Notes on Programming Language Concepts and More

Apiros3

First Version : Jan 29, 2025
Last Update : Jan 29, 2025

Contents

1 Introduction

Much of these notes come from a variety of existing notes.

2 Basics of Category Theory

We cover basic definitions for category theory such that the later sections become clear.

2.1 Categories and Commutative Diagrams

Definition 2.1.1 *A **cateogy** C consists of*

- *a class* $\mathrm{ob}(C)$ *of **objects***
- *a class* $\mathrm{mor}(C)$ *of **morphisms** or **arrows***
- *a **domain** or **source** class function* $\mathrm{dom} : \mathrm{mor}(C) \to \mathrm{ob}(C)$
- *a **codomain** or **target** class function* $\mathrm{cod} : \mathrm{mor}(C) \to \mathrm{ob}(C)$
- *for any three* $a, b, c \in \mathrm{ob}(C)$*, a binary operation of type* $\hom(a, b) \times \hom(b, c) \to \hom(a, c)$ *called **composition of morphisms**.*

Note here that $\hom(a, b)$ *denotes the subclass of morphisms* f *in* $\mathrm{mor}(C)$ *such that* $\mathrm{dom}(f) = a$ *and* $\mathrm{cod}(f) = b$*. Morphisms in this subclass are written* $f : a \to b$*, and composition is written* $g \circ f$ *or* gf*.*

- *associativity : if* $f : a \to b$*,* $g : b \to c$*,* $h : c \to d$*, then* $h \circ (g \circ f) = (h \circ g) \circ f$*.*
- *for every* $x \in \mathrm{ob}(C)$*, there exists a morphism* id_x *such that for all* $f : a \to x$*,* $id_x \circ f = f$ *and for all* $g : x \to b$*,* $g \circ id_x = g$*.*

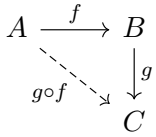
Example 2.1.2 *Some examples of cateogories:*

- *The class of all sets together with functions between them as morphisms and the canonical function composition forms the category **Set***
- *The category **Rel** of all sets with binary relations* $R \subseteq A \times B$ *between them, with composition of two relations* $R : A \to B, S : B \to C$ *given by* $(a, c) \in S \circ R \iff \exists\, b \in B, (a, b) \in R$ *and* $(b, c) \in S$*.*
- *Given a set* I*, the **discrete** category is the category where elements of* I *are objects and the only morphisms are the identity morphisms*
- *Any preordered set* $(P, \leq)$ *forms a category, where objects are members of* P *and morphisms are arrows from* x *to* y *when* $x \leq y$*. If* $\leq$ *is antisymmetric, there can be at most one morphism between any two objects.*
- *Any monoid is a category for a fixed set* x*, where morphisms from* x *to* x *corresponds to the elements of the monoid and composition based on what the monoid operation sends the two elements to, and identity morphism is the identity of the monoid.*
- *A group is a category with a single object in which every morphism is invertible; every morphism has a morphism that is both left and right inverses under composition.*
- *Any directed graph (augmented with self-loops) with composition by concatenation of paths forms a category.*

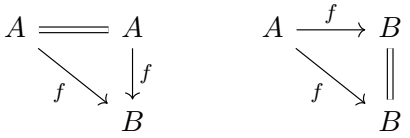
Definition 2.1.3 *Given a category C, the **opposite category** or **dual category** written C^{op} is a category formed by reversing the morphisms in C. Note here that $(C^{\text{op}})^{\text{op}} = C$. To be explicit, $\hom_C(A, B) = \hom_{C^{\text{op}}}(B, A)$.*

Notation 2.1.4 *We will often make use of commutative diagrams to represent connections between maps.*

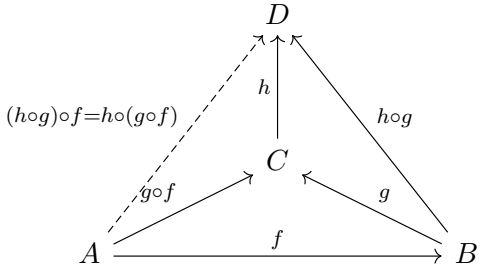
We use a dashed line to denote that it is the unique arrow that makes the map commute.



Moreover, we use a double line to indicate identities:



Remark 2.1.5 *Using the notation above, we can represent what it means for a map to be associative by the following commutative diagram:*

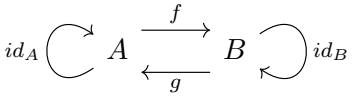


Definition 2.1.6 *A **terminal object** $\mathbf{1}$is an object such that there is exactly one morphism from any other object. That is,*

$$A \dashrightarrow^{!_A} \mathbf{1}$$

*For instance, one element sets are terminal objects in **Set**.*

Definition 2.1.7 *Two objects A, B are said to be **isomorphic** if there are morphisms $f : A \to B$ and $g : B \to A$ such that $f \circ g = id_B$ and $g \circ f = id_A$.*



*We call the morphisms f, g to be **isomorphisms**. We write $f : A \cong B$, or $A \cong B$ to denote that A and B are isomorphic.*

Example 2.1.8 *Given two terminal objects, the unique morphisms between them are isomorphisms. That is, terminal objects are equivalent up to isomorphism.*

Definition 2.1.9 *A **initial object** $\mathbf{0}$ is an object where there is a unique morphism from it to any other object.*

$$\mathbf{0} \xrightarrow{?_A} A$$

*For instance, in **Set**, the empty set is the initial object.*

Note that initial objects are unique up to isomorphism. This is clear from the fact that there is a unique map between any two initial object and also to itself, so composition is forced to be the identity map.

Also, by definition, initial objects are terminal objects in the opposite category, and vice versa a terminal object is an initial object in the opposite category.

Example 2.1.10 *Consider the category where objects are natural numbers and morphisms are unique maps between numbers m, n such that $m \leq n$ (Laws follow as it is a preorder relation). Now consider the category with one object and its morphisms are the natural numbers, with composition given by addition (Laws follow as it is a monoid). These are two categories where in one the objects are $\mathbb{N}$ and in the other the morphisms correspond to $\mathbb{N}$.*

Definition 2.1.11 *A category is said to be a **groupoid** if for every morphism $f : A \to B$ there is an inverse $f^{-1} : B \to A$ such that $f \circ f^{-1} = id$ and $f^{-1} \circ f = id$*

In this sense, a groupoid with one object is a group. Furthermore, a groupoid where the homsets are propositions forms an equivalence relation. As another example, a category where objects are sets and morphisms are isomorphisms between the sets forms a groupoid.

We have showcased how a set equipeed with a structure is a method of creating new categories. For instance, the category of preorders **Pre** is one where the objects are preorders (that is, a set A with a relation $R : A \to A \to \mathbf{Prop}$ where R is reflexive and transitive.) Given two preorders (A, R) and (B, S), a morphism is a function $f : A \to B$ which preserves relations such that $R\ x\ y$ implies $S\ (f\ x)\ (f\ y)$. Identity and composition takes the same structure from **Set**.

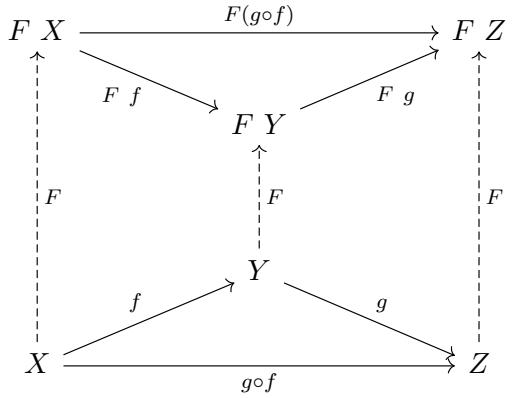
Another instance is the category of monoids **Mon**, where objects are monoids (that is, a set A with a binary operation $\_ * \_$ where the monoid laws hold). Then, the morphisms between objects are the structure preserving maps in the sense that given a function $f : A \to B$ and two monoids $(A, e, \_ * \_), (B, e', \_ *' \_)$, we have $f\ e = e'$, $f\ (x * y) = (f\ x) *' (f\ y)$.

2.1.1 Functors

Definition 2.1.12 *Let C, D be categories. A **functor** F from C to D is a mapping such that*

- *it associates each $X \in \mathrm{ob}(C)$ to $F\ X \in \mathrm{ob}(D)$*
- *it associates each morphism $f : X \to Y \in \mathrm{mor}(C)$ to a morphism $F\ f : F\ X \to F\ Y \in \mathrm{mor}(D)$ such that*
  - *for all $X \in \mathrm{ob}(C), F\ id_X = id_{F\ X}$*
  - *for all $f : X \to Y, g : Y \to Z \in \mathrm{mor}(C)$, $F\ (g \circ f) = (F\ g) \circ (F\ f)$*

That is, it preserves identity morphisms and composition of morphisms.



The canonical example for a functor in computer science is the List functor, which is an endofunctor on the category of sets, which we write List : **Set** → **Set**. The list functor has an effect on objects, where given A : **Set** we obtain List A : **Set**, which is the set of lists over A, written as $[a_0, a_1, \ldots, a_n]$ including the empty list $[\,]$. This functor also is equipped with a maps a function $f : A \to B$ to List f : List $A \to$ List B, defined as

$$\text{List } f[a_0, a_1, \ldots, a_n] = [f\ a_0, f\ a_1, \ldots, f\ a_n]$$

Notice that this preserves the categorical structure, mapping the identity to the identity in the sense that List $id_A = id_{\text{List } A}$ and List $(f \circ g) = (\text{List } f) \circ (\text{List } g)$, which can be quickly proved by an induction on the structure of the list.

Another example is the powerset functor $\mathcal{P}$: **Set** → **Set**. It maps objects $X \mapsto \{S : S \subseteq X\}$, and morphisms $f : X \to Y$ to $\mathcal{P}\ f : \mathcal{P}\ X \to \mathcal{P}\ Y$ which sends $S \subseteq X$ to $f\ S \subseteq Y$. This clearly satisfies our condition to be a functor.

Proposition 2.1.13 *Every functor preserves isomorphisms. That is, if $\phi : A \to B$ is an isomorphism, so is $F\ \phi : F\ A \to F\ B$.*

Proof. Suppose that $\phi : A \to B$ is an isomorphism with an inverse ψ. Then, we have

$$(F\ \phi) \circ (F\ \psi) = F\ (\phi \circ \psi) = F\ id_B = id_{F\ B}$$

$$(F\ \psi) \circ (F\ \phi) = F\ (\psi \circ \phi) = F\ id_A = id_{F\ A}$$

It follows that $F\ \phi$ has an inverse $F\ \psi$, showing it is an isomorphism. ■.

Functors may also be composed. That is, given functors

$$C \xrightarrow{T} B \xrightarrow{S} A$$

between categories A, B, C, we can define a functor $S \circ T : C \to A$ with the maps $c \mapsto S(T\ c)$ and $f \mapsto S(T\ f)$. where $c \in \text{ob}(C)$ and $f \in \text{mor}(C)$. Extending from this, for every category C, there is an identity functor $I_C : C \to C$ which acts as an identity for this composition as well.

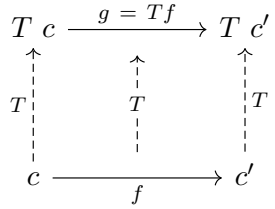
Definition 2.1.14 *Given categories C, C', and a functor $T : C \to C'$, T is an **isomorphism** if T is a bijection both on objects and on arrows.*

Proposition 2.1.15 *Given a functor $T : C \to C'$, T is an isomorphism if and only if there exists a functor $S : C' \to C$ such that $S \circ T$ and $T \circ S$ are both identity functors. We write $S = T^{-1}$ to represent the two-sided inverse.*

Proof. TODO!!!!!!

We further define some weaker but useful notions to functors being isomorphisms.

Definition 2.1.16 *A functor $T : C \to C'$ is **full** if for all $c, c' \in C$, given $g \in \hom(T\,c, T\,c')$, there exists $f : c \to c'$ with $g = Tf$. As a diagram, we find an f that makes the correspondence below:*



Notice that the composition of two full functors gives a full functor.

2.1.2 Natural Transformation

Example 2.1.17 *We can observe that functors give rise to forming categories of categories. For instance,*

- *Functors between preorder categories are monotone maps, which has a correspondence with morphisms in* **Pre**
- *Functors between monoids are monoid morphisms which corresponds to morphisms in* **Mon**.

2.2 Monos and Epis

The notions of injection and surjection can be placed into the language of category theory by utilizing the fact that an injection is precicely when function compositions are left-cancellable and surjections are when they are right-cancellable.

Definition 2.2.1 *A function $i : A \to B$ is a **monomorphism** or **mono** if given any $f, g : C \to A$, if $i \circ f = i \circ g$, then $f = g$. We use a special arrow to represent monos:*

$$A \xrightarrow{i} B$$

Definition 2.2.2 *A function $e : A \to B$ is said to be an **epimorphism** or **epi** for short, if $f \circ e = g \circ e$ implies that $f = g$.*

$$A \xrightarrow{e} B$$

Proposition 2.2.3 *The monos in* **Set** *are exactly the injective functions and the epis are the surjective functions.*

Proof. Note first that the direction injective implies mono and surjective implies epi is immediate from definitions.

To show that mono implies injective, given a monomorphism $f : X \to Y$, suppose we have $x, y \in X$ such that $f(x) = f(y)$. Define functions $g, h : \mathbf{1} \to X$ by $g(*) = x$ and $h(*) = y$. Then, $f(g(*)) = f(h(*))$, so by function extentionality we have $f \circ g = f \circ h$. As f is mono, $g = h$. It follows that $a = b$.

To show that epi implies surjective, given an epimorphism $f : X \to Y$, define $g, h : Y \to \{0, 1\}$ with

$$g(y) = 0 \quad \text{for all } y \in Y \qquad h(y) = \begin{cases} 0 & \text{if } y \in \text{im}(f) \\ 1 & \text{otherwise} \end{cases}$$

Then for any $x \in X$, we see that $g(f(x)) = 0 = h(f(x))$. As f is an epimorphism, $g = h$, which is only the case if $\text{im}(f) = Y$. ■

Proposition 2.2.4 *If $f : X \to Y$ has a left inverse, it is mono. It is has a right inverse, it is epi.*

Proof. Suppose that f has a left inverse $l : Y \to X$ with $l \circ f = id$. Then,

$$\begin{aligned} f \circ g = f \circ h &\implies l \circ (f \circ g) = l \circ (f \circ h) \\ &\implies (l \circ f) \circ g = (l \circ f) \circ h \\ &\implies id \circ g = id \circ h \\ &\implies g = h \end{aligned}$$

Similarly, if f has a right inverse $r : Y \to X$ with $f \circ r = id$, then,

$$\begin{aligned} g \circ f = h \circ f &\implies (g \circ f) \circ r = (h \circ f) \circ r \\ &\implies g \circ (f \circ r) = h \circ (f \circ r) \\ &\implies g \circ id = h \circ id \\ &\implies g = h \end{aligned}$$

■

Corollary 2.2.5 *In* **Set**, *a morphism that is both mono and epi is an isomorphism.*

Proof. By Proposition 2.2.3 that a morphism that is both mono and epi is a bijection. Then, this is an invertible function, and it follows that this forms an isomorphism.

Remark 2.2.6 *Note that the above corollary is not true for a general category. Consider the embedding $i : \mathbb{N} \to \mathbb{Z}$ is a monoid morphism with $i : \mathbf{Mon}((\mathbb{N}, \_ + \_, 0), (\mathbb{Z}, \_ + \_, 0))$. This is clearly mono and epi, but is not surjective.*

However, in any category, if the function has both a left and right inverse, it is always an isomorphism, as it can be shown that the left and right inverses are equal to each other.

2.3 Monads

2.3.1 In Category Theory

A monad is an endofunctor with additional structure (two natural transformations). In pop culture, we often say that a monad is a "monoid in the category of endofunctors" (for some fixed category).

Definition 2.3.1 (Monoid) *A **monoid** is a Set X equipped with an operator $\_ * \_$ which is associative and has an identity element $1_X \in X$ for the $\oplus$ operator.*

Example 2.3.2 *The following are some examples of monoids.*

- *$X = \mathbb{Z}$ with the addition operator, and additive identity 0.*

- $X = \mathbb{R}_{>0}$ *with the multiplication operator, and multiplicative identity 1.*
- $X =$ List A *with concat as the operator, and the empty list as the identity.*

Definition 2.3.3 (Monad) *Let C be a category. A **monad** on C is a triple (T, η, μ) where $T : C \to C$ is an endofunctor, $\eta : id_C \to T$ (where id_C is the identity functor on C), $\mu : T^2 \to T$, satisfying the following coherence conditions:*

- $\mu \circ T\mu = \mu \circ \mu T$ *where $T\mu$ and μT are formed by horizontal composition*
- $\mu \circ T\eta = \mu \circ \eta T = id_T$ *where id_T is the identity transformation from T to T.*

Alternatively, we can write these using two commutative diagrams

$$(i)\quad \begin{array}{ccc} T^3 & \xrightarrow{T\mu} & T^2 \\ {\scriptstyle \mu T}\downarrow & & \downarrow{\scriptstyle \mu} \\ T^2 & \xrightarrow[\mu]{} & T \end{array} \qquad (ii)\quad \begin{array}{ccc} T & \xrightarrow{\eta T} & T^2 \\ {\scriptstyle T\eta}\downarrow & \searrow & \downarrow{\scriptstyle \mu} \\ T^2 & \xrightarrow[\mu]{} & T \end{array}$$

3 Lambda Calculus

4 Definitional Interpreter

In this section we aim to explain the concept and fundamentals of a definitional interpreter under various contexts.

We will base the language to be implemented to be based off the "Fun" programming language as specificed by the Principles of Programming Languages course at Oxford University.

4.1 Defining Fun

The first step in describing a fixed program is to specify the set of legal phrases (the concrete syntax), then by describing in the language that interprets the program a set of trees that capture the structure of legal phrases (the abstract syntax). For the rest of the section, we will form a Haskell datatype that captures the structure of these legal phrases.

At the simplest level, we have a function

```
parse :: String → Phrase
```

where the `String` is any line of text in the concrete syntax, then produces a corresponding tree in the abstract syntax, which we give the type `Phrase`.

Often the abstract syntax is much more simple than the concrete syntax, as the concrete syntax allows for convinient abbreviations (syntactic sugars).

Basic fun does not have type-checking, so we regard the set of valid expressions to be produced by a context free grammar, rejecting those which do not "make sense".

4.1.1 Abstract Syntax

The abstract syntax of a language can be expressed as a collection of mutually dependent datatype definitions. In Fun, there is `Expr` for expressions, `Defn` for definitions, and `Phrase` for top level phrases.

```
data Expr =
      Number Integer
    | Variable Ident
    | Apply Expr [Expr]
    | If Expr Expr Expr
    | Lambda [Ident] Expr
    | Let Defn Expr
```

Note that we can have functions with no arguemnts.

The definitions that appear after `let` also appear in the abstract syntax as

```
data Defn =
      Val Ident Expr
    | Rec Ident Expr
```

which correspond to giving variables denoted by `Ident` expressions in `Expr`.

In this way, the concrete form val $x(x_1, \ldots, x_n)$ e is syntactic sugar for val x = lambda$(x_1, \ldots, x_n)$ e. We use empty "()" if the function has no inputs, and the constructor `Rec` is for a definition that starts with a lambda (but is not enforced at the datatype level).

The top-level phrase that is typed in the prompt (or included as code in fun) is either an expression which is to evaluated, or a definition to be added into the environment. So,

```
data Phrase =
      Calculate Expr
    | Define Defn
```

Remark 4.1.1 *In the abstract syntax for fun, identifiers (Ident) are represented by strings. This limits efficiencies, and in more optimized languages have indexing into a global list of identifiers (and thus can avoid string comparisons).*

4.1.2 Interpreter for Fun

The main component of an interpreter is a function `eval` which takes an abstract syntax tree with an environment and turns it into a value of that expression. Specifically,

```
eval :: Expr → Env → Value
```

where `Env` is the type of environments which mapps identifiers to values, with `Value` representing possible values computed by Fun programs.

At the simplest level, values are denoted by

```
data Value =
      IntVal Integer
    | BoolVal Bool
    | Nil
    | Cons Value Value
```

5 Domain Theory

This section aims to cover concepts in domain theory, which is a study on a special kind of posets. There is a great connection between the concepts in domain theory to areas like denotational semantics (the motivation of this field comes initially from Dana Scott's search for denotational semantics of the lambda calculus). We can then equip notions like approximations and convergence by defining a suitable topology in which it models our intuition. Hence, this section has close ties with concepts including category theory, functional programming, and topology. Notes here are mainly based off of the following:

- Barendregt, Syntax and Semantics, Chapter 1
- Abramsky and Jung, Domain Theory
- Winskel, The Formal Semantics of Programming Languages, Chapter 8

5.1 Partial Orders, Suprema, and Continuity

Definition 5.1.1 *A set P with a binary relation $\sqsubseteq$ is a partially ordered set (or poset) if the operation is reflexive, transitive, and antisymmetric.*

Definition 5.1.2 *Given a partially ordered set D, we define a **directed set** to be a nonempty $S \subseteq D$ such that every pair of elements of S has an upper bound in D. That is, for any $x, y \in D$, there exists a $z \in D$ such that $x \sqsubseteq z$ and $y \sqsubseteq z$.*

For instance, a chain, which is a non-empty subset which are totally ordered are directed sets. The chain of natural numbers with their natural orders is indeed a chain, and subsets of posets which are isomorphic to these are called ω-chains.

Definition 5.1.3 *Let D be a partially ordered set (poset) with order $\sqsubseteq$. We define the **least upper bound** (lub) or **supremum** of a subset $S \subseteq D$ denoted $\bigsqcup S$ if it satisfies:*

- *$\bigsqcup S$ is greater than or equal to every element in S; $\forall x \in S,\ x \sqsubseteq \bigsqcup S$*
- *$\bigsqcup S$ is the smallest such element; $\forall y \in D, (\forall x \in S, x \sqsubseteq y) \implies (\bigsqcup S \sqsubseteq y)$*

*We give the dual notion (greatest lower bound) as the **infimum** with symbol $\bigsqcap$.*

Notation 5.1.4 *We write $\bigsqcup^{\uparrow} A$ to denote $\bigsqcup A$ if A is directed and has a supremum.*

Notation 5.1.5 *Given a partially ordered set $(D, \sqsubseteq)$ and a set $X \subseteq D$, we write*

- *$\uparrow X = \{d \in D \mid \exists x \in X, x \sqsubseteq d\}$*
- *$\downarrow X = \{d \in D \mid \exists x \in X, d \sqsubseteq x\}$*

Where it is clear, we may write $\uparrow x$ to represent $\uparrow \{x\}$ and similarly for $\downarrow x$.

Definition 5.1.6 *We call a set to be an **upper set** if $\uparrow X = X$. If $\downarrow X = X$, then we say that X is a **lower set**. Furthermore, we call an element $x \in D$ to be maximal if there are no elements above it. That is, $\uparrow x \cap D = \{x\}$. An element $x \in D$ is similarly minimal if $\downarrow x \cap D = \{x\}$.*

Proposition 5.1.7 *Let D be a poset such that the following suprema and infima exist. Then,*

1. $A \subseteq B$ *implies* $\bigsqcup A \sqsubseteq \bigsqcup B$ *and* $\bigsqcap B \sqsubseteq \bigsqcap A$
2. $\bigsqcup A = \bigsqcup(\downarrow A)$ *and* $\bigsqcap A = \bigsqcap(\uparrow A)$
3. $A = \bigcup_{i \in I} A_i$ *implies* $\bigsqcup A = \bigsqcup_{i \in I}(\bigsqcup A_i)$ *and similarly for the infimum*

Proof. The first two cases are obvious. For the third case, note that $\bigsqcup A$ is each above $\bigsqcup A_i$ so by taking the least upper bound, it follows that $\bigsqcup_{i \in I}(\bigsqcup A_i) \sqsubseteq \bigsqcup A$. Conversely, each $a \in A$ is in A_i so is below $\bigsqcup A_i$, which is below $\bigsqcup_{i \in I} \bigsqcup A_i$. Thus, this is an upper bound for A, and $\bigsqcup A$ is the least, giving $\bigsqcup A \sqsubseteq \bigsqcup_{i \in I}(\bigsqcup A_i)$. ■

This means that we are allowed to take supremums of parts of a set and take the whole later, so long as the A_i's cover the entire set.

Definition 5.1.8 *We say that a directed lower set is an* ***ideal****. We say it is principal if it is of the form* $\downarrow x$*. The dual notion is filtered set and principal filter.*

Remark 5.1.9 *As* $\sqsubseteq$ *is reflexive, we have* $X \subseteq \uparrow X$*. Therefore, to prove* X *is an upper set, it suffices to show that* $\uparrow X \subseteq X$*.*

Definition 5.1.10 *We say that a partial ordered set* $(D, \sqsubseteq)$ *is an* ***ω-complete partial order*** *(ω-cpo) if every countable ascending chain* $(d_0 \sqsubseteq d_1 \sqsubseteq \dots)$ *has a least upper bound, written* $\bigsqcup_{n \geq 0} d_n$*. Additionally, we say* $(D, \sqsubseteq)$ *is a cpo* ***with bottom*** *or is* ***pointed****, if it has a least element* $\bot \in D$ *(over* $\sqsubseteq$*).*

Definition 5.1.11 *We say that poset* D *is an* ***directed-complete partial order*** *(dcpo) if every directed subset* $X \subseteq D$ *has a supremum. It is also referred to as the* ***up-complete poset****. We write* $\bigsqcup X$ *for the suprema.*

In general, it is difficult to make a non-dcpo into a dcpo. For instance, the natural numbers by the usual order does not form a dcpo (as it has no supremum). On the other hand, every finite poset is a dcpo.

Definition 5.1.12 *A poset* D *is a* ***pointed-directed-complte partial order (pointed dcpo)*** *or* ***ccpo*** *is a dcpo with a least element* $\bot \in D$*. Alternatively, it is a poset which has a supremum for every directed or empty subset.*

When we refer to a "cpo" in a proof, we usually mean dcpo, and when $\bot$ is mentioned, a ccpo.

Definition 5.1.13 *A poset* D *is a* ***$\bigsqcup$-semilattice*** *if the supremum for each pair of elements exists.*

Remark 5.1.14 *We dually give notions for infimum as the greatest lower bound, using the* $\bigsqcap$ *notation. Similar notions apply for definitions like* $\bigsqcap$*-semilattice.*

Definition 5.1.15 *A* ***complete lattice*** *is a poset* D *where every* $X \subseteq D$ *has a supremum and infimum (where infimum is defined similarly).*

Remark 5.1.16 *We can clearly see that a ccpo is a dcpo is a ω-cpo.*

It is also important to distinguish between complete partial orders (including dcpo and ccpo) and complete lattices. Notably, we don't force every subset to have a suprema. A complete lattice is a cpo with bottom, as $\bot = \bigsqcup \emptyset$*, but not vice versa.*

Proposition 5.1.17 *A poset D is a dcpo if and only if each chain in D has a supremum.*

Proof. Uses Axiom of Choice. Proof is out of scope of notes but can be found in [2].

Definition 5.1.18 *Given cpos D, D', we say that a function $f : D \to D'$ is **monotonic** if*

$$\forall d, d' \in D, d \sqsubseteq d' \implies f(d) \sqsubseteq f(d')$$

Remark 5.1.19 *The set $[P \xrightarrow{m} Q]$ of all monotone functions betwene posets ordered pointwise give rise to another poset, the **monotone function space** between P and Q.*

Proposition 5.1.20 *Let A be a non-empty subset of a $\bigsqcup$-semilattice for which $\bigsqcup A$ exists. Then,*

$$\bigsqcup{}^{\uparrow} \{ \bigsqcup M \mid M \subseteq A \text{ finite and non-empty} \}$$

Proof. Note that by Proposition 5.1.7, as the set is a cover for A, the suprema are equal. We now need to show that the internal set is directed as $(\bigsqcup A) \sqcup (\bigsqcup B) = \bigsqcup(A \cup B)$ and $A \cup B$ is finite if A and B are both finite. ■

Definition 5.1.21 *Let D be an cpo and $f : D \to D$. We say that $d \in D$ is a **fixed point** of f if $f(d) = d$. We say it is a **prefixed point** if $f(d) \sqsubseteq d$*

Proposition 5.1.22 *If D is a complete lattice then every monotone $f : D \to D$ has a fixpoint. The least is*

$$\bigsqcap \{ x \in D \mid f(x) \sqsubseteq x \}$$

The largest with

$$\bigsqcup \{ x \in D \mid x \sqsubseteq f(x) \}$$

Proof. For the least fixed point, let $X = \{x \in D \mid f(x) \sqsubseteq x\}$. Now take $d = \bigsqcap X$. Now, for each $x \in X$, we have $d \sqsubseteq x$ and $f(d) \sqsubseteq f(x) \sqsubseteq x$. By taking the infimum, we get $f(d) \sqsubseteq \bigsqcap f(X) \sqsubseteq \bigsqcap A = d$. As f is monotonic, $d \sqsubseteq f(d)$, so $d = f(d)$.

For the greatest fixed point, let $x = \bigsqcup\{x \in D \mid x \sqsubseteq f(x)\}$. If $f(x) \not\sqsubseteq x$, $f(x)$ is strictly greater than x, but $f(x) \sqsubseteq f(f(x))$ so $f(x) \in \{x \in D \mid x \sqsubseteq f(x)\}$. ■

Proposition 5.1.23 *A pointed poset D is a dcpo if and only if every monotone map on D has a fixpoint.*

Follows from the proof structure used in Proposition 5.1.17 and 5.1.22. ■

From here onwards, unless explicitly stated otherwise, we shorthand $D = (D, \sqsubseteq), D' = (D', \sqsubseteq)$, which will range over cpos and interpreted in the Scott topology.

Proposition 5.1.24 *Given D, D', define $D \times D'$ to be the cartesian product partially ordered by*

$$\langle x, x' \rangle \sqsubseteq \langle y, y' \rangle \quad \text{iff} \quad x \sqsubseteq y \text{ and } x' \sqsubseteq' y'$$

Then, $D \times D'$ is a cpo with for any directed $X \subseteq D \times D'$,

$$\bigsqcup{}^{\uparrow} X = \langle \bigsqcup{}^{\uparrow} X_0, \bigsqcup{}^{\uparrow} X_1 \rangle$$

where

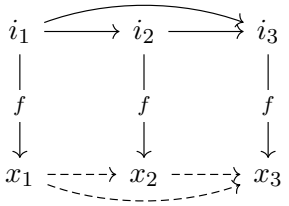
$$X_0 = \{x \in D : \exists x' \in D', \langle x, x' \rangle \in X\}$$
$$X_1 = \{x' \in D' : \exists x \in D, \langle x, x' \rangle \in X\}$$

Proof. First note that if $X \subseteq D \times D'$ is directed, so are X_0 and X_1. It can then be seen that these both must have defined suprema, thus $\bigsqcup^{\uparrow} X$ is defined. To show this is actually the suprema follows trivially from construction. ■

Definition 5.1.25 *A **monotone net** in a poset D is a monotone function f from a directed set I into D. The set I is called the **index set** of the net.*

Let $f : I \to D$ be a monotone net. If we are given a monotone function $f' : J \to I$ where J is directed and for all $i \in I$ there is $j \in J$ with $i \sqsubseteq f'(j)$, we say that $f \circ f'$ to be a subnet of f.

We say that a monotone net $f : I \to D$ has a supremum in D if the set $\{f(i) \mid i \in I\}$ has a supremum in D.



Of course, every directed set as an embedding can be seen as a monotone net. On the other hand, the image of a monotone net is a directed set in D.

Lemma 5.1.26 *Let D be a poset and $f : I \to D$ be a monotone net. Then f has a subnet $f \circ f' : J \to D$ whose index J is a lattice in which every principal ideal is finite.*

Proof. Let J be the set of finite subsets of I. Note that J is a lattice in which every principal ideal is finite (bounded by the empty set). Define a map $f' : J \to I$ by induction by the cardinality of elements of J, by

$$f'(\emptyset) = \text{any element of } I$$
$$f'(A) = \text{any upper bound of } A \cup \{f'(B) \mid B \subsetneq A\}$$

Then, f' is monotone by construction. It is a subnet as $A \sqsubseteq f'(A)$. ■

Proposition 5.1.27 *Given a directed I and $f : I \times I \to D$ to be a monotone net, if any of the following suprema exists then then the following equality holds:*

$$\bigsqcup\nolimits^{\uparrow}_{i,j \in I} f(i,j) = \bigsqcup\nolimits^{\uparrow}_{i \in I} (\bigsqcup\nolimits^{\uparrow}_{j \in J} f(i,j)) = \bigsqcup\nolimits^{\uparrow}_{j \in J} (\bigsqcup\nolimits^{\uparrow}_{i \in I} f(i,j)) = \bigsqcup\nolimits^{\uparrow}_{i \in I} f(i,i)$$

Proof. TODO!!!!

Remark 5.1.28 *General directed sets may not have enough structure to allow for swapping of $\bigsqcup^{\uparrow}$, as it might not be directed after swaps. Proposition 5.1.27 is making the claim that we are allowed to make these swaps under above conditions, and Proposition 5.1.7 says that in the case we are allowed to swap, they are equal. Thus, when we are sure the suprema exists, it is safe to swap $\bigsqcup^{\uparrow}$.*

Definition 5.1.29 *Let D be a partially ordered set over $\sqsubseteq$. We say that a function $f : D \to D'$ is **continuous** (or Scott continuous) if it preserves least upper bounds for directed sets. That is, for every directed set S,*

$$f(\bigsqcup\nolimits^{\uparrow} S) = \bigsqcup\nolimits^{\uparrow} f(S)$$

*A function between pointed dcpos which preserves the bottom element is called **strict**.*

Proposition 5.1.30 *Scott continuous functions are monotonic.*

Proof. Let $f : D \to D'$ over cpos D, D'. Suppose we take $d, d' \in D$ such that $d \sqsubseteq d'$. Then, by continuity,

$$f(d') = f(d \sqcup d') = f(d) \sqcup f(d')$$

It therefore follows from the definition of directed suprema that $f(d) \sqsubseteq f(d')$. ■

5.2 Family of Continuous Functions

Definition 5.2.1 *Fix cpos D and D'. Define*

$$[D \to D'] := \{f : D \to D' \mid f \text{ continuous}\}$$

We equip this with a partial order:

$$f \sqsubseteq g \iff \forall x \in D, f(x) \sqsubseteq' g(x)$$

Note that this gives a well-defined poset. We further write

$$[D \overset{\perp!}{\to} D']$$

to define the set of all continuous strict functions.

Lemma 5.2.2 *Take $\{f_i\}_{i \in I} \subseteq [D \to D']$ be a directed family of maps. Define*

$$f(x) = \bigsqcup_{i \in I}^{\uparrow} f_i(x)$$

Then, f is well-defined and continuous.

Proof. Since $\{f_i\}_{i \in I}$ is directed, $\{f_i(x)\}_{i \in I}$ is directed for any x, meaning f exists. Furthermore, given any directed $X \subseteq D$,

$$f(\bigsqcup^{\uparrow} X) = \bigsqcup_{i \in I}^{\uparrow} \bigsqcup_{x \in X}^{\uparrow} f_i(x) = \bigsqcup_{x \in X}^{\uparrow} \bigsqcup_{i \in I}^{\uparrow} f_i(x) = \bigsqcup^{\uparrow} f(X)$$

where the second equality is well defined as $f_i(x)$ is directed. ■

That is, $[D \to D']$ is a dcpo if D and D' are dcpos.

5.2.1 Into Lambda Calculus

Notation 5.2.3 *Given a function f, we write $\lambda x.f\ x$ to refer to the function who maps $x \mapsto f\ x$.*

Proposition 5.2.4 *$[D \to D']$ is a ccpo with the supremum of a directed $F \subseteq [D \to D']$ given by*

$$(\bigsqcup^{\uparrow} F)(x) = \bigsqcup^{\uparrow} \{f(x) \mid f \in F\}$$

Proof. First note that $\lambda x.\perp'$ is the bottom element of $[D \to D']$. By Lemma 5.2.2, the map $\lambda x.\bigsqcup^{\uparrow}\{f(x) : f \in F\}$ is continuous. This is clearly the supremum of F, and so the proof follows. ■

That is, we are allowed to replace the order of supremum and evaluation of a function.

Lemma 5.2.5 *Let $f : D \times D' \to D''$. Then, f is continuous if and only if it is continuous in its arguments separately. Specifically, both $\lambda x.f(x, x'_0)$ and $\lambda x'.f(x_0, x')$ are continuous for any x_0, x'_0.*

Proof. ($\Rightarrow$) Let $g = \lambda x.f(x, x'_0)$. Then, for any directed $X \subseteq D$, we have

$$\begin{aligned} g(\bigsqcup^{\uparrow} X) &= f(\bigsqcup^{\uparrow} X, x'_0) \\ &= f(\bigsqcup^{\uparrow} \{(x, x'_0) \mid x \in X\}) \\ &= \bigsqcup^{\uparrow} \{f(x, x'_0) \mid x \in X\} \\ &= \bigsqcup^{\uparrow} g(X) \end{aligned}$$

Thus g is continuous, and we can do the same proof for $\lambda x'.f(x_0, x')$.

($\Leftarrow$) Let $X \subseteq D \times D'$ be directed. Then,

$$\begin{aligned} f(\bigsqcup^{\uparrow} X) &= f(\bigsqcup^{\uparrow} X_0, \bigsqcup^{\uparrow} X_1) \\ &= \bigsqcup_{x \in X_0}^{\uparrow} f(x, \bigsqcup^{\uparrow} X_1) \\ &= \bigsqcup_{x \in X_0}^{\uparrow} \bigsqcup_{x' \in X_1}^{\uparrow} f(x, x') \\ &= \bigsqcup_{\langle x, x' \rangle \in X}^{\uparrow} f(x, x') \qquad \text{as } X \text{ is directed} \\ &= \bigsqcup^{\uparrow} f(X) \end{aligned}$$

Therefore f is continuous. ■

Proposition 5.2.6 (Continuity of Application) *Define the application,*

$$\mathrm{Ap} : [D \to D'] \times D \to D'$$

by $f \mapsto x \mapsto f(x)$. Then, Ap *is continuous with respect to the Scott topology on $[D \to D'] \times D$.*

Proof. Note first that $\lambda x.f(x) = f$ is continuous. Let $h = \lambda f.f(x)$. Taking any directed $F \subseteq [D \to D']$,

$$\begin{aligned} h(\bigsqcup^{\uparrow} F) &= (\bigsqcup^{\uparrow} F)(x) \\ &= \bigsqcup^{\uparrow} \{f(x) \mid f \in F\} \qquad \text{by Proposition 5.2.4} \\ &= \bigsqcup^{\uparrow} \{h(f) \mid f \in F\} \\ &= \bigsqcup^{\uparrow} h(F) \end{aligned}$$

Therefore, h is continuous. We finish by applying Lemma 5.2.5. ■

Proposition 5.2.7 (Continuity of Abstraction) *Let $f \in [D \times D' \to D'']$. Define $\hat{f}(x) = \lambda y.f(x, y)$. Then,*

- $\hat{f}$ *is continuous. That is,* $\hat{f} \in [D \to [D' \to D'']]$
- $\lambda f.\hat{f} : [D \times D' \to D''] \to [D \to [D' \to D'']]$ *is continuous.*

Proof. (i) Let $X \subseteq D$ be directed. Then,

$$\begin{aligned}
\hat{f}(\bigsqcup^{\uparrow} X) &= \lambda y.f(\bigsqcup^{\uparrow} X, y) \\
&= \lambda y.\bigsqcup_{x \in X}^{\uparrow} f(x,y) \\
&= \bigsqcup_{x \in X}^{\uparrow} (\lambda y.f(x,y)) \qquad \text{by proposition 5.2.4} \\
&= \bigsqcup^{\uparrow} \hat{f}(x,y)
\end{aligned}$$

(ii) Let $L = \lambda f.\hat{f}$. Then for $F \subseteq [D \times D' \to D'']$ directed,

$$\begin{aligned}
L(\bigsqcup^{\uparrow} F) &= \lambda x.\lambda y.(\bigsqcup^{\uparrow} F)(x,y) \\
&= \lambda x.\lambda y.\bigsqcup_{f \in F}^{\uparrow} f(x,y) \\
&= \bigsqcup_{f \in F}^{\uparrow} \lambda x.\lambda y.f(x,y) \\
&= \bigsqcup^{\uparrow} L(F)
\end{aligned}$$

■

That is, functions are continuous after currying, and the function itself that curries another function is also continuous.

Definition 5.2.8 *We define* **DCPO** *to be the category of cpo's with continuous maps.*

Theorem 5.2.9 DCPO *is a cartesian closed category.*

Proof. TODO!!!

5.2.2 Least fixed point in $[D \to D]$

Theorem 5.2.10 (Kleene's Fixed Point Theorem for pointed cpos) *Let* D *be a pointed cpo, and* $f : D \to D$ *be a continuous function. Define*

$$\mathrm{lfp}(f) = \bigsqcup_{n \in \mathbb{N}}^{\uparrow} f^n(\bot)$$

Then $\mathrm{lfp}(f)$ *is the least fixed point of* f*.*

Proof. We first show that $\mathrm{lfp}(f)$ is a fixed point of f. Noting that f is continuous, we have

$$\begin{aligned} f(\mathrm{lfp}(f)) &= f(\bigsqcup_{n\in\mathbb{N}}^{\uparrow} f^n(\bot)) \\ &= \bigsqcup_{n\in\mathbb{N}}^{\uparrow} f^{n+1}(\bot) \\ &= \bigsqcup_{n\in\mathbb{N}}^{\uparrow} f^{n+1}(\bot) \sqcup \{\bot\} \\ &= \bigsqcup_{n\in\mathbb{N}}^{\uparrow} f^{n}(\bot) \\ &= \mathrm{lfp}(f) \end{aligned}$$

This shows $\mathrm{lfp}(f)$ is a fixed point.

Let d be any prefixed point. Noting that $\bot \sqsubseteq d$, as scott-continuous functions are monotone, we have $f(\bot) \sqsubseteq f(d)$. As d is a prefixed point, $f(\bot) \sqsubseteq d$, and inductively $f^n(\bot) \sqsubseteq d$. This gives

$$\mathrm{lfp}(f) = \bigsqcup_{n\in\mathbb{N}}^{\uparrow} f^n(\bot) \sqsubseteq d$$

As all fixed points are prefixed points, this shows $\mathrm{lfp}(f)$ is the least fixed point of f. ■

Lemma 5.2.11 *The function* $\mathrm{lfp} : [D \to D] \to D$ *is continuous.*

Proof. By Lemma 5.2.2, it is sufficient to show that every $\mathrm{it}_n : [D \to D] \to D$ defined by $f \mapsto f^n(\bot)$ is continuous. We proceed by induction on n. Noting that the base case is trivial, we proceed by taking a directed family F of continuous functions on D:

$$\begin{aligned} \mathrm{it}_{n+1}(\bigsqcup^{\uparrow} F) &= (\bigsqcup^{\uparrow} F)\mathrm{it}_n(\bigsqcup^{\uparrow} F) \\ &= (\bigsqcup^{\uparrow} F)(\bigsqcup_{f\in F}^{\uparrow} \mathrm{it}_n(f)) && \text{by inductive hypothesis} \\ &= \bigsqcup_{g\in F}^{\uparrow} g(\bigsqcup_{f\in F}^{\uparrow} \mathrm{it}_n(f)) && \text{by Lemma 5.2.2} \\ &= \bigsqcup_{g\in F}^{\uparrow} \bigsqcup_{f\in F}^{\uparrow} g(\mathrm{it}_n(f)) && \text{by continuity on } g \\ &= \bigsqcup_{f\in F}^{\uparrow} (\mathrm{it}_{n+1}(f)) \end{aligned}$$

Note in the final line that we are allowed to swap the order of suprema by Proposition 5.1.27, as the map $(-)(\mathrm{it}_n(-))$ is a monotone net. ■

We can now define fixpoint induction. First, call a dcpo **admissible** if it contains $\bot$ and is closed under suprema of ω-chains. We have a nice property about admissible predicates.

Lemma 5.2.12 *Let D be a dcpo. Let $P \subseteq D$ is an admissible predicate and $f : [D \to D]$. If x satisfies P implies that $f(x)$ satisfies P, then* $\mathrm{lfp}(f)$ *satisfies P.*

Proof. Follows easily from utilizing the fact that P is admissible. ■

Lemma 5.2.13 *Let D and D' be ccpos, and let*

$$\begin{array}{ccc} D & \xrightarrow{h} & D' \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle g} \\ D & \xrightarrow[h]{} & D' \end{array}$$

be a commutative diagram of continuous functions where h is strict. Then, $\mathrm{lfp}(g) = h(\mathrm{lfp}(f))$.

Proof. By the continuity of h, we have

$$\begin{aligned} h(\mathrm{lfp}(f)) &= h(\bigsqcup\nolimits^{\uparrow}_{n\in\mathbb{N}} f^n(\bot)) \\ &= \bigsqcup\nolimits^{\uparrow}_{n\in\mathbb{N}} h \circ f^n(\bot) \quad \text{by continuity of } h \\ &= \bigsqcup\nolimits^{\uparrow}_{n\in\mathbb{N}} g^n \circ h(\bot) \quad \text{by commutativity of diagram} \\ &= \mathrm{lfp}(g) \quad \text{by strictness} \end{aligned}$$

■

5.3 Approximations

We want to somehow move this notion of dcpos and move them into the space of computation. We start first by giving a notion of approximation.

Definition 5.3.1 *We say that x **approximates** y if for all directed $X \subseteq D$, $y \sqsubseteq \bigsqcup^{\uparrow} X$ implies $x \sqsubseteq x_0$ for some $x_0 \in X$. We write $x \ll y$. We may read this as x is way-below y.*

This notion of approximation can be thought of as an 'order of definite refinement', or 'order of approximation'.

5.3.1 Basis, Compactness, and Algebraicity

We can connect this notion back into topology.

Definition 5.3.2 *We say that $x \in D$ is **compact** if for every directed $X \subseteq D$, we have*

$$x \sqsubseteq \bigsqcup^{\uparrow} X \qquad \Longrightarrow \qquad x \sqsubseteq x_0 \quad \text{for some } x_0 \in X$$

Alternatively, x is compact if $x \ll x$.

Notice that the order of approximation is not necessarily reflexive unless compact. We may refer to compact as being 'finite' or 'isolated'. The intuition behind these phrases will be seen later.

Notation 5.3.3 *We will use the following notation for $x, y \in D$ and $A \subseteq D$.*

$$\begin{aligned} \mathrel{\downarrow} x &= \{y \in D \mid y \ll x\} \\ \mathrel{\uparrow} x &= \{y \in D \mid x \ll y\} \\ \mathrel{\uparrow} A &= \bigcup_{a\in A} \mathrel{\uparrow} a \\ \mathrm{K}(D) &= \{x \in D \mid x \text{ compact}\} \end{aligned}$$

Proposition 5.3.4 *Let D be a dcpo. Then for any $x, x', y, y' \in D$, we have*

- *$x \ll y$ implies $x \sqsubseteq y$*
- *$x' \sqsubseteq x \ll y \sqsubseteq y'$ implies $x' \ll y'$*

Proof. Note the first case is obvious, as the set $\{y\}$ is directed. The second also follows, as given any directed $X \subseteq D$, $y \sqsubseteq y' \sqsubseteq \bigsqcup^{\uparrow} X$ and $x' \sqsubseteq x \sqsubseteq x_0$ for some $x_0 \in X$. ■

Definition 5.3.5 *We say that a subset B of a dcpo D is a **basis** for D if every element x of D, the set $B_x = \{\downarrow x \cap B\}$ contains a directed subset with supremum x. We call elements of B_x **approximants to** x **relative to** B.*

This means that the elements in the basis which are way-below an element $x \in D$ are directed and has supremum x.

For instance, we can think of this notion in $\mathbb{R}$ (with a top element added) as some 'dense from below' viewing the reals as Dedekind cuts. Then, $\mathbb{Q}, \mathbb{R}\backslash\mathbb{Q}$, dyadic numbers (rationals with denominator powers of 2) are all basis for $\mathbb{R}$.

Proposition 5.3.6 *Let D be a dcpo with basis B. Then,*

1. *For every $x \in D$, B_x is directed, and $x = \bigsqcup^{\uparrow} B_x$*
2. $\mathrm{K}(D) \subseteq B$
3. *Every superset of B is also a basis for D*

Proof. (1) Equality follows from definition given directedness. We wish to show directedness. By definition, B_x contains a directed A with $\bigsqcup^{\uparrow} A = x$. Now take any $y, y' \in B_x$. These must be approximating x. As $x \sqsubseteq \bigsqcup^{\uparrow} A$, there exists $a, a' \in A$ such that $y \sqsubseteq a$ and $y' \sqsubseteq a'$. As A is directed, this has an upper bound in A, which is a subset of B_x. Thus B_x is directed.

(2) Taking $x \in \mathrm{K}(D)$, using the fact that $x = \bigsqcup^{\uparrow} B_x$, as x is compact, there exists an $x_0 \in B_x$ such that $x \sqsubseteq x_0$. Thus, $x = x_0$ and so $x \in B$.

(3) Follows from definition. ■

Definition 5.3.7 *A dcpo is called **continuous** if it has a basis. We say that it is ω-continuous if there exists a countable basis.*

Proposition 5.3.8 *If x is a non-compact element of a basis B for a continuous D, then $B\backslash\{x\}$ is still a basis.*

Proof. TODO!!!!(Hint, interpolation property)

Corollary 5.3.9 *The largest basis for D is D itself. On the contrary, B is the smallest basis of D if and only if $B = \mathrm{K}(D)$.*

Proof. The first statement and the if condition for the second statement follow from Proposition 5.3.6. The only if condition follows from Proposition 5.3.8, as we can remove non-compact elements and still be a basis. ■

Remark 5.3.10 *Note that this does not imply that if D is continuous then $\mathrm{K}(D)$ is a basis. However, we are allowed to remove any finite number of non-compact elements from a basis. The next part covers when $\mathrm{K}(D)$ is indeed a basis.*

Definition 5.3.11 *A dcpo is called* ***algebraic*** *or is an* ***algebraic domain*** *if it has a basis of compact elements. We say that it is ω-algebraic if $\mathrm{K}(D)$ is a countable basis.*

Remark 5.3.12 *We use the notion of domain to refer to a space in which we can talk about some notion of convergence and approximation.*

Proposition 5.3.13 *We can combine our notion with Proposition 5.3.6 to redefine the notion of continuity and algebraicity. That is,*

1. *A dcpo D is continuous if and only if for all $x \in D$, $x = \bigsqcup^{\uparrow} \Downarrow x$.*
2. *A dcpo D is algebraic if and only if for all $x \in D$, $x = \bigsqcup^{\uparrow} \mathrm{K}(D)_x$*

Proof. (1) ($\Rightarrow$) Suppose that D has a basis. Then D is a basis for D. Specifically, for any $x \in D$, $D_x = \Downarrow x \cap D = \Downarrow x$. By Proposition 5.3.6, $x = \bigsqcup^{\uparrow} D_x = \bigsqcup^{\uparrow} \Downarrow x$.

($\Leftarrow$) Suppose that $x = \bigsqcup^{\uparrow} \Downarrow x = \bigsqcup^{\uparrow} D_x$ for any x. Then, clearly D is a basis as each D_x is a directed subset with supremum x.

(2) ($\Rightarrow$) Suppose that D has a basis of compact elements. By Proposition 5.3.6, any basis is larger than $\mathrm{K}(D)$, so $\mathrm{K}(D)$ is a basis for D. Now, by the same proposition, it follows that for any x, $x = \bigsqcup^{\uparrow} \mathrm{K}(D)_x$.

($\Leftarrow$) Suppose that for any x, $x = \bigsqcup^{\uparrow} \mathrm{K}(D)_x$. Then, as $\{x\} \subseteq \mathrm{K}(D)_x$ is trivially directed with supremum x. ■

5.3.2 Closure Systems and Algebraicity

There is a reason why we use the word "algebraic" when $\mathrm{K}(D)$ is a basis for D. We cover this here.

Definition 5.3.14 *Let X be a set and $\mathcal{L}$ be a family of subsets of X. We say that*

- *$\mathcal{L}$ is* ***closure system*** *if it closed under the formation of intersections. That is, if $A_i \in \mathcal{L}$ for $i \in I$, $\bigcap_{i \in I} A_i \in \mathcal{L}$. Note that as I can be empty, $X \in \mathcal{L}$.*
- *We call the members of $\mathcal{L}$* ***hulls*** *or* ***closed sets****.*
- *Given an arbitrary $A \subseteq X$, the least superset of A contained in $\mathcal{L}$, or explicitly, $\bigcap\{B \in \mathcal{L} \mid A \subseteq B\}$ is called the* ***hull*** *or* ***closure*** *of A.*

Proposition 5.3.15 *Every closure system is a complete lattice with respect to inclusion.*

Proof. Infima is given trivially by intersection, while the supremum is given by the closure of the union. ■

Definition 5.3.16 *A closure system $\mathcal{L}$ is called* ***inductive*** *if it is closed under directed union (directed sets are closed under union).*

Proposition 5.3.17 *Every inductive closure system $\mathcal{L}$ is an algebraic lattice. The compact elements are precisely the finitely generated closed sets (closure of finite sets).*

Proof. First, if A is the closure of a finite set M and $(B_i)_{i\in I}$ is a directed family of closed sets such that $\bigsqcup^{\uparrow}_{i\in I} B_i = \bigcup_{i\in I} B_i \supseteq A$, then M is contained in some B_i as M is finite and $(B_i)_{i\in I}$ is directed. Thus closures of finite sets are compact in $\mathcal{L}$.

On the other hand, every closed set is the directed union of finitely generated closed sets. That is, given a closed set A, the set $\{B \subseteq A \mid B \text{ is finitely generated}\}$ is a directed set whose union is equal to A. As finitely generated closed sets are compact, it is way-below anything greater than x. Thus, the set of finitely generated elements form a basis for $\mathcal{L}$.

By Proposition 5.3.6, as every basis contains the set of compact elements, it follows that compact elements are precicely the finitely generated closed sets, which forms a basis. ■

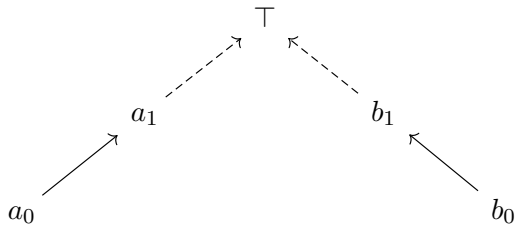
Example 5.3.18 *We denote some examples of algebraic domains.*

- *Given a group, there are two canonical inductive closure systems associated to it, the lattice of subgroups and the lattice of normal subgroups.*
- *Any set with discrete order is an algebraic domain. In semantics, we may add a bottom to obtain a flat domain. In both cases, a basis will contain all elements.*
- *Every finite poset.*

Example 5.3.19 *We further illustrate some examples of continuous domains.*

- *Every algebraic approximation is characterized by $x \ll y$ if and only if there exists a compact element between x and y. This follows from the fact that every element is a supremum of a set of compact elements.*
- *The unit interval is a continuous lattice.*

Example 5.3.20 *On the contrary, we can construct non-continuous dcpos. We illustrate this with the following example:*



This dcpo has no order of approximation. Pairs (a_i, b_j) or (b_i, a_j) are not related to one another, and two points $a_i \sqsubseteq a_j$ (for a_j possibly $\top$) is not approximating as $(b_n)_{n\in\mathbb{N}}$ is a directed set with supremum above a_j but has no element above a_i.

On the other hand, if we have a pointed poset, then it has non-empty approximation as the bottom element approximates every other element.

Basis elements have many nice properties, as it not only gives approximations for elements but also for the order relation.

Proposition 5.3.21 *Let D be a continuous domain with basis B and let $x, y \in D$. Then the following are equivalent.*

1. $x \sqsubseteq y$
2. $B_x \subseteq B_y$

3. $B_x \subseteq \downarrow y$

Proof. (1) $\implies$ (2) Follows immediately from the fact that anything approximating x is also approximating y.

(2) $\implies$ (3) Follows from the fact that $B_y \subseteq \downarrow y$.

(3) $\implies$ (1) By taking the suprema on both sides, we get $x \sqsubseteq y$. ■

Remark 5.3.22 *By the above proposition, if D is continuous, given $x \not\sqsubseteq y$, there exists $b \in B_x$ with $b \not\sqsubseteq y$ (by using (1) and (3)).*

Furthermore, we can also see that for a continuous domain, the information about how the elements are related are already contained in the basis. The fact that $\bigsqcup^{\uparrow}_{n\in\mathbb{N}} a_n = \bigsqcup^{\uparrow}_{n\in\mathbb{N}} b_n = \top$ is precisely what is not visible in a 'basis', if it were to exist. Thus, by separating $\top$ to be distinct for (a_i) and (b_i), it gives us an algebraic domain.

Proposition 5.3.23 *Let D and D' be continuous domains with bases B and B'. Then, $f : D \to D'$ is continuous if and only if for each $x \in D$ and $y \in B'_{f(x)}$, there exists $z \in B_x$ with $f(\uparrow z) \subseteq \uparrow y$.*

Proof. ($\Rightarrow$) Suppose f is continuous. Fixing an x and y, we have

$$f(x) = f(\bigsqcup{}^{\uparrow} B_x) = \bigsqcup^{\uparrow}_{d\in B_x} f(d)$$

and as y approximates $f(x)$, there exists a $z \in B_x$ such that $y \sqsubseteq f(z)$. By monotonicity, $f(\uparrow z) \subseteq \uparrow y$.

($\Leftarrow$) We first show that f is monotonic. Suppose $x \sqsubseteq x'$ but $f(x) \not\sqsubseteq f(x')$. By Proposition 5.3.21, there exists a $y \in B'_{f(x)}$ with $y \not\sqsubseteq f(x')$. By assumption, there exists a $z \in B_x$ such that $f(\uparrow z) \subseteq \uparrow y$. As $x \in \uparrow z$, so is x', but now $f(x') \in \uparrow y$, which is a contradiction.

Now, let X be a directed subset of D with $x = \bigsqcup^{\uparrow} X$. By monotonicity,

$$\bigsqcup{}^{\uparrow} f(X) \sqsubseteq f(\bigsqcup{}^{\uparrow} X) = f(x)$$

If $f(x) \not\sqsubseteq \bigsqcup^{\uparrow} f(X)$, we can again find a $y \in B'_{f(x)}$ with $y \not\sqsubseteq \bigsqcup^{\uparrow} f(X)$. By assumption, there exists a $z \in B_x$ such that $f(\uparrow z) \subseteq \uparrow y$. As z approximates x, some $x' \in X$ is above z, giving

$$y \sqsubseteq f(z) \sqsubseteq f(x') \sqsubseteq \bigsqcup{}^{\uparrow} f(X)$$

contradicting our choice of y. ■

Proposition 5.3.24 *let D be a pointed ω-continuous domain with basis B. If $f : [D \to D]$, there exist an ω-chain $b_0 \sqsubseteq b_1 \sqsubseteq \ldots$ of basis elements such that the following hold:*

1. $b_0 = \bot$
2. $\forall n \in \mathbb{N}$, $b_{n+1} \sqsubseteq f(b_n)$
3. $\bigsqcup^{\uparrow}_{n\in\mathbb{N}} b_n = \mathrm{lfp}(f)$

Proof. Out of scope, but proof may be found in [1]. The point is that as continuous functions don't preserve compactness nor order of approximation, $f^n(\bot)$ need not consist of basis elements. Nonetheless, we can construct a chain of elements which are prefixpoints of the previous item such that the suprema is the fixpoint. ■

Proposition 5.3.25 *Let D be algebraic and $f : D \to D$. Then f is continuous if and only if for any x, $f(x) = \bigsqcup^{\uparrow}\{f(e) \mid e \sqsubseteq x, e \text{ compact}\}$.*

Proof. ($\Rightarrow$) Let f be continuous. Then,

$$\begin{aligned} f(x) &= f(\bigsqcup^{\uparrow}\{e \sqsubseteq x \mid e \text{ compact}\}) \\ &= \bigsqcup^{\uparrow}\{f(e) \mid e \sqsubseteq x, e \text{ compact}\} \end{aligned}$$

($\Leftarrow$) First note that f is monotinic. That is, given $x \sqsubset y$,

$$\{e \sqsubseteq x \mid e \text{ compact}\} \subseteq \{e \sqsubseteq y \mid e \text{ compact}\}$$

Then,

$$\begin{aligned} f(x) &= \bigsqcup^{\uparrow}\{f(e) \mid e \sqsubseteq x, e \text{ compact}\} \\ &\sqsubseteq \bigsqcup^{\uparrow}\{f(e) \mid e \sqsubseteq y, e \text{ compact}\} = f(y) \end{aligned}$$

Suppose now that $X \subseteq D$ is directed. Then,

$$\begin{aligned} f(\bigsqcup^{\uparrow} X) &= \bigsqcup^{\uparrow}\{f(e) \mid e \sqsubseteq \bigsqcup^{\uparrow} X, e \text{ compact}\} \\ &\sqsubseteq \bigsqcup^{\uparrow}\{f(x) \mid x \in X\} \quad \text{by compactness} \\ &\sqsubseteq f(\bigsqcup^{\uparrow} X) \quad \text{by monotonicity} \end{aligned}$$

Henceforth $f(\bigsqcup^{\uparrow} X) = \bigsqcup^{\uparrow} f(X)$. ■

Proposition 5.3.26 *Let D be algebraic. Define $O_e = \{x \in D \mid e \sqsubseteq x\} = \uparrow \{e\}$. Then,*

$$\{O_e \mid e \text{ compact}\}$$

is a basis for the topology on D.

Proof. We first show that O_e is open when e is compact. Note that O_e is an upper set. Taking any directed $X \subseteq D$ such that $\bigsqcup^{\uparrow} X \in O_e$, as $e \sqsubseteq \bigsqcup^{\uparrow} X$, by compactness, we have $e \sqsubseteq x_0$ for some $x_0 \in X$. Specifically, the intersection of X and O_e is nontrivial, hence O_e is Scott open.

Now, take any $x \in O$ where O is open. As $x = \bigsqcup^{\uparrow}\{e \sqsubseteq x \mid e \text{ compact}\}$ is directed suprema,

$$\exists e \sqsubseteq x \quad e \in O \quad e \text{ compact}$$

using the fact O is open. Thus, $x \in O_e \in O$ (noting that O_e is the smallest open set that contains e). Therefore, each open O is the union of opens in the basis set. ■

5.4 Topological Interpretation of Domains

Definition 5.4.1 *Let $(D, \sqsubseteq)$ be a partially ordered set. A subset $O \subseteq D$ is called **Scott-open** if is an upper set that is inaccessible by directed suprema. That is, all directed sets S with a supremum in O have a non-empty intersection with O.*

Proposition 5.4.2 *The Scott-open subsets of a partially ordered set $(D, \sqsubseteq)$ form a topology on D, which is called the* ***Scott topology*** *and written as (D, τ).*

Proof. Start by noting that $\emptyset$ and D are both trivially Scott open.

Consider a family of Scott open sets $\mathcal{U} = \{U_i\}_{i \in I}$. Take any $x \in \uparrow (\bigcup \mathcal{U})$. Then, there exists an $i \in I$ such that $y \in U_i$ and $y \sqsubseteq x$. Now,

$$x \in \uparrow U_i = U_i \subseteq \bigcup \mathcal{U}$$

Henceforth $\uparrow (\bigcup \mathcal{U}) \subseteq \bigcup \mathcal{U}$ and equality follows. Now, let $X \subseteq D$ be a directed subset such that $\bigsqcup^{\uparrow} X \in \bigcup \mathcal{U}$. Then, there exists a $i \in I$ such that $\bigsqcup^{\uparrow} X \in U_i$. Then as U_i is Scott open, $X \cap U_i \neq \emptyset$. Using

$$X \cap U_i \subseteq \bigcup_{i \in I} (X \cap U_i) = X \cap \bigcup_{i \in I} U_i = X \cap (\bigcup \mathcal{U})$$

We see that $X \cap (\bigcup \mathcal{U}) \neq \emptyset$. Thus $\bigcup \mathcal{U}$ is Scott open.

Take any $U, V \in \tau$ and let $x \in \uparrow (U \cap V)$. Then, there exists a $y \in U \cap V$ with $y \sqsubseteq x$. Now, as U and V are both upper sets, we have

$$x \in (\uparrow U) \cap (\uparrow V) = U \cap V$$

This gives $\uparrow (U \cap V) \subseteq U \cap V$. Now, let $X \subseteq D$ be a directed subset such that $\bigsqcup^{\uparrow} X \in U \cap V$. As U and V are Scott open, there exists a $u \in X \cap U$ and $v \in X \cap V$. As X is directed, there exists a $x \in X$ such that $u \sqsubseteq x$ and $v \sqsubseteq x$. Now,

$$x \in X \cap ((\uparrow U) \cap (\uparrow V)) = X \cap (U \cap V)$$

Hence, $D \cap (U \cap V) \neq \emptyset$. Thus $U \cap V$ is Scott open. ■

Proposition 5.4.3 *A subset of a poset D is closed in the Scott topology if and only if it is a lower set and is closed under the suprema of directed subsets.*

Proof. ($\Rightarrow$) Let U be Scott open. That is, it it is an upper set and every directed set with suprema in U has non-empty intersection with U. Now, take any $y \in D \backslash U$ and let $x \sqsubseteq y$. If $x \in U$, as U is open, $y \in U$, a contradiction. Thus, $D \backslash U$ is a lower set. Take any directed subset $X \subseteq D \backslash U$ with a suprema. Then, if the suprema lies in U, X has nonempty intersection with U, a contradiction.

($\Leftarrow$) Let U be a lower set and closed under the suprema of directed subsets. Then, takinga any $x \in D \backslash U$ with $x \sqsubseteq y$, if $y \in U$, we contradict with U being a lower set. Now, take any directed subset X with suprema. If this set is contained in U, it's suprema is contained in U. Otherwise, X has suprema in $D \backslash U$ and has non-empty intersection with $D \backslash U$. ■

Proposition 5.4.4 *Let $(D, \sqsubseteq)$ be a partially ordered set. For any $d \in D$, $D \backslash (\downarrow d)$ is Scott open. We write U_d for this set.*

Proof. Let $x \in \uparrow (D \backslash \downarrow d)$. Then there exists a $y \in D \backslash \downarrow d$ such that $y \sqsubseteq x$. Suppose for a contradiction that $x \in \downarrow d$. That is, $x \sqsubseteq d$. By transitivity of $\sqsubseteq$, $y \sqsubseteq p$. This contradicts $y \in D \backslash \downarrow d$. Therefore $x \in D \backslash \downarrow d$. Thus, $D \backslash \downarrow d$ is an upper set.

Now consider a directed set $X \subseteq D$ such that $\bigsqcup^{\uparrow} X \in D \backslash \downarrow d$. Then, $\bigsqcup^{\uparrow} X \not\sqsubseteq d$. Suppose for a contradiction that $X \cap (D \backslash \downarrow d) = \emptyset$. This gives $X \subseteq \downarrow d$. This means that d is an upper bound for X, giving $\bigsqcup^{\uparrow} X \sqsubseteq d$, which is a contradiction. Thus, $X \cap (D \backslash \downarrow d) \neq \emptyset$. This shows $D \backslash \downarrow d$ is inaccessible by directed suprema, meaning it is Scott open. ■

Corollary 5.4.5 *D is a T_0 space which is not necessarily T_1.*

Proof. Take $x, y \in D$ with $x \neq y$. Suppose now that without loss of generality, $x \not\sqsubseteq y$. Then, $x \in U_y, y \neq U_y$ and U_y is open. Thus D is T_0. On the other hand, if $x \sqsubseteq y$, then every neighborhood of x contains y as it must be an upper set. Thus, D need not be T_1. ■

5.4.1 On continuity

Scott continuity has an interpretation under continuity in the topological sense, which we will discuss in this subsection.

Lemma 5.4.6 *If f is continuous under the Scott topology, it is monotonic.*

Proof. Let $f : D \to D'$ be continuous under the Scott topology. Take $x, x' \in D$ such that $x \sqsubseteq x'$. Supose for a contradiction that $f(x) \not\sqsubseteq f(x')$. Then, $f(x) \in D' \setminus \downarrow f(x')$. Noting this set is Scott open, we have $x \in f^{-1}(D' \setminus \downarrow f(x'))$ which is also Scott open by continuity. As this set is upper closed, it follows that $x' \in f^{-1}(D' \setminus \downarrow f(x'))$. Now,

$$\begin{aligned} x' \in f^{-1}(D' \setminus \downarrow f(x')) &\implies f(x) \in D' \setminus \downarrow f(x') \\ &\implies f(x') \sqsubseteq f(x') \end{aligned}$$

which is a contradiction. Hence, it follows that $f(x) \sqsubseteq f(x')$. ■

Theorem 5.4.7 *A function between partially ordered sets $(D, \sqsubseteq)$ is Scott continuous if and only if it is continuous with respect to the Scott topology.*

Proof. ($\Rightarrow$) Suppose that $f : D \to D'$ is Scott continuous. Take any Scott open set U in E. We want to show that $f^{-1}(U)$ is Scott open. Specifically, we wish to show that U is (i) an upper set and (ii) all directed sets with a supremum in $f^{-1}(U)$ has a non-empty intersection with $f^{-1}(U)$.

- For (i), take any $x \in f^{-1}(U)$ such that $f(x) \in U$. Given $x \sqsubseteq x'$, by monotonicity of Scott continuous functions we have $f(x) \sqsubseteq f(x')$. As U is an upper set, we have $f(x') \in U$. It follows that $x' \in f^{-1}(U)$.
- For (ii), Take $X \subseteq D$ be any directed set such that $\bigsqcup^{\uparrow} X \in f^{-1}(U)$. That is, $f(\bigsqcup^{\uparrow} X) \in U$. By Scott continuity, $\bigsqcup^{\uparrow} f(X) \in U$. As U is Scott open, we have $f(X) \cap U \neq \emptyset$. Equivalently, there is a $x \in X$ such that $f(x) \in U$. Thus, $x \in f^{-1}(U)$, therefore it is inaccessible by a directed suprema.

($\Leftarrow$) Suppose that f is continuous in the Scott topology, such that for any Scott open set $U \in D'$, $f^{-1}(U)$ is Scott open in D. We wish to show that f is Scott continuous, such that for any directed set $X \subseteq D$ with a supremum,

$$f(\bigsqcup^{\uparrow} X) = \bigsqcup^{\uparrow} f(X)$$

We prove this by showing that $f(\bigsqcup^{\uparrow} X)$ is (i) an upper bound and (ii) the least upper bound with respect to $f(X)$.

- For (i), note that as f is monotone from Lemma 5.4.6, given any $x \in X$, we have $x \sqsubseteq \bigsqcup^{\uparrow} X$, meaning $f(x) \sqsubseteq f(\bigsqcup^{\uparrow} X)$. This shows $\bigsqcup^{\uparrow} f(X) \sqsubseteq f(\bigsqcup^{\uparrow} X)$.

- For (ii), Suppose for a contradiction that $f(\bigsqcup^{\uparrow} X) \not\sqsubseteq \bigsqcup^{\uparrow} f(X)$. Then, $f(\bigsqcup^{\uparrow} X) \in D' \setminus \downarrow \bigsqcup^{\uparrow} f(X)$. It follows that $\bigsqcup^{\uparrow} X \in f^{-1}(D' \setminus \downarrow \bigsqcup^{\uparrow} f(X))$. As this is Scott open, $X \cap f^{-1}(D' \setminus \downarrow \bigsqcup^{\uparrow} f(X)) \neq \emptyset$. Taking x to be an element in this, we have $f(x) \in f(X)$ and $f(x) \in D' \setminus \downarrow \bigsqcup^{\uparrow} f(X)$. The latter transforms into $f(x) \not\sqsubseteq \bigsqcup^{\uparrow} f(X)$, contradicting with $f(x) \in f(X)$. We therefore have $f(\bigsqcup^{\uparrow} X) \sqsubseteq \bigsqcup^{\uparrow} f(X)$.

■

5.4.2 Projective Limits

Definition 5.4.8 *Let $D_0, D_1, \ldots$ be a countable sequence of cpo's and let $f_i \in [D_{i+1} \to D_i]$. We now define*

- *The sequence (D_i, f_i) is called the **projective** or **inverse system** of cpos.*
- *The **projective** or **inverse limit** of the system (D_i, f_i) with notation $\varprojlim(D_i, f_i)$ is the poset $(D_\infty, \sqsubseteq_\infty)$ with*

$$D_\infty = \{\langle x_0, x_1, \ldots \rangle \mid \forall i, x_i \in D_i \wedge f_i(x_{i+1}) = x_i\}$$

where,

$$\langle \vec{x} \rangle \sqsubseteq_\infty \langle \vec{y} \rangle \quad \text{iff} \quad \forall i, x_i \sqsubseteq y_i$$

Note that as usual, we interpret an infinite sequence $\langle x_0, x_1, \ldots \rangle$ with a map $x : \mathbb{N} \to \cup_i D_i$ such that $x(i) = x_i \in D_i$ for all i. Where it is clear, we may write $\varprojlim(D_i)$ for $\varprojlim(D_i, f_i)$.

Proposition 5.4.9 *Let (D_i, f_i) be a projective system. Then $\varprojlim(D_i, f_i)$ is a cpo with*

$$\bigsqcup^{\uparrow} X = \lambda i. \bigsqcup^{\uparrow} \{x(i) \mid x \in X\}$$

for directed $X \subseteq \varprojlim(D_i)$.

Proof. If X is directed, then $\{x(i) \mid x \in X\}$ is directed for each i. Let

$$y_i = \bigsqcup^{\uparrow} \{x(i) \mid x \in X\}$$

Then by the conitnuity of f_i, we have

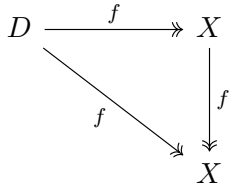
$$\begin{aligned} f_i(y_{i+1}) &= \bigsqcup^{\uparrow} f_i\{x(i+1) \mid x \in X\} \\ &= \bigsqcup^{\uparrow} \{x(i) \mid x \in X\} \\ &= y_i \end{aligned}$$

Therefore $\langle y_0, y_1, \ldots \rangle \in \varprojlim(D_i)$. This is clearly the supremum of X. ■

Definition 5.4.10 *Let D be a cpo and let $X \subseteq D$. Then,*

- *$f \in [D \to D]$ is a **retraction** map of D onto X if $X = \mathrm{im}(f)$ and $f = f \circ f$.*
- *X is a **retract** of D if there is a retraction map f of D onto X.*

Alternatively, we can think of a retraction map as a continuous function f for which the following diagram commutes:



Proposition 5.4.11 *Let D be a cpo with retract X. Then X is a cpo whose directed subsets have the same supremum as in D whose topology is the subspace topology.*

Proof. Let $f : D \to X$ be a retraction map. Let $Y \subseteq X$ be directed. Then Y is also directed as a subset of D and $\bigsqcup^{\uparrow} Y = y$ exists in D. Now,

$$\begin{aligned} f(y) &= f(\bigsqcup^{\uparrow} Y) \\ &= \bigsqcup^{\uparrow} f(Y) \\ &= \bigsqcup^{\uparrow} Y \qquad \text{as } Y \subseteq X \\ &= y \end{aligned}$$

Therefore $y \in X$ and is also clearly the supremum of Y in X. ■

5.5 Function Approximations

Definition 5.5.1 *Given two sets A and B, an **empty function** is a partial function $A \rightharpoonup B$ that has no defined maps from elements of A. Alternatively, if we interpret this as a function from A to* Option B *that takes $a \mapsto$* Nothing*. We often write $\perp$ to represent this function.*

Proposition 5.5.2 *The set of partial functions from $\mathbb{N}$ to $\mathbb{N}$ with the standard $\sqsubseteq$ on partial functions is a ccpo.*

Proof. TODO!!

5.5.1 The Factorial Function

Example 5.5.3

Consider the factorial function which might be recursively defined as

```
int factorial(nat n) {
    if (n == 0)
        then 1;
    else
        n * factorial(n-1);
}
```

under a programming context. We will write f to represent this function.

To give meaning to this f, we model this through an approximation as a partial function $\mathbb{N} \rightharpoonup \mathbb{N}$ starting with the empty function. We introduce a function $F : (\mathbb{N} \rightharpoonup \mathbb{N}) \to (\mathbb{N} \rightharpoonup \mathbb{N})$ defined by the map

$$f \mapsto \{0 \mapsto 1\} \ \oplus \ \{n \mapsto n * f(n-1) \mid n \in \mathbb{N} \backslash \{0\}\}$$

where $\oplus$ represents an overloading of function maps. Then we define $F^0(\bot) = \bot$ and $F^{n+1}(\bot) = F(F^n(\bot))$. This process builds a sequence of $\mathbb{N} \rightharpoonup \mathbb{N}$. Note that this sequence satisfies $F^n \sqsubseteq F^{n+1}$. As $\mathbb{N} \rightharpoonup \mathbb{N}$ is an ω-cpo, by Kleene's Fixed Point Theorem, setting

$$f := \bigsqcup_{n \in \mathbb{N}}^{\uparrow} F^n(\bot)$$

this define a suitable interpretation of the recursive function defined above. Notice that we take the least fixed point on F, as we are talking about functions that terminate.

5.6 Negative Definitions (In Haskell)

6 Dependent Type Theory

The basics covered in the first part of this section will also have been covered (in a similar fashion) in the Lambda Calculus section. We first start from a simple type system and extend this into a dependent type system.

We first define **types** to be the expressions generated by the following context-free grammar:

$$\textit{Types} \quad A, B := \mathbf{b} \mid A \times B \mid A \to B$$

with some base type $\mathbf{b}$ to ensure that the grammar is nonempty.

We also give rise to **contexts**, which give some notion of types to variables. As a context free grammar,

$$\textit{Contexts} \quad \Gamma := \mathbf{1} \mid \Gamma, x : A$$

where $\mathbf{1}$ represents the empty context, and $\Gamma, x : A$ to mean the extention of Γ by a term variable x of type A. In this section, we will assume that there is an infinite set of variables $x, y, z, \ldots$ and that variables occuring in a given context or term are distinct (otherwise we can define some notion of a well-formed context). Equivalently, in inference rule notation, we write

$$\frac{}{\vdash \mathbf{1}\ \mathrm{cx}} \qquad \frac{\vdash \Gamma, x : A\ \mathrm{cx}}{\vdash \Gamma\ \mathrm{cx} \qquad A\ \mathrm{type}}$$

To define terms which are typed by this context, first fix a finite indexing set I such that for any $i \in I$ the base term $\mathbf{c}_i$ has type $\mathbf{b}$. Then, typing rules can be expressed as follows:

$$\frac{\Gamma \vdash x : A}{(x : A) \in \Gamma} \qquad \frac{\Gamma \vdash \mathbf{c}_i : \mathbf{b}}{i \in I} \qquad \frac{\Gamma \vdash (a, b) : A \times B}{\Gamma \vdash a : A \qquad \Gamma \vdash b : B} \qquad \frac{\Gamma \vdash \mathbf{fst}(p) : A}{\Gamma \vdash p : A \times B}$$

$$\frac{\Gamma \vdash \mathbf{snd}(p) : B}{\Gamma \vdash p : A \times B} \qquad \frac{\Gamma \vdash \lambda x.b : A \to B}{\Gamma, x : A \vdash b : B} \qquad \frac{\Gamma \vdash f\ a : B}{\Gamma : f : A \to B \qquad \Gamma \vdash a : A}$$

Notions of typing allow us to force the existence only of terms we wish to have creating some notion of well-formed terms, which then gives us some additional structure to symbols (in this case some well-behaved computability). It is worth noting the existence of preterms:

$$\textit{Preterms} \quad a, b := \mathbf{c}_i \mid x \mid (a, b) \mid \mathbf{fst}(a) \mid \mathbf{snd}(a) \mid \lambda x.a \mid a\ b$$

which gives typeless terms like $\mathbf{fst}(\lambda x.x)$. Nonetheless, an extention from terms to preterms often allows us to prove statements that regard well-typed terms, so we place this here for completeness.

6.1 Quotienting Terms by Equality

Typed terms are yet to hold any meaning in the sense that terms which are not syntactically equal are treated differently. To give a notion of "sameness", we can take the set of terms and quotient it by a congruence rule that describes the intended sameness. In our current case, we want:

$$\frac{\Gamma \vdash \mathbf{fst}((a, b)) = a : A}{\Gamma \vdash a : A \qquad \Gamma \vdash b : B} \qquad \frac{\Gamma \vdash \mathbf{snd}((a, b)) = b : B}{\Gamma \vdash a : A \qquad \Gamma \vdash b : B} \qquad \frac{\Gamma \vdash p = (\mathbf{fst}(p), \mathbf{snd}(p)) : A \times B}{\Gamma \vdash p : A \times B}$$

$$\frac{\Gamma \vdash (\lambda x.b)a = b[a/x] : B}{\Gamma, x : A \vdash b : B \qquad \Gamma \vdash a : A} \qquad \frac{\Gamma \vdash f = \lambda x.(f\ x) : A \to B}{\Gamma \vdash f : A \to B}$$

That is, we impose rules that represent our projections, destruction (of p), and β, η-equivalences. Further note that these equivalences are up to any constructor composition, meaning if we have a unary hole context $C[X]$ and two terms x, y such that $x = y$, there is some notion of equality (in a possibly different context due to the λ binders) such that $C[x] = C[y]$. For the scope of the section, we will assume for simplicity that substitution is capture avoiding and the variable to substitute into does not coincide with the bound variable. A more proper notion of this is covered in the Lambda Calculus Section.

Remark 6.1.1 *It should be noted here that we must technically show substitution is well-defined. That is, substitution must map equal terms to equal terms (such that it is a proper function in the quotient space). Proving this is a quick check over each constructor of equality and hence omitted.*

This then gives us a notion of a set of contexts $\mathrm{Cx} = \{\Gamma \mid \vdash \Gamma \text{ cx}\}$, a set of types $\mathrm{Ty} = \{A \mid A \text{ type}\}$, and for every $\Gamma \in \mathrm{Cx}, A \in \mathrm{Ty}$ a set of terms $\mathrm{Tm}(\Gamma, A)$ which correspond to the set of a such that there exists a derivation $\Gamma \vdash a : A$ modulo $\sim$, where $a \sim b \iff$ there exists a derivation of $\Gamma \vdash a = b : A$.

6.2 Into Dependent Types

The difference between standard types and dependent types is that while in the simple type system (created in the previous section) only terms are context sensitive, in dependent type theory both types and terms are context sensitive as they refer to each other. For instance, if we take $(n : \mathbf{Nat}) \to \mathbf{Vec\ String}\ (\mathrm{suc}\ n)$, the well-formedness of the codomain is dependent on the fact that suc n is a well-formed type of **Nat**. Consequently, the type judgement must also have some notion of a context, giving us a new pair of rules:

$$\frac{}{\vdash \mathbf{1} \text{ cx}} \qquad \frac{\vdash \Gamma, x : A \text{ cx}}{\vdash \Gamma \text{ cx} \qquad \Gamma \vdash A \text{ type}}$$

6.3 Martin-Löf's Extentional Type Theory

7 Homotopy Type Theory

This section will primarily be based on the Homotopy Type Theory (Univalent Foundations of Mathematics) book.

In bridging the concepts between the two fields, consider a notion from type theory which says that *term a is of type A*. This is written $a : A$. Then, we have a correspondence in homotopy type theory where we say that *a is a point in the space A*. Similarly, every function $f : A \to B$ in type theory corresponds to a continuous map between spaces A and B. It must be noted that there is a distinction between homotopical spaces to topological spaces; we don't present the notion of open subsets and convergence, but only notions like paths between points and homotopies between paths.

Remark 7.0.1 *As such, it is often better to state that types are treated as ∞-groupoids. We come back to this later.*

7.1 Basic Notions and Intuition

At the basic level, if A is a type representing a proposition, then $a : A$ is a **witness** to the provability of A, or **evidence** of the truth of A. We note the difference between the set-theoretic statement like $a \in A$, because $a : A$ is a judgement whereas $a \in A$ is a proposition.

There is another difference where "let x be a natural number" in set theory is "let x be a thing in which we assume $x \in \mathbb{N}$", whereas in type theory we put "let $x : \mathbb{N}$".

We finally note the treatment of equality; where equality is a proposition (which is a type), so equality is a type. That is, given any elements $a, b : A$, we have a corresponding type $a =_A b$. When this type is inhabited,we say that a and b are propositionally equal. We then add an equality **judgement**, which is at the same level as the judgement $x : A$. This is called **judgemental equality** or **definitional equality**, writing $a \equiv b : A$ or $a \equiv b$. For instance, if we define a function $f : \mathbb{N} \to \mathbb{N}$ by $x \mapsto x^2$, then $f(3) \equiv 3^2$ by definition. Therefore, it doesn't make sense to negate or assume definitional equality. This lets us take witnesses for proofs along judgemental equalities, such that if $a : A$ and $A \equiv B$, then $a : B$. We may also write "$:\equiv$" for defining a function map. Finally, note that $:$ and $\equiv$ binds least.

Judgements can also depend on assumptions of the form $x : A$, where x is a variable and A is a type. For instance, we can construct an object $m + n : \mathbb{N}$ from $m, n : \mathbb{N}$. Alternatively, given $p : x =_A y$, we can construct $p^{-1} : y =_A x$. Collections of such assumptions are called the **context**, which can be thought of as a parameter space from a topological view. If the type A is an assumption, $x : A$ represents a proposition, where we assume the proposition A holds.

Note that these assumptions cannot be judgemental equality, as it is not a type with an element. We can mimic this behavior by taking $x : A$ and substituting $a : A$ to obtain a more specific type or element.

Similarly, we cannot prove judgemental equality as it cannot exhibit a witness. For instance if we take a statement like "there exists $f : A \to B$ such that $f(x) \equiv y$", we can treat this as taking two separate judgements where $f : A \to B$ for some f and make the additional judgement that $f(x) \equiv y$.

7.1.1 Function Types

Given A, B : Type, we can construct the type $A \to B$ of **functions** with domain A and codomain B. We may also call these **maps**. Given a function $f : A \to B$ and an element $a : A$, we can apply the function to obtain an element of B denoted $f(a)$ or $f\ a$.

We can construct elements of $A \to B$ by starting with $f : A \to B$ and taking

$$f(x) :\equiv \Phi$$

where x is a variable and $Phi : B$ assuming $x : A$. Alternatively, we can define this through a lambda abstraction to remove the need for naming this function:

$$(\lambda(x : A).\Phi) : A \to B$$

We may equivalently write

$$(x \mapsto \Phi) : A \to B$$

As notation, we can write "$-$" to denote an implicit λ abstraction. That is, $g(x, -)$ is equivalent to $\lambda y.g(x, y)$.

Definition 7.1.1 *The **computation rule** is a definitional equality that comes from β-reductions, such that*

$$(\lambda x.\Phi)(a) \equiv \Phi[a/x]$$

Definition 7.1.2 *We also take the **uniqueness principle for function types**, which comes from η-equivalence, such that*

$$f \equiv (\lambda x.f(x))$$

Finally, up to currying, we can define functions that take tuples of inputs like

$$f(x, y) :\equiv \Phi$$

to be

$$f :\equiv \lambda x.\lambda y.\Phi$$

or

$$f :\equiv x \mapsto y \mapsto \Phi$$

7.1.2 Universes and Families

To avoid Russell's paradox, while ensuring that we have some notion of a "universe" which is a type where elements are types, we introduce a hierarchy of universes

$$\mathcal{U}_0 : \mathcal{U}_1 : \mathcal{U}_2 : \dots$$

where every universe $\mathcal{U}_i$ is an element of $\mathcal{U}_{i+1}$. Furthermore, universes are cumulative, such that if $A : \mathcal{U}_i$, $A : \mathcal{U}_{i+1}$. Note that this implies that elements do not have unique types.

Then, when we say that A is a type, we mean that there exists some i such that $A : \mathcal{U}_i$. When we write $\mathcal{U} : \mathcal{U}$, this means $\mathcal{U}_i : \mathcal{U}_{i+1}$, with indices implicit.

To model a collection of types which vary over a given type A, we use functions $B : A \to \mathcal{U}$, which we call **families of types** (or dependent types). For instance, we can define a type family that is a familiy of finite sets $\mathrm{Fin} : \mathbb{N} \to \mathcal{U}$ where $\mathrm{Fin}(n)$ is a type with exactly n elements, which we may write $0_n, 1_n, \dots, (n-1)_n$ in order to differenciate between $\mathrm{Fin}(n)$ and $\mathrm{Fin}(m)$ for $n \neq m$. We also have a constant type family at $B : \mathcal{U}$ which is just $(\lambda(x : A).B) : A \to \mathcal{U}$. Finally note that we cannot have type families like $\lambda(i : \mathbb{N}).\mathcal{U}_i$ as there is no fixed i such that $\mathcal{U}_i$ is its codomain.

7.1.3 Dependent Function Types

Definition 7.1.3 *A* Π-***type*** *or* ***dependent function type*** *are functions whose codomain can vary depending on the element of the domain, which we call* ***dependent functions****.*

Given a type $A : \mathcal{U}$ and a family $B : A \to \mathcal{U}$, we may construct the type of dependent functions $\Pi_{(x:A)} B(x) : \mathcal{U}$. We can also write

$$\Pi_{(x:A)} B(x) \qquad \prod_{(x:A)} B(x) \qquad \prod (x : A), B(x)$$

If B is a constant family, the dependent product type is ordinary function type

$$\prod (x : A), B \equiv (A \to B)$$

The important notion is the interpretation of $a = b : A$ to be an existence of a path $p : a \rightsquigarrow b$.

8 Sources

References

[1] S. Abramsky. A generalized Kahn principle for abstract asynchronous networks. In M. Main, A. Melton, M. Mislove, and D. Schmidt, editors, Mathematical Foundations of Programming Semantics, volume 442 of Lecture Notes in Computer Science, pages 1–21. Springer Verlag, 1990.

[2] G. Markowsky. Chain-complete p.o. sets and directed sets with applications. Algebra Universalis, 6:53–68, 1976.