complete posets, their cartesian product $P \times Q$ ordered pairwise is an ω-complete poset

In categorical terms, Ω-Pos **has products**

A category with exponentials and products is called *cartesian closed*

Tarski's Fixpoint Theorem

Definition (Fixed Point)

A fixed point of a function $f : X \to X$ is an element $x \in X$ such that $f(x) = x$

Definition (Least Fixed Point)

$x \in X$ is a least fixed point if $x \leq x'$ for all fixed points x'

Theorem (Tarski)

Let $f \in \Omega\text{-}Pos(P, P)$ where P has a least element $\bot$. Then f has a least fixed point $fix(f) = \bigsqcup\{f^n(\bot) \mid n \in \mathbb{N}\}$

As mentioned with the Y-combinator, the existence of fixpoints allows for recursive functions. Thus in a model of a programming language, we'd have least fixed points correspond to recursive algorithms.

Tarski's Fixpoint Theorem

Definition (Fixed Point)

A fixed point of a function $f : X \to X$ is an element $x \in X$ such that $f(x) = x$

Definition (Least Fixed Point)

$x \in X$ is a least fixed point if $x \leq x'$ for all fixed points x'

Theorem (Tarski)

Let $f \in \Omega\text{-}Pos(P, P)$ where P has a least element $\bot$. Then f has a least fixed point $fix(f) = \bigsqcup\{f^n(\bot) \mid n \in \mathbb{N}\}$

As mentioned with the Y-combinator, the existence of fixpoints allows for recursive functions. Thus in a model of a programming language, we'd have least fixed points correspond to recursive algorithms.

Tarski's Fixpoint Theorem

Definition (Fixed Point)

A fixed point of a function $f : X \to X$ is an element $x \in X$ such that $f(x) = x$

Definition (Least Fixed Point)

$x \in X$ is a least fixed point if $x \leq x'$ for all fixed points x'

Theorem (Tarski)

Let $f \in \Omega\text{-}Pos(P, P)$ where P has a least element $\bot$. Then f has a least fixed point $fix(f) = \bigsqcup\{f^n(\bot) \mid n \in \mathbb{N}\}$

As mentioned with the Y-combinator, the existence of fixpoints allows for recursive functions. Thus in a model of a programming language, we'd have least fixed points correspond to recursive algorithms.

Tarski's Fixpoint Theorem

Definition (Fixed Point)

A fixed point of a function $f : X \to X$ is an element $x \in X$ such that $f(x) = x$

Definition (Least Fixed Point)

$x \in X$ is a least fixed point if $x \leq x'$ for all fixed points x'

Theorem (Tarski)

Let $f \in \Omega\text{-}Pos(P, P)$ where P has a least element $\bot$. Then f has a least fixed point $fix(f) = \bigsqcup\{f^n(\bot) \mid n \in \mathbb{N}\}$

As mentioned with the Y-combinator, the existence of fixpoints allows for recursive functions. Thus in a model of a programming language, we'd have least fixed points correspond to recursive algorithms.

Tarski's Fixpoint Theorem

Definition (Fixed Point)

A fixed point of a function $f : X \to X$ is an element $x \in X$ such that $f(x) = x$

Definition (Least Fixed Point)

$x \in X$ is a least fixed point if $x \leq x'$ for all fixed points x'

Theorem (Tarski)

Let $f \in \Omega\text{-}Pos(P, P)$ *where* P *has a least element* $\bot$*. Then* f *has a least fixed point* $fix(f) = \bigsqcup\{f^n(\bot) \mid n \in \mathbb{N}\}$

As mentioned with the Y-combinator, the existence of fixpoints allows for recursive functions. Thus in a model of a programming language, we'd have least fixed points correspond to recursive algorithms.

LFPs for Denotational Semantics

- Recall Ω-Pos is cartesian closed; cartesian closed categories are known to be models of typed λ-calculi
- In a domain model of a computer program, the least element $\perp$ can be used to represent non-terminating computations.
- Recursive functions correspond to fixed points of continuous maps on a ω-CPO
- Continuity and approximation in domains allows one to capture fundamental aspects of computability like iterative computation and partial functions.

Thank You!