Universiteit Utrecht

Department of Mathematics

Bachelor Thesis

Interpreting the Second-Order Lambda Calculus using Qualitative Domains

Author:
Anton Golov

Supervisor:
Dr. Jaap van Oosten

September 2014-July 2015

Contents

Abstract

We demonstrate how the qualitative domain model for the second-order lambda calculus given by Girard [Gir86] can be formulated in terms of a **PL-Category** as described by Seely [See87].

Chapter 1

Introduction

In his paper *Categorical Semantics For Higher Ordered Polymorphic Lambda Calculus* [See87], Seely defines the concept of a **PL-Category**, which provides an abstract description of a model of the higher-order lambda calculus. We modify the requirements on a **PL-Category** to describe a model of the second-order lambda calculus and give an outline to show that the qualitative domain model for the second-order lambda calculus given by Girard [Gir86] is an instance of a **PL-Category**.

1.1 Model Theory

Most of this thesis consists of an introduction to the lambda calculus and its models. We briefly introduce the properties of category theory that we use, but readers entirely unfamiliar with it are advised to start with an introduction such as Mac Lane's *Category Theory for the Working Mathematician* [ML98] before reading this text.

The lambda calculus was invented by Alonzo Church to formalize the concept of computability. Computation occurs by mechanical rewriting of terms according to some basic rules. The two key operations are abstraction and application. Abstraction parametrises a term by a variable, and application then substitutes all occurences of the variable with the given argument.

The first formulation of the lambda calculus allowed any term to be applied to any other term; this is the so-called untyped lambda calculus. Later, it was discovered that assigning every term a type and restricting application to terms of suitable type had a number of advantages. For example, a type of natural numbers could be defined, and a natural number could then not be used as a function.

Informally, one can see this as each type specifying a set and every term describing a computation that yields an element of this set. Such an assignment of types to sets and terms to functions gives a model of the lambda calculus. Having a model gives us new ways to reason about the language as a whole; for example, if we can show that two terms are unequal in some model, then they cannot describe the same computation.

In order to allow more general models than just set-theoretic ones we will define a general concept of 'sets and functions between them', that of a category. This gives us a way to construct models that interpret types as something other than sets. We will use this to provide a categorical description of the domain-theoretic model of the lambda calculus presented by Girard [Gir86]. Unfortunately, due to time constraints, we do not provide the full verification of the validity of this model. In particular, we omit several proofs that the coherence conditions, as well as the proof that the Π functor is right-adjoint to the weakening functor. These shortcomings are described in more detail in section 8.5.

1.2 Terminology

The paper by Seely [See87] describes the higher order lambda calculus in terms of orders, operators, types and terms. In computer science circles, ‘kind’ is usually used instead of ‘order’ and ‘type constructor’ instead of ‘operator’. For the sake of consistency, I have stuck with Seely's notation, except that function orders are indicated using $A \Rightarrow \Omega$ in place of Ω^A.

Chapter 2

The Simply-Typed Lambda Calculus

The lambda calculus is a language for expressing computation. The system was originally introduced by Church [Chu32], but has since then been expanded with support for types. For a good introduction to the variations of the lambda calculus, see Barendregt [Bar13]. The variation we define is similar to the one given by Sørensen and Urzyczyn [SU98].

The two principal objects we concern ourselves with are *terms* and *types*. A type specifies the possible results of a computation, while a term specifies the computation itself. Every term must have a single type; given a term t and a type τ we say $t \in \tau$ if t has type τ. We use Ω to denote the collection of all types.

Informally, we can see a type τ as a set of values. The judgement $t \in \tau$ then states that computing t gives a value v and $v \in \tau$. This provides a good intuition for the meaning of $t \in \tau$ but only works if all computations terminate, which is not the case in general. We use a similar notation $\tau \in \Omega$ to indicate that τ is a type. As terms must have types, the judgement $t \in \tau$ implicitly requires $\tau \in \Omega$.

The language presented here is called the simply-typed lambda calculus and is denoted $\lambda_\to$.[1] In chapter 5, we will expand this system to the second-order lambda calculus.

2.1 Formal Definition

We will now introduce type and term formation rules for $\lambda_\to$.

We require the existence of a unit type $\top$. This is a type with exactly one value $*$. We see this requirement as the existence of following inference rules for typing:

$$\frac{}{\vdash \top \in \Omega} \qquad \frac{}{\vdash * \in \top}\,\top\mathcal{I}.$$

In general, when we say "there exists a type" or "there exists a term" we mean that there exist corresponding typing rules. Such rules may depend on earlier deductions. For example, we require the existence of product and function types as follows:

$$\frac{\Gamma \vdash \tau \in \Omega \quad \Gamma \vdash \sigma \in \Omega}{\Gamma \vdash \tau \wedge \sigma \in \Omega} \qquad \frac{\Gamma \vdash \tau \in \Omega \quad \Gamma \vdash \sigma \in \Omega}{\Gamma \vdash \tau \supset \sigma \in \Omega}.$$

[1]More formally, this is the simply typed lambda calculus with exponential and product types $\lambda_{(\to,\times)}$. However, from the point of view of models, this is inessential.

In cases where the context Γ is not of interest, we may instead simply write

$$\frac{\tau \in \Omega \quad \sigma \in \Omega}{\tau \wedge \sigma \in \Omega} \qquad \frac{\tau \in \Omega \quad \sigma \in \Omega}{\tau \supset \sigma \in \Omega}.$$

These types have the same role as the sets $A \times B$ and B^A have for sets A and B. For product types this is easily expressed by requiring the following terms:

$$\frac{t \in \tau \quad s \in \sigma}{\langle t, s \rangle \in \tau \wedge \sigma} \wedge\mathfrak{I} \qquad \frac{a \in \tau \wedge \sigma}{\pi_1 a \in \tau} \wedge\mathcal{E} \qquad \frac{a \in \tau \wedge \sigma}{\pi_2 a \in \tau} \wedge\mathcal{E}.$$

In order to properly specify the rules for functions we first have to introduce variables. Given a type τ, we can introduce a variable of type τ using the identity typing rule:

$$\frac{}{\Gamma, x \in \tau \vdash x \in \tau} Var\mathfrak{I}.$$

If a term $t \in \tau$ requires $x \in \sigma$ to be deduced, then we say t has a free variable x of type σ. In chapter 5 will see a similar construction for free type variables. A term with no free variables (or free type variables) is called closed.

The introduction of function terms corresponds to binding a free variable. Function application is expressed by juxtaposition and is left-associative; $f\,x$ means "f applied to x" and $f\,x\,y$ should be read as $(f\,x)\,y$. The typing rules are as follows:

$$\frac{\Gamma, x \in \sigma \vdash t \in \tau}{\Gamma \vdash \lambda x \in \sigma.\, t \in \sigma \supset \tau} \supset\mathfrak{I} \qquad \frac{f \in \sigma \supset \tau \quad s \in \sigma}{f\,s \in \tau} \supset\mathcal{E}$$

We will frequently abbreviate $\lambda x \in \sigma.\, t$ as $\lambda x^\sigma.\, t$ or, if the type of x is clear from the context, $\lambda x.\, t$.

This defines all the terms available in the simply typed lambda calculus. However, we must also define equalities between terms if our models are to satisfy the intuitive requirements described above. An equality judgement is of the form $t = t'$, and means that the interpretation of t and the interpretation of t' must be equal in all models. Just as $t \in \tau$ implicitly requires $\tau \in \Omega$, so does $t = t'$ require that $t \in \tau$ and $t' \in \tau$.

We require the following equalities to hold for the unit type,

$$\frac{t \in \top}{t = *} \top\mathrm{red}$$

for products,

$$\frac{t \in \tau \quad s \in \sigma}{\pi_1 \langle t, s \rangle = t} \wedge\mathrm{red} \qquad \frac{t \in \tau \quad s \in \sigma}{\pi_2 \langle t, s \rangle = s} \wedge\mathrm{red} \qquad \frac{a \in \tau \wedge \sigma}{a = \langle \pi_1 a, \pi_2 a \rangle} \wedge\mathrm{exp}$$

and for functions

$$\frac{\Gamma, x \in \sigma \vdash t \in \tau \quad \Gamma \vdash s \in \sigma}{\Gamma \vdash (\lambda x.\, t)\, s = t[s/x]} \supset\mathrm{red} \qquad \frac{\Gamma \vdash f \in \sigma \supset \tau}{\Gamma \vdash f = \lambda x.\, f\,x} \supset\mathrm{exp}.$$

The notation $t[s/x]$ denotes t with all occurrences of x replaced by s. Care must be taken to avoid changing the meaning of bound variables, and which variables exactly must be changed. For example, we do not want the following equality to hold:

$$(\lambda y.\, \lambda y.\, y)\lambda x.\, x = \lambda y.\, \lambda x.\, x.$$

On the other hand, we also do not want the names of the variables to have any influence on the meaning of the term; for example, we want to have $\lambda x.\, x = \lambda y.\, y$.

For a formal definition of terms and substitution that addresses these issues, see Sørensen and Urzyczyn's book [SU98], which provides an explicit construction of both. We side-step the issue by requiring that all variables in a term be bound at most once, and assume that they are renamed after substitution if this property is not satisfied. Similarly, if a deduction rule mentions a variable, we assume that that variable does not occur free in any terms, unless explicitly mentioned. For example, in $\supset$exp we assume that x is not free in f.

Finally, we require that equality be a congruence relation on terms, and that if two terms differ only in the names of bound variables (*alpha-equivalent* terms) that they be equal. This is done by adding additional equality constraints where necessary. In further sections we assume that these properties are preserved when adding new equality deductions and binding operations.

2.2 $\lambda_\to$Theories

For the language above to be useful it must be possible to require the existence of more types and terms than just those that we have already named. However, it is inconvenient to modify the definition of the language simply for the sake of adding a type. When constructing a model for our new language, we would have to check a great many properties, most of which have nothing to do with the changes we have made.

In order to make this more manageable we define what it means to be a $\lambda_\to$ *theory*, a structure that can represent most variations that we may want to express. Rather than building a model for the lambda calculus, we build a model for the lambda calculus together with an arbitrary $\lambda_\to$ theory. This makes the model harder to construct initially, but allows us to then reuse it.

We define a $\lambda_\to$ theory to be a set of judgements of the form $\vdash \tau \in \Omega$, $\Gamma \vdash t \in \tau$, and $\Gamma \vdash t = t'$. As above, a judgement of the form $\Gamma \vdash t \in \tau$ or $\Gamma \vdash t = t'$ may only be present if the judgements required to make it ($\vdash \tau \in \Omega$ and $\Gamma \vdash t \in \tau, \Gamma \vdash t' \in \tau$ respectively) are also present. Furthermore, we require that our theories be coherent; that they are closed under weakening (of Γ), transitivity of equality, substitution, abstraction and application. For a formal description of the requirements of the requirements we refer the reader to chapter 7 of Van Oosten's lecture notes [vO02].

Chapter 3

Cartesian Closed Categories

We now take a step away from the lambda calculus and look at the language we will be using to describe models of it: that of category theory.

A *category* $\mathbf{C}$ is a collection of *objects* $\mathcal{O}$ together with for every two objects A, B a collection $\mathrm{Hom}_{\mathbf{C}}(A, B)$ of *morphisms.* We write $f : A \to B$ for $f \in \mathrm{Hom}_{\mathbf{C}}(A, B)$. For all $A, B, C \in \mathcal{O}$ and every $f : A \to B$ and $g : B \to C$, the *composition* $g \circ f : A \to C$ is defined. Moreover, composition is associative and every object A has an identity morphism $1_{\mathbf{A}}$ with respect to it.

The motivating example of a category is the category **Set** of sets. Let $\mathcal{O}$ be the collection of all sets and $\mathrm{Hom}_{\mathbf{Set}}(A, B)$ the set of functions from A to B. Composition is simply function composition, of which the identity function is the identity.

We will use categories as models of the lambda calculus as follows. Every type will be assigned an object, and every term of type B with a free variable of type A will be assigned a morphism $A \to B$. Composition of morphisms will correspond to substitution of terms; if $t(x)$ is a term of type C with a free variable of type B, and $t'(x)$ is a term of type B with a free variable of type A, then $t(t'(x))$ is a term of type C with a free variable of type A. The identity is simply the identity term x, where x is the free variable.

At first glance, it is not clear that this also allows us to interpret closed terms and open terms with multiple free variables, and even if we can, that there may be no connection between the interpretation of a term $t(x)$ with x a free variable and a term $\lambda x \in \tau.\, t(x)$. The rest of this chapter will introduce the concepts necessary to impose the conditions we must place on a category for the model to be well-behaved. The next chapter will show how these conditions lead to a correspondence between such categories and lambda calculi.

3.1 Examples

It is useful to first look at some examples of categories. In this chapter, a recurring theme will be that we find some collection of objects and morphisms, notice that there is an obvious composition relation to be defined and that it has an identity, and then put these together into a category.

A number of examples come from algebraic structures involving a set together with some operations on it. For example, vector spaces and linear transformations form a category, as do groups and group homomorphisms, rings and ring homomorphisms, topological spaces and continuous functions, and so forth.

Another important example is the category with propositions as objects and a unique morphism $P \to Q$ if Q is provable from P. For example, there would then be an morphism $P \to P \vee Q$, but no morphism $P \vee Q \to P$. Interestingly, there is a close relationship between such categories and models of the lambda calculus. This is known as the Curry-Howard Isomorphism.

Furthermore, for every category $\mathbf{C}$ we have a dual category $\mathbf{C}^{\mathrm{op}}$ which has the same objects as C but where all morphisms are reversed.

3.2 Categorical Products

We will start by solving the problem of terms with $n \neq 1$ free variables. Suppose a term $t(\vec{x})$ of type σ depends on free variables $x_1, \ldots, x_n$ with types $\tau_1, \ldots, \tau_n$. We can just as well regard this as a term $t(x)$ which depends on a free variable x of type $\tau_1 \times \ldots \times \tau_n$. However, for this intepreration to be coherent we must specify how the interpretation of $\tau_1 \times \ldots \times \tau_n$ must relate to the interpretations of $\tau_1, \ldots, \tau_n$.

Given a collection of objects $\vec{A} = A_1, \ldots, A_n$, we say that the *(categorical) product* of $\vec{A}$ is an object B together with for each $1 \leq i \leq n$ a morphism $\pi_i : B \to A_i$, such that for any other object C together with morphisms $\pi'_i : C \to A_i$ there exists a unique morphism $f : C \to B$ such that $\pi_i \circ f = \pi'_i$ for all $1 \leq i \leq n$.

If $n = 1$, then $B \cong A_1$ by taking $C = A_1$ and $\pi'_1 = 1_{A_1}$. The case for $n = 2$ can be illustrated as follows:

$$\begin{tikzcd}
 & C \arrow[dl, "\pi'_1"'] \arrow[d, "!f"] \arrow[dr, "\pi'_B"] & \\
A_1 & A_1 \times A_2 \arrow[l, "\pi_1"] \arrow[r, "\pi_2"'] & A_2
\end{tikzcd}$$

Note that products are defined up to isomorphism, and are associative and commutative up to isomorphism.

An interesting special case is $n = 0$. This is called the *terminal object* of the category, and is determined by the property that if B is terminal, then for every object A, there is a unique morphism $A \to B$. We will usually denote the terminal object with 1. It is worth noting that 1 is the identity of the product.

In the category **Set**, every singleton set is a terminal object. Note that there is a bijective correspondence between functions $\{*\} \to A$ and the set-theoretic elements of A for every A. We will employ this terminology for the general case as well: a morphism $f : 1 \to A$ is called a (global) element of A. In the context of the categories we will work with, the elements will play a meaningful role, but this is not always the case: for example, in the category of groups, every object has exactly one element.

3.3 Functors

A *functor* is a structure-preserving mapping between categories. It sends an object A to an object $F(A)$, a morphism $f : A \to B$ to a morphism $F(f) : F(A) \to F(B)$, and preserves composition and identities. Composition of functors is defined in the obvious way, and every category has an identity functor that sends all objects and morphisms to themselves. Given a collection of categories, we can thus construct a category with these categories as objects and functors between them as morphisms. We will use **Cat** to denote the category of all categories $\mathbf{C}$ where for every object A, B of $\mathbf{C}$, $\mathrm{Hom}_{\mathbf{C}}(A, B)$ is a set (as opposed to a proper class).[1]

An example of a functor from **Set** to itself is $(-) \times B$ which sends every set A to $A \times B$ and every function $f : A \to A'$ to a function $f \times id_B : A \times B \to A' \times B$ which sends the pair (a, b) to $(f(a), b)$. This functor will have an important relation to the functor $(-)^B$, which sends every object A to A^B, the set of functions from B to A, and which sends a function $f : A \to A'$ to the function $f^B : A^B \to A'^B$ which acts on functions $B \to A$ by composition. That is, if we regard $g : B \to A$ as an element of A^B, then $f^B(g) = f \circ g$.

An example of a functor that has different source and target categories is the forgetful functor from the category of groups to **Set**. This sends a group $(A, \cdot)$ to its underlying set A and a morphism $A \to B$ to the set-theoretic function between the underlying sets.

If a functor $F : \mathbf{C} \to \mathbf{D}$ and $G : \mathbf{D} \to \mathbf{C}$ have the property that $G \circ F = 1_{\mathbf{C}}$ and $F \circ G = 1_{\mathbf{D}}$ then we say F and G are each other's inverses and are thus an isomorphism between $\mathbf{C}$ and $\mathbf{D}$. The categories $\mathbf{C}$ and $\mathbf{D}$ are then said to be isomorphic.

A functor $F : \mathbf{C}^{\mathrm{op}} \to \mathbf{D}$ is sometimes also called a contravariant functor from $\mathbf{C}$ to $\mathbf{D}$.

3.4 Natural Transformations

Let $F, G : \mathbf{C} \to \mathbf{D}$ be functors. We can regard $\mathbf{C}$ and $\mathbf{D}$ as graphs, with objects being nodes and morphisms being paths. Given an object A in $\mathbf{C}$, both $F(A)$ and $G(A)$ are objects in $\mathbf{D}$, which we can regard as nodes. The question then arises whether we can connect them, and whether this connection can be done uniformly over all objects.

A *natural transformation* $\alpha : F \Rightarrow G$ assigns to every object A of $\mathbf{C}$ a morphism $\alpha_A : F(A) \to G(A)$. This can be seen as a connection from $F(A)$ to $G(A)$ in $\mathbf{D}$. The *naturality condition* is that given a morphism $f : A \to B$ in $\mathbf{C}$, we have $G(f) \circ \alpha_A = \alpha_B \circ F(f)$.

Visually, this means that for every two objects A, B of $\mathbf{C}$ and every morphism $f : A \to B$, the following diagram commutes:

$$\begin{tikzcd}
F(A) \arrow[r, "\alpha_A"] \arrow[d, "F(f)"'] & G(A) \arrow[d, "G(f)"] \\
F(B) \arrow[r, "\alpha_B"'] & G(B)
\end{tikzcd}$$

Every functor F has an associated identity natural transformation, with the identity for each componenent. Given a natural transformation $\alpha : F \Rightarrow G$ and a natural transformation $\beta : G \Rightarrow H$ we can obtain a natural transformation $\beta \circ \alpha : F \Rightarrow H$ by composing each component. Given categories $\mathbf{C}, \mathbf{D}$ this gives rise to a category with functors $\mathbf{C} \to \mathbf{D}$ as objects and natural transformations as morphisms.

An important related concept is that of a natural isomorphism. A natural isomorphism between F and G is a natural transformation that has an inverse. This is usually phrased as "$F(A)$ and $G(A)$ are isomorphic naturally in A".

Natural isomorphisms allow us to express a notion of equivalence of categories that is weaker than isomorphism but still preserves many useful properties. We say that $\mathbf{C}$ and $\mathbf{D}$ are equivalent

[1]Most texts also require that the collection of objects of $\mathbf{C}$ form a set. However, for our purposes it is convenient if **Set** is an object of **Cat** which this condition would make impossible.

if there exist functors $F : \mathbf{C} \to \mathbf{D}$ and $G : \mathbf{D} \to \mathbf{C}$ such that $G \circ F$ is naturally isomorphic to $1_{\mathbf{C}}$ and $F \circ G$ is naturally isomorphic to $1_{\mathbf{D}}$.

3.5 Adjoints

Let $\mathbf{C}, \mathbf{D}$ be categories and $F : \mathbf{C} \to \mathbf{D}$ and $G : \mathbf{D} \to \mathbf{C}$ be functors. We say that F is *left adjoint* to G, or equivalently, G is *right adjoint* to F, if there is a natural isomorphism

$$\mathrm{Hom}_{\mathbf{D}}(F(A), B) \cong \mathrm{Hom}_{\mathbf{C}}(A, G(B)).$$

This is denoted $F \dashv G$.

Adjoints play a key role in category theory. In this text we will use them as a generalisation of the relation

$$\mathrm{Hom}_{\mathbf{Set}}(A \times B, C) \cong \mathrm{Hom}_{\mathbf{Set}}\left(A, C^B\right)$$

between $(-) \times B$ and $(-)^B$ in **Set** which characterises the set of functions C^B. In the following section, we will generalise this idea to define so-called exponential objects in more general categories, where C^B represents the morphisms from B to C.

This is by far not the only example of an adjunction. The forgetful functor mentioned above is right adjoint to the free functor that sends every set A to the free group generated by A and every function $f : A \to B$ to the homomorphism induced by how f maps the generators.

3.6 Exponential Objects

As remarked above, the functors $(-) \times B$ and $(-)^B$ are related by an adjunction. This means that the sets $\mathrm{Hom}_{\mathbf{Set}}(A \times B, C)$ and $\mathrm{Hom}_{\mathbf{Set}}\left(A, C^B\right)$ are isomorphic. In particular, if we choose $A = 1$, we have

$$\mathrm{Hom}_{\mathbf{Set}}(B, C) \cong \mathrm{Hom}_{\mathbf{Set}}(1 \times B, C) \cong \mathrm{Hom}_{\mathbf{Set}}\left(1, C^B\right).$$

Since, in **Set** we can identify the set of morphisms from 1 to C^B with C^B, this shows that C^B is isomorphic to set of morphisms $B \to C$. We now specify this for arbitrary categories.

Let $\mathbf{C}$ be a category with all finite products and with a functor $(-)^B$ that is a right adjoint to $(-) \times B$. For every object C of $\mathbf{C}$, we call C^B an *exponential object*. We say that $\mathbf{C}$ has all exponential objects if a functor $(-)^B$ that is right-adjoint to $(-) \times B$ exists for every B. Notice that this means that

$$\mathrm{Hom}_{\mathbf{C}}(B, C) \cong \mathrm{Hom}_{\mathbf{C}}(1 \times B, C) \cong \mathrm{Hom}_{\mathbf{C}}\left(1, C^B\right).$$

In other words, the set of elements of C^B is isomorphic to the set of morphisms $B \to C$.

An alternative definition of the exponential object can be obtained by generalising the concept of evaluation. In **Set**, for all sets X, Y there is a morphism $ev : X \times Y^X \to Y$. We say that Y^X is the exponential object in some category $\mathbf{C}$ if there is a morphism $ev : X \times Y^X \to Y$ such that for every object Z and every morphism $f : X \times Z \to Y$ there exists a unique morphism $g : Z \to Y^X$ such that $ev \circ (1_X \times g) = f$. In a diagram:

We will use the ev morphism to interpret application of terms in our model.

$$\begin{tikzcd}
X \times Z \arrow[d, "1_X \times !g"'] \arrow[dr, "f"] & \\
X \times Y^X \arrow[r, "ev"'] & Y
\end{tikzcd}$$

3.7 Cartesian Closedness

We call a category *Cartesian closed* if it has all finite products and all exponential objects. Note that this is equivalent to a category having a terminal object, binary products, and all exponential objects.

Cartesian closed categories are exactly the categories that make for good models of the lambda calculus. Products are used to interpret product types and exponential objects are used to interpret function types. Furthermore, the terminal object allows us to represent closed terms as elements of some type A via a morphism $1 \to A$.

Chapter 4

CCCs as models of $\lambda_{\rightarrow}$

In this chapter we will define what it means to be a model of the simply-typed lambda calculus, and demonstrate a correspondence between $\lambda_{\rightarrow}$theories and Cartesian closed categories. The material here is based strongly on chapters 10 and 11 of Lambek and Scott's book [LS88].

Given a $\lambda_{\rightarrow}$ theory T, we can construct a Cartesian closed category $\mathbf{Cl}(T)$ in which every term can be interpreted, and which is the most general possible model, so two terms have the same interpretation if and only if they are identical up to equalities in the theory.

Given a Cartesian closed category $\mathbf{C}$, we can perform the opposite transformation by taking a $\lambda_{\rightarrow}$ theory $\mathcal{L}(C)$ that has a type for every object of $\mathbf{C}$ and a term for every morphism $A \rightarrow B$, together with equalities for every composition. Moreover, every term that can be formed from $\mathcal{L}(C)$ is equal to a term that is contained in $\mathcal{L}(C)$; that is, to a morphism of $\mathbf{C}$.

Composing these transformations in either order does not necessarily give the identity, as both the resulting category and the resulting theory can be larger than the category or theory we started with. However, it is still essentially correct to regard them as each other's inverse. If we start with a category $\mathbf{C}$, construct a theory T, and then construct a category $\mathbf{C}'$ from that theory, the categories $\mathbf{C}$ and $\mathbf{C}'$ will be equivalent. On the other hand, if we start with a theory T, construct a category $\mathbf{C}$, and then construct a theory T' from it, any term we can form from T' will be equal to a term we can form from T. In this sense, neither transformation allows to express anything new.

Once we see that a theory T and its corresponding category $\mathbf{Cl}(T)$ are essentially the same, we can define a model of T entirely in terms of $\mathbf{Cl}(T)$. A model of T is a category $\mathbf{C}$ together with a functor $\mathbf{Cl}(T) \rightarrow \mathbf{C}$ that preserves finite products and exponents. The image of such a functor is again a Cartesian closed category.

4.1 A Naive Model

We will start by showing how a model of the lambda calculus can be defined naively, by directly mapping types to objects and terms to morphisms of some given Cartesian closed category $\mathbf{C}$.

Suppose we have a $\lambda_{\rightarrow}$ theory T with type judgements $\vdash T_1, \vdash T_2, \ldots$, term judgements $\Gamma_1 \vdash t_1 \in \tau_1, \Gamma_2 \vdash t_2 \in \tau_2, \ldots$ and equality judgements $\Delta_1 \vdash s_1 = s_1', \Delta_2 \vdash s_2 = s_2', \ldots$.

We now define a function $[\![.]\!]$ that maps each type to an object of $\mathbf{C}$. Suppose we already have a definition $[\![T_i]\!]$ for every type judgement $\vdash T_i$ in our theory T. Then we can extend $[\![.]\!]$ to all types as follows:

- $[\![\top]\!]$ is the terminal object of $\mathbf{C}$.

- $[\![A \wedge B]\!] = [\![A]\!] \times [\![B]\!]$.
- $[\![A \supset B]\!] = [\![B]\!]^{[\![A]\!]}$.

Suppose now that $\Gamma = x_1 \in \sigma_1, x_2 \in \sigma_2, \ldots$. We define $[\![\Gamma]\!] = [\![\sigma_1]\!] \times [\![\sigma_2]\!] \times \ldots$. As a context is always finite, $[\![\Gamma]\!]$ is a well-defined object.

We now extend $[\![.]\!]$ to map every term judgement $\Gamma \vdash t \in \tau$ to a morphism $[\![t]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$. Suppose that we have already defined $[\![.]\!]$ on the term judgements in our theory. We then extend it as follows:

- $[\![\Gamma \vdash x_i \in \tau]\!]$ is the projection morphism from $[\![\Gamma]\!]$ to its i-th component.
- $[\![\Gamma \vdash st \in \tau]\!]$ is $ev([\![s]\!], [\![t]\!])$, where ev is the evaluation morphism introduced in section 3.6. This is well-typed, as s is necessarily of the type $\sigma \supset \tau$ for some σ, and t is then of type σ.
- $[\![\Gamma \vdash \lambda x^\sigma . t \in \tau]\!]$ is $ab([\![\Gamma, x \in \sigma \vdash t \in \tau]\!])$, where ab is the isomorphism $\mathrm{Hom}_{\mathbf{C}}([\![\Gamma]\!] \times \sigma, \tau) \to \mathrm{Hom}_{\mathbf{C}}([\![\Gamma]\!], \tau^\sigma)$ that exists because $(-)^\sigma$ is right-adjoint to $(-) \times \sigma$.

This gives us an interpretation of the lambda calculus with theory T in some Cartesian closed category $\mathbf{C}$. This interpretation gives a model if $[\![\Gamma \vdash s_i]\!] = [\![\Gamma \vdash s'_i]\!]$ for every judgement $\Gamma \vdash s_i = s'_i$ in T.

4.2 Classifying Categories

We will now construct, given a $\lambda_\to$ theory, the most general category that can be treated as its model. Any other model can be constructed by mapping this category into some other Cartesian closed category.

For the sake of simplicity, we will first define a category that clearly satisfies the requirements, but is unnecessary large. We will then show that certain objects are isomorphic and we can thus restrict ourselves to a simpler category.

Let T be a $\lambda_\to$ theory and let X be the set of types we can form from types in T together with the unit type, product types, and arrow types. Let $\mathbf{Cl}^-(T)$ be the category with tuples of elements of X as objects and for two objects $\vec{\tau} = (\tau_1, \ldots, \tau_n), \vec{\sigma} = (\sigma_1, \ldots, \sigma_m)$ let the morphisms $\vec{\tau} \to \vec{\sigma}$ be m-tuples $\vec{f} = (f_1, \ldots, f_m)$ with f_i an equivalence class of terms of type σ_i with free variables x_j for each τ_j. Two terms are in the same equivalence class if they are equal by the equalities of the simply-typed lambda calculus and the theory T.

By identifying objects of the form (τ) with the corresponding type τ we see that we can interpret any type as an object and any term t of type σ with free variables of types $\tau_1, \ldots, \tau_n$ as a corresponding morphism $(t) : (\tau_1, \ldots, \tau_n) \to (\sigma)$. It is also easy to see that this category has finite products simply by concatenating the tuples in question, with the empty tuple being the terminal object.

However, while $\mathbf{Cl}^-(T)$ satisfies our requirements, it has far more objects than necessary. Notice that the n-tuple $(\tau_1, \ldots, \tau_n)$ is naturally isomorphic to the product $(\tau_1) \times \ldots \times (\tau_n)$, with the tuple of projection functions as the morphism one way, and the pairing function as the morphism the other way. This means that rather than interpreting a term of type σ with open variables of type $\tau_1, \ldots, \tau_n$ as a morphism $(\tau_1, \ldots, \tau_n) \to \sigma$, we can regard it as a morphism $\tau_1 \times \ldots \times \tau_n \to \sigma$. The only objects that we still use are those that have been identified with types. We can thus use the following, simpler, category.

Definition 4.1. The classifying category $\mathbf{Cl}(T)$ of a $\lambda_\to$ theory T has as its objects the types of T together with the unit type, product types, and function types. Given objects σ, τ, the morphisms $\sigma \to \tau$ are equivalence classes of terms of type τ with a free variable of type σ, where two terms are equivalent if they are equal by the equations of the simply-typed lambda calculus together with the theory T.

Note that $\mathbf{Cl}(T)$ is a subcategory of $\mathbf{Cl}^{-}(T)$. We can thus interpret any term first in $\mathbf{Cl}^{-}(T)$, and then precompose it with the corresponding isomorphism to get a morphism in $\mathbf{Cl}(T)$.

4.3 Internal Languages

To show that the construction of the classifying category does not lose us any properties we define an inverse operations that constructs a theory given a Cartesian closed category. The λ-calculus given by this theory is called the *internal language* of the Cartesian closed category. Conceptually, the language of a theory T is equivalent to the internal language of $\mathbf{Cl}(T)$, and the classifying category of the internal language of some Cartesian closed category $\mathbf{C}$ is equivalent to $\mathbf{C}$ itself.

Both of these claims are discussed and proved by Lamkek and Scott [LS88]. We will simply state the construction of the theory of the internal language.

Given a Cartesian closed category $\mathbf{C}$ the theory of its internal language is denoted $\mathcal{L}(\mathbf{C})$. Its types are the objects of $\mathbf{C}$. For every element f of A (that is, for every morphism $f : 1 \to A$), there is a constant f of type A in the theory. As $\mathbf{C}$ is Cartesian closed, this means that given a morphism $f : A \to B$ there is a corresponding morphism $f' : 1 \to B^A$. Letting x be a free variable of type A, we get a term $f'x$ of type B which we can express in $\mathcal{L}(\mathbf{C})$. This means that we can find, for every morphism of $\mathbf{C}$, a corresponding morphism of $\mathbf{Cl}(\mathcal{L}(\mathbf{C}))$ in an injective manner.

The problem is that this mapping does not respect composition and is thus not a functor. For this purpose we need to include equalities in our theory. Let $f : A \to B$ and $g : B \to C$ be morphisms in $\mathbf{C}$. We require that the theory also contain the equality $\lambda x \in A.\, g'(f'x) = \lambda x \in A.\, (g \circ f)'x$, where we indicate with f' the element of B^A corresponding to f, and idem for g and $g \circ f$.

With this extension, the inclusion of $\mathbf{C}$ into $\mathbf{Cl}(\mathcal{L}(\mathbf{C}))$ is an injective functor, and furthermore this injection is an equivalence of categories.

4.4 Models of $\lambda_{\to}$

We have now established that the classifying category of a theory is equivalent to the theory itself. Given a theory T, we can definitely see $\mathbf{Cl}(T)$ as a model of it. After all, just as we wanted, we have an object for every type and a morphism for every term. However, this isn't very interesting, as every type and term are simply interpreted as themselves.

We define a model of a $\lambda_{\to}$ theory T by mapping $\mathbf{Cl}(T)$ into a different Cartesian closed category using some functor F. This allows us to interpret a type or term first by looking at the corresponding object or morphism in $\mathbf{Cl}(T)$, and then following the functor to find the interpretation in the category we are making the model in. Coherence demands a number of further requirements:

Definition 4.2. A *model* F of a $\lambda_{\to}$ theory T is a Cartesian closed category $\mathbf{C}$ together with a functor $F : \mathbf{Cl}(T) \to \mathbf{C}$ which preserves the terminal object, products, and exponents.

Let us finish with an example of a model. Let T be the theory with one type N and the constants $0 \in N$ and $s \in N \supset N$ as well as a recursor $r \in ((N \supset N) \times N \times N) \supset N$ with the equalities $r(f, 0, x) = x$ and $r(f, sn, x) = f(r(f, n, x))$ (where the parametrisation over f, n, and x is implied). In this case, $\mathbf{Cl}(T)$ will have as objects 1, N, $N \times N$, $N \supset N$, $(N \supset N) \supset N$, etc. A possible set-theoretic model of this theory would be a functor that sends N to $\mathbb{N}$, 1 to the singleton set, the term 0 (seen as a morphism $1 \to N$) to the constant 0 function, s to the successor function, and r to the recursor function. This would be a fairly natural choice of a model, but it is by far not the only one. For example, $\mathbb{N}$ can be replaced in this construction by any other model of Peano arithmetic.

Chapter 5

The Second-Order Lambda Calculus

In chapter 2 we introduced the simply-typed lambda calculus. We will now expand this definition with the second-order parts and expand $\lambda_{\rightarrow}$ theories to PL theories by allowing types to be parametrised by type variables and introducing types and terms that make use of these parametrised types.

The approach taken here is syntactically based on Seely's description of the polymorphic lambda calculus [See87]. However, as we are limiting ourselves to the second order, we omit the existential type $\Sigma\alpha.\,\tau$ from our language and show in section 5.3 how it can be expressed in terms of the universal type.

5.1 Intuition

Suppose that τ is a type expression with one free variable α. It will be convenient to say that t is a term of type τ, meaning simply that t is well-formed under the assumption that α is a type. In general, t may have free (term) variables with types that contain α. This may appear like abuse of terminology, but it will be justified when we rephrase it in terms of contexts: by $t \in \tau$, we simply mean that $\alpha \in \Omega \vdash t \in \tau$ is derivable.

Given a type σ, it makes sense to talk about terms of type $\tau[\sigma/\alpha]$. This immediately suggests a kind of "terms t of parametrised types": namely, we require that $t[\sigma/\alpha] \in \tau[\sigma/\alpha]$ for all $\sigma \in \Omega$. We introduce a type $\Pi\alpha.\,\tau$ to represent such terms. In order to distinguish between the term t with a free type variable and a corresponding term of type $\Pi\alpha.\,\tau$, we denote the latter by $\Lambda\alpha.\,t$.

To give an example, the term $\lambda f \in \alpha \supset \alpha.\, \lambda x \in \alpha.\, f(fx)$ has type $(\alpha \supset \alpha) \supset (\alpha \supset \alpha)$ for all $\alpha \in \Omega$. We can thus parametrise the type by a type variable and get the term $\Lambda\alpha.\,\lambda f \in \alpha \supset \alpha.\,\lambda x \in \alpha.\, f(fx)$ of type $\Pi\alpha.\,(\alpha \supset \alpha) \supset (\alpha \supset \alpha)$. When we want to apply the term, we specify the type we want to apply it with just as we would specify a value we give as an argument.

We start by introducing the *universal type*. Suppose that τ is a type expression with a free type variable α. We can then always form the type $\Pi\alpha.\,\tau$ (though of course, this type need not be inhabited). Just like we consider terms that are alpha-equivalent to be equal, so too do we consider types equal if they are alpha-equivalent.

Given a term t of type τ with one free type variable α, we can form the term $\Lambda\alpha.\,t$ if for every $\sigma \in \Omega$, $t[\sigma/\alpha]$ is a term of type α. Note that this binds the type variable α, so that it is no longer free in $\Lambda\alpha.\,t$. The inverse operation specialises a term of universal type to a specific type.

If $\Lambda\alpha.\, t \in \Pi\alpha.\, \tau$, then $(\Lambda\alpha.\, t)\{\sigma\}\tau[\sigma/\alpha]$ and furthermore, $(\Lambda\alpha.\, t)\{\sigma\} = t[\sigma/\alpha]$.

To formalise this approach, we reformulate it in terms of judgements. This also immediately extends the construction to terms and types with multiple free type variables.

5.2 Judgement Trees

We will now rephrase the above in terms of contexts and judgement trees. We start by allowing type variables:

$$\frac{}{\alpha \in \Omega \vdash \alpha \in \Omega}.$$

We now introduce the type formation and alpha-equivalence rules. Note that in $\alpha -$ equiv we assume that α is not free in σ, β is not free in τ, and γ is free in neither τ nor σ.

$$\frac{\Gamma, \alpha \in \Omega \vdash \tau \in \Omega}{\Gamma \vdash \Pi\alpha.\, \tau \in \Omega} \qquad \frac{\Gamma \vdash \Pi\alpha.\, \tau \in \Omega \quad \Gamma \vdash \Pi\beta.\, \sigma \in \Omega \quad \Gamma \vdash \tau[\gamma/\alpha] = \sigma[\gamma/\beta]}{\Gamma \vdash \Pi\alpha.\, \tau = \Pi\beta.\, \sigma}\,\alpha\text{−equiv}$$

The term formation rules are somewhat more complicated, as we must ensure that the type α does not occur in any free term variables of t. We do this by making $\alpha \in \Omega$ the rightmost judgement of the context of $\mathcal{J}$. Hence, no judgement in Γ can depend on $\alpha \in \Omega$, and thus α does not occur in the type of any free variable of t.

The rules are as follows:

$$\frac{\Gamma, \alpha \in \Omega \vdash t \in \tau}{\Gamma \vdash \Lambda\alpha.\, t \in \Pi\alpha.\, \tau}\,\Pi\mathcal{I} \qquad \frac{\Gamma \vdash t \in \Pi\alpha.\, \tau \quad \Gamma \vdash \sigma \in \Omega}{\Gamma \vdash t\{\sigma\} \in \tau[\sigma/\alpha]}\,\Pi\mathcal{E}.$$

The equality rules mentioned above are, of course, also present:

$$\frac{\Gamma, \alpha \in \Omega \vdash t \in \tau \quad \sigma \in \Omega}{\Gamma \vdash (\Lambda\alpha.\, t)\{\sigma\} = t[\sigma/\alpha]}\,\Pi\text{red} \qquad \frac{\Gamma \vdash t \in \Pi\alpha.\, \tau}{\Gamma \vdash t = \Lambda\beta.\, \tau\{\beta\}}\,\Pi\text{exp}.$$

In Πexp we assume that β is not a free type variable of τ

Note that this formulation allows terms such as $\Lambda\alpha.\, \Lambda\beta.\, \lambda x^{\alpha}.\, \lambda y^{\beta}.\, x$, which has type $\Pi\alpha.\, \Pi\beta.\, \alpha \supset (\beta \supset \alpha)$. Here, the subterm $\lambda x^{\alpha}.\, \lambda y^{\beta}.\, x$ depends on two type variables.

The notion of a $\lambda_{\rightarrow}$ theory extends immediately to a PL theory by allowing terms and types to use universal types, as well as allowing equalities between these terms and types.

5.3 Recovering Σ types

A possible further addition to the system would be an *existential type* $\Sigma\alpha.\, \sigma$, a term s of which would represent a term of type $\sigma[\tau/\alpha]$ for *some* (rather than all) $\tau \in \Omega$. We have chosen not to make this part of our language, as it can be expressed in terms of the universal type. Namely, let

$$\Sigma\alpha.\, \sigma = \Pi\beta.\, (\Pi\alpha.\, \sigma \supset \beta) \supset \beta.$$

We can define Seely's $I_{\Sigma\alpha.\, \sigma, \tau}$ by

$$I_{\Sigma\alpha.\, \sigma, \tau} = \lambda x^{\tau}.\Lambda\beta.\, \lambda f^{\Pi\sigma.\, \alpha \supset \beta}.\, f\{\tau\}\, x$$

and V is obtained by treating a as a term of type $\Pi\alpha.\,\sigma \supset \rho$, where α is not free in ρ, and then defining

$$(V\alpha.\,a)(I_{\Sigma\alpha.\,\sigma,\tau}t) = (I_{\Sigma\alpha.\,\sigma,\tau}t)\{\rho\}a.$$

Once the definitions are expanded, we see that this is equal to $a\{\tau\}t$.

Chapter 6

PL Categories

In order to define what a model of a $\lambda_{\to}$ theory is, we defined a classifying category that corresponds to the theory and then defined a model as a functor from the classifying category to some category $\mathbf{C}$ which preserved the Cartesian closed structure. Once we had done this, it became clear that we can regard any Cartesian closed category for which we can interpret all terms as a model.

We will do the same to define a model of a PL theory, by finding a "second-order classifying category" and stating what properties the 'functor' will preserve. Once we have done this, we will characterise the image of this functor, which will give us the concept of a **PL-Category**. We will then speak of a certain **PL-Category** being a model of a PL theory, leaving the corresponding functor implicit.

The definition of a PL category given here is a modification of Seely's work [See87] to adapt it to the second-order lambda calculus. The primary change is that (as suggested in section 2.6 of Seely's paper) we have removed the requirement that $\mathbf{S}$ be exponentiable. Additionally, we only require that the weakening functor have a right adjoint, as opposed to both a right and a left adjoint. This is a consequence of Σ being expressible in terms of Π in the second order lambda calculus, as we saw in the previous chapter.

6.1 Polymorphic Classifying Categories

Unlike the simply-typed lambda calculus, where the only dependency was of terms on terms, we have three kinds of dependencies here: terms can still depend on terms, but additionally terms and types can each depend on types.

First of all, we solve the problem of types depending on types by introducing something akin to a classifying category $\mathbf{S}$ for types. For every natural number n, $\mathbf{S}$ has an object Ω^n representing the set of n-tuples of types. A morphism $f : \Omega^n \to \Omega^m$ is an m-tuple of type expressions in n free type variables. Note that for any n, $\mathrm{Hom}_{\mathbf{S}}(\Omega^n, \Omega)$ then corresponds to the set of type expressions with n free variables. We abbreviate $\Omega^1 = \Omega$ and $\Omega^0 = 1$. Note that $\mathbf{S}$ has all finite products, with 1 being the terminal object and $\Omega^n \times \Omega^m = \Omega^{n+m}$.

We will still use a Cartesian closed category to model how terms can depend on terms; in fact, given that the simply-typed lambda calculus is contained in the second-order lambda calculus, we should expect every model of the latter to contain a model of the former. We will do this by making use of the fact that we can assume that the free variables in a term depend on the same set of free type variables as the term itself does. We can thus have separate categories for terms that depend on different sets of free type variables.

This idea gives rise to the following structure. For every $n \in \mathbb{N}$, let $\mathbf{G}_n$ be the category with as objects types with n free type variables and as morphisms $X \to Y$ terms of type Y with a free variable of type X. Note that by the definition of $\mathbf{S}$, we can regard an object of $\mathbf{G}_n$ as a morphism $\Omega^n \to \Omega$ in $\mathbf{S}$; in fact, these two collections are isomorphic. Given an object A in $\mathbf{G}_n$ and a morphism $f : \Omega^m \to \Omega^n$, we can thus compose them to get an object in $\mathbf{G}_m$. Defining $\mathbf{G}_f$ by this, we see that $\mathbf{G}$ is a contravariant functor that sends every object of $\mathbf{S}$ to the classifying category of a simply-typed lambda calculus and every morphism of $\mathbf{S}$ to a Cartesian structure-preserving functor between these categories.

Finally, we need to be able to bind free type variables when constructing existential types. Given an object A in G_n with $n > 0$, there should be a corresponding object $\Pi\tau_n.\, A$ in G_{n-1}, where τ_n is the type variable we are binding. Similarly, given a morphism f in G_n with $n > 0$, there should be a corresponding morphism $\Lambda\tau_n.\, f$ in G_{n-1}. This boils down to a functor Π from G_n to G_{n-1} for every $n \in \mathbb{N}$.

Let κ_n be the functor $G_n \to G_{n+1}$ which sends objects and morphisms to themselves, but with an extra free type variable added. The following condition holds naturally in A an object of G_{n+1} and B an object of G_n:

$$\mathrm{Hom}_{\mathbf{G_{n+1}}}\left(\kappa_n(B), A\right) \cong \mathrm{Hom}_{\mathbf{G_n}}\left(B, \Pi(A)\right).$$

Suppose, namely, that $f : \kappa_n(B) \to A$. That is, f is a term of type A with $n+1$ free type variables and a free term variable of type B. We can then bind one type variable to get a term $\Lambda\tau_{n+1}.\, f$ of type $\Pi\tau_{n+1}.\, A$ with n free type variables and a free term variable of type B. This gives a morphism $B \to \Pi(A)$ in G_n.

On the other hand, suppose $g : B \to \Pi(A)$ in G_{n+1}. Then g is a term of type $\Pi\alpha.\, A$ with n free type variables and a free term variable of type B. Let τ_{n+1} be a free type variable that doesn't occur in g. Then $g\{\tau_{n+1}\}$ is a term of type A with $n+1$ free type variables; that is, a morphism $\kappa_n(B) \to A$ in G_{n+1}. It is easy to see that this is the inverse of the operation above. The verification of the naturality condition is long but straightforward.

This shows that Π is the right-adjoint of κ_n.

6.2 Models of PLC

Just like a model of a simply-typed lambda calculus is a Cartesian structure-preserving functor from its classifying category to a Cartesian closed category, a model of a second-order lambda calculus is a mapping from its classifying category to a category with the appropriate structure. We will complete this chapter by describing what this structure is. This will involve a functor T to interpret $\mathbf{S}$ and a family of functors F_n for $n \in \mathbb{N}$ to interpret G_n. Recall that G_n is the classifying category of a simply-typed lambda calculus with n fresh types. Each F_n is thus a model of a simply-typed lambda calculus. The role of T is to ensure that these models are coherent.

First of all, we require a category that we can map $\mathbf{S}$ into. Let us call this $\mathbf{S}'$. This category must have all finite products, and the model is given by a functor $T : \mathbf{S} \to \mathbf{S}'$ which must preserve these products. This is entirely analogous to the simply-typed case, except that we do not have exponential objects.

Secondly, we require a contravariant functor G' that sends every object of $\mathbf{S}'$ to a Cartesian closed category. We will use G'_n to denote $G'(T(\Omega^n))$ and G'_f to denote $G'(T(f))$ in analogy with G_n and G_f. Together with this, we require a functor F that associates to every object Ω^n in $\mathbf{S}$ a functor from G_n to G'_n and to every morphism $f : \Omega^n \to \Omega^m$ a natural transformation from G_f to G'_f. Note that T and F should commute with G and G'. We get the following diagram:

$$\begin{tikzcd}
\mathbf{S} \arrow[r, "T"] \arrow[d, "G"'] & \mathbf{S}' \arrow[d, "G'"] \\
G(\mathbf{S}) \arrow[r, "F"'] & \mathbf{Cat}
\end{tikzcd}$$

The requirements on F may appear involved, but actually come down to little more than the simply-typed case. For every $n \in \mathbb{N}$, G_n is the classifying category of a simply-typed lambda calculus with the theory expanded with n fresh types (without terms or equalities). $F(\Omega^n)$ is then simply a model of that simply-typed lambda calculus. All other conditions are only necessary to ensure that the interpretation is coherent.

Finally, we require that the functors Π' be defined on G' so that Π and Π' commute with G and G'. This is easily achieved by requiring that Π' be right-adjoint to the weakening functor, which is induced by the projection morphism in $\mathbf{S}$ from $A \times \Omega$ to A.

We will from now on leave the functors implicit, and simply speak of $\mathbf{S}$ when we mean $\mathbf{S}'$ and G when we mean G'. We will also use Ω to refer to the image of Ω under T. All together, this gives the following definition:

Definition 6.1. A ***PL-Category*** is a pair $(\mathbf{S}, G)$ where

- $\mathbf{S}$ is a category with finite products and a distinguished object Ω
- G is a contravariant functor that assigns to every object A in $\mathbf{S}$ a Cartesian closed category $G(A)$ where $\mathrm{Ob}(G(A)) \cong \mathrm{Hom}_{\mathbf{S}}(A, \Omega)$ and to every morphism $f : A \to B$ in $\mathbf{S}$ a functor $G(B) \to G(A)$ which acts by composition on objects.
- For every object A in $\mathbf{S}$, the weakening functor $\kappa_A : G(A) \to G(A \times \Omega)$ has a right adjoint Π natural in A.

Chapter 7

Qualitative Domains

We have seen that we can use **Set** to model a simply-typed lambda calculus. However, as shown by Reynolds, this no longer works in the second order case [Rey84]. Instead, we construct the model in a different category: that of qualitative domains.

The idea behind qualitative domains is that rather than assigning to every type a set of values and to every term a value from that set, we assign to each type a set of 'properties' that its terms might have and then to each term a set of 'properties' that it does have. Functions between qualitative domains specify what properties the result has based on some finite set of properties of the argument.

The definition we provide here is based on the treatment of complete partial orders by Girard, Taylor, and Lafont [GTL89]; in particular, their concept of a web is used. This system is equivalent to the system presented in Girard's earlier paper [Gir86].

7.1 Definition

The above leads to the following definition:

Definition 7.1. A *qualitative domain* $\mathbf{X}$ is a set $|\mathbf{X}|$ of *tokens* together with a binary relation *coherence* that is reflexive and symmetric.

The binary relation is denoted $x \smile y$.[1] A *point* a of X is a set $a \subseteq |\mathbf{X}|$ where all elements are pairwise coherent; that is, which satisfies $\forall x, y \in a, x \smile y$. A point a is called *total* if it is maximal with respect to inclusion, so if for every point b with $a \subseteq b$, $a = b$.

A qualitative domain that will come up often is **Bool**, which has tokens $\mathbf{t}$ and $\mathbf{f}$ which are not coherent. The points of **Bool** are $\emptyset$, $\{\mathbf{t}\}$, and $\{\mathbf{f}\}$, of which only $\emptyset$ is not total.

If we restrict our attention to the simply typed lambda calculus, we can interpret every type as a qualitative domain and every term as a point in this domain. A `Bool` type with two values would be interpreted as the qualitative domain **Bool**. A term t of type `Bool` would be interpreted as one of the points. Under this interpretation, we can see each token as a predicate on the value. If the interpretation of t is $\{\mathbf{t}\}$, then we have derived that 't is true' holds. If the interpretation is $\emptyset$, we have not yet derived anything. For example, if t is a non-terminating computation then both 't is true' and 't is false' fail to hold.

An important note is that while we may derive that some predicate represented by the token x holds on some term t (that is, t is interpreted as a and $x \in a$), we may not directly derive that

[1]We may use $x \smile_{\mathbf{X}} y$ if it is unclear in what qualitative domain x and y are coherent.

a predicate x does not hold on some term t. The closest we can come is to find a token y that is not coherent with x and derive $y \in a$. The statement $x \notin a$ states only that x has not yet been derived, not that it does not hold. We will see the significance of this when we introduce stable functions between qualitative domains.

7.1.1 Set of Points

It is convenient to see a qualitative domain X as the set of its points. We will thus use notation such as $x \in X$ to indicate that x is a point of X. The following theorem gives a characterisation of what sets correspond to qualitative domains.

Theorem 7.1. *A qualitative domain X satisfies the following properties:*

1. *if $a \in X$ and $b \subset a$ then $b \in X$,*
2. *if A is a set of points of X and for all $a_1, a_2 \in A$, $a_1 \cup a_2 \in X$ then $\bigcup A \in X$, and*

Furthermore, any set of sets X that satisfies the above can be seen as a qualitative domain with $|X| = \bigcup X$ and for all $x, y \in |X|$, $x \smile y$ iff $\{x, y\} \in X$.

Proof. Let X be a qualitative domain. Recall that $a \in X$ simply means that every two elements of a are coherent in X. Hence, if a is coherent and $b \subset a$, then for any $x, y \in b$ we have $x, y \in a$ and thus $x \smile y$.

Let A be a set of points of X such that for all $a_1, a_2 \in A$, $a_1 \cup a_2 \in X$. Let $x, y \in \bigcup A$ and choose $a_1, a_2 \in A$ such that $x \in a_1$ and $y \in a_2$. Then $a_1 \cup a_2 \in X$, and thus $x \smile y$. Hence every two elements of $\bigcup A$ are coherent and thus $\bigcup A$ is itself a point of X.

Now suppose X is a set of sets satisfying the above. If $x \in \bigcup X$ then there exists some $a \in X$ so that $x \in a$, and then by property (1), $\{x\} \in X$, hence $x \smile x$, which gives us reflexivity. Symmetry is trivial, as $\{x, y\} = \{y, x\}$.

We prove that $a \in X$ iff a is coherent in the sense defined in the theorem. Let $a \in X$ and let $x, y \in a$. By property (1), $\{x, y\} \in X$, thus $x \smile y$, hence a is coherent. Now suppose a is a coherent set. Let $A = \{\{x\} | x \in a\}$. If $a_1, a_2 \in A$ then $a_1 \cup a_2 = \{x, y\}$ for some $x, y \in a$. As a is coherent, $x \smile y$, and thus $\{x, y\} \in X$. By property (2) it follows that $\bigcup A = a \in X$. $\square$

Property (2) is convenient when we work with sets, but it does not immediately generalise to other categories. We will instead work with directed sets, which generalise as direct limits.

Definition 7.2. A *directed set* is a non-empty partially ordered set A such that for all $a_1, a_2 \in A$, there exists a $b \in A$ with $a_1 \leq b$ and $a_2 \leq b$.

We will be particularly interested in the case that A is a set of points ordered by inclusion. It is a simple consequence that X contains the union of any directed set of its points.

Theorem 7.2. *Given a qualitative domain X and directed set of points A ordered by inclusion, $\bigcup A \in X$.*

Proof. If $a_1, a_2 \in A$ then there is a $b \in A$ such that $a_1 \cup a_2 \subset b$, and as $b \in X$, it follows that $a_1 \cup a_2$. Therefore, A satisfies the requirements for property (2) and $\bigcup A \in X$. $\square$

7.1.2 Examples

Some examples of qualitative domains are:

- Given a set A we can construct a qualitative domain $\mathbf{X}$ by taking $|\mathbf{X}| = A$ and $a \smile b$ iff $a = b$. We call this the discrete qualitative domain. Given a point of $\mathbf{X}$, it either uniquely determines an element of A, or leaves the element fully unspecified.
- Let $|\mathbf{N}| = \mathbb{N} \cup \{n^+ | n \in \mathbb{N}\}$, and let coherence be the least reflexive and transitive relation with $n^+ \smile m^+$ for all $n, m \in \mathbb{N}$ and $n^+ \smile m$ iff $n \leq m$. This domain is suitable for interpreting the natural numbers in a computational context, as we shall see in the next section.
- Given qualitative domains $\mathbf{X}$ and $\mathbf{Y}$, we can construct the domain $\mathbf{X} \times \mathbf{Y}$ by taking $|\mathbf{X} \times \mathbf{Y}| = |\mathbf{X}| \sqcup |\mathbf{Y}|$ and $x \smile y$ iff x and y are both in X or both in Y and coherent within that domain, or if they are in different domains. Once we formalise qualitative domains as a category, we will see that this is indeed a product.
- Taking $|\mathbf{1}| = \emptyset$ trivially gives a qualitative domain. This domain has no tokens, but still has $\emptyset$ as a point. This will eventually be the terminal object of our category.

7.2 Stable Functions

Informally, a stable function can be seen as a mapping that is fully determined by rules of the form "If the input contains tokens $x_0, x_1, \ldots, x_n$, then include the token y in the output." Additionally, it must respect the structure of the qualitative domains; given a point in $\mathbf{X}$, it should return a point in $\mathbf{Y}$.

We formalise this idea by introducing a suitable qualitative domain $\mathbf{X} \Rightarrow \mathbf{Y}$, and then defining a mapping Θ from points in this domain to functions $\mathbf{X} \to \mathbf{Y}$. We will say that a function $f : \mathbf{X} \to \mathbf{Y}$ is stable if there exists a point F in $\mathbf{X} \Rightarrow \mathbf{Y}$ such that $\Theta(F) = f$.

Define $|\mathbf{X} \Rightarrow \mathbf{Y}| = \{(a, y) | a \in \mathbf{X}, y \in |\mathbf{Y}|, |a| \in \mathbb{N}\}$. We define coherence as the least reflexive and symmetric relation such that

1. If a, a' are points of $\mathbf{X}$ and $y, y' \in |\mathbf{Y}|$ with $y \neq y'$, then $(a, y) \smile (a', y')$ if $y \smile y'$ or $a \cup a' \notin \mathbf{X}$.
2. If a, a' are points of $\mathbf{X}$ and $y \in |\mathbf{Y}|$, then $(a, y) \smile (a', y)$ if $a = a'$ or if $a \cup a'$ is not a point. We will refer to this property as minimality.

Note that the minimality condition implies that (a, y) and (a', y) are not coherent if a is a strict subset of a'.

Now let $F \in \mathbf{X} \Rightarrow \mathbf{Y}$. We define

$$\Theta(F)(a) = \{y | \exists (a', y) \in F, a' \subset a\}$$

For this definition to be valid, we need to prove that if $a \in \mathbf{X}$ then $\Theta(F)(a) \in \mathbf{Y}$. This is a consequence of condition (1) above: if $y_1, y_2 \in \Theta(F)(a)$ then there exist a_1, a_2 such that $(a_1, y_1), (a_2, y_2) \in F$ with $a_1 \subset a$ and $a_2 \subset a$. Thus $a_1 \cup a_2 \subset a$ and hence $a_1 \cup a_2$ is a point, from which follows that $y_1 \smile y_2$.

7.2.1 Traces

By definition, every stable function $f : \mathbf{X} \to \mathbf{Y}$ is represented by a point $F \in \mathbf{X} \Rightarrow \mathbf{Y}$ called its *trace*. We will now prove that this representation is unique and set $Tr(f) = F$, and then give a characterisation of the elements of $Tr(f)$. This shows that the qualitative domain $\mathbf{X} \Rightarrow \mathbf{Y}$ fully represents the stable functions $\mathbf{X} \to \mathbf{Y}$. We will later show that this is indeed the case, and that it is an exponential object in our category.

Theorem 7.3. *The function Θ defined above is injective.*

Proof. Let $F, G \in \mathbf{X} \Rightarrow \mathbf{Y}$ such that for all $a \in \mathbf{X}$, $\Theta(F)(a) = \Theta(G)(a)$. Suppose $(a, y) \in F$. We prove $(a, y) \in G$, which gives us $F \subset G$ and $G \subset F$ follows by symmetry. As $(a, y) \in F$, $y \in \Theta(F)(a)$, and thus $y \in \Theta(G)(a)$. Thus there exists an $a' \subset a$ such that $(a', y) \in G$. Then $y \in \Theta(G)(a')$ and thus $y \in \Theta(F)(a')$. But now there exists an $a'' \subset a'$ such that $(a'', y) \in F$, so $(a'', y) \smile (a, y)$. As $a'' \subset a$, it follows by minimality that $a'' = a$, hence $a' = a$ and $(a, y) \in G$. $\square$

As a corollary, since we defined Tr as a section of Θ and the section of an injective function is its inverse, Θ and Tr give an isomorphism between stable functions and their traces.

Theorem 7.4. $(a, y) \in Tr(f)$ *iff* $y \in f(a)$ *and for all* $a' \subsetneq a$, $y \notin f(a)$.

Proof. Suppose $(a, y) \in Tr(f)$. Then certainly $y \in \Theta(Tr(f))(a) = f(a)$, as Tr is a section of Θ. Furthermore, if $a' \subsetneq a$ and $y \in \Theta(Tr(f))(a')$ then there exists an $a'' \subset a'$ such that $(a'', y) \in Tr(f)$, which is not coherent with (a, y) by minimality. On the other hand, suppose $y \in f(a)$ and for all $a' \subsetneq a$, $y \notin f(a)$. Then there exists an $a' \subset a$ such that $(a', y) \in Tr(f)$. However, for all $a' \subsetneq a$, $(a', y) \notin Tr(f)$. Hence $a' = a$, and thus $(a, y) \in Tr(f)$. $\square$

7.2.2 Functions on Points

Similarly to how we often wish to regard a qualitative domain as the set of its points, it is often convenient to regard a stable function as a function between points. The following theorem gives us such an equivalence.

Theorem 7.5. *Given qualitative domains $\mathbf{X}$, $\mathbf{Y}$ and a function $f : \mathbf{X} \to \mathbf{Y}$, f is stable iff the following conditions hold:*

1. *For all $a, a' \in \mathbf{X}$, if $a \subset a'$ then $f(a) \subset f(a')$.*
2. *For every directed collection $A \subset \mathbf{X}$, $f(\bigcup A) = \bigcup_{a \in A} f(a)$.*
3. *For all $a, a' \in \mathbf{X}$, if $a \cup a' \in X$, then $f(a \cap a') = f(a) \cap f(a')$.*

Proof. Suppose f is stable and let F be the associated element of $\mathbf{X} \Rightarrow \mathbf{Y}$.

Let $a, a' \in \mathbf{X}$ with $a \subset a'$, and let $y \in f(a)$. By definition, $y \in \Theta(F)(a)$ and so there exists a $a'' \subset a$ such that $(a'', b) \in F$. But then $y \in \Theta(F)(a')$, as $a'' \subset a'$.

Let $A \subset \mathbf{X}$ be a directed collection. We prove that $f(\bigcup A) = \bigcup_{a \in A} f(a)$ by proving that each includes the other. Let $y \in f(\bigcup A)$. Then there exists a finite $a' \subset \bigcup A$ such that $(a', y) \in F$. Note that for each $x \in a'$ there exists a $b \in A$ such that $x \in b$. Hence, by directedness of A, there is a $b \in A$ such that $a' \subset b$. Thus $y \in f(b)$, and so $y \in \bigcup_{a \in A} f(a)$. In the other direction, let $y \in \bigcup_{a \in A} f(a)$. Choose $a \in A$ such that $y \in f(a)$. As $a \subset \bigcup A$, $y \in f(\bigcup A)$ follows immediately.

Let $a, a' \in \mathbf{X}$ such that $a \cup a' \in X$ and let $y \in f(a \cap a')$. Choose $a'' \subset a \cap a'$ such that $(a'', y) \in F$. Note that $a'' \subset a$ and $a'' \subset a'$, so $y \in f(a)$ and $y \in f(a')$, hence $y \in f(a) \cap f(a')$. Now let $y \in f(a) \cap f(a')$. There exist $b \subset a$ and $b' \subset a'$ such that $(b, y) \in F$ and $(b', y) \in F$. As $a \cup a'$ is coherent, so is $b \cup b'$. But then by minimality, $b = b'$, hence $b \subset a \cap a'$ and thus $y \in f(a \cap a')$. $\square$

Theorem 7.6. *The identity function is stable.*

Proof. Using theorem 7.5, it suffices to check that the identity function preserves subsets, directed unions, and intersections, which it trivially does. □

Theorem 7.7. *Stable functions are closed under composition.*

Proof. Again using theorem 7.5, we need only check that if $f : \mathbf{X} \to \mathbf{Y}$ and $g : \mathbf{Y} \to \mathbf{Z}$ preserves subsets, directed unions, and intersections, that $g \circ f$ preserves them as well. In the case of subsets and intersections this is obviously so. For directed unions, we need to ensure that the image of a directed set under a stable function is itself a directed set. This follows directly from stable functions preserving subsets: let $f : \mathbf{X} \to \mathbf{Y}$ be a stable function, $A \subset X$ a directed set, and $a, b \in A$. As A is directed, there is a $c \in A$ such that $a \cup b \subset c$. Now, as f preserves subsets, $f(a) \subset f(c)$ and $f(b) \subset f(c)$, so $f(a) \cup f(b) \subset f(c)$. Hence $f(A)$ is itself a directed set, and so given $g : \mathbf{Y} \to \mathbf{Z}$, $g(f(\bigcup A)) = g(\bigcup_{a \in A} f(a)) = \bigcup_{a \in A} g(f(a))$. □

That the identity function is the identity element of composition follows from considering stable functions as set-theoretic functions on the set of points.

It follows that qualitative domains and stable functions between them form a category. We will call this category **Stab**.

7.3 The Category Stab

In subsection 7.1.2 we have already shown how to define $\mathbf{X} \times \mathbf{Y}$ and $\mathbf{1}$, and claimed that these are the product and terminal object respectively. We can now prove this formally.

Theorem 7.8. *Given* $\mathbf{X}, \mathbf{Y}$ *objects of* **Stab**, $\mathbf{X} \times \mathbf{Y}$ *as defined in subsection 7.1.2 is a product.*

Proof. Let $\mathbf{X}, \mathbf{Y}$, and $\mathbf{Z}$ be qualitative domains and $q_X : \mathbf{Z} \to \mathbf{X}$ and $q_Y : \mathbf{Z} \to \mathbf{Y}$ be projection functions. For simplicity, we assume $|\mathbf{X}|$ and $|\mathbf{Y}|$ are disjoint, which allows us to use the usual union rather than the disjoint union of $|\mathbf{X}|$ and $|\mathbf{Y}|$ for $|\mathbf{X} \times \mathbf{Y}|$. Let $\mathbf{X} \times \mathbf{Y}$ be defined as above, with $p_X : \mathbf{X} \times \mathbf{Y} \to X$ be $\Theta(\{(\{x\}, x) | x \in |\mathbf{X}|\})$ and let p_Y be defined analogously. We prove there is a unique stable function h such that $p_X \circ h = q_X$ and $p_Y \circ h = q_Y$.

We start by proving existence. Let $h(a) = q_X(a) \cup q_Y(a)$. Then $p_X(h(a)) = \{x | \exists a' \subset q_X(a) \cup q_Y(a). (a', x) \in Tr(p_X)\}$. Given that we know $Tr(p_X)$ this can be simplified to

$$p_X(h(a)) = \{x | x \in q_X(a) \cup q_Y(a), (\{x\}, x) \in Tr(p_X)\}.$$

Since $(\{x\}, x) \in Tr(p_X)$ iff $x \in |\mathbf{X}|$, this can be further simplified to

$$p_X(h(a)) = \{x | x \in q_X(a) \cup q_Y(a), x \in |\mathbf{X}|\}$$

and since $q_Y(a) \cap |\mathbf{X}| = \emptyset$ and $q_X(a) \subset |\mathbf{X}|$, this is simply $p_X(h(a)) = q_X(a)$, as desired.

By symmetry, $p_Y \circ h = q_Y$.

We now prove uniqueness. Suppose h and h' both satisfy this requirement with $h \neq h'$. Let $a \in \mathbf{Z}$ such that $h(a) \neq h'(a)$. Choose $x \in h(a)$ such that $x \in \mathbf{X}$ and $x \notin h'(a)$ (if no such x exists, switch the roles of h and h' or of $\mathbf{X}$ and $\mathbf{Y}$; by symmetry, these choices are immaterial). Now by definition of p_A, $x \in p_X(h(a))$ but $x \notin p_X(h'(a))$. However, $p_X \circ h = q_X = p_X \circ h'$, a contradiction. □

Theorem 7.9. *The* $\mathbf{1}$ *qualitative domain defined in subsection 7.1.2 is a terminal object.*

Proof. Let $\mathbf{X}$ be a qualitative domain. As $\emptyset$ is the only point of $\mathbf{1}$, there is only one set-theoretic function $f : \mathbf{X} \to \mathbf{1}$, namely the function that sends everything to the empty set. This is trivially stable, being given by the empty trace. $\square$

Furthermore, in section 7.2 we have introduced $\mathbf{X} \Rightarrow \mathbf{Y}$, and claimed it will be the internal hom. We will now prove this.

Theorem 7.10. *Given* $\mathbf{Y}, \mathbf{Z}$ *objects of* $\mathbf{Stab}$*,* $\mathbf{Y} \Rightarrow \mathbf{Z}$ *is an exponential object.*

Proof. Let $\mathbf{X}, \mathbf{Y}$, and $\mathbf{Z}$ be qualitative domains. For simplicity, again assume their token sets are disjoint. We prove

$$\mathrm{Hom}_{\mathbf{Stab}}(\mathbf{X} \times \mathbf{Y}, \mathbf{Z}) \cong \mathrm{Hom}_{\mathbf{Stab}}(\mathbf{X}, \mathbf{Y} \Rightarrow \mathbf{Z})$$

naturally in $\mathbf{X}, \mathbf{Y}$, and $\mathbf{Z}$.

We define functions

$$\begin{aligned} Ap &: \mathrm{Hom}_{\mathbf{Stab}}(\mathbf{X} \times \mathbf{Y}, \mathbf{Z}) \to \mathrm{Hom}_{\mathbf{Stab}}(\mathbf{X}, \mathbf{Y} \Rightarrow \mathbf{Z}) \\ Ab &: \mathrm{Hom}_{\mathbf{Stab}}(\mathbf{X}, \mathbf{Y} \Rightarrow \mathbf{Z}) \to \mathrm{Hom}_{\mathbf{Stab}}(\mathbf{X} \times \mathbf{Y}, \mathbf{Z}) \end{aligned}$$

as follows:

$$\begin{aligned} Ap(f)(a) &= Tr(\lambda b. f(a \cup b)) \\ Ab(f)(a) &= \{z \in |\mathbf{Z}| | \exists a' \subset p_Y(a).\, (a', z) \in f(p_X(a))\} \end{aligned}$$

We prove that $Ap \circ Ab = 1_{\mathrm{Hom}_{\mathbf{Stab}}(\mathbf{X}, \mathbf{Y} \Rightarrow \mathbf{Z})}$ and $Ab \circ Ap = 1_{\mathrm{Hom}_{\mathbf{Stab}}(\mathbf{X} \times \mathbf{Y}, \mathbf{Z})}$. We start with the first.

Let $f : \mathbf{X} \to (\mathbf{Y} \Rightarrow \mathbf{Z})$ and $a \in \mathbf{X}$. Then by filling in the definitions we get

$$\begin{aligned} (Ap \circ Ab)(f)(a) &= Tr(\lambda b. Ab(f)(a \cup b)) \\ &= Tr(\lambda b. \{z \in |\mathbf{Z}| | \exists b' \subset p_Y(a \cup b).(b', z) \in f(p_X(a \cup b))\}). \end{aligned}$$

Note that as $a \in \mathbf{X}$ and $b \in \mathbf{Y}$ we have $p_X(a \cup b) = a$ and $p_Y(a \cup b) = b$. Hence the above can be simplified to

$$(Ap \circ Ab)(f)(a) = Tr(\lambda b. \{z \in |\mathbf{Z}| | \exists b' \subset b.(b', z) \in f(a)\}).$$

But this is exactly $Tr(\lambda b.\Theta(f(a))(b))$, or equivalently, $Tr(\Theta(f(a)))$. As Θ is the inverse of Tr, we have $(Ap \circ Ab)(f)(a) = f(a)$ for all $a \in \mathbf{X}$, hence $(Ap \circ Ab)(f) = f$ and $Ap \circ Ab = 1_{\mathrm{Hom}_{\mathbf{Stab}}(\mathbf{X}, \mathbf{Y} \Rightarrow \mathbf{Z})}$.

We now show the converse equality. Let $f : \mathbf{X} \times \mathbf{Y} \to \mathbf{Z}$ and let $d \in \mathbf{X} \times \mathbf{Y}$. We can decompose d into $a \cup b$ with $a \in \mathbf{X}$ and $b \in \mathbf{Y}$ by setting $a = p_X(d)$ and $b = p_Y(d)$. Now

$$\begin{aligned} (Ab \circ Ap)(f)(d) &= \{z \in |\mathbf{Z}| | \exists b' \subset p_Y(d).(b', z) \in Tr(\lambda c. f(p_X(d) \cup c))\} \\ (Ab \circ Ap)(f)(a \cup b) &= \{z \in |\mathbf{Z}| | \exists b' \subset b.(b', z) \in Tr(\lambda c. f(a \cup c))\}. \end{aligned}$$

Note now that $(b', z) \in Tr(\lambda c. f(a \cup c))$ iff $z \in f(a \cup b')$. Hence

$$\begin{aligned} (Ab \circ Ap)(f)(a \cup b) &= \{z \in |\mathbf{Z}| | \exists b' \subset b. z \in f(a \cup b') \\ &= \bigcup_{b' \subset b} f(a \cup b') \\ &= f(a \cup b) \end{aligned}$$

$$\begin{tikzcd}
\mathrm{Hom}_{\mathbf{Stab}}\left(\mathbf{X}\times\mathbf{Y},\mathbf{Z}\right) \arrow[r, "Ap"] \arrow[d, "\mathrm{Hom}_{\mathbf{Stab}}\left(f\times g,h\right)"'] & \mathrm{Hom}_{\mathbf{Stab}}\left(\mathbf{X},\mathbf{Y}\Rightarrow\mathbf{Z}\right) \arrow[d, "\mathrm{Hom}_{\mathbf{Stab}}\left(f,g\Rightarrow h\right)"] \\
\mathrm{Hom}_{\mathbf{Stab}}\left(\mathbf{X}'\times\mathbf{Y}',\mathbf{Z}'\right) \arrow[r, "Ap"'] & \mathrm{Hom}_{\mathbf{Stab}}\left(\mathbf{X}',\mathbf{Y}'\Rightarrow\mathbf{Z}'\right)
\end{tikzcd}$$

where the last equality follows from the fact that $f(a\cup b')\subset f(a\cup b)$ for all $a\cup b'\subset a\cup b$. We thus have $Ab\circ Ap = 1_{\mathrm{Hom}_{\mathbf{Stab}}(\mathbf{X}\times\mathbf{Y},\mathbf{Z})}$.

To complete the proof of adjointness, it remains to prove the naturality of Ap. That is, that given morphisms $f:\mathbf{X}'\to\mathbf{X}, g:\mathbf{Y}'\to\mathbf{Y}$, and $h:\mathbf{Z}\to\mathbf{Z}'$, the following diagram commutes:

In other words, that for all $i:\mathbf{X}\times\mathbf{Y}\to\mathbf{Z}$, we have $Ap(h\circ i\circ f\times g) = g\Rightarrow h\circ Ap(i)\circ f$, where $(f\times g)(a\cup b) = f(a)\cup g(b)$ if $a\in\mathbf{X}$ and $b\in\mathbf{Y}$, and where $(g\Rightarrow h)(F) = Tr(h\circ\Theta(F)\circ g)$.

Expanding the definitions, we get

$$\begin{aligned}
\Theta(Ap(h\circ i\circ(f\times g))(a))(b) &= \Theta(Tr(\lambda b'.(h\circ i\circ(f\times g))(a\cup b')))(b)\\
&= (h\circ i\circ(f\times g))(a\cup b)\\
&= h(i(f(a)\circ g(b)))
\end{aligned}$$

and

$$\begin{aligned}
\Theta(((g\Rightarrow h)\circ Ap(i)\circ f)(a))(b) &= \Theta((g\Rightarrow h)(Ap(i)(f(a))))(b)\\
&= \Theta((g\Rightarrow h)(Tr(\lambda b'.i(f(a)\cup b'))))(b)\\
&= \Theta(Tr(h\circ\Theta(Tr(\lambda b'.i(f(a)\cup b')))\circ g))(b)\\
&= (h\circ(\lambda b'.i(f(a)\cup b'))\circ g)(b)\\
&= h(i(f(a)\cup g(b))).
\end{aligned}$$

This shows that the isomorphism is indeed natural (as Ab is simply Ap^{-1} and inverses preserve naturality), and thus that $Y\Rightarrow(-)$ is an exponential object. □

Theorem 7.11. **Stab** *is a Cartesian closed category.*

Proof. Immediate consequence of theorems 7.8, 7.9, and 7.10. □

Chapter 8

Qualitative Domains as a PL Category

Before we formally define $\mathbf{S}$ and $G : \mathbf{S}^{\mathrm{op}} \to \mathbf{Cat}$, let us give a sketch of what follows.

The only orders present are products of Ω and 1. Hence, up to isomorphism, every order is a product of n copies of Ω, which we will denote Ω^n. An operator expression with a free operator variable of order Ω^n can thus be seen as an operator expression with n free operator variables of order Ω. Hence, an object of $G(\Omega^n)$ should be a type expression that depends on n type variables.

In chapter 7 we have seen that **Stab**, the category of qualitative domains and functions between them, is Cartesian closed (theorem 7.11). We would will interpret every type expression as a qualitative domain. In other words, we would like to see a morphism $\Omega^n \to \Omega$ as a qualitative domain. But this is a problem we have already solved for stable functions! We thus require $\Omega^n \to \Omega$ to have the structure of a stable function and then construct traces on this level. $G(\Omega^n)$ then becomes a subcategory of **Stab**.

Once this construction is complete, it remains to check that $G(\Omega^n)$ is Cartesian closed, that for any $f : \Omega^n \to \Omega^m$, $G(f) : G(\Omega^m) \to G(\Omega^n)$ preserves the Cartesian closed structure, and that there exists an adjoint Π to the weakening functor $G(\pi) : G(\Omega^n) \to G(\Omega^{n+1})$ that satisfies the naturality conditions.

8.1 The Category qD

Given qualitative domains $\mathbf{X}, \mathbf{Y}$ we can define an inclusion $i : \mathbf{X} \to \mathbf{Y}$ as an injective function $|i| : |\mathbf{X}| \to |\mathbf{Y}|$ such that for all $x, x' \in |\mathbf{X}|$, $x \smile_{\mathbf{X}} x'$ iff $i(x) \smile_{\mathbf{Y}} i(x')$. It is immediately clear that the identity is an inclusion and that inclusions are closed under composition. Let **qD** be the category with qualitative domains as objects and inclusions as morphisms.

Intuitively, we want to regard **qD** modulo isomorphism as a qualitative domain, with object equivalence classes being points and inclusion of qualitative domains as being the inclusion of points. However, the equivalence classes are not sets themselves and do not form a set, so we cannot treat them as a qualitative domain directly. Instead, we will reconstruct the machinery of stable functions and traces in this new setting.

Our stable functions will be functors $\mathbf{qD} \to \mathbf{qD}$. Recall the requirements for a function f to be stable in the usual sense (theorem 7.5):

1. For all $a, a' \in \mathbf{X}$, if $a \subset a'$ then $f(a) \subset f(a')$.

2. For every directed collection $(a_i)_{i\in I} \subset \mathbf{X}$, $f(\bigcup^{\uparrow}_{i\in I} a_i) = \bigcup^{\uparrow}_{i\in I} f(a_i)$.
3. For all $a, a' \in \mathbf{X}$, if $a \cup a' \in X$, then $f(a \cap a') = f(a) \cap f(a')$.

By using inclusion rather than the subset relation we can reformulate these requirements for a functor $F : \mathbf{qD} \to \mathbf{qD}$ as follows:

1. For every two qualitative domains $\mathbf{X}, \mathbf{Y}$, if there exists an $f : \mathbf{X} \to \mathbf{Y}$, then $F(f) : F(\mathbf{X}) \to F(\mathbf{Y})$.
2. For every directed collection $(\mathbf{X}_i)_{i\in I}$ of objects of $\mathbf{qD}$, $F(\lim_{i\in I} \mathbf{X}_i) = \lim_{i\in I} F(\mathbf{X}_i)$.
3. For every three qualitative domains $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$, if $f : \mathbf{X} \to \mathbf{Z}$ and $g : \mathbf{Y} \to \mathbf{Z}$ and $\mathbf{X} \times_{\mathbf{Z}} \mathbf{Y}$ is the pullback, then $F(\mathbf{X} \times_{\mathbf{Z}} \mathbf{Y}) = F(\mathbf{X}) \times_{F(\mathbf{Z})} F(\mathbf{Y})$.

The first requirement is implied by F being a functor. The second and third state that F preserves direct limits and pullbacks respectively. We take this as our definition.

Definition 8.1. A stable functor from $\mathbf{A}$ to $\mathbf{B}$ is a functor $\mathbf{A} \to \mathbf{B}$ that preserves pullbacks and direct limits.

It is easy to show that the identity functor is stable, and that the composition of two stable functors is itself stable. This means that we can consider the category with $\mathbf{qD}^n$ for $n \in \mathbf{N}$ as objects and stable functors as morphisms. We define $\mathbf{S}$ to be this category.

8.2 The Category S

The only requirement on $\mathbf{S}$ is that it have all finite products, as we have already chosen the distinguished object $\mathbf{qD}$. We claim that the singleton category $\mathbf{qD}^0$ (also referred to as 1) is a terminal object and that the product of $\mathbf{qD}^n$ and $\mathbf{qD}^m$ is $\mathbf{qD}^{n+m}$. This verification amounts to the projections and the pairing morphism being stable.

Theorem 8.1. *$\mathbf{qD}^0$ is the terminal object of $\mathbf{S}$.*

Proof. Let $\mathbf{S}'$ be the category of powers of $\mathbf{qD}$ and functors between them. Note that $\mathbf{S}$ is a subcategory of $\mathbf{S}'$ and that the properties of the categorical product show that $\mathbf{qD}^0 = 1$ is the terminal object of $\mathbf{S}'$. There is thus at most one morphism $\mathbf{qD}^n \to \mathbf{qD}^0$ in $\mathbf{S}$, and it suffices to show that this morphism is stable.

Let $n \in \mathbb{N}$. We prove that the morphism $F : \mathbf{qD}^n \to \mathbf{qD}^0$ is a stable functor. This requires that

$$F(A \times_C B) = F(A) \times_{F(C)} F(B)$$
$$F(\lim_{i\in I} X_i) = \lim_{i\in I} F(X_i).$$

As $\mathbf{qD}^0$ is a singleton category, these equalities on object equality are trivially satisfied. □

Theorem 8.2. *Given $n, m \in \mathbb{N}$, $\mathbf{qD}^{n+m}$ is the categorical product of $\mathbf{qD}^n$ and $\mathbf{qD}^m$.*

Proof. We again use $\mathbf{S}'$ for the proof. Note that $\mathbf{qD}^{n+m}$ is the categorical product of $\mathbf{qD}^n$ and $\mathbf{qD}^m$ in $\mathbf{S}'$. Let p_n and p_m be the projections in $\mathbf{S}'$. As pullbacks and direct unions can be taken componentwise, we see that p_n and p_m are stable. We thus claim that $\mathbf{qD}^{n+m}$ together with p_n and p_m are also the product of $\mathbf{qD}^n$ and $\mathbf{qD}^m$ in $\mathbf{S}$.

Let $k \in \mathbb{N}$ and let $q_n : \mathbf{qD}^k \to \mathbf{qD}^n$ and $q_m : \mathbf{qD}^k \to \mathbf{qD}^m$ be morphisms. We show that there is a unique morphism h in $\mathbf{S}$ such that $p_n \circ h = q_n$ and $p_m \circ h = q_n$. Uniqueness follows from the uniqueness of such a morphism in $\mathbf{S}'$. For existence, we prove that the pairing morphism in $\mathbf{S}'$ is a stable functor. Again, this is a direct consequence of pullback and direct union distributing over products. □

8.3 The Tr Functor

We can now begin to define how $\mathbf{S}$ relates to $\mathbf{Stab}$. Note that for all $n \in \mathbb{N}$, the stable functors $\mathbf{qD}^n \to \mathbf{qD}$ form a category, with natural transformations as morphisms. By the above, the objects of this category represent types that depend on n free type variables, and we will thus call this category $\mathbf{Type_n}$. Soon we will see that given F, G objects of $\mathbf{Type_n}$, a morphism $\alpha : F \Rightarrow G$ represents a term expression of type G with a free term variable of type F.

However, though we will have $G(\mathbf{qD}^n) \cong \mathbf{Type_n}$, this interpretation is still too abstract to be a good model. We will thus define a functor $\mathbf{Tr} : \mathbf{Type_n} \to \mathbf{Stab}$ and set $G(\mathbf{qD}^n) = \mathbf{Tr}(\mathbf{Type_n})$. This approach will mirror how we used traces of stable functions to define exponential types in $\mathbf{Stab}$.

Definition 8.2. Let $T : \mathbf{qD}^n \to \mathbf{qD}$ in $\mathbf{S}$ be a stable functor. The *graph* of T is the collection of all pairs $(\mathbf{X_1}, \dots, \mathbf{X_n}, x)$ where $x \in |\mathbf{T}(\mathbf{X_1}, \dots, \mathbf{X_n})|$. We call an element of the graph of T a *point on the graph of* T, or simply a *point of* T where no confusion can arise. We call a point *finite* if the domains $\mathbf{X_1}, \dots, \mathbf{X_n}$ are finite.

We define a preorder relation on points of T. Let $(\overrightarrow{\mathbf{X}}, x)$ and $(\overrightarrow{\mathbf{Y}}, y)$ be points. We say that $(\overrightarrow{\mathbf{X}}, x) \leq (\overrightarrow{\mathbf{Y}}, y)$ if there exist morphisms $\vec{f} : \overrightarrow{\mathbf{X}} \to \overrightarrow{\mathbf{Y}}$ such that $y = T(\vec{f})(x)$.

A *minimal point* X of T is a point on its graph that is minimal with regard to the preorder relation. That is, if X is a minimal point and $Y \leq X$, then $X \leq Y$.[1]

Given a point X of T we say that X^0 is a *normal form* of X if X^0 is a minimal point and $X^0 \leq X$.

Intuitively, the minimal points play the same role in the traces of stable functors as (a', y) pairs with minimal a' play in the traces of stable functions. The following few theorems will be of use later.

Theorem 8.3. *Every minimal point of T is a finite point.*

Proof. Let $(\overrightarrow{\mathbf{X}}, x)$ be a minimal point of T. Let $\overrightarrow{\mathbf{X_i^0}}$ be the directed set of tuples of finite subdomains of $\overrightarrow{\mathbf{X}}$ with the arrows being inclusions. We can then take the direct limit of $T(\overrightarrow{\mathbf{X_i^0}})$ and as T is stable, we have

$$\lim_{i \in I} T(\overrightarrow{\mathbf{X_i^0}}) = T(\lim_{i \in I} \overrightarrow{\mathbf{X_i^0}}) = T(X).$$

Hence $x \in T(\overrightarrow{\mathbf{X_i^0}})$ for some $i \in I$ and we have the inclusions $\vec{f} : \overrightarrow{\mathbf{X_i^0}} \to \overrightarrow{\mathbf{X}}$, thus $(\overrightarrow{\mathbf{X_i^0}}, x) \leq (\overrightarrow{\mathbf{X}}, x)$. By minimality of $(\overrightarrow{\mathbf{X}}, x)$, however, we have functions $\vec{g} : \overrightarrow{\mathbf{X}} \to \overrightarrow{\mathbf{X_i^0}}$. As both $\vec{f}$ and $\vec{g}$ are injective, by the Cantor-Schroder-Bernstein theorem there are bijections between $\overrightarrow{\mathbf{X}}$ and $\overrightarrow{\mathbf{X_i^0}}$. The latter are all finite, and thus so are the former. □

[1]Note that this does not imply $X = Y$, as we are dealing with a preorder, not a partial order.

Theorem 8.4. *Let* $(\overrightarrow{\mathbf{X}}, x), (\overrightarrow{\mathbf{Y}}, y) \leq (\overrightarrow{\mathbf{Z}}, z)$ *with* $(\overrightarrow{\mathbf{X}}, x)$ *a minimal element and let* $\vec{f} : \overrightarrow{\mathbf{X}} \to \overrightarrow{\mathbf{Z}}$ *and* $\vec{g} : \overrightarrow{\mathbf{Y}} \to \overrightarrow{\mathbf{Z}}$ *be the morphisms that prove the relations. Then there exists a unique factorisation of* $\vec{f}$ *through* $\vec{g}$*, and hence* $(\mathbf{X}, x) \leq (\overrightarrow{\mathbf{Y}}, y)$*.*

Proof. We first prove existence. Note that we can construct the pullback $\overrightarrow{\mathbf{X}'}$ of $\overrightarrow{\mathbf{X}}$ and $\overrightarrow{\mathbf{Y}}$ along $\vec{f}$ and and $\vec{g}$ respectively. We can set $\overrightarrow{\mathbf{X}'} \subset \overrightarrow{\mathbf{X}}$ so that the induced arrow $\vec{f'} : \overrightarrow{\mathbf{X}'} \to \overrightarrow{\mathbf{X}}$ is the inclusion. Note that by the same argument as in theorem 8.3 we have that there exist bijections between $\overrightarrow{\mathbf{X}'}$ and $\overrightarrow{\mathbf{X}}$, and hence each of $\vec{f'}$ is simply the identity; this follows from the finiteness of $\overrightarrow{\mathbf{X}}$. We thus have morphisms $\vec{h} : \mathbf{X} \to \mathbf{Y}$. By commutation of the pullback square, we have $g_i \circ h_i = f_i$.

The uniqueness of the factorisation follows from the injectivity of morphisms in **qD**. If we have factorisations $\vec{h}$ and $\vec{h'}$, then from the fact $g_i \circ h_i = f_i = g_i \circ h'_i$ follows $h_i = h'_i$ because g_i is injective, and thus monic. □

Note that this implies that all normal forms of any point are equivalent.

Theorem 8.5. *Let* $T : \mathbf{qD}^n \to \mathbf{qD}$ *be a stable functor. Every point of* T *has a normal form.*

Proof. Let $(\overrightarrow{\mathbf{X}}, x)$ be a point of T. By the same reasoning as in the proof of theorem 8.3, $(\overrightarrow{\mathbf{X}}, x)$ has a finite subpoint $(\overrightarrow{\mathbf{X}'}, x)$. But there are (up to renaming) only finitely many points less than $(\overrightarrow{\mathbf{X}'}, x)$, the number being bounded by the cardinalities of $\overrightarrow{\mathbf{X}'}$. There is thus a minimal point $(\overrightarrow{\mathbf{X}^0}, x^0)$ less than $(\overrightarrow{\mathbf{X}'}, x)$. Now by transitivity we have $(\overrightarrow{\mathbf{X}^0}, x^0) \leq (\overrightarrow{\mathbf{X}}, x)$, and so $(\overrightarrow{\mathbf{X}^0}, x^0)$ is a normal form of $(\overrightarrow{\mathbf{X}}, x)$. □

This already hints that any stable functor T can be expressed as a combination of its least points.

Definition 8.3. The *trace* of a stable functor T is a collection $\|\mathbf{Tr}(T)\|$ that for every point $(\mathbf{X}, x)$ of T contains a unique normal form $(\mathbf{X}^0, x^0)$ of $(\mathbf{X}, x)$.

We regard $\|\mathbf{Tr}(T)\|$ as a superset of the set of tokens of a qualitative domain $\mathbf{Tr}(T)$ and define coherence as follows: $(\overrightarrow{\mathbf{X}}, x) \smile (\overrightarrow{\mathbf{Y}}, y)$ iff for every n-tuple of finite qualitative domains $\overrightarrow{\mathbf{Z}}$ and morphisms $\vec{f} : \overrightarrow{\mathbf{X}} \to \overrightarrow{\mathbf{Z}}$ and $\vec{g} : \overrightarrow{\mathbf{Y}} \to \overrightarrow{\mathbf{Z}}$ we have $\{T(\vec{f})(x), T(\vec{g})(y)\} \in T(\vec{Z})$.

We now set $|\mathbf{Tr}(T)| = \{X \in \|\mathbf{Tr}(T)\| \mid X \smile X\}$. This gives us a qualitative domain $\mathbf{Tr}(T)$ for every stable functor T. An important note which we unfortunately cannot prove is that $\mathbf{Tr}$ is an injective mapping.

It remains to extend this mapping to morphisms, and we will have a functor that maps each $\mathbf{Type_n}$ into **Stab**. The following rather technical reinterpretation of $\mathbf{Tr}(T)$ gives us a way of defining the mapping on morphisms by composition.

Definition 8.4. With every **qD**-morphism $f : \mathbf{X} \to \mathbf{Y}$ we associate two stable functions $f^+ : \mathbf{X} \to \mathbf{Y}$ and $f^- : \mathbf{Y} \to \mathbf{X}$ given by $f^+(a) = f[a]$ and $f^-(b) = f^{-1}[b]$.

Theorem 8.6. $f^- \circ f^+ = 1_X$.

Proof. Let $a \in \mathbf{X}$. Clearly, $a \subset f^{-1}[f[a]]$ as if $x \in a$ then $f(x) \in f[a]$ and so $x \in f^{-1}[f[a]]$. Recall that f is injective. Now suppose $x \in f^{-1}[f[a]]$. Then $f(x) \in f[a]$. But as f is injective, it follows that $x \in a$. Thus $a = f^{-1}[f[a]]$, as desired. □

Theorem 8.7. *For every* $b \in \mathbf{Y}$*,* $f^+(f^-(b)) \subset b$*.*

Proof. Let $b \in \mathbf{Y}$. Suppose $y \in f[f^{-1}[b]]$. Then there is an $x \in f^{-1}[b]$ such that $f(x) = y$. As f is injective, $y \in b$. $\square$

Definition 8.5. Given T a stable functor $\mathbf{qD}^n \to \mathbf{qD}$, define $T^- : \mathbf{qD}^n \to \mathbf{Stab}^{\mathrm{op}}$ as T on objects and as $T^-(f_1, \ldots, f_n) = T(f_1, \ldots, f_n)^-$ on morphisms.

Define $\mathbb{1} : \mathbf{qD}^n \to \mathbf{Stab}^{\mathrm{op}}$ as the constant functor which sends everything to the terminal object of **Stab**.

With these details out of the way, we can finally state the result that makes this worth our while:

Theorem 8.8. *There is a canonical bijection between elements of* $\mathbf{Tr}(T)$ *and natural transformations* $\mathbb{1} \to T^-$.

Unfortunately, we can only state it, not prove it.

Recall that we wish $\mathbf{Tr}$ to be a functor $\mathbf{Type_n} \to \mathbf{Stab}^{\mathrm{op}}$. Given objects T, S of $\mathbf{Type_n}$ (that is, stable functors $\mathbf{qD}^n \to \mathbf{qD}$) and a natural transformation $\mu : T \to S$ we can define $\mu^- : S^- \to T^-$ by $(\mu_{\overrightarrow{\mathbf{X}}})^- : S(\overrightarrow{\mathbf{X}}) \to T(\overrightarrow{\mathbf{X}})$, which by composition maps natural transformations $\mathbb{1} \to S^-$ to natural transformations $\mathbb{1} \to T^-$. This induces a map $\mathbf{Tr}(S) \to \mathbf{Tr}(T)$, which is a stable function.

Theorem 8.9. $\mathbf{Tr}$ *is a functor* $\mathbf{Type_n} \to \mathbf{Stab}^{op}$.

Proof. Let $T : \mathbf{qD}^n \to \mathbf{qD}$. We have already seen how $\mathbf{Tr}(T)$ can be regarded as a qualitative domain and as a natural transformation $\mathbb{1} \to T^-$. Given a morphism $\mu : T \to S$ in $\mathbf{Type_n}$, the corresponding map $S^- \to T^-$ induces a transformation $\mathbb{1} \to T^-$ to $\mathbb{1} \to S^-$. Hence, by the above we have a corresponding stable function $\mathbf{Tr}(\mu) : \mathbf{Tr}(S) \to \mathbf{Tr}(T)$.

The preservation of composition and identities follows from the way $\mathbf{Tr}$ is defined on functions. If $\mu : R \to T$ and $\nu : T \to S$ are morphisms in $\mathbf{Type_n}$, then they induce two transformation $\mathbf{Tr}(T) \to \mathbf{Tr}(R)$ and $\mathbf{Tr}(S) \to \mathbf{Tr}(T)$ simply by how they act on $1 \to T^-$ and $1 \to S^-$. However, the action on these respects composition, and thus so does the induced action on traces. Identities follow the same way. $\square$

With this theorem, everything is in place to define G.

Definition 8.6. Let G be a category-valued contravariant functor from $\mathbf{S}$ with $G(\mathbf{qD}^n)$ the image of $\mathbf{Type_n}$ under $\mathbf{Tr}$ and with $G(f)$ acting by composition. This is uniquely determined as we can regard an object of G_n as a functor $\mathbf{qD}^n \to \mathbf{qD}$ and a morphism in G_n as a natural transformation between two functors $\mathbf{qD}^n \to \mathbf{qD}$, each of which can be precomposed with a morphism $\mathbf{qD}^k \to \mathbf{qD}^n$.

This has fixed our choices of $\mathbf{S}$ and G. It remains to verify that all requirements are satisfied, namely that G_n is a Cartesian closed category and that for every morphism f, G_f preserves the Cartesian structure. Unfortunately, we are unable to provide a proof of these assertions.

Theorem 8.10. *For every* $n \in \mathbb{N}$, G_n *is Cartesian closed.*

Theorem 8.11. *For every* $f : \mathbf{qD}^n \to \mathbf{qD}^m$, $G(f)$ *preserves the Cartesian closed structure.*

It is worth noting how, given $S : \mathbf{qD}^{n+k} \to \mathbf{qD}$ and $\mathbf{X_1}, \ldots, \mathbf{X_n}$ we can apply a term $t \in Tr(\lambda \mathbf{Y_1}, \ldots, \mathbf{Y_k}. S(\overrightarrow{\mathbf{X}}, \overrightarrow{\mathbf{Y}}))$ to a k-tuple of domains $\mathbf{Z_1}, \ldots, \mathbf{Z_k}$. Let $z \in |\mathbf{S}(\overrightarrow{\mathbf{X}}, \overrightarrow{\mathbf{Z}})|$. Then $z \in t(\overrightarrow{\mathbf{Z}})$ iff there exists a $(\overrightarrow{\mathbf{Y}}, y) \in t$ such that $(\overrightarrow{\mathbf{Y}}, y) \leq (\overrightarrow{\mathbf{Z}}, z)$ according to the ordering defined above. This way, $t(\overrightarrow{\mathbf{Z}}) \in S(\overrightarrow{\mathbf{X}}, \overrightarrow{\mathbf{Z}})$.

8.4 The Π Functor

It remains to show that the weakening functor has the appropriate adjoint. Choose $n \in \mathbb{N}$. The projection morphism $p_n : \mathbf{qD}^{n+1} \to \mathbf{qD}^n$ is mapped by G to the weakening functor $G(p_n) : G_n \to G_{n+1}$. This should be seen as taking a type or term with n free type variables and adding as dependencies one further type variable that is not used in the term itself.

The final condition on G is that this weakening functor has a right-adjoint Π. This functor is given as follows:

Definition 8.7. Given S an object of G_{n+1}, note that we can regard S as a morphism $\mathbf{qD}^{n+1} \to \mathbf{qD}$ in $\mathbf{S}$. Let

$$\Pi(S)(\mathbf{X_1}, \dots \mathbf{X_n}) = \mathbf{Tr}(\lambda \mathbf{Y}.\, S(\mathbf{X_1}, \dots, \mathbf{X_n}, \mathbf{Y})).$$

This is a morphism $\mathbf{qD}^n \to \mathbf{qD}$ in $\mathbf{S}$ and thus an object of G_n.

Given S, T objects of G_{n+1} and a morphism $h : S \to T$, we can regard h as a set of morphisms $h_{\overrightarrow{\mathbf{X}}, \mathbf{Y}} : S(\overrightarrow{\mathbf{X}}, \mathbf{Y}) \to T(\overrightarrow{\mathbf{X}}, \mathbf{Y})$ for all $\overrightarrow{\mathbf{X}}$ and $\mathbf{Y}$. Let $t_{\overrightarrow{\mathbf{X}}} \in \Pi(S)(X_1, \dots X_n)$ and let $\mathbf{Y}$ a qualitative domain. Define:

$$\Pi(h)(t_{\overrightarrow{\mathbf{X}}})(\mathbf{Y}) = h_{\overrightarrow{\mathbf{X}}, \mathbf{Y}}(t_{\overrightarrow{\mathbf{X}}}(\mathbf{Y}))$$

The above definition requires some verification to ensure that the types are correct.

We state the following theorem without proof.

Theorem 8.12. *The functor Π defined above is right-adjoint to the weakening functor $G(p_n)$.*

Theorem 8.13. *The pair $(\mathbf{S}, G)$ forms a PL-category.*

Proof. Immediate consequence of theorems 8.1, 8.2, 8.10, 8.11, and 8.12. □

8.5 Conclusion

We have now constructed **Stab**, **Tr**, G, and S. Due to time constraints, we were unable to everywhere verify that this construction is valid. The greatest omission is the construction of the canonical bijection between elements of $\mathbf{Tr}(T)$ and natural transformations $\mathbb{1} \to T^-$ claimed by theorem 8.8. The theorem is a generalisation of theorem 2.9 from Girard's article [Gir86]. The crucial step is that a natural transformation $\mathbb{1} \to T^-$ can be seen as a function t mapping $\mathbf{X_1}, \dots, \mathbf{X_n}$ to an element $t(\mathbf{X_1}, \dots, \mathbf{X_n})$ of $T(\mathbf{X_1}, \dots, \mathbf{X_n})$ such that given $(f_1, \dots, f_n) : (\mathbf{X_1}, \dots, \mathbf{X_n}) \to (\mathbf{Y_1}, \dots, \mathbf{Y_n})$ such that $T^-(f_1, \dots, f_n)(t(\mathbf{Y_1}, \dots, \mathbf{Y_n})) = t(\mathbf{X_1}, \dots, \mathbf{X_n})$.

Additionally, we have not verified that the rest of our construction satisfies the requirements for G_n to be a Cartesian closed category for each n, and for $G(f)$ to preserve the Cartesian structure. Finally, we have not verified that Π is well-defined and a right adjoint to the weakening functor. This was caused primarily by a lack of time.

Chapter 9

Bibliography

[Bar13] Henk P Barendregt. *The Lambda Calculus: Its Syntax and Semantics.* Studies in Logic and the Foundations of Mathematics vol. 103. Elsevier Science, 2013.

[Chu32] Alonzo Church. A set of postulates for the foundation of logic. *Annals of Mathematics*, 33(2):pp. 346–366, 1932.

[Gir86] Jean-Yves Girard. The system F of variable types, fifteen years later. *Theor. Comput. Sci.*, 45(2):159–192, September 1986.

[GTL89] Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and Types.* Cambridge University Press, New York, NY, USA, 1989.

[LS88] Joachim Lambek and Philip J Scott. *Introduction to Higher-Order Categorical Logic.* Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1988.

[ML98] Saunders Mac Lane. *Categories for the Working Mathematician.* Graduate Texts in Mathematics. Springer New York, 1998.

[Rey84] John C Reynolds. *Polymorphism is not set-theoretic.* Springer, 1984.

[See87] Robert AG Seely. Categorical semantics for higher order polymorphic lambda calculus. *The Journal of Symbolic Logic*, 52(04):969–989, 1987.

[SU98] Morten Heine B Sørensen and Pawel Urzyczyn. *Lectures on the Curry-Howard Isomorphism.* Studies in Logic and the Foundations of Mathematics vol. 149. Elsevier Science, 1998.

[vO02] Jaap van Oosten. *Basic Category Theory.* 2002.