SJÄLVSTÄNDIGA ARBETEN I MATEMATIK

MATEMATISKA INSTITUTIONEN, STOCKHOLMS UNIVERSITET

Formalizing Cartesian Cubical Sets in UniMath

av

Elisabeth Bonnevier

2020 - No M2

MATEMATISKA INSTITUTIONEN, STOCKHOLMS UNIVERSITET, 106 91 STOCKHOLM

Formalizing Cartesian Cubical Sets in UniMath

Elisabeth Bonnevier

Självständigt arbete i matematik 30 högskolepoäng, avancerad nivå

Handledare: Anders Mörtberg

2020

Abstract

Homotopy type theory is a new formal system for doing constructive mathematics. As a system for logical deductions, its consistency must be verified by models. We will look at an axiomatization of constructive presheaf models of homotopy type theory and show that cartesian cubical sets satisfy four of these axioms. These results are formalized using the UniMath library for the proof assistant Coq, to which we also give an introduction.

Acknowledgements

I would like to thank my thesis advisor Anders Mörtberg for his unlimited patience with my many questions and for always finding time for me when I needed guidance. I would also like to thank Guillaume Brunerie for helpful discussions about Coq.

Contents

1 Introduction

Like set theory, dependent type theory is a framework for logical deduction. It was created by Swedish logician, philosopher and mathematical statistician Per Martin-Löf [11]. It is constructive in nature and implements the Curry-Howard isomorphism between proofs and computation. Therefore dependent type theory is very well suited for computer aided theorem proving. As an example of this, Coq is a tool for (among other things) proving mathematical theorems with the help of a computer and is built upon a version of dependent type theory.

In its original formulation, dependent type theory does not have a canonical way of transporting constructions of one type to a construction of an equivalent type, which is a property we often want and use in mathematics, i.e. that any structure/proof that holds for one type holds for any equivalent type. Due to the late Vladimir Voevodsky, this problem was solved by adding an axiom called the Univalence Axiom [14, Axiom 2.10.3] to dependent type theory, which allows us to do such transportations between equivalent types. This version of dependent type theory is called Homotopy Type Theory (HoTT). The name comes from the observation that equality between terms behaves very much like paths between points in a topological space and that we can consider equalities between equalities like we can consider homotopies between paths.

As Voevodsky added the univalence axiom to dependent type theory he also began proving basic mathematical results in Coq with the addition of the univalence axiom [15]. This set of formalized mathematics based on univalence has since been expanded upon and is now called the UniMath library. It contains among other things formalizations of basic results in category theory which have been used in the work of this thesis.

With an axiom added to dependent type theory one needs to construct a model to show that the theory is still consistent, which Voevodsky also did using simplicial sets [6]. But this model relied on a non-constructive metatheory. In the quest to find models of HoTT in a constructive metatheory Bezem et al. found the first such model using cubical sets [2]. After this several more models using different cubical sets have been found (for an overview see [3, Section 1]).

From these different cubical set models of HoTT, an axiomatization was formulated by Orton and Pitts [12] to construct models of HoTT using presheaf categories. A summary and updated version of this axiomatization is found in Coquand's article *A Survey of Constructive Presheaf Models of Univalence* [5]. In this article Coquand presents eight axioms on presheaf categories that when satisfied produce a model of HoTT. Four of these (the axioms labeled B1-B4 in Coquand's article) concern properties of a certain presheaf, denoted $\mathbb{I}$, that the category needs to contain. The focus of this thesis has been the presheaf $\mathbb{I}$ and the axioms concerning it. In a recent paper Cavallo, Mörtberg and Swan show that the fourth of these axioms is actually not needed [3], so it is enough to consider only on the first three.

The first two axioms are straightforward to show for a given presheaf category while the third requires more effort. In their article on universes in models of

HoTT, Licata et al. give a proof that the interval $\mathbb{I}$ in their cubical set model satisfies the third axiom [9, p. 10]. Their proof is very short and leaves out many details. The main work of this thesis has been to expand the details of this proof and generalize it and to formalize this result using the UniMath library. The remaining work has been to formalize that the interval in the cartesian cubical sets model of HoTT satisfies the three axioms.

It is important that (at least one of) the constructive models of the univalence axiom have been formally verified since they show the consistency of homotopy type theory. Only some of Coquand's axioms have been verified for the cartesian cubical set model in this thesis, but this formalization may make the formal verification of other cubical set models simpler since the code can be used as a template to start the new formalization from. Moreover, the generalization of the proof by Licata et al. [9, p. 10] resulted in a set of sufficient conditions on the cube category used, in order for the interval $\mathbb{I}$ to satisfy the third axiom (see Theorem 4.7). As this theorem has been formalized it can be used for the verification of this axiom for other presheaf models, again making such a formalization simpler.

2 Dependent type theory

The usual foundation for mathematics is set theory, which is a system for making logical deductions. But a proposed new candidate as a foundation is dependent type theory, which was created by Swedish logician Per Martin-Löf [11]. While set theory is built on first order logic by adding a set of axioms on top of it, dependent type theory instead embeds all information needed to do everyday mathematics within the inference rules. In its original formulation by Martin-Löf, no axioms were added on top of the theory. However, as we will see later, the univalence axiom was eventually added to improve the properties of the system and this extended type theory was given the name of homotopy type theory.

As the work of this thesis has been to formalize some aspects of constructive models of the univalence axiom and since the theoretical foundation of the UniMath library used for this is dependent type theory we begin by giving an introduction to the subject. We end this section by stating the univalence axiom and by discussing how to construct models of dependent type theory with this axiom using presheaf categories. First, we begin by a description of the Curry-Howard isomorphism upon which dependent type theory is built.

2.1 The Curry-Howard isomorphism

We will assume that the reader is familiar with the λ-calculus and natural deduction for first order logic. The Curry-Howard isomorphism gives us a one-to-one correspondence between proofs of propositions and elements of sets.

Say that we, given a variable $x \in A$, have an element $u \in B$ (depending on x). Then we can construct the function $\lambda x.u : A \to B$ (this is the function that maps x to u). We can write this as the deduction rule

$$\frac{x \in A \vdash u \in B}{\vdash \lambda x.u : A \to B}\ .$$

Similarly, say that we, given that proposition P is true, can show that proposition Q is true. Then we can conclude that P implies Q. We can write this as the deduction rule

$$\frac{P \vdash Q}{\vdash P \to Q}\ .$$

We see that these deduction rules have the same form. Now suppose the other way around, that we have a function $f : A \to B$ and an element $a \in A$. Then we can get an element in B by applying f to a. We can write this as the deduction rule

$$\frac{\vdash f : A \to B \quad \vdash a \in A}{\vdash f(a) \in B}\ .$$

Similarly, say that we know that P implies Q and that P holds. Then we can conclude that Q also holds. We can write this as the deduction rule

$$\frac{\vdash P \to Q \quad \vdash P}{\vdash Q}\ .$$

We see again that these rules have the same form. This can be understood by the Brouwer–Heyting–Kolmogorov interpretation (BHK-interpretation) of what a proof of a proposition is. A proof that P implies Q is a function that, given a proof of P returns a proof of Q. So implication on the propositional side corresponds to functions on the computational side.

Now suppose that we have elements $a \in A$ and $b \in B$. Then we can create the pair (a, b) in the cartesian product $A \times B$. We can write this as the deduction rule

$$\frac{\vdash a \in A \quad \vdash b \in B}{\vdash (a, b) \in A \times B}\ .$$

Similarly, say that P is true and Q is true, then we can conclude that $P \wedge Q$ is true. We can write this as the deduction rule

$$\frac{\vdash P \quad \vdash Q}{\vdash P \wedge Q}\ .$$

These rules both have the same form.

Now suppose the other way around that we have an element $u \in A \times B$. Then we can take the first projection (pr_1) of u to get an element in A and the second projection (pr_2) of u to get an element in B. This can be written as the deduction rules

$$\frac{\vdash u \in A \times B}{\vdash \mathsf{pr}_1(u) \in A} \qquad \text{and} \qquad \frac{\vdash u \in A \times B}{\vdash \mathsf{pr}_2(u) \in B}\ .$$

Similarly, if we know that $P \wedge Q$ is true then we can conclude that P is true and that Q is true. These are the deduction rules:

$$\frac{\vdash P \wedge Q}{\vdash P} \qquad \text{and} \qquad \frac{\vdash P \wedge Q}{\vdash Q}\ .$$

The fact that the rules for sets and propositions are of the same form can be understood by the BHK-interpretation in that a proof of the proposition $P \wedge Q$ is a pair (p, q) where p is a proof of P and q is a proof of Q.

In the same way as above, logical disjunction corresponds to disjoint union, for an element in a disjoint union $A \sqcup B$ is either a copy of an element in A or an element in B. Similarly, a proof of the proposition $P \vee Q$ either consists of a proof of P or a proof of Q.

The true and false propositions $\top$ and $\bot$ correspond to the set with one element and the empty set respectively. The true proposition $\top$ only has one canonical proof; it is always true. The false proposition $\bot$ has no proof.

By implementing this correspondence, dependent type theory handles both propositions and computations simultaneously using the same language.

2.2 Judgments and structural typing rules

The objects of dependent type theory are terms and types. Each term has a type. Through the Curry-Howard isomorphism, types can be seen both as sets and as propositions and terms can be seen both as elements and as proofs of propositions.

The terms and types come with typing rules which we will present in the same form as the inference rules in natural deduction of first order logic (some of which we have already discussed in Section 2.1).

2.2.1 Judgments

There are five types of judgments in dependent type theory:

- $\Gamma\,\mathsf{ctx}$, which states that Γ is a well-typed context,
- $\Gamma \vdash A\,\mathsf{type}$, which states that A is a well-typed type in context Γ,
- $\Gamma \vdash A \equiv B\,\mathsf{type}$, which states that A and B are definitionally equal types in context Γ,
- $\Gamma \vdash a : A$, which states that a is a term of type A in context Γ and
- $\Gamma \vdash a \equiv b : A$, which states that a and b are definitionally equal terms of type A in context Γ.

Any deduction in dependent type theory consists of different instances of these five judgments. (Some authors leave out the first judgment and consider it instead as a finite series of judgments of the second type.)

2.2.2 Contexts

A context can be thought of as a list of declared variables on the computational side, or as a list of assumptions on the propositional side. It is a list of the form

$$x_1 : A_1, x_2 : A_2, ..., x_n : A_n$$

or an empty list. Such a context is well-typed if it can be derived using the inference rules

$$\frac{}{\emptyset\ \mathsf{ctx}}\ \emptyset\text{-}\mathsf{ctx} \qquad \frac{\Gamma\ \mathsf{ctx} \quad \Gamma \vdash A_{n+1}\ \mathsf{type}}{\Gamma, x_{n+1} : A_{n+1}\ \mathsf{ctx}}\ \mathsf{ctx}\text{-ext}$$

where in the second rule, x_{n+1} must be distinct from all variables in Γ.

2.2.3 Structural rules

Before detailing the different type formers in dependent type theory, we begin by introducing some general structural rules that ensure that the judgments behave as expected.

We have the variable rule:

$$\frac{x_1 : A_1, ..., x_n : A_n\ \mathsf{ctx}}{x_1 : A_1, ..., x_n : A_n \vdash x_i : A_i}\ \text{var} .$$

which states that if we have term of type A_i in our context then we can conclude that we have a term of type A_i.

Then we have the rules that ensure that definitional equality is an equivalence relation:

$$\frac{\Gamma \vdash A\ \mathsf{type}}{\Gamma \vdash A \equiv A\ \mathsf{type}} \qquad \frac{\Gamma \vdash A \equiv A'\ \mathsf{type}}{\Gamma \vdash A' \equiv A\ \mathsf{type}} \qquad \frac{\Gamma \vdash A \equiv A'\ \mathsf{type} \quad \Gamma \vdash A' \equiv A''\ \mathsf{type}}{\Gamma \vdash A \equiv A''\ \mathsf{type}}$$

$$\frac{\Gamma \vdash a : A}{\Gamma \vdash a \equiv a : A} \qquad \frac{\Gamma \vdash a \equiv a' : A}{\Gamma \vdash a' \equiv a : A} \qquad \frac{\Gamma \vdash a \equiv a' : A \quad \Gamma \vdash a' \equiv a'' : A}{\Gamma \vdash a \equiv a'' : A} .$$

We also need to ensure that if the type of a term is definitionally equal to another type then the term also has that type:

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash A \equiv A'\ \mathsf{type}}{\Gamma \vdash a : A'} \qquad \frac{\Gamma \vdash a \equiv a' : A \quad \Gamma \vdash A \equiv A'\ \mathsf{type}}{\Gamma \vdash a \equiv a' : A'}$$

2.2.4 Admissible rules

There are several rules that cannot be derived in dependent type theory but that nonetheless do not need to be added because they can be proved using induction on all possible derivations. These are called admissible rules. Two such rules are substitution and weakening. We will use a double line when writing admissible rules to distinguish them from the other typing rules. Let $\mathcal{J}$ be one of the last four types of judgments in Section 2.2.1 and let Γ, Δ be arbitrary well-typed contexts. Then we have the admissible rules

$$\frac{\Gamma \vdash a : A \quad \Gamma, x : A, \Delta \vdash \mathcal{J}}{\Gamma, \Delta[x := a] \vdash \mathcal{J}[x := a]}\ \mathsf{subst} \qquad \frac{\Gamma \vdash A\ \mathsf{type} \quad \Gamma, \Delta \vdash \mathcal{J}}{\Gamma, x : A, \Delta \vdash \mathcal{J}}\ \mathsf{wkg}$$

where x must be distinct from the variables in Γ, Δ in the second rule.

2.2.5 Universes

In order to be able to quantify over types we need some object to quantify over since everything in type theory is typed. So we need essentially a type of types, i.e. the terms in this type are types. Such types of types are called universes and will be denoted by $\mathcal{U}$. There are different ways of constructing universes, the main two being universes à la Russel and universes à la Tarski. Coq uses cumulative Russell style universes. So there is a countably infinite hierarchy $\mathcal{U}_1 : \mathcal{U}_2 : ... : \mathcal{U}_n : ...$ of universes in which each universe is a term of the next universe and the terms of the first universe are the types. Moreover, cumulativity means that if $A : \mathcal{U}_i$ then $A : \mathcal{U}_{i+1}$.

2.2.6 Groups of typing rules

The typing rules of each type former can be categorized into groups as follows:

Formation rule A rule that describes when the type is well-typed.

Introduction rules Zero, one or more rules which describe how to construct terms of the type.

Elimination rules One or more rules which describe how to use terms of the type.

β-reduction For every pair of an introduction and an elimination rule, a β-reduction rule that states what we get when constructing a term and immediately eliminating it.

η-expansion Some types with exactly one introduction rule have an η-expansion rule which states that any term of the type is equal to the constructor applied to appropriate arguments.

Congruence rules For each formation, introduction and elimination rule, a congruence rule which states that this rule respects definitional equality.

2.3 Dependent function types

We now move to one of the basic type formers in dependent type theory, the dependent function type, or Π-type. Given a type A and a family B of types dependent on A, i.e. for every $a : A$ we have a type $B(a)$, the dependent function type can be thought of as the type of functions from A to B (note that the codomain here varies). We denote this type by $\prod_{x:A} B(x)$. The typing rules are:

$$\frac{\Gamma \vdash A\,\mathsf{type} \quad \Gamma, x : A \vdash B(x)\,\mathsf{type}}{\Gamma \vdash \prod_{x:A} B(x)\,\mathsf{type}}\ \Pi\text{-}\mathsf{form}$$

$$\frac{\Gamma, x : A \vdash f(x) : B(x)}{\Gamma \vdash \lambda x.f(x) : \prod_{x:A} B(x)}\ \Pi\text{-}\mathsf{intro}$$

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash f : \prod_{x:A} B(x)}{\Gamma \vdash f(a) : B(a)}\ \Pi\text{-elim}$$

$$\frac{\Gamma \vdash a : A \quad \Gamma, x : A \vdash f(x) : B(x)}{\Gamma \vdash (\lambda x.f(x))(a) \equiv f(a) : B(a)}\ \Pi\text{-}\beta$$

$$\frac{\Gamma \vdash f : \prod_{x:A} B(x)}{\Gamma \vdash f = \lambda x.f(x) : \prod_{x:A} B(x)}\ \Pi\text{-}\eta$$

$$\frac{\Gamma \vdash A \equiv A' \ \mathsf{type} \quad \Gamma, x : A \vdash B(x) \equiv B'(x) \ \mathsf{type}}{\Gamma \vdash \prod_{x:A} B(x) \equiv \prod_{x:A'} B'(x) \ \mathsf{type}}\ \Pi\text{-}\mathsf{cong}_1$$

$$\frac{\Gamma, x : A \vdash f(x) \equiv f'(x) : B(x)}{\Gamma \vdash \lambda x.f(x) \equiv \lambda x.f'(x) : \prod_{x:A} B(x)}\ \Pi\text{-}\mathsf{cong}_2$$

$$\frac{\Gamma \vdash a \equiv a' : A \quad \Gamma \vdash f \equiv f' : \prod_{x:A} B(x)}{\Gamma \vdash f(a) \equiv f'(a') : B(a)}\ \Pi\text{-}\mathsf{cong}_3$$

On the propositional side, the Π-type corresponds to universal quantification, because for any proposition $P(x)$ dependent on an element $x \in A$, a proof of the proposition

$$\forall x \in A : P(x)$$

is a function that for every element $a \in A$ returns a witness (i.e. a term of) $P(a)$.

If the type B does not depend on A then a term in $\prod_{x:A} B$ is simply a (non-dependent) function that for every term $a : A$ returns a term $b : B$. So in this case the Π-type is corresponds to the ordinary function type from A to B and we denote it by $A \to B$.

2.4 Dependent pair types

Another basic type former is the dependent pair type, or Σ-type. Given a type A and a family B of types dependent on A a term of the dependent pair type is an object of the form (a, b) where $a : A$ and $b : B(a)$. The typing rules are:

$$\frac{\Gamma \vdash A \ \mathsf{type} \quad \Gamma, x : A \vdash B(x) \ \mathsf{type}}{\Gamma \vdash \sum_{x:A} B(x) \ \mathsf{type}}\ \Sigma\text{-}\mathsf{form}$$

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B(a) \quad \Gamma, x : A \vdash B(x) \ \mathsf{type}}{\Gamma \vdash (a, b) : \sum_{x:A} B(x)}\ \Sigma\text{-}\mathsf{intro}$$

$$\frac{\Gamma \vdash u : \sum_{x:A} B(x)}{\Gamma \vdash \mathsf{pr}_1(u) : A}\ \Sigma\text{-}\mathsf{elim}_1 \qquad \frac{\Gamma \vdash u : \sum_{x:A} B(x)}{\Gamma \vdash \mathsf{pr}_2(u) : B(\mathsf{pr}_1(u))}\ \Sigma\text{-}\mathsf{elim}_2$$

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B(a) \quad \Gamma, x : A \vdash B(x)\ \mathsf{type}}{\Gamma \vdash \mathsf{pr}_1((a,b)) \equiv a : A}\ \Sigma\text{-}\beta_1$$

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B(a) \quad \Gamma, x : A \vdash B(x)\ \mathsf{type}}{\Gamma \vdash \mathsf{pr}_2((a,b)) \equiv b : B(a)}\ \Sigma\text{-}\beta_2$$

$$\frac{\Gamma \vdash u : \sum_{x:A} B(x)}{\Gamma \vdash u \equiv (\mathsf{pr}_1(u), \mathsf{pr}_2(u)) : \sum_{x:A} B(x)}\ \Sigma\text{-}\eta$$

We also have congruence rules which state that the type and term formers and the elimination rules respect definitional equality, but we will henceforth not write these out.

On the propositional side, Σ-types correspond to strong existential quantification (strong existential quantification requires the construction of an explicit witness of the proposition under the quantifier), because for any proposition $P(x)$ dependent on an element $x \in A$, a witness of the proposition

$$\exists x \in A : B(x)$$

consists of a pair (a, b) where a is a witness of A and b is a witness of $B(a)$.

Observe that if B does not depend on A then a term in the type $\sum_{x:A} B$ is a pair (a, b) where $a : A$ and $b : B$. So in this case the Σ-type corresponds to cartesian product and we denote it by $A \times B$.

2.5 Coproducts

The coproduct type can be thought of as disjoint union, in terms of sets, and disjunction, in terms of propositional logic. The typing rules are:

$$\frac{\Gamma \vdash A\ \mathsf{type} \quad \Gamma \vdash B\ \mathsf{type}}{\Gamma \vdash A \sqcup B\ \mathsf{type}}\ \sqcup\text{-}\mathsf{form}$$

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash B\ \mathsf{type}}{\Gamma \vdash \mathsf{inl}(a) : A \sqcup B}\ \sqcup\text{-}\mathsf{intro}_1 \qquad \frac{\Gamma \vdash A\ \mathsf{type} \quad \Gamma \vdash b : B}{\Gamma \vdash \mathsf{inr}(b) : A \sqcup B}\ \sqcup\text{-}\mathsf{intro}_2$$

$$\frac{\begin{array}{cc} \Gamma \vdash u : A \sqcup B & \Gamma, y : B \vdash d_r : P(\mathsf{inr}(y)) \\ \Gamma, w : A \sqcup B \vdash P(w)\ \mathsf{type} & \Gamma, x : A \vdash d_l : P(\mathsf{inl}(x)) \end{array}}{\Gamma \vdash \mathsf{elim}_{\sqcup, d_l, d_r}(u) : P(u)}\ \sqcup\text{-}\mathsf{elim}$$

$$\frac{\begin{array}{cc}\Gamma \vdash a : A & \Gamma, x : A \vdash d_l : P(\mathsf{inl}(x)) \\ \Gamma, w : A \sqcup B \vdash P(w)\,\mathsf{type} & \Gamma, y : B \vdash d_r : P(\mathsf{inr}(y))\end{array}}{\Gamma \vdash \mathsf{elim}_{\sqcup, d_l, d_r}(\mathsf{inl}(a)) \equiv d_l(a) : P(\mathsf{inl}(a))}\ \sqcup\text{-}\beta_1$$

$$\frac{\begin{array}{cc}\Gamma \vdash b : B & \Gamma, x : A \vdash d_l : P(\mathsf{inl}(x)) \\ \Gamma, w : A \sqcup B \vdash P(w)\,\mathsf{type} & \Gamma, y : B \vdash d_r : P(\mathsf{inr}(y))\end{array}}{\Gamma \vdash \mathsf{elim}_{\sqcup, d_l, d_r}(\mathsf{inr}(b)) \equiv d_r(b) : P(\mathsf{inr}(b))}\ \sqcup\text{-}\beta_2$$

We will write $[d_l, d_r]$ for the term $\lambda w.\mathsf{elim}_{\sqcup, d_l, d_r}(w) : \prod_{w:A\sqcup B} P(w)$.

2.6 General inductive types

Coproducts are an example of inductive types. One can define a general notion of inductive types with an algorithm that generates all the typing rules given all the introduction rules of the type. Loosely speaking, we define zero, one or more introduction rules for our inductive type. Then we have an induction rule that states that for any type family B over our inductive type, if we can construct a term in $B(x)$ for any term x resulting from an introduction rule, then we can construct a term in $B(w)$ for any term w in the inductive type.

It is also possible to define rules that allow us to use pattern matching when defining inductive types, which is usually what is done in practice in proof assistants.

Some common examples of inductive types include coproducts, the natural numbers ($\mathbb{N}$), the empty type ($\emptyset$), the unit type ($\mathbf{1}$) and the booleans ($\mathbf{2}$). We will now show how the last four of these are defined.

2.6.1 The natural numbers

The type most mathematicians are familiar with as an inductive type are the natural numbers. We define the 'base element' 0 and a successor function $\mathsf{S} : \mathbb{N} \to \mathbb{N}$ that inductively gives us all the natural numbers. The introduction rules in dependent type theory for the natural numbers are

$$\frac{\Gamma\,\mathsf{ctx}}{\Gamma \vdash 0 : \mathbb{N}}\ \mathbb{N}\text{-}\mathsf{intro}_1 \qquad \frac{\Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathsf{S}n : \mathbb{N}}\ \mathbb{N}\text{-}\mathsf{intro}_2.$$

The induction rule generated by the algorithm for typing rules of general inductive types then becomes

$$\frac{\Gamma, x : \mathbb{N} \vdash P(x)\,\mathsf{type} \quad \Gamma \vdash p_0 : P(0) \quad \Gamma, m : \mathbb{N} \vdash p_{S(m)} : P(m) \to P(S(m))}{\Gamma \vdash \mathsf{elim}_{\mathbb{N}, p_0, p_{S(m)}}(n) : P(n)}\ \mathbb{N}\text{-}\mathsf{elim},$$

i.e. if we can construct a term of type $P(0)$ and for every natural number m a function that sends a term of type $P(m)$ to a term of type $P(\mathsf{S}(m))$, then we can construct a term of type $P(n)$ for any natural number n. When P is seen as a proposition that depends on $\mathbb{N}$, then this is the usual induction principle on the natural numbers. When P is seen as a set that instead does not depend on $\mathbb{N}$ then this is the usual recursion on the natural numbers, i.e. we define a function on $\mathbb{N}$ by stating where it maps 0 and the successor of any number.

2.6.2 The empty type

The empty type does not contain any terms so we define it as the inductive type without introduction rules. Since there are no introduction rules the elimination rule becomes

$$\frac{\Gamma, x : \emptyset \vdash P(x)\ \mathsf{type}}{\Gamma \vdash \mathsf{elim}_{\emptyset} : \prod_{x:\emptyset} P(x)}\ \emptyset\text{-}\mathsf{elim}.$$

As a proposition the empty type corresponds to $\bot$ and $\emptyset$-elim corresponds to ex falso quodlibet.

2.6.3 The unit type

The unit type should contain only one term. So the only introduction rule is

$$\frac{\Gamma\ \mathsf{ctx}}{\Gamma \vdash tt : \mathbf{1}}\ \mathbf{1}\text{-}\mathsf{intro}$$

and the elimination rule is

$$\frac{\Gamma, x : \mathbf{1} \vdash P(x)\ \mathsf{type} \quad \Gamma \vdash p : P(tt)}{\Gamma \vdash \mathsf{elim}_{\mathbf{1},p} : \prod_{x:\mathbf{1}} P(x)}\ \mathbf{1}\text{-}\mathsf{elim}.$$

As a proposition the unit type corresponds to $\top$.

2.6.4 The booleans

The type of booleans contains exactly two terms, 0_2 and 1_2 which correspond to true and false. There is one introduction rule for each of these terms. Explicitly:

$$\frac{\Gamma\ \mathsf{ctx}}{\Gamma \vdash 0_2 : \mathbf{2}}\ \mathbf{2}\text{-}\mathsf{intro}_1 \qquad \frac{\Gamma\ \mathsf{ctx}}{\Gamma \vdash 1_2 : \mathbf{2}}\ \mathbf{2}\text{-}\mathsf{intro}_2.$$

The elimination rule then becomes

$$\frac{\Gamma, x : \mathbf{2} \vdash P(x)\ \mathsf{type} \quad \Gamma \vdash p_0 : P(0_2) \quad \Gamma \vdash p_1 : P(1_2)}{\Gamma \vdash \mathsf{elim}_{\mathbf{2},p_0,p_1} : \prod_{x:\mathbf{2}} P(x)}\ \mathbf{2}\text{-}\mathsf{elim}$$

2.7 The identity type

We now come to a family of types that is perhaps not as intuitively clear as the previous types: types corresponding to propositional equality.

In our type system we have a version of definitional equality, but it is not enough to express all of mathematics. For example, the expressions $x + 2$ and $2 + x$ evaluate to the same natural number for every natural number x so we would like to have a witness of the equality "$x + 2 = 2 + x$" (in the context $x : \mathbb{N}$), but this does not hold as a definitional equality.

Consider the statement "a = b". This is a statement that can either be true or false, so it is a proposition. Therefore it should correspond to a type (rather than a term) in dependent type theory. The question is now, how much structure should this type have? It turns out that it is enough to have a canonical term of the type $a = a$ and an induction principle for the identity type to behave as desired. The formation and introduction rules of the identity type are:

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash b : A}{\Gamma \vdash a =_A b\,\mathsf{type}} = \text{-form} \qquad \frac{\Gamma \vdash a : A}{\Gamma \vdash \mathsf{refl}_a : a =_A a} = \text{-intro},$$

i.e. for any two terms a and b of the same type we can form the proposition that a is equal to b and we always have a witness that a is equal to itself. We often leave out the subscript A of the equality sign as this type can be inferred from the terms a and b. Now, we would like propositional equality to behave in such a way that if a proposition $P(x)$ dependent on $x \in A$ has a witness for some element $a \in A$, then it should have a witness for any element $x \in A$ such that $a = x$ (propositionally). This is exactly how we define the elimination rule for propositional equality:

$$\frac{\Gamma \vdash a : A \quad \Gamma, x : A, p : a =_A x \vdash P(x,p)\,\mathsf{type} \quad \Gamma \vdash p_a : P(a, \mathsf{refl}_a)}{\Gamma \vdash \mathsf{elim}_{=,a} : \prod_{x:A} \prod_{p:a=_A x} P(x,p)} = \text{-elim}$$

(note that P depends on the witness p of the propositional equality). We will also refer to this rule as path induction for reasons that will be discussed in Section 2.8. A β-reduction rule is needed that states what happens when we apply the elimination term to the terms a and refl_a:

$$\frac{\Gamma \vdash a : A \quad \Gamma, x : A, p : a =_A x \vdash P(x,p)\,\mathsf{type} \quad \Gamma \vdash p_a : P(a, \mathsf{refl}_a)}{\Gamma \vdash \mathsf{elim}_{=,a}(a, \mathsf{refl}_a) \equiv p_a : P(a, \mathsf{refl}_a)} = \text{-}\beta$$

It is possible to define the identity type with both sides of the equality being variables (see [14, Section 1.12.2]) but in this text we use based identity types as that is what is used in the UniMath library.

This is enough structure on the equality types because using path induction we can show that propositional equality is an equivalence relation.

2.7.1 Symmetry

We begin by constructing a term of the type $a =_A b \to b =_A a$. It is possible to do this as a full deduction tree, but such trees easily become very large and cumbersome so we will instead write the deduction in text.

By path induction, in order to construct a term of type $a = b \to b = a$ it is enough to construct a term of type $a = a \to a = a$. For this we can simply take the identity function

$$\mathsf{id}_{a=a} :\equiv \lambda p.p : a = a \to a = a.$$

Then we apply path induction to get the term

$$\_^{-1} :\equiv \mathsf{elim}_{=,a}(b) : (a =_A b) \to (b =_A a),$$

i.e. $p^{-1} : b =_A a$ for any term $p : a =_A b$. We leave a, b and A as implicit arguments because these can all be inferred from p.

2.7.2 Transitivity

To show transitivity we again use path induction. In order to construct a term of type

$$(a =_A b) \to (b =_A c) \to (a =_A c)$$

it is by path induction enough to construct a term of the type

$$(a =_A a) \to (a =_A c) \to (a =_A c).$$

For this we can simply take the function that returns the second argument:

$$\lambda p.\lambda q.q : (a =_A a) \to (a =_A c) \to (a =_A c).$$

Then we apply path induction to get the term

$$\_ \cdot \_ :\equiv \mathsf{elim}_{=,a}(b) : (a =_A b) \to (b =_A c) \to (a =_A c),$$

i.e. $p \cdot q : a =_A c$ for any terms $p : a =_A b$ and $q : b =_A c$.

2.8 Homotopy type theory

An observation from Section 2.7.1 and 2.7.2 is that types behave like topological spaces with the terms corresponding to the points in the space and terms of the identity type of two terms behave like paths between the points: for any term $p : a =_A b$ if we view it as a path between points a and b in a space A then we can construct the inverse path $p^{-1} : b =_A a$ from b to a. Given two paths $p : a =_A b$ and $q : b =_A c$, we can compose them to get a path $p \cdot q : a =_A c$. We also always have the constant path $\mathsf{refl}_a : a =_A a$ from a to a.

Since the identity type can be formed between any two terms of any type, we can form the identity type $p =_{a =_A b} q$ for $p, q : a =_A b$, that is, we can look at paths between paths, i.e. homotopies between paths. One can continue to iterate the identity type to get a hierarchy of homotopies between homotopies one level lower in the hierarchy. In this way it turns out that the identity type in dependent type theory has a non-trivial structure. The view of the identity type as a set of paths gave rise to the area of homotopy type theory and it is due to this view of the identity type that equality elimination is often called path induction.

Let us now see some more examples of path induction and how to endow the identity type with a groupoid structure (in the category theoretic sense) viewing $\_ \cdot \_$ as composition.

Proposition 2.1. *For any term $p : x = y$ we have the equalities $\mathsf{refl}_x \cdot p = p$ and $p \cdot \mathsf{refl}_y = p$ (i.e. the types are inhabited).*

Proof. By the definition of $\_ \cdot \_$ and the β-reduction of the identity type we have

$$\mathsf{refl}_x \cdot p \equiv p.$$

It follows that

$$(\mathsf{refl}_x \cdot p = p) \equiv (p = p)$$

as types, and for the latter we have the term

$$\mathsf{refl}_p : p = p.$$

For the type $p \cdot \mathsf{refl}_y = p$ we use path induction. Thus it suffices to construct a term of type

$$\mathsf{refl}_x \cdot \mathsf{refl}_x = \mathsf{refl}_x\,.$$

Again, by definition of $\_ \cdot \_$ and β-reduction we have

$$(\mathsf{refl}_x \cdot \mathsf{refl}_x = \mathsf{refl}_x) \equiv (\mathsf{refl}_x = \mathsf{refl}_x)$$

as types, and for the latter we have the term

$$\mathsf{refl}_{\mathsf{refl}_x} : \mathsf{refl}_x = \mathsf{refl}_x\,.$$

□

So refl is both a left and a right unit to composition. We now show that $\_^{-1}$ is an inverse.

Proposition 2.2. *For any term $p : x = y$ we have the equalities $p \cdot p^{-1} = \mathsf{refl}_x$ and $p^{-1} \cdot p = \mathsf{refl}_y$.*

Proof. For $p \cdot p^{-1} = \mathsf{refl}_x$ it is enough, by path induction, to construct a term of type

$$\mathsf{refl}_x \cdot \mathsf{refl}_x^{-1} = \mathsf{refl}_x\,.$$

By the definition of $\_^{-1}$ and the β-reduction for the identity type we have

$$\mathsf{refl}_x^{-1} \equiv \mathsf{refl}_x\,.$$

It follows that

$$(\mathsf{refl}_x \cdot \mathsf{refl}_x^{-1} = \mathsf{refl}_x) \equiv (\mathsf{refl}_x \cdot \mathsf{refl}_x = \mathsf{refl}_x)$$

for which the latter type is inhabited by Proposition 2.1.

For $p^{-1} \cdot p = \mathsf{refl}_y$ it is enough, by path induction, to construct a term of type

$$\mathsf{refl}_x^{-1} \cdot \mathsf{refl}_x = \mathsf{refl}_x\,.$$

We have the definitional equality

$$(\mathsf{refl}_x^{-1} \cdot \mathsf{refl}_x = \mathsf{refl}_x) \equiv (\mathsf{refl}_x \cdot \mathsf{refl}_x = \mathsf{refl}_x)$$

of types, for which the latter type is again inhabited by Proposition 2.1. □

We show that $\_ \cdot \_$ respects propositional equality. This result will then be used to show that composition is associative.

Proposition 2.3. *For any terms $p : x = y$, $q, r : y = z$ and $s : z = w$ each of the following types is inhabited:*

$$q = r \to p \cdot q = p \cdot r \qquad \textit{and} \qquad q = r \to q \cdot s = r \cdot s.$$

Proof. For the first type, by path induction it is enough to construct a term of type

$$q = r \to \mathsf{refl}_x \cdot q = \mathsf{refl}_x \cdot r$$

where $q, r : x = z$. By definition and β-reduction, we have

$$(\mathsf{refl}_x \cdot q = \mathsf{refl}_x \cdot r) \equiv (q = r)$$

as types. So it is enough to construct a term of type

$$q = r \to q = r.$$

For this we can simply take the identity function

$$\lambda\alpha.\alpha : q = r \to q = r.$$

For the second type, by path induction it is enough to construct a term of type

$$q = r \to q \cdot \mathsf{refl}_z = r \cdot \mathsf{refl}_z .$$

By Proposition 2.1 we have terms

$$\varphi : q \cdot \mathsf{refl}_z = q \quad \text{and} \quad \psi : r \cdot \mathsf{refl}_z = r.$$

Thus, for any term $\sigma : q = r$ we have the term

$$(\varphi \cdot \sigma) \cdot \psi^{-1} : q \cdot \mathsf{refl}_z = r \cdot \mathsf{refl}_z .$$

Thus, by λ-abstraction, we have the term

$$\lambda\sigma.((\varphi \cdot \sigma) \cdot \psi^{-1}) : q = r \to q \cdot \mathsf{refl}_z = r \cdot \mathsf{refl}_z$$

which is of the desired type. $\square$

We now show that $\_ \cdot \_$ is associative.

Proposition 2.4. *For any terms $p : x = y$, $q : y = z$ and $r : z = w$ we have $(p \cdot q) \cdot r = p \cdot (q \cdot r)$.*

Proof. By path induction it is enough to construct a term of type

$$(p \cdot \mathsf{refl}_y) \cdot r = p \cdot (\mathsf{refl}_y \cdot r)$$

where $r : y =_A w$. By Proposition 2.1 the types $p \cdot \mathsf{refl}_y = p$ and $\mathsf{refl}_y \cdot r = r$ are inhabited and thus, by Proposition 2.3 we have terms

$$\varphi : (p \cdot \mathsf{refl}_y) \cdot r = p \cdot r$$

and

$$\psi : p \cdot (\mathsf{refl}_y \cdot r) = p \cdot r.$$

It follows that the term

$$\varphi \cdot \psi^{-1} : (p \cdot \mathsf{refl}_y) \cdot r = p \cdot (\mathsf{refl}_y \cdot r)$$

is of the desired type. □

By the propositions above it follows that we can endow any type with a groupoid structure by defining $\_ \cdot \_$ as morphism composition, $\_^{-1}$ as the inverse and refl as the unit.

Given two paths $p, q : a = b$ one can ask if we always have $p = q$. The proposition that this holds is called *Uniqueness of Identity Proofs* (UIP). An interesting consequence of the definition of the identity type is that one cannot prove UIP. Indeed, this property is independent of dependent type theory since there are models that satisfy UIP and models that do not satisfy UIP. The groupoid model [13, p. 83-111] of dependent type theory was the first model to contradict UIP showing that it cannot be proved. In fact, as observed previously, any type has a hierarchy of structure consisting of homotopies between lower level homotopies. For some types this structure may become trivial after a certain number of levels, but for some types this hierarchy may be non-trivial for all infinite number of levels. Types in dependent type theory have been shown to form weak ω-categories by Lumsdaine [10] with respect to this hierarchy of identity types.

From the homotopy theoretic view of dependent type theory it is not surprising that UIP is not provable because any two paths in a topological space are not necessarily homotopic and the infinite structure of paths between paths is much more natural.

2.9 The univalence axiom

In mathematics we usually have some notion of equivalence in the context we are working in (bijections of functions, group/ring isomorphisms, homeomorphisms of topological spaces, etc.) for which two equivalent structures behave the same. We would like this for our type theory, that two equivalent types have all the same properties, i.e. that we can transport any property of a type A to any equivalent type B. It is not inherent in dependent type theory that we can do this, so the late Vladimir Voevodsky proposed to add an axiom, which he named the *Univalence Axiom* (UA), to the type theory that would make this possible. But in order to state this we must first define equivalence of types.

2.9.1 Homotopies

We say that two functions $f, g : A \to B$ are equal if we have $f(x) = g(x)$ for all $x : A$. In dependent type theory we denote this by $\sim$ as follows:

$$f \sim g :\equiv \prod_{x:A} f(x) = g(x)$$

and call this the type of homotopies from f to g. The name comes from the homotopy interpretation of type theory, i.e. that the identity type is the type of paths between the two endpoints. Then terms of the type $f \sim g$ are homotopies from f to g, and the identification of two functions that are equal on all elements of the domain is the assumption that we have a function

$$f \sim g \to f = g.$$

It is not possible to show that this type is inhabited in dependent type theory, so we need to add this assumption as an axiom which we call *Function Extensionality*. But it turns out that this axiom actually follows from the univalence axiom [14, Section 4.9] which we will now introduce.

2.9.2 Equivalence of types

We need to define equivalence between types. For this we take our intuition from sets. We say that two sets A and B are equivalent if there is a function $f : A \to B$ that has a two-sided inverse $g : B \to A$ such that $g(f(x)) = x$ and $f(g(y))$ for all $x \in A$ and all $y \in B$, i.e. that

$$f \circ g \sim \mathsf{id}_B \quad \text{and} \quad g \circ f \sim \mathsf{id}_A .$$

But it turns out that the corresponding type

$$\sum_{g:B \to A} (f \circ g \sim \mathsf{id}_B) \times (g \circ f \sim \mathsf{id}_A)$$

does not behave well [14, Section 2.4]. Fortunately, one can show that this type is logically equivalent to the much better behaved type

$$\left(\sum_{g:B \to A} f \circ g \sim \mathsf{id}_B \right) \times \left(\sum_{h:B \to A} h \circ f \sim \mathsf{id}_A \right),$$

see [14, Chapter 4]. We denote this type by $\mathsf{isequiv}(f)$ and we say that two types A and B are equivalent if the type

$$(A \simeq B) :\equiv \sum_{f:A \to B} \mathsf{isequiv}(f)$$

is inhabited. As an example, for any type A we have the identity equivalence $\mathsf{ideq}_A : A \simeq A$ defined as the term

$$\mathsf{ideq}_A :\equiv (\mathsf{id}_A, ((\mathsf{id}_A, \lambda x.\, \mathsf{refl}_x), (\mathsf{id}_A, \lambda x.\, \mathsf{refl}_x))).$$

Thus for any two types A and B we have the function

$$\mathsf{idtoequiv} : A = B \to A \simeq B$$

defined by path induction by sending $\mathsf{refl}_A : A = A$ to $\mathsf{ideq}_A : A \simeq A$.

We can now formulate the univalence axiom.

Axiom 2.5 (The Univalence Axiom)**.** *The function* $\mathsf{idtoequiv}$ *is an equivalence.*

It follows by the univalence axiom that the following type is inhabited:

$$\prod_{A,B\ \mathsf{type}} (A = B) \simeq (A \simeq B),$$

i.e. equivalence between two types is equivalent to equality between the types.

By the univalence axiom we get an inverse to idtoequiv:

$$\mathsf{equivtoid} : (A \simeq B) \to (A = B),$$

which is often very useful because it is in general easier to show that two types are equivalent than that they are equal. Given such an equivalence we can use equivtoid to construct a path along which we can transport any results from one type to the other.

2.10 Cubical set models of univalence

When Voevodsky proposed the univalence axiom, he also showed its consistency with a model in Kan simplicial sets [6]. But this model was carried out using classical logic. In a desire to find a model of univalence in a constructive mathematical setting, models using cubical sets, a type of presheaf categories, were created [4, 1] (a presheaf is a functor into the category of sets, see Definition 4.5).

There was a first axiomatization of sufficient properties of a presheaf category in order to model type theory with UA by Orton and Pitts [12]. These axioms have since been revised and condensed by Cavallo et al. [3]. A recent survey article by Coquand [5] contains a compilation of eight axioms on presheaf categories that, if satisfied, produces a model of type theory with UA. We will look at four of these axioms.

Coquand states that the category needs to contain a special presheaf $\mathbb{I}$ (thought of as modeling the unit interval $[0, 1] \in \mathbb{R}$) satisfying the axioms

(B1) $\mathbb{I}$ has two distinct global elements 0 and 1,

(B2) $\mathbb{I}(J)$ has decidable equality for every object J,

(B3) $\mathbb{I}$ is tiny,

(B4) $\mathbb{I}$ has connections.

Axiom (B1) means that there should be two distinct natural transformations from the terminal object to $\mathbb{I}$. Decidable equality in axiom (B2) means that for every pair of elements in $\mathbb{I}(J)$ we can decide if they are equal or if they are not. Axiom (B3) means that the exponential functor by $\mathbb{I}$ has a right adjoint (see Definition 4.14).

In a recent paper, Cavallo, Mörtberg and Swan show that axiom (B4) is not needed [3] by building on a model in cubical sets by Angiuli et al. [1]. Therefore only axioms (B1)-(B3) have been formally verified to hold for cartesian cubical sets in the work of this thesis. This formalization was done using the proof assistant Coq with the UniMath library, which we will now give a brief introduction to.

3 Coq

Coq is a proof assistant that implements the calculus of inductive constructions, a variant of type theory. The idea of a proof assistant is that you write code corresponding to what you want to prove. Then you run the code line by line and another window will show you at every step what you have yet to prove, called the goal(s), and all your assumptions/declared variables. If there is any problem in the proof the assistant will give you an error message and not run that line. In this way you know that if your proof goes through it holds.

Code in Coq is written using so called *tactics* which implement backwards reasoning, which is the reasoning where a deduction rule is read as 'in order to prove this I have to prove these premises'. A tactic is followed by a term (of a specific type) and applied to one of the active goals. Let us do some examples proving some simple propositions in Coq.

In order to prove something in Coq one first has to state the type that one wishes to construct a term of, beginning with one of the *commands* `Theorem`, `Lemma`, `Proposition` and a few other similar commands. This is then followed by a name of the theorem. You then write any arguments/assumptions of the proposition and finally you write the type that you want to create a term of. For example, if we would like to prove modus ponens, i.e. $P \to (P \to Q) \to Q$ for propositions P and Q then we would write

```
Proposition modus_ponens (P Q : Type) :
   P -> (P -> Q) -> Q.
```

Every line of code ends with a dot. Here we state that given types P and Q there is a term, called `modus_ponens`, of type $P \to (P \to Q) \to Q$. By the Curry-Howard isomorphism, this can also be seen as stating that there is a function that takes an element in P, a function from P to Q and returns an element in Q. This is the interpretation we will use when constructing the desired term.

The execution of this line in our proof assistant will create an output that specifies the context and the goal. The goal states the type we need to construct a term of in order to prove the theorem and the context states all our assumptions/declared variables. So after running the above line of code our goal would be

```
P, Q : Type
------------------------------------------
P -> (P -> Q) -> Q
```

After this, one begins the proof by the command `Proof` (with no arguments), after which one uses tactics to construct a term of the type in the goal. So the next line of code in our example would be

```
Proof.
```

This command does not change anything in the goal window.

We now begin constructing a term of type $P \to (P \to Q) \to Q$ using tactics. When the type of the goal is a (dependent or non-dependent) product type we may use the tactic `intro` to apply the λ-abstraction rule to the goal in a 'backwards reasoning' way, i.e. in order to construct a term of a product type it is enough to construct a term of the codomain type given a term of the domain type in the context. So `intro` essentially moves the domain into the context. We write a name after `intro` which is the name given to the term moved into the context. So in our example we would write

```
intro p.
```

which would produce the output

```
P, Q : Type
p :  P
-----------------------------------------
(P -> Q) -> Q
```

when run. The type of the goal is again a product type, so we may use `intro` once more:

```
intro f.
```

which would produce the output

```
P, Q : Type
p :  P
f :  P -> Q
-----------------------------------------
Q
```

So in order to prove our theorem we need only exhibit a term of type Q (because that is the type in our goal). For this we can apply f to p. In order to pass $f(p)$ as the term of our goal type we use the tactic `exact`, which should be followed by a term of the exact (or convertible to the) type of the goal. So in our example we would write

```
exact (f p).
```

which would complete the goal (in Coq, function application is written without parentheses around the arguments). After we have proved a theorem we can end with the command `Qed` or `Defined`. The difference between these two is that `Qed` makes the term opaque which means that the term cannot be expanded when used later. On the other hand, `Defined` makes the term transparent, so that it is possible to expand later on. The choice between `Qed` and `Defined` depends on the situation. While `Qed` can make the code faster if the term is very large, `Defined` may allow simplifications because the term can be expanded. In our case the term is not very large and it would be useful to be able to expand it so we will use

```
Defined.
```

This finishes the proof and the term is stored given the name in the beginning. All together the code above is:

```
Proposition modus_ponens (P Q : Type) :
   P -> (P -> Q) -> Q.
Proof.
   intro p.
   intro f.
   exact (f p).
Defined.
```

We will now introduce the UniMath library and show some examples of code using it because it introduces some new notation and tactics.

3.1 The UniMath library

The UniMath library is a library of formalized mathematics in Coq with the univalence axiom added. It is based on a repository started by Vladimir Voevodsky in 2010 [15] that was expanded and given the name UniMath in 2014. It has since been contributed to by 50 people and is still being actively developed. The repository can be found at https://github.com/UniMath/UniMath.

All the basic types are redefined in the UniMath library and it implements the notation used in Section 2 ($\prod$ for dependent function types, $\sum$ for dependent sum types, etc.). UniMath also defines its own universe UU as the sort Type, with the property that UU : UU. This is technically inconsistent. Perhaps it will be resolved in some future version of UniMath. In the meantime, the formalization done for this thesis is only of results that do not (explicitly) need several different universes.

Code using the UniMath library looks slightly different from writing in native/vanilla Coq. There are also some additional tactics defined in the UniMath library, most notably the tactic `use` which tries to match the given term to the goal but allows variables to be left out in the term. If it succeeds to match the term to the goal, the tactic returns a new subgoal for each variable left out in the term (it is just an application of the tactic `simple refine` in the background).

Let us see some examples of code using the UniMath library. By importing the file Foundations/Preamble.v using the line

```
Require Import UniMath.Foundations.Preamble.
```

we may define the identity function with the code:

```
Proposition idfun (A : UU) : A → A.
Proof.
   exact (λx,x).
Defined.
```

In the UniMath library we can use the notation of λ-calculus. The identity function can also be directly defined without having to 'prove' it using the command `Definition` as follows:

```
Definition idfun {A : UU} :  A → A := λx, x.
```

The command `Definition` is used like `Proposition` but at the end it is followed by := and then a term of the proposed type. `Definition` then stores that term under the name given. The {}-brackets around the type A make this argument implicit which means that Coq will try to infer this type from the context in which this definition is used. If it fails to infer it an error message is produced. It is possible to give implicit parameters explicitly by writing `@` before the term. So if we for example wanted the identity function on the unit type, which is denoted `unit` in the UniMath library, we would write `@idfun unit`. From here on we will use the second construction of the identity function, where the type is an implicit argument.

We now explore the category theory section of the library. Let us define the identity functor. For this we use the definitions of categories and functors already formalized. First we need to import the relevant part of the library:

```
Require Import UniMath.CategoryTheory.Core.Categories.
Require Import UniMath.CategoryTheory.Core.Functors.
```

We open the notation for categories defined in Core/Categories with the line

```
Open Scope Cat.
```

A functor is defined as a term of a Σ-type where the first projection is the functor data, i.e. the map on objects and morphisms, and the second projection is a proof that this data satisfies the identity and composition axioms. For many constructions in the UniMath library, definitions beginning with 'make' have been implemented in order to more easily create terms of a desired type. We begin by defining the map on objects and morphisms using '`make_functor_data`' and the identity function defined previously:

```
Definition id_functor_data (C : category) :
   functor_data C C :=
   make_functor_data idfun (λ _ _, idfun).
```

The first argument to '`make_functor_data`' specifies the function on objects, i.e. it is a term of type `ob C` $\to$ `ob C`. Here we may simply write `idfun` since we made the type in the definition of the identity function an implicit argument. In this case Coq can infer this argument. The second argument specifies the map on morphisms, i.e. it is a term of the type $\prod_{a,b:\texttt{ob C}} \mathrm{Hom}(a,b) \to \mathrm{Hom}(Fa, Fb)$, where F is the map on objects given in the first argument. The underscores next to λ are used because we do not need to refer to those variables, so we do not name them, but we still need some placeholder for them. In this case the two underscores correspond to the objects a and b. Note that we again do not have to specify the type of `idfun` because Coq can infer that argument.

We now prove that this definition satisfies the identity and composition axioms of functors, named in UniMath as `functor_idax` and `functor_compax`. We write this as a lemma as follows:

```
Lemma id_functor_is_functor (C : category) :
   is_functor (id_functor_data C).
Proof.
   split.
   - unfold functor_idax.
     simpl.
     unfold idfun.
     intro.
     apply idpath.
   - unfold functor_compax.
     simpl.
     unfold idfun.
```

```
      intros.
      apply idpath.
  Defined.
```

The type `is_functor F` is defined as `(functor_idax F)` × `(functor_compax F)`. The tactic `split` divides a goal of the type $A \times B$ into the two subgoals A and B, so in our case we divide the goal into the two subgoals `functor_idax (id_functor_data C)` and `functor_compax (id_functor_data C)`. We now use the 'bullet' `-` to focus on the first subgoal where we begin by unfolding the term `functor_idax`, which replaces the name with its definition. The tactic `simpl` then tries to reduce the goal while still attempting to keep it readable (so not fully normalizing it). At this stage our goal window looks something like

```
C : category
------------------------------------------
∏ a :  C, idfun (identity a) = identity (idfun a)
```

together with a message that we have an unfocused subgoal left as well.

We need to unfold `idfun` in order for the goal to reduce. The equality in the goal then reduces to `identity a = identity a`. Since the goal at this point is a dependent product type we can now use the tactic 'intro' to apply the λ-abstraction rule using backwards reasoning, i.e. this moves the object `a :  C` into the context instead (by leaving out the name after `intro` we let Coq decide the name of the term introduced in the context). The goal window at this point looks like

```
C : category
a :  C
------------------------------------------
identity a = identity a
```

and we give the reflexivity term by the tactic `apply` followed by `idpath` (which is UniMath's notation for refl with all arguments implicit), for which we do not specify the arguments. Coq infers these when using `apply` (as opposed to `exact` where we would need to write out all arguments). This finishes the first subgoal.

The second subgoal is essentially the same with the only difference that we must unfold instead `functor_compax`. We end with the command `Defined` because we want this term to be expandable and it is not so large as to be likely to notably slow down any code using it if leaving it transparent.

Now that we have defined the data of the identity functor and proved that it satisfies the identity and composition axioms we create the functor using `make_functor` as follows:

```
Definition id_functor (C : category) :  C ⟶ C :=
   make_functor (id_functor_data C)
                (id_functor_is_functor C).
```

This is only a brief introduction to Coq and the UniMath library, but it should hopefully make it easier for the reader previously unfamiliar with these to understand the formalization done for this thesis.

Since the axioms in Coquand's article [5] concern objects of certain categories we will now give the necessary background in category theory needed to state and prove some results about these axioms.

4 Category theory

We will assume that the reader is familiar with the basic notions of category theory (categories, functors, natural transformations, etc.). The goal of this section is to generalize and expand the details of the proof found in Licata et al. [9] that the interval object is tiny (see Definition 4.14). We also define the cartesian cubical sets and show that this particular presheaf category satisfies axioms (B1)-(B3) in Coquand's article [5]. This section builds mostly upon the book *Categories for the Working Mathematician* by Saunders Mac Lane [7] together with some results from *Sheaves in Geometry and Logic* by Saunders Mac Lane and Ieke Moerdijk [8]. The proofs in the first of these books are often leave out many details. Therefore, many results in this section build upon results from the book but the details have been worked out by the author. Some results do not occur in the same formulation in Mac Lane but may be found as corollaries of more general results. As the results needed in this section are not very difficult, the ones not in any of the references have been worked out by the author (although other people have most certainly found these results previously). We will indicate when results are expansions of results in any of the references and when the results were worked out by the author. We begin with some definitions that will be needed.

Definition 4.1. *Given a category $\mathcal{C}$ the* ***opposite category*** *$\mathcal{C}^{op}$ has the same objects as $\mathcal{C}$ but the direction of the morphisms is reversed. For a morphism $f \in \mathrm{Hom}_{\mathcal{C}}(X, Y)$ we write f^{op} to denote the corresponding (reversed) morphism in $\mathcal{C}^{op}$.*

Definition 4.2. *Given categories $\mathcal{C}$ and $\mathcal{D}$ and a functor $F : \mathcal{C} \to \mathcal{D}$ between these, the* ***opposite functor*** *$F^{op} : \mathcal{C}^{op} \to \mathcal{D}^{op}$ is given by*

- *$F^{op}(X) = F(X)$ for objects $X \in \mathcal{C}$,*
- *$F^{op}(f^{op}) = F(f)$ for morphisms $f^{op} : Y \to X$.*

Definition 4.3. *Given a category $\mathcal{C}$, a* ***terminal object*** *of $\mathcal{C}$ is an object $T \in \mathcal{C}$ such that there exists exactly one arrow $f : Y \to T$ for every object $Y \in \mathcal{C}$.*

Definition 4.4. *Given two categories $\mathcal{C}$ and $\mathcal{D}$ we denote by $[\mathcal{C}, \mathcal{D}]$ the* ***functor category*** *from $\mathcal{C}$ to $\mathcal{D}$, that is, the category with objects the functors from $\mathcal{C}$ to $\mathcal{D}$ and morphisms the natural transformations between these.*

We will use the arrow $\dot{\to}$ to denote natural transformations and we will sometimes say that a family of maps is **natural** in some object to mean that the family of maps form a natural transformation between some, often implicitly understood, functors.

Definition 4.5. *Given a category $\mathcal{C}$, a functor $F : \mathcal{C}^{op} \to \mathbf{Set}$ is called a* ***presheaf*** *on $\mathcal{C}$. The category $[\mathcal{C}^{op}, \mathbf{Set}]$ of presheaves on $\mathcal{C}$ will be denoted by $\widehat{\mathcal{C}}$.*

Definition 4.6. *For any morphism $f \in \mathrm{Hom}_{\mathcal{C}}(X, Y)$ let f^* denote* ***precomposition*** *by f, i.e. the map that sends any morphism $g \in \mathrm{Hom}_{\mathcal{C}}(Y, Z)$ to $g \circ f \in \mathrm{Hom}_{\mathcal{C}}(X, Z)$. Similarly, let f_* denote* ***postcomposition*** *by f, i.e. the map that sends $h \in \mathrm{Hom}_{\mathcal{C}}(W, X)$ to $f \circ h \in \mathrm{Hom}_{\mathcal{C}}(W, Y)$. The category $\mathcal{C}$ will often be left implicit.*

In this text we say that a category is **small** if both the collection of objects and every hom-set is a set and not a proper class. Similarly, we say that a category is **locally small** to mean that every hom-set is a set and not a proper class. Thus a category that is small is locally small, but not every locally small category is small, e.g. the category **Set** of sets and functions between these.

Definition 4.7. *Given a locally small category $\mathcal{C}$ and an object $X \in \mathcal{C}$ we use the notation $\mathrm{Hom}(X, \_)$ to denote the* ***covariant hom-functor****, i.e. the functor from $\mathcal{C}$ to* **Set** *given by*

- $\mathrm{Hom}(X, \_)(Y) = \mathrm{Hom}_{\mathcal{C}}(X, Y)$ *for objects $Y \in \mathcal{C}$ (note that here we need $\mathcal{C}$ to be locally small to ensure that $\mathrm{Hom}(X, Y)$ is a set),*
- $\mathrm{Hom}(X, \_)(f) = f_* : \mathrm{Hom}_{\mathcal{C}}(X, Y) \to \mathrm{Hom}_{\mathcal{C}}(X, Z)$ *for morphisms $f : Y \to Z$.*

Similarly we use the notation $\mathrm{Hom}(\_, X)$ to denote the ***contravariant hom-functor****, i.e. the functor from $\mathcal{C}^{op}$ to* **Set** *given by*

- $\mathrm{Hom}(\_, X)(Y) = \mathrm{Hom}_{\mathcal{C}}(Y, X)$ *for objects $Y \in \mathcal{C}$,*
- $\mathrm{Hom}(\_, X)(f) = f^* : \mathrm{Hom}_{\mathcal{C}}(Z, X) \to \mathrm{Hom}_{\mathcal{C}}(Y, X)$ *for morphisms $f : Y \to Z$.*

Here we use postcomposition for the map on morphisms by the covariant hom-functor and precomposition for the contravariant hom-functor. If we swap pre- and postcomposition, we can instead define natural transformations between covariant hom-functors using precomposition and between contravariant hom-functors using postcomposition.

Proposition 4.1. *Given a locally small category $\mathcal{C}$, objects $X, Y \in \mathcal{C}$ and a morphism $f : X \to Y$, precomposition by f gives a natural transformation from $\mathrm{Hom}(Y, \_)$ to $\mathrm{Hom}(X, \_)$ and postcomposition by f gives a natural transformation from $\mathrm{Hom}(\_, X)$ to $\mathrm{Hom}(\_, Y)$.*

Proof. Both statements follow from the associativity of composition of morphisms. More specifically, for any morphism $g : Z \to W$ the following diagrams both commute:

$$\begin{tikzcd}
\mathrm{Hom}_{\mathcal{C}}(Y,Z) \arrow[r, "g_*"] \arrow[d, "f^*"'] & \mathrm{Hom}_{\mathcal{C}}(Y,W) \arrow[d, "f^*"] \\
\mathrm{Hom}_{\mathcal{C}}(X,Z) \arrow[r, "g_*"'] & \mathrm{Hom}_{\mathcal{C}}(X,W)
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathrm{Hom}_{\mathcal{C}}(W,X) \arrow[r, "g^*"] \arrow[d, "f_*"'] & \mathrm{Hom}_{\mathcal{C}}(Z,X) \arrow[d, "f_*"] \\
\mathrm{Hom}_{\mathcal{C}}(W,Y) \arrow[r, "g^*"'] & \mathrm{Hom}_{\mathcal{C}}(Z,Y).
\end{tikzcd}$$

□

This proposition does not occur explicitly in Mac Lane [7], but we add it here as it will be useful later.

4.1 The evaluation functor

For the functor category $[\mathcal{C},\mathcal{D}]$ we can define a functor $\mathrm{eval}_X : [\mathcal{C},\mathcal{D}] \to \mathcal{D}$ that evaluates a functor in $[\mathcal{C},\mathcal{D}]$ at the object $X \in \mathcal{C}$. This is an expansion of the details of the definition found in Mac Lane [7, p. 61]. Explicitly, for functors F and G and natural transformation $\alpha : F \dot{\to} G$ we define eval_X as follows:

- $\mathrm{eval}_X(F) = F(X)$,
- $\mathrm{eval}_X(\alpha) = \alpha(X)$.

We verify that this definition respects identity morphisms and compositions.

For the identity $\mathrm{id}_F : F \dot{\to} F$ we have

$$\mathrm{eval}_X(\mathrm{id}_F) = \mathrm{id}_F(X) = \mathrm{id}_{F(X)} = \mathrm{id}_{\mathrm{eval}_X(F)}$$

so the map respects identity morphisms.

For any functors $F, G, H \in [\mathcal{C},\mathcal{D}]$ and any morphisms $\alpha : F \dot{\to} G$ and $\beta : G \dot{\to} H$ we have

$$\mathrm{eval}_X(\beta \circ \alpha) = (\beta \circ \alpha)(X) = \beta(X) \circ \alpha(X) = \mathrm{eval}_X(\beta) \circ \mathrm{eval}_X(\alpha),$$

so the map respects composition of morphisms. Thus it is a functor from $[\mathcal{C},\mathcal{D}]$ to $\mathcal{D}$.

4.2 Adjunctions

When we have two functors $F : \mathcal{C} \to \mathcal{D}$ and $G : \mathcal{D} \to \mathcal{C}$ for some categories $\mathcal{C}$ and $\mathcal{D}$ we may ask if these are related in some way. One way in which two such functors can be related is by being adjoint.

Definition 4.8. *An adjunction from $\mathcal{C}$ to $\mathcal{D}$ is a pair of functors $F : \mathcal{C} \to \mathcal{D}$ and $G : \mathcal{D} \to \mathcal{C}$ together with a family of bijections*

$$\varphi : \mathrm{Hom}_{\mathcal{D}}(FX, Y) \cong \mathrm{Hom}_{\mathcal{C}}(X, GY)$$

for all $X \in \mathcal{C}$ and $Y \in \mathcal{D}$ which is natural in both X and Y, i.e. for any morphisms $f : X' \to X$ and $g : Y \to Y'$ the following diagrams commute:

$$\begin{tikzcd}
\mathrm{Hom}_{\mathcal{D}}(FX,Y) \arrow[r, "F(f)^*"] \arrow[d, "\varphi"'] & \mathrm{Hom}_{\mathcal{D}}(FX',Y) \arrow[d, "\varphi"] \\
\mathrm{Hom}_{\mathcal{C}}(X,GY) \arrow[r, "f^*"'] & \mathrm{Hom}_{\mathcal{C}}(X',GY)
\end{tikzcd}$$

and

$$\begin{tikzcd}
\mathrm{Hom}_{\mathcal{D}}(FX,Y) \arrow[r, "g_*"] \arrow[d, "\varphi"'] & \mathrm{Hom}_{\mathcal{D}}(FX,Y') \arrow[d, "\varphi"] \\
\mathrm{Hom}_{\mathcal{C}}(X,GY) \arrow[r, "G(g)"'] & \mathrm{Hom}_{\mathcal{C}}(X,GY').
\end{tikzcd}$$

4.3 Products

In many different settings of mathematics we encounter some form of product of objects, for example the cartesian product of sets, groups and rings. Such products come with natural projections on the components. This concept can be generalized to arbitrary categories.

Definition 4.9. *Let $\mathcal{C}$ be a category. For any two objects $X_1, X_2 \in \mathcal{C}$ the* ***product*** *$X_1 \times X_2$ of X_1 and X_2 (if it exists) is an object in $\mathcal{C}$ together with a pair of morphisms $\pi_1 : X_1 \times X_2 \to X_1$ and $\pi_2 : X_1 \times X_2 \to X_2$ such that for any object $Y \in \mathcal{C}$ and any pair of morphisms $f_1 : Y \to X_1$ and $f_2 : Y \to X_2$ there exists a unique morphism $f : Y \to X_1 \times X_2$ such that the following diagram commutes:*

$$\begin{tikzcd}
 & Y \arrow[dl, "f_1"'] \arrow[d, dashed, "f"] \arrow[dr, "f_2"] & \\
X_1 & X_1 \times X_2 \arrow[l, "\pi_1"] \arrow[r, "\pi_2"'] & X_2.
\end{tikzcd}$$

The unique morphism f will often be denoted by (f_1, f_2).

When the product $X_1 \times X_2$ exists for every pair of objects $X_1, X_2 \in \mathcal{C}$ we say that $\mathcal{C}$ has binary products.

For the special case when Y is itself a product we will use the notation $f_1 \times f_2$ for the unique morphism making the following diagram commute:

$$\begin{tikzcd}
Y_1 \arrow[d, "f_1"'] & Y_1 \times Y_2 \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[d, dashed, "f_1 \times f_2"] & Y_2 \arrow[d, "f_2"] \\
X_1 & X_1 \times X_2 \arrow[l, "\pi_1"] \arrow[r, "\pi_2"'] & X_2.
\end{tikzcd}$$

We observe that the morphism $\mathrm{id}_{X_1 \times X_2}$ makes the following diagram commute:

$$\begin{tikzcd}
X_1 \arrow[d, "\mathrm{id}_{X_1}"'] & X_1 \times X_2 \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[d, "\mathrm{id}_{X_1 \times X_2}"] & X_2 \arrow[d, "\mathrm{id}_{X_2}"] \\
X_1 & X_1 \times X_2 \arrow[l, "\pi_1"] \arrow[r, "\pi_2"'] & X_2.
\end{tikzcd}$$

Since $\mathrm{id}_{X_1} \times \mathrm{id}_{X_2}$ is the unique morphism making this diagram commute it follows that $\mathrm{id}_{X_1 \times X_2} = \mathrm{id}_{X_1} \times \mathrm{id}_{X_2}$.

Moreover, the diagram below commutes:

$$\begin{tikzcd}
Z_1 \arrow[d, "f_1"'] & Z_1 \times Z_2 \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[d, "f_1 \times f_2"] & Z_2 \arrow[d, "f_2"] \\
Y_1 \arrow[d, "g_1"'] & Y_1 \times Y_2 \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[d, "g_1 \times g_2"] & Y_2 \arrow[d, "g_2"] \\
X_1 & X_1 \times X_2 \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] & X_2.
\end{tikzcd}$$

Therefore $(g_1 \times g_2) \circ (f_1 \times f_2)$ makes the following diagram commute:

$$\begin{tikzcd}
Z_1 \arrow[d, "g_1 \circ f_1"'] & Z_1 \times Z_2 \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[d, "(g_1 \times g_2) \circ (f_1 \times f_2)"] & Z_2 \arrow[d, "g_2 \circ f_2"] \\
X_1 & X_1 \times X_2 \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] & X_2.
\end{tikzcd}$$

Since $(g_1 \circ f_1) \times (g_2 \circ f_2)$ is the unique morphism making this diagram commute it follows that $(g_1 \times g_2) \circ (f_1 \times f_2) = (g_1 \circ f_1) \times (g_2 \circ f_2)$.

From these two observations we note that, given an object $X \in \mathcal{C}$ such that all binary products with X exist we can define an endofunctor that gives the product with X.

Definition 4.10. *Given a category $\mathcal{C}$ and an object $X \in \mathcal{C}$ such that all binary products with X exist we define the functor $\_ \times X : \mathcal{C} \to \mathcal{C}$ by*

- $(\_ \times X)(Y) = Y \times X$ *for objects* $Y \in \mathcal{C}$,
- $(\_ \times X)(f) = f \times \mathrm{id}_X$ *for morphisms* $f : Y \to Z$.

We verify that this definition respects identity morphisms and compositions of morphisms. For the identity id_Y we have

$$(\_ \times X)(\mathrm{id}_Y) = \mathrm{id}_Y \times \mathrm{id}_X = \mathrm{id}_{Y \times X} = \mathrm{id}_{(\_ \times X)(Y)}$$

and for any morphisms $f : Y \to Z$ and $g : Z \to W$ we have

$$(\_ \times X)(g \circ f) = (g \circ f) \times \mathrm{id}_X = (g \times \mathrm{id}_X) \circ (f \times \mathrm{id}_X) = (\_ \times X)(g) \circ (\_ \times X)(f).$$

If a category $\mathcal{D}$ has binary products then we can also define the product of two functors $F, G : \mathcal{C} \to \mathcal{D}$ by pointwise product.

Definition 4.11. *Let $\mathcal{D}$ be a category with binary products. Given a category $\mathcal{C}$ and functors $F, G \in [\mathcal{C}, \mathcal{D}]$ we define the product functor $F \times G$ by*

- $(F \times G)(X) = F(X) \times G(X)$ *for any* $X \in \mathcal{C}$,
- $(F \times G)(f) = F(f) \times G(f)$ *for any* $f \in \mathrm{Hom}_{\mathcal{C}}(X, Y)$.

The projections, which we denote by Π_1 and Π_2 to distinguish them from the projections π_1 and π_2 on the products in $\mathcal{D}$, are defined by

- $\Pi_1 : F \times G \dot{\to} F$ *is the natural transformation given by*
$$\Pi_1(X) := \pi_1 : F(X) \times G(X) \to F(X),$$
- $\Pi_2 : F \times G \dot{\to} G$ *is the natural transformation given by*
$$\Pi_2(X) := \pi_2 : F(X) \times G(X) \to G(X).$$

We now show that this satisfies the definition of binary products. First, we verify that $F \times G$ is a functor.

For the identity $\mathrm{id}_X : X \to X$ we have

$$(F\times G)(\mathrm{id}_X) = F(\mathrm{id}_X)\times G(\mathrm{id}_X) = \mathrm{id}_{F(X)} \times \mathrm{id}_{G(X)} = \mathrm{id}_{F(X)\times G(X)} = \mathrm{id}_{(F\times G)(X)}$$

and for morphisms $f : X \to Y$ and $g : Y \to Z$ we have

$$\begin{aligned}(F \times G)(g \circ f) &= F(g \circ f) \times G(g \circ f)\\ &= (F(g) \circ F(f)) \times (G(g) \circ G(f))\\ &= (F(g) \times G(g)) \circ (F(f) \times G(f))\\ &= (F \times G)(g) \circ (F \times G)(f).\end{aligned}$$

So the pointwise product of two functors is indeed a functor.

We now verify that $F \times G$ is a product in $[\mathcal{C}, \mathcal{D}]$. Given a morphism $f : X \to Y$, by the definition of Π_1 and Π_2, the following diagram commutes:

$$\begin{tikzcd}
F(X) \arrow[d, "F(f)"'] & F(X) \times G(X) \arrow[l, "\Pi_1(X)"'] \arrow[r, "\Pi_2(X)"] \arrow[d, "F(f)\times G(f)"] & G(X) \arrow[d, "G(f)"] \\
F(Y) & F(Y) \times G(Y) \arrow[l, "\Pi_1(Y)"] \arrow[r, "\Pi_2(Y)"'] & G(Y).
\end{tikzcd}$$

Thus Π_1 and Π_2 are natural transformations.

Given a functor $H \in [\mathcal{D}, \mathcal{C}]$ and natural transformations $\alpha_1 : H \dot{\to} F$ and $\alpha_2 : H \dot{\to} G$, for any object $X \in \mathcal{C}$ the morphism

$$(\alpha_1(X), \alpha_2(X)) : H(X) \to F(X) \times G(X)$$

is the unique morphism making the following diagram commute:

$$\begin{tikzcd}
 & H(X) \arrow[dl, "\alpha_1(X)"'] \arrow[d, "(\alpha_1(X){,}\alpha_2(X))"] \arrow[dr, "\alpha_2(X)"] & \\
F(X) & F(X) \times G(X) \arrow[l, "\Pi_1(X)"] \arrow[r, "\Pi_2(X)"'] & G(X).
\end{tikzcd}$$

It follows that there is a unique natural transformation $(\alpha_1, \alpha_2) : H \dot{\to} F \times G$ making the diagram below commute,

$$\begin{tikzcd}
 & H \arrow[dl, "\alpha_1"'] \arrow[d, "(\alpha_1{,}\alpha_2)"] \arrow[dr, "\alpha_2"] & \\
F & F \times G \arrow[l, "\Pi_1"] \arrow[r, "\Pi_2"'] & G
\end{tikzcd}$$

namely the natural transformation defined by

$$(\alpha_1, \alpha_2)(X) = (\alpha_1(X), \alpha_2(X)).$$

This is a natural because for any morphism $f : X \to Y$ we have

$$\begin{aligned}
\Pi_1(Y) \circ (F(f) \times G(f)) \circ (\alpha_1(X), \alpha_2(X)) &= F(f) \circ \Pi_1(X) \circ (\alpha_1(X), \alpha_2(X)) \\
&= F(f) \circ \alpha_1(X) \\
&= \alpha_1(Y) \circ H(f) \\
&= \Pi_1(Y) \circ (\alpha_1(Y), \alpha_2(Y)) \circ H(f)
\end{aligned}$$

and

$$\begin{aligned}
\Pi_2(Y) \circ (F(f) \times G(f)) \circ (\alpha_1(X), \alpha_2(X)) &= G(f) \circ \Pi_2(X) \circ (\alpha_1(X), \alpha_2(X)) \\
&= G(f) \circ \alpha_2(X) \\
&= \alpha_2(Y) \circ H(f) \\
&= \Pi_2(Y) \circ (\alpha_1(Y), \alpha_2(Y)) \circ H(f).
\end{aligned}$$

Since the morphisms $(F(f) \times G(f)) \circ (\alpha_1(X), \alpha_2(X))$ and $(\alpha_1(Y), \alpha_2(Y)) \circ H(f)$ agree on both projections it follows that

$$(F(f) \times G(f)) \circ (\alpha_1(X), \alpha_2(X)) = (\alpha_1(Y), \alpha_2(Y)) \circ H(f)$$

and thus (α_1, α_2) is a natural transformation from H to $F \times G$. The proofs of the properties of products in this subsection were done by the author as these were not found in Mac Lane [7] but were not difficult to work out the details of.

4.4 Exponentials

In the category **Set**, the functions $\mathrm{Hom}_{\mathbf{Set}}(Y, Z)$ between two sets Y and Z form a set which we denote by the 'exponential' Z^Y. So the exponential is an object in the category that represents a hom-set between two objects. We may therefore ask if this can be generalized to arbitrary categories, i.e. for two objects in the category, is there an object representing the hom-set between them? To answer this we begin by observations of some properties of the set Z^Y of functions from Y to Z.

Given a function $f \in Z^Y$ we can evaluate it at a point $y \in Y$ to obtain an element $f(y) \in Z$, i.e. we have a function $\mathbf{e} : Z^Y \times Y \to Z$ defined by $\mathbf{e}(f, y) = f(y)$. Now, given any function $g : X \times Y \to Z$ in two variables, it can either be seen as a function $X \times Y \to Z$ or as a function $X \to Z^Y$. For if we fix an element $x \in X$, we get the function $\lambda g(x) : Y \to Z$ defined by

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

The relation between $\lambda g : X \to Z^Y$ and the evaluation $\mathbf{e} : Z^Y \times Y \to Z$ can be described as the commutativity of the diagram

$$\begin{tikzcd}
X \times Y \arrow[d, "\lambda g \times \mathrm{id}_Y"'] \arrow[dr, "g"] & \\
Z^Y \times Y \arrow[r, "\mathbf{e}"'] & Z.
\end{tikzcd}$$

In fact, λg is the unique function making this diagram commute.

From this observation we define exponential objects for an arbitrary category $\mathcal{C}$. This definition is found in Mac Lane and Moerdijk [8, Section I.6] but here we write out all the details.

Definition 4.12. *Let Y and Z be objects in a category $\mathcal{C}$ such that all binary products with Y exist. An **exponential object** is an object $Z^Y \in \mathcal{C}$ together with a morphism $\mathbf{e} : Z^Y \times Y \to Z$ such that for any object X and morphism $g : X \times Y \to Z$ there is a unique morphism $\lambda g : X \to Z^Y$ such that the following diagram commutes:*

$$\begin{tikzcd}
X \times Y \arrow[d, dashed, "\lambda g \times \mathrm{id}_Y"'] \arrow[dr, "g"] & \\
Z^Y \times Y \arrow[r, "\mathbf{e}"'] & Z.
\end{tikzcd}$$

Since the morphism λg is required to be unique there is a one-to-one correspondence

$$\mathrm{Hom}_{\mathcal{C}}(X \times Y, Z) \cong \mathrm{Hom}_{\mathcal{C}}(X, Z^Y).$$

We will now define the exponential functor and show that this bijection defines an adjunction.

Definition 4.13. *Let Y be an object in a category $\mathcal{C}$ such that Z^Y exists for all objects Z. Define the **exponential functor** $\_^Y : \mathcal{C} \to \mathcal{C}$ by*

- *$Z \mapsto Z^Y$ for any object $Z \in \mathcal{C}$ and*
- *$f \mapsto f^Y := \lambda(f \circ \mathbf{e})$ for any morphism $f : X \to Z$, i.e. f^Y is the unique morphism making the following diagram commute:*

$$\begin{tikzcd}
X^Y \times Y \arrow[r, "\mathbf{e}"] \arrow[d, "f^Y \times \mathrm{id}_Y"'] & X \arrow[d, "f"] \\
Z^Y \times Y \arrow[r, "\mathbf{e}"] & Z.
\end{tikzcd}$$

We verify that this map respects identity morphisms and composition of morphisms.

Since $\mathrm{id}_{X^Y} \times \mathrm{id}_Y = \mathrm{id}_{X^Y \times Y}$, the diagram below commutes

$$\begin{tikzcd}
X^Y \times Y \arrow[r, "\mathbf{e}"] \arrow[d, "\mathrm{id}_{X^Y} \times \mathrm{id}_Y"'] & X \arrow[d, "\mathrm{id}_X"] \\
X^Y \times Y \arrow[r, "\mathbf{e}"] & X.
\end{tikzcd}$$

It thus follows that $(\mathrm{id}_X)^Y = \mathrm{id}_{X^Y}$.

Given two morphisms $f : X \to Z$ and $g : Z \to W$, the following diagram commutes:

$$\begin{tikzcd}
X^Y \times Y \arrow[r, "\mathbf{e}"] \arrow[d, "f^Y \times \mathrm{id}_Y"'] & X \arrow[d, "f"] \\
Z^Y \times Y \arrow[r, "\mathbf{e}"] \arrow[d, "g^Y \times \mathrm{id}_Y"'] & Z \arrow[d, "g"] \\
W^Y \times Y \arrow[r, "\mathbf{e}"] & W
\end{tikzcd}$$

since both squares commute by the definition of f^Y and g^Y. It follows that $(g^Y \times \mathrm{id}_Y) \circ (f^Y \times \mathrm{id}_Y)$ makes the diagram below commute

$$\begin{tikzcd}
X^Y \times Y \arrow[r, "\mathbf{e}"] \arrow[d, "(g^Y \times \mathrm{id}_Y) \circ (f^Y \times \mathrm{id}_Y)"'] & X \arrow[d, "g \circ f"] \\
W^Y \times Y \arrow[r, "\mathbf{e}"] & W.
\end{tikzcd}$$

Since $(g^Y \times \mathrm{id}_Y) \circ (f^Y \times \mathrm{id}_Y) = (g^Y \circ f^Y) \times \mathrm{id}_Y$ it follows that $(g \circ f)^Y = g^Y \circ f^Y$. So $\_^Y$ is indeed a functor from $\mathcal{C}$ to $\mathcal{C}$.

Now, as noted above, we have a bijection

$$\varphi : \mathrm{Hom}_{\mathcal{C}}(X \times Y, Z) \cong \mathrm{Hom}_{\mathcal{C}}(X, Z^Y)$$

for all objects $X, Z \in \mathcal{C}$, defined by $\varphi(g) = \lambda g$. Seen as a candidate for an adjunction from $\_ \times Y$ to $\_^Y$ we may ask if this bijection is natural in X and Z.

For naturality in X we need to check that for any morphism $f : W \to X$ the diagram below commutes:

$$\begin{tikzcd}
\mathrm{Hom}_{\mathcal{C}}(X \times Y, Z) \arrow[r, "(f \times \mathrm{id}_Y)^*"] \arrow[d, "\varphi"'] & \mathrm{Hom}_{\mathcal{C}}(W \times Y, Z) \arrow[d, "\varphi"] \\
\mathrm{Hom}_{\mathcal{C}}(X, Z^Y) \arrow[r, "f^*"'] & \mathrm{Hom}_{\mathcal{C}}(W, Z^Y).
\end{tikzcd}$$

For any morphism $g \in \mathrm{Hom}_{\mathcal{C}}(X \times Y, Z)$ the following diagram commutes:

$$\begin{tikzcd}
W \times Y \arrow[d, "f \times \mathrm{id}_Y"'] & \\
X \times Y \arrow[d, "\lambda g \times \mathrm{id}_Y"'] \arrow[dr, "g"] & \\
Z^Y \times Y \arrow[r, "\mathbf{e}"'] & Z
\end{tikzcd}$$

by the definition of λg. Thus the diagram below commutes:

$$\begin{tikzcd}
W \times Y \arrow[d, "(\lambda g \times \mathrm{id}_Y) \circ (f \times \mathrm{id}_Y)"'] \arrow[dr, "g \circ (f \times \mathrm{id}_Y)"] & \\
Z^Y \times Y \arrow[r, "\mathbf{e}"'] & Z.
\end{tikzcd}$$

Since $(\lambda g \times \mathrm{id}_Y) \circ (f \times \mathrm{id}_Y) = (\lambda g \circ f) \times \mathrm{id}_Y$ it follows that $\lambda(g \circ (f \times \mathrm{id}_Y)) = \lambda g \circ f$. Hence

$$\varphi \circ (f \times \mathrm{id}_Y)^*(g) = f^* \circ \varphi(g),$$

i.e. φ is natural in X.

For naturality in Z we need to check that for any morphism $f : Z \to W$ the following diagram commutes:

$$\begin{tikzcd}
\mathrm{Hom}_{\mathcal{C}}(X \times Y, Z) \arrow[r, "f_*"] \arrow[d, "\varphi"'] & \mathrm{Hom}_{\mathcal{C}}(X \times Y, W) \arrow[d, "\varphi"] \\
\mathrm{Hom}_{\mathcal{C}}(X, Z^Y) \arrow[r, "(f^Y)_*"'] & \mathrm{Hom}_{\mathcal{C}}(X, W^Y).
\end{tikzcd}$$

For any morphism $g \in \mathrm{Hom}_{\mathcal{C}}(X \times Y, Z)$, we have the following commutative diagram:

$$\begin{tikzcd}
X \times Y \arrow[d, "\lambda g \times \mathrm{id}_Y"'] \arrow[dr, "g"] & \\
Z^Y \times Y \arrow[r, "\mathbf{e}"'] \arrow[d, "f^Y \times \mathrm{id}_Y"'] & Z \arrow[d, "f"] \\
W^Y \times Y \arrow[r, "\mathbf{e}"'] & W.
\end{tikzcd}$$

This commutes since the top half commutes by the definition of λg and the bottom half commutes by the definition of f^Y. Thus the following diagram commutes:

$$\begin{tikzcd}
X \times Y \arrow[d, "(f^Y \times \mathrm{id}_Y) \circ (\lambda g \times \mathrm{id}_Y)"'] \arrow[dr, "f \circ g"] & \\
W^Y \times Y \arrow[r, "\mathbf{e}"'] & W.
\end{tikzcd}$$

Since $(f^Y \times \mathrm{id}_Y) \circ (\lambda g \times \mathrm{id}_Y) = (f^Y \circ \lambda g) \times \mathrm{id}_Y$ it follows that $\lambda(f \circ g) = f^Y \circ \lambda g$. Hence

$$\varphi \circ f_*(g) = (f^Y)_* \circ \varphi(g),$$

i.e. φ is natural in Z.

It thus follows that the exponential functor $\_^Y$ is right adjoint to the product functor $\_ \times Y$. One may now ask if the exponential functor is also a left adjoint, i.e. does it have a right adjoint? This property is given its own name and turns out to be useful for constructing internal universes in presheaf models of HoTT [9].

Definition 4.14. *Let Y be an object in a category $\mathcal{C}$ such that Z^Y exists for all objects Z. We say that Y is* ***tiny*** *if the exponential functor $\_^Y : \mathcal{C} \to \mathcal{C}$ has a right adjoint.*

In order to give conditions under which presheaves are tiny we will need a basic result in category theory called the Yoneda lemma.

4.5 The Yoneda lemma

The Yoneda lemma is an important result in category theory. It is one of the main results we will use to show that the interval in cartesian cubical sets is tiny. We begin by defining the Yoneda embedding.

Definition 4.15. *Let $\mathcal{C}$ be a locally small category. The* ***Yoneda embedding*** *is the functor* $\mathrm{yon} : \mathcal{C} \to \widehat{\mathcal{C}}$ *that sends an object $X \in \mathcal{C}$ to the contravariant hom-functor* $\mathrm{Hom}(\_, X)$ *and a morphism $f : X \to Y$ to the natural transformation $f_* : \mathrm{Hom}(\_, X) \dot{\to} \mathrm{Hom}(\_, Y)$ given by postcomposition by f as shown in Proposition 4.1.*

We verify that the Yoneda embedding respects identity morphisms and compositions of morphisms.

For the identity $\mathrm{id}_X : X \to X$, the Yoneda embedding is, by the property of identity morphisms, the natural transformation $\mathrm{Hom}(\_, X) \dot{\to} \mathrm{Hom}(\_, X)$ that for every object $Y \in \mathcal{C}$ sends a morphism $g \in \mathrm{Hom}_{\mathcal{C}}(Y, X)$ to itself. Hence $\mathrm{yon}(\mathrm{id}_X) = \mathrm{id}_{\mathrm{yon}(X)}$.

The fact that $\mathrm{yon}(f \circ g) = \mathrm{yon}(f) \circ \mathrm{yon}(g)$ follows from the associativity of composition of morphisms.

Theorem 4.2 (The Yoneda Lemma)**.** *For any locally small category $\mathcal{C}$, functor $F : \mathcal{C}^{op} \to \mathbf{Set}$ and object $X \in \mathcal{C}$ there is a bijection of sets*

$$\mathrm{Hom}_{\widehat{\mathcal{C}}}(\mathrm{yon}(X), F) \cong F(X)$$

that is natural in both F and X.

Proof. Consider the map

$$\begin{aligned} \Phi : \mathrm{Hom}_{\widehat{\mathcal{C}}}(\mathrm{yon}(X), F) &\to F(X) \\ \alpha &\mapsto \alpha(X)(\mathrm{id}_X) \end{aligned}$$

and set for the moment $u := \alpha(X)(\mathrm{id}_X)$. For any morphism $f : Y \to X$ the diagram below commutes since α is a natural transformation, so chasing the identity morphism id_X around the diagram we have:

$$\begin{tikzcd}
\mathrm{Hom}_{\mathcal{C}}(X,X) \arrow[rrr, "\mathrm{yon}(X)(f)"] \arrow[ddd, "\alpha(X)"'] & & & \mathrm{Hom}_{\mathcal{C}}(Y,X) \arrow[ddd, "\alpha(Y)"] \\
& \mathrm{id}_X \arrow[r, mapsto] \arrow[d, mapsto] & \mathrm{id}_X \circ f = f \arrow[d, mapsto] & \\
& u \arrow[r, mapsto] & F(f)(u) = \alpha(Y)(f) & \\
F(X) \arrow[rrr, "F(f)"'] & & & F(Y).
\end{tikzcd}$$

We see that α is completely determined by u.

On the other hand, for any element $v \in F(X)$, define the natural transformation $\Psi(v) : \mathrm{yon}(X) \dot{\to} F$ by $\Psi(v)(Y)(f) = F(f)(v)$. This is a natural transformation since for any morphism $g : Z \to Y$ we have

$$\begin{aligned}(F(g) \circ \Psi(v)(Y))(f) &= F(g)(\Psi(v)(Y)(f)) = F(g)(F(f)(v)) \\ &= (F(g) \circ F(f))(v) = F(f \circ g)(v) \\ &= \Psi(v)(Z)(f \circ g) = (\Psi(v)(Z) \circ \mathrm{yon}(X)(g))(f).\end{aligned}$$

So Ψ is a map from $F(X)$ to $\mathrm{Hom}_{\widehat{\mathcal{C}}}(\mathrm{yon}(X), F)$. We now show that Ψ is a two-sided inverse to Φ.

First, for any element $v \in F(X)$ we have

$$(\Phi \circ \Psi)(v) = \Phi(\Psi(v)) = \Psi(v)(X)(\mathrm{id}_X) = F(\mathrm{id}_X)(v) = \mathrm{id}_{F(X)}(v) = v,$$

and thus Ψ is a right-sided inverse to Φ.

Second, for any natural transformation $\alpha : \mathrm{yon}(X) \dot{\to} F$ we have

$$\begin{aligned}(\Psi \circ \Phi)(\alpha)(Y)(f) &= \Psi(\Phi(\alpha))(Y)(f) \\ &= \Psi(\alpha(X)(\mathrm{id}_X))(Y)(f) \\ &= F(f)(\alpha(X)(\mathrm{id}_X)) \\ &= \alpha(Y)(f),\end{aligned}$$

and thus Ψ is a left-sided inverse to Φ. It follows that Φ defines a bijection between $\mathrm{Hom}_{\widehat{\mathcal{C}}}(\mathrm{yon}(X), F)$ and $F(X)$.

In order to show that Φ is natural in X we consider the functors $\mathrm{Hom}_{\widehat{\mathcal{C}}}(\mathrm{yon}(\_), F)$ and F from $\mathcal{C}^{op}$ to **Set** where

$$\mathrm{Hom}_{\widehat{\mathcal{C}}}(\mathrm{yon}(\_), F) = \mathrm{yon}(F) \circ \mathrm{yon}^{op}$$

i.e. for any $f : Y \to X$ the element $\alpha \in \mathrm{Hom}_{\widehat{\mathcal{C}}}(\mathrm{yon}(X), F)$ is mapped to the element $\beta \in \mathrm{Hom}_{\widehat{\mathcal{C}}}(\mathrm{yon}(Y), F)$ defined by

$$\beta(Z)(g) = \alpha(Z)(f \circ g).$$

Then we have

$$\begin{aligned}(\Phi(F, Y) \circ \mathrm{Hom}_{\widehat{\mathcal{C}}}(\mathrm{yon}(\_), F)(f))(\alpha) &= \Phi(F, Y)(\beta) = \beta(Y)(\mathrm{id}_Y) \\ &= \alpha(Y)(f \circ \mathrm{id}_Y) = \alpha(Y)(f) \\ &= F(f)(\alpha(X)(\mathrm{id}_X)) = F(f)(\Phi(F, X)(\alpha)) \\ &= (F(f) \circ \Phi(F, X))(\alpha),\end{aligned}$$

so Φ is natural in X.

Now, consider instead the functors eval_X and $\mathrm{Hom}(\mathrm{yon}(X), \_)$ from $\widehat{\mathcal{C}}$ to **Set**. We have for any natural transformation $\gamma : F \dot{\to} G$ between presheaves $F, G \in \widehat{\mathcal{C}}$

$$\begin{aligned}(\Phi(G, X) \circ (\mathrm{Hom}(\mathrm{yon}(X), \_)(\gamma))(\alpha) &= \Phi(G, X)(\gamma \circ \alpha) = (\gamma \circ \alpha)(X)(\mathrm{id}_X) \\ &= \gamma(X)(\alpha(X)(\mathrm{id}_X)) = \gamma(X)(\Phi(F, X)(\alpha)) \\ &= (\mathrm{eval}_X(\gamma) \circ \Phi(F, X))(\alpha),\end{aligned}$$

so Φ is natural in F. □

This theorem is found in Mac Lane [7, p. 61] but the all details of the proof are left out. The details here have been worked out by the author.

So far, all previous results in this section were already formalized in the category theory section of the UniMath library at the start of this thesis. All formalizations done by the author are of results from here on. We will indicate with footnotes when a proposition or theorem was formalized in connection with this thesis and give a link to where the code can be found.

A result that will be needed later is that the Yoneda embedding commutes with binary products. We will now show this.

Recall that if the target category has binary products, then we can define the product of any functors into that category as the pointwise product (see Definition 4.11).

Theorem 4.3. *Let $\mathcal{C}$ be a locally small category with binary products. For any two objects $X, Y \in \mathcal{C}$ the functors* $\mathrm{yon}(X \times Y)$ *and* $\mathrm{yon}(X) \times \mathrm{yon}(Y)$ *are naturally isomorphic.*[1]

Proof. For any object $Z \in \mathcal{C}$ define the map $\alpha(Z) : \mathrm{Hom}_{\mathcal{C}}(Z, X \times Y) \to \mathrm{Hom}_{\mathcal{C}}(Z, X) \times \mathrm{Hom}_{\mathcal{C}}(Z, Y)$ by

$$\alpha(Z)(f) = (\pi_1 \circ f, \pi_2 \circ f).$$

By the definition of binary products, this is a bijection (which is an isomorphism in **Set**).

Given a morphism $g : W \to Z$, consider the diagram:

$$\begin{tikzcd}
\mathrm{Hom}_{\mathcal{C}}(Z, X \times Y) \arrow[r, "g^*"] \arrow[d, "\alpha(Z)"'] & \mathrm{Hom}_{\mathcal{C}}(W, X \times Y) \arrow[d, "\alpha(W)"] \\
\mathrm{Hom}_{\mathcal{C}}(Z, X) \times \mathrm{Hom}_{\mathcal{C}}(Z, Y) \arrow[r, "(g,g)^*"'] & \mathrm{Hom}_{\mathcal{C}}(W, X) \times \mathrm{Hom}_{\mathcal{C}}(W, Y).
\end{tikzcd}$$

Since $(\pi_1 \circ f, \pi_2 \circ f) \circ (g, g) = (\pi_1 \circ f \circ g, \pi_2 \circ f \circ g)$ for any morphism $f \in \mathrm{Hom}_{\mathcal{C}}(Z, X \times Y)$, the diagram commutes. Thus α is a natural isomorphism from $\mathrm{yon}(X \times Y)$ to $\mathrm{yon}(X) \times \mathrm{yon}(Y)$. $\square$

This theorem occurs as part of the proof in Licata et al. [9, p. 10] that the interval object is tiny but here we have written all the details out. These were worked out by the author.

4.6 Exponentiation by representable objects

Let $\mathcal{C}$ be a small category. By Proposition 1 in Mac Lane and Moerdijk [8, p. 46] the category $[\mathcal{C}^{op}, \mathbf{Set}]$ of presheaves on $\mathcal{C}$ has all binary products and exponential objects. Thus for any object $X \in \mathcal{C}$ the exponential functor $\_^{\mathrm{yon}(X)}$:

[1]The formalization of this theorem can be found at `https://github.com/UniMath/UniMath/blob/0e1bd59/UniMath/CategoryTheory/YonedaBinproducts.v#L89-L97`

$\widehat{\mathcal{C}} \to \widehat{\mathcal{C}}$ exists. We will now show that if $\mathcal{C}$ has binary products this functor has a right adjoint.

In order to show that the exponential functor $\_^{\mathrm{yon}(X)}$ has a right adjoint, we show that it is naturally isomorphic to the precomposition functor $((\_ \times X)^{op})^*$, because this functor does have a right adjoint.

First we prove a technical result that gives a sufficient condition for constructing an isomorphism of endofunctors on presheaf categories. We will then use this result to construct the isomorphism mentioned above. This theorem has not been found in any of Mac Lane [7] or Mac Lane and Moerdijk [8] but is used implicitly in the proof in Licata et al. [9, p. 10]. The details of the proof were worked out by the author.

Proposition 4.4. *Let $\mathcal{C}$ be a category and let $\Phi, \Psi : \widehat{\mathcal{C}} \to \widehat{\mathcal{C}}$ be functors. If there is a bijection of sets*

$$\Phi(F)(X) \cong \Psi(F)(X)$$

for all $F \in \widehat{\mathcal{C}}$ and all $X \in \mathcal{C}$ which is natural in both F and X then the functors Φ and Ψ are naturally isomorphic.[2]

Proof. Suppose there is a bijection

$$\alpha(F, X) : \Phi(F)(X) \cong \Psi(F)(X)$$

that is natural in both F and X. The assumption that α is natural in X is exactly that $\alpha(F) : \Phi(F) \to \Psi(F)$, defined by $\alpha(F)(X) = \alpha(F, X)$, is a natural transformation. Since $\alpha(F, X)$ is a bijection for every $X \in \mathcal{C}$, i.e. an isomorphism in **Set**, it follows that $\alpha(F)$ is a natural isomorphism between the functors $\Phi(F)$ and $\Psi(F)$.

The assumption that α is natural in F is exactly that $\alpha(\_, X) : \mathrm{eval}_X \circ \Phi \to \mathrm{eval}_X \circ \Psi$, defined by $\alpha(\_, X)(F) = \alpha(F, X)$ is a natural transformation.

Now, given a morphism $f : F \dot{\to} G$ of presheaves, consider the diagram

$$\begin{tikzcd}
\Phi(F) \arrow[r, "\Phi(f)"] \arrow[d, "\alpha(F)"'] & \Phi(G) \arrow[d, "\alpha(G)"] \\
\Psi(F) \arrow[r, "\Psi(f)"'] & \Psi(G).
\end{tikzcd}$$

By the assumption that α is natural in F we have

$$\begin{aligned}
(\Psi(f) \circ \alpha(F))(X) &= \Psi(f)(X) \circ \alpha(F)(X) \\
&= \Psi(f)(X) \circ \alpha(\_, X)(F) \\
&= \alpha(\_, X)(G) \circ \Phi(f)(X) \\
&= \alpha(G, X) \circ \Phi(f)(X) \\
&= (\alpha(G) \circ \Phi(f))(X)
\end{aligned}$$

[2]The formalization of this proposition was done by Anders Mörtberg in connection with this thesis and can be found at `https://github.com/UniMath/UniMath/blob/0e1bd59/UniMath/CategoryTheory/Presheaf.v#L389-L419`

and thus $\Psi(f) \circ \alpha(F) = \alpha(G) \circ \Phi(f)$ so the diagram commutes. Therefore α defines a natural transformation $\Phi \dot\to \Psi$.

Since $\alpha(F)$ is an isomorphism for every $F \in \widehat{\mathcal{C}}$ it follows that α is a natural isomorphism from Φ to Ψ. $\square$

We are now finally ready to prove our main theorem which gives sufficient conditions for the exponential functor of a representable presheaf to have a right adjoint. This is an expansion of the details of the proof found in Licata et al. [9, p. 10]. First we show that the exponential functor is naturally isomorphic to the precomposition functor mentioned in the beginning of this section.

Theorem 4.5. *Let $\mathcal{C}$ be a locally small category with binary products. For any object $X \in \mathcal{C}$, the functors $\_^{\mathrm{yon}(X)}$ and $((\_ \times X)^{op})^*$ are naturally isomorphic.*[3]

Proof. By the Yoneda lemma we have a bijection

$$F^{\mathrm{yon}(X)}(Y) \cong \mathrm{Hom}_{\widehat{\mathcal{C}}}(\mathrm{yon}(Y), F^{\mathrm{yon}(X)}) \tag{1}$$

of sets that is natural in both F and Y.

By the definition of exponentials, the functor $\_^{\mathrm{yon}(X)}$ is right adjoint to the functor $\_ \times \mathrm{yon}(X)$. It follows that there is a bijection

$$\mathrm{Hom}_{\widehat{\mathcal{C}}}(\mathrm{yon}(Y), F^{\mathrm{yon}(X)}) \cong \mathrm{Hom}_{\widehat{\mathcal{C}}}(\mathrm{yon}(Y) \times \mathrm{yon}(X), F) \tag{2}$$

of sets that is natural in both Y and F.

By Theorem 4.3 there is an isomorphism

$$\alpha : \mathrm{yon}(Y \times X) \cong \mathrm{yon}(Y) \times \mathrm{yon}(X).$$

We thus have a bijection

$$\alpha^* : \mathrm{Hom}_{\widehat{\mathcal{C}}}(\mathrm{yon}(Y) \times \mathrm{yon}(X), F) \cong \mathrm{Hom}_{\widehat{\mathcal{C}}}(\mathrm{yon}(Y \times X), F) \tag{3}$$

of sets. This bijection is natural in F since for any morphism $\gamma : F \dot\to G$ of presheaves, the following diagram commutes:

$$\begin{tikzcd}
\mathrm{Hom}_{\widehat{\mathcal{C}}}(\mathrm{yon}(Y) \times \mathrm{yon}(X), F) \arrow[r, "\gamma_*"] \arrow[d, "\alpha^*"'] & \mathrm{Hom}_{\widehat{\mathcal{C}}}(\mathrm{yon}(Y) \times \mathrm{yon}(X), G) \arrow[d, "\alpha^*"] \\
\mathrm{Hom}_{\widehat{\mathcal{C}}}(\mathrm{yon}(Y \times X), F) \arrow[r, "\gamma_*"'] & \mathrm{Hom}_{\widehat{\mathcal{C}}}(\mathrm{yon}(Y \times X), G).
\end{tikzcd}$$

Since α is a natural transformation we have the equality

$$(\mathrm{yon}(f) \times \mathrm{yon}(\mathrm{id}_X)) \circ \alpha = \alpha \circ \mathrm{yon}(f \times \mathrm{id}_X)$$

[3]The formalization of this theorem can be found at `https://github.com/UniMath/UniMath/blob/0e1bd59/UniMath/CategoryTheory/ExponentiationLeftAdjoint.v#L357-L365`

for any morphism $f : Z \to Y$. Hence the diagram below commutes:

$$\begin{tikzcd}
\mathrm{Hom}_{\widehat{\mathcal{C}}}(\mathrm{yon}(Y) \times \mathrm{yon}(X), F) \arrow[r, "(\mathrm{yon}(f)\times\mathrm{yon}(\mathrm{id}_X))^*"] \arrow[d, "\alpha^*"'] & \mathrm{Hom}_{\widehat{\mathcal{C}}}(\mathrm{yon}(Z) \times \mathrm{yon}(X), F) \arrow[d, "\alpha^*"] \\
\mathrm{Hom}_{\widehat{\mathcal{C}}}(\mathrm{yon}(Y \times X), F) \arrow[r, "\mathrm{yon}(f\times\mathrm{id}_X)^*"'] & \mathrm{Hom}_{\widehat{\mathcal{C}}}(\mathrm{yon}(Z \times X), F).
\end{tikzcd}$$

So the bijection is also natural in Y.

Finally, by the Yoneda lemma we have a bijection

$$\mathrm{Hom}_{\widehat{\mathcal{C}}}(\mathrm{yon}(Y \times X), F) \cong F(Y \times X) = ((\_ \times X)^{op})^*(F)(Y) \tag{4}$$

of sets that is natural in both F and Y. It thus follows from equations (1), (2), (3) and (4) that there is a bijection

$$F^{\mathrm{yon}(X)}(Y) \cong ((\_ \times X)^{op})^*(F)(Y)$$

which is natural in both F and Y. Therefore, by Proposition 4.4, the functors $\_^{\mathrm{yon}(X)}$ and $((\_ \times X)^{op})^*$ are naturally isomorphic. □

The fact that $\_^{\mathrm{yon}(X)}$ has a right adjoint now follows from the next proposition.

Proposition 4.6. *Let $\mathcal{C}$ be a small category with binary products. For any object $X \in \mathcal{C}$ the precomposition functor $((\_ \times X)^{op})^* : \widehat{\mathcal{C}} \to \widehat{\mathcal{C}}$ has a right adjoint.*[4]

Proof. By Theorem 1 in Mac Lane [7, p. 110], **Set** is complete. It thus follows by Corollary 2 in Mac Lane [7, p. 239] that the precomposition functor $((\_ \times X)^{op})^*$ has a right adjoint. □

We can now give a sufficient condition for the exponentiation of a representable presheaf to have a right adjoint.

Theorem 4.7. *Let $\mathcal{C}$ be a small category with binary products. Any representable presheaf $F \in \widehat{\mathcal{C}}$ is tiny.*[5]

Proof. Assume F is naturally isomorphic to $\mathrm{yon}(X)$, for some object $X \in \mathcal{C}$. By Theorem 4.5 the exponential functor $\_^{\mathrm{yon}(X)}$ is naturally isomorphic to the precomposition functor $((\_ \times X)^{op})^* : \widehat{\mathcal{C}} \to \widehat{\mathcal{C}}$. By Proposition 4.6 the precomposition functor has a right adjoint. Thus the exponential functor $\_^{\mathrm{yon}(X)}$ has a right adjoint and it follows that $\_^F$ has a right adjoint. □

[4]The formalization of this proposition, which uses the formalization of right Kan extensions done by Ahrens, Matthes and Mörtberg, can be found at `https://github.com/UniMath/UniMath/blob/0e1bd59/UniMath/CategoryTheory/ExponentiationLeftAdjoint.v#L78-L82`

[5]The formalization of this theorem can be found at `https://github.com/UniMath/UniMath/blob/0e1bd59/UniMath/CategoryTheory/ExponentiationLeftAdjoint.v#L367-L375`

5 Cartesian cubical sets

In this section we will define the cartesian cubical sets and show that this presheaf category satisfies axioms (B1)-(B3) in Coquand's article [5]. We begin by defining the cartesian cube category which we denote by $\Box$. In this section we will use inl to denote the function $A \to A \sqcup B$ that maps an element a to its copy in the disjoint union. Similarly, inr will denote the function $B \to A \sqcup B$ that maps an element b to its copy in the disjoint union. Given two functions $f : A \to C$ and $g : B \to C$ we will denote by $[f, g] : A \sqcup B \to C$ the function defined by

$$[f,g](w) = \begin{cases} f(w), & w \in A \\ g(w), & w \in B. \end{cases}$$

Definition 5.1. *The* ***cartesian cube category*** *is the category with*

- ***objects:*** *discrete finite sets, i.e. finite sets with decidable equality,*
- ***morphisms:*** *the morphisms* $\mathrm{Hom}(I, J)$ *are the functions* $J \to I \sqcup \{0, 1\}$*,*
- ***composition:*** *the composition of morphisms* $f \in \mathrm{Hom}(I, J)$ *and* $g \in \mathrm{Hom}(J, K)$ *is given by*
$$g \star f = [f, \mathrm{inr}] \circ g$$
where the composition on the right hand side is the usual function composition,
- ***identity:*** *the identity morphism is given by*
$$\mathrm{id}_I := \mathrm{inl} : I \to I \sqcup \{0, 1\} .$$[6]

We verify that the composition is associative. For any three morphisms $f \in \mathrm{Hom}(I, J)$, $g \in \mathrm{Hom}(J, K)$ and $h \in \mathrm{Hom}(K, L)$ we have

$$\begin{aligned}(h \star g) \star f &= ([g, \mathrm{inr}] \circ h) \star f = [f, \mathrm{inr}] \circ ([g, \mathrm{inr}] \circ h) \\ &= ([f, \mathrm{inr}] \circ [g, \mathrm{inr}]) \circ h = [[f, \mathrm{inr}] \circ g, \mathrm{inr}] \circ h \\ &= [g \star f, \mathrm{inr}] \circ h = h \star (g \star f)\end{aligned}$$

so the composition is associative.

We have
$$f \star \mathrm{id}_I = [\mathrm{inl}, \mathrm{inr}] \circ f = f$$
and
$$\mathrm{id}_J \star f = [f, \mathrm{inr}] \circ \mathrm{inl} = f$$
so the identity morphism is a two-sided identity for the composition. Thus the definition above defines a category.

[6]The formalization of the cartesian cube category can be found at `https://github.com/UniMath/UniMath/blob/0e1bd59/UniMath/CategoryTheory/categories/CartesianCubicalSets.v#L47-L85`

Definition 5.2. *A* ***cartesian cubical set*** *is a presheaf on the cartesian cube category, i.e. a functor $\square^{op} \to \mathbf{Set}$.*

In the cartesian cubical sets model of HoTT we model the interval type $\mathbb{I}$ by the object $\text{yon}(\{0\})$. We will now show that $\mathbb{I}$ satisfies axioms (B1) and (B2). In order to show (B1) we need to find the terminal object of presheaf categories.

Proposition 5.1. *For any category $\mathcal{C}$, the constant functor $T : \mathcal{C}^{op} \to \mathbf{Set}$ that sends all objects to $\{0\}$ and all arrows to $\text{id}_{\{0\}}$ is a terminal object of $\widehat{\mathcal{C}}$.*

Proof. Let $F \in \widehat{\mathcal{C}}$ be an arbitrary presheaf. For any object $X \in \mathcal{C}$ there exists a unique function $\alpha(X) : F(X) \to T(X)$, namely the function mapping every element in $F(X)$ to 0. Therefore α determines the unique family of maps from F to T. This family is natural since for any morphism $f : Y \to X$ the following diagram commutes:

$$\begin{tikzcd}
F(X) \arrow[r, "F(f)"] \arrow[d, "\alpha(X)"'] & F(Y) \arrow[d, "\alpha(Y)"] \\
T(X) \arrow[r, "\text{id}_{\{0\}}"'] & T(Y).
\end{tikzcd}$$

□

We can now state and prove axiom (B1) for cartesian cubical sets.

Theorem 5.2. *The interval object $\mathbb{I}$ in cartesian cubical sets has two distinct global elements, i.e. there exists two distinct natural transformations*

$$\mathbf{0} : T \xrightarrow{\cdot} \mathbb{I} \quad \text{and} \quad \mathbf{1} : T \xrightarrow{\cdot} \mathbb{I}$$

where T is the terminal object in $\widehat{\square}$.[7]

Proof. For each finite set $J \in \square$ there are two distinct morphisms f_0 and f_1 in $\text{hom}_\square(J, \{0\})$ given by

$$f_0(0) = \text{inr}(0) \quad \text{and} \quad f_1(0) = \text{inr}(1)$$

Now define the family of maps $\mathbf{0} : T \to \mathbb{I}$ by $\mathbf{0}(J)(0) = f_0$ and the family of maps $\mathbf{1} : T \to \mathbb{I}$ by $\mathbf{1}(J)(0) = f_1$. These are both natural since for any morphism $g \in \text{hom}_\square(K, J)$ the following diagrams commute:

$$\begin{tikzcd}
\{0\} \arrow[r, "\text{id}_{\{0\}}"] \arrow[d, "\mathbf{0}(J)"'] & \{0\} \arrow[d, "\mathbf{0}(K)"] \\
\text{hom}_\square(J, \{0\}) \arrow[r, "g^*"'] & \text{hom}_\square(K, \{0\})
\end{tikzcd}
\qquad
\begin{tikzcd}
\{0\} \arrow[r, "\text{id}_{\{0\}}"] \arrow[d, "\mathbf{1}(J)"'] & \{0\} \arrow[d, "\mathbf{1}(K)"] \\
\text{hom}_\square(J, \{0\}) \arrow[r, "g^*"'] & \text{hom}_\square(K, \{0\}).
\end{tikzcd}$$

[7]The formalization of (a slight variation of) this theorem can be found at `https://github.com/UniMath/UniMath/blob/0e1bd59/UniMath/CategoryTheory/categories/CartesianCubicalSets.v#L139-L152`

The natural transformations $\mathbf{0}$ and $\mathbf{1}$ are distinct since the maps f_0 and f_1 are distinct. $\square$

The fact that axiom (B2) holds for cartesian cubical sets follows almost immediately from the decidable equality on the objects of $\square$.

Theorem 5.3. *For every object $J \in \square$, the set $\mathbb{I}(J)$ has decidable equality.*[8]

Proof. The set $\mathbb{I}(J)$ is the set $\hom_{\square}(J, \{0\})$ of functions from $\{0\}$ to $J \sqcup \{0,1\}$. These functions can be identified with their image of 0. Since J and $\{0,1\}$ have decidable equality it follows that $J \sqcup \{0,1\}$ has decidable equality. Therefore the set $\hom_{\square}(J, \{0\})$ has decidable equality. $\square$

5.1 Binary products in the cartesian cube category

We want to use Theorem 4.7 to show that axiom (B3) holds for cartesian cubical sets. For this we need to show that $\square$ has binary products.

Theorem 5.4. *The cartesian cube category has binary products.*[9]

Proof. Define $I \times J$ as the disjoint union $I \sqcup J$ and define the projections by

$$\pi_1 := \mathrm{inl} \circ \mathrm{inl} \in \mathrm{Hom}(I \times J, I) \qquad \text{and} \qquad \pi_2 := \mathrm{inl} \circ \mathrm{inr} \in \mathrm{Hom}(I \times J, J).$$

Now, given any finite set K and morphisms $f_1 \in \mathrm{Hom}(K, I)$, $f_2 \in \mathrm{Hom}(K, J)$ consider the diagram

$$\begin{tikzcd}
& K \arrow[dl, "f_1"'] \arrow[dr, "f_2"] & \\
I & I \times J \arrow[l, "\pi_1"] \arrow[r, "\pi_2"'] & J
\end{tikzcd}$$

in $\square$. This corresponds to the diagram

$$\begin{tikzcd}
& K \sqcup \{0,1\} & \\
I \arrow[ur, "f_1"] \arrow[r, "\mathrm{inl} \circ \mathrm{inl}"'] & I \sqcup J \sqcup \{0,1\} & J \arrow[ul, "f_2"'] \arrow[l, "\mathrm{inl} \circ \mathrm{inr}"]
\end{tikzcd}$$

in **Set**. A morphism $f \in \mathrm{Hom}_{\square}(K, I \times J)$ corresponds to a function $f : I \sqcup J \to K \sqcup \{0,1\}$. Now any function from $I \sqcup J$ to $K \sqcup \{0,1\}$ making the diagram commute must send an element $i \in I$ to $f_1(i)$ and an element $j \in J$ to $f_2(j)$. The unique such function is $[f_1, f_2] : I \sqcup J \to K \sqcup \{0,1\}$. Thus the function $[f_1, f_2] \in \mathrm{Hom}_{\square}(K, I \times J)$ is the unique morphism making the first diagram commute. It follows that the cartesian cube category has binary products. $\square$

[8]The formalization of this theorem can be found at `https://github.com/UniMath/UniMath/blob/0e1bd59/UniMath/CategoryTheory/categories/CartesianCubicalSets.v#L154-L166`

[9]The formalization of this theorem can be found at `https://github.com/UniMath/UniMath/blob/0e1bd59/UniMath/CategoryTheory/categories/CartesianCubicalSets.v#L87-L117`

With this result we can now prove that axiom (B3) holds for the cartesian cubical sets.

Theorem 5.5. *The interval $\mathbb{I}$ in cartesian cubical sets is tiny.*[10]

Proof. By Theorem 5.4 and the fact that the category of finite sets is small, it follows from Theorem 4.7 that exponentiation by $\mathbb{I}$ has a right adjoint, i.e. that $\mathbb{I}$ is tiny. □

6 The formalization

The work on this thesis began with the author learning the foundations of homotopy type theory, Coq and category theory, the first two of which were completely new subjects and the third of which the author knew only the basic notions. After this, the first formalization was of the cartesian cubical sets including that the interval in this specific presheaf category is tiny following the proof in Licata et al. [9, p. 10].

After the formalization that the interval in cartesian cubical sets is tiny, an interesting inversion of the usual workflow followed: this result was generalized by testing what assumptions on the interval and the containing category was necessary for the proof to go through. In this way it was found that the only properties needed of the category was that it should be small and have binary products and no assumptions about the object was needed, from which the formulation and proof of Theorem 4.7 followed. Hence, instead of first doing the theoretical proof and then formalizing it, the formalization helped produce the theoretical proof of a more general theorem.

All in all, the formalization done by the author consisted of about 500 significant lines of code of which there were 30 lemmas/theorems and 22 definitions. The code has been merged into the UniMath library. A full overview of all the formalization done in association with this thesis (including the formalization of Proposition 4.4 by Anders Mörtberg) can be found at `https://github.com/UniMath/UniMath/compare/3832438...0e1bd59`.

The formalization built heavily upon already formalized category theory and other results in the UniMath library. The scope would have been far too large to encompass in this thesis had the formalization needed to be done from the very beginning, without any pre-formalized results. So the author is indebted to the amazing work done by the contributors to the UniMath library that made this work possible.

7 Conclusion and future work

The formalization done in this thesis for the cartesian cubical sets only concerns four of the eight axioms stated in Coquand [5] as sufficient for a presheaf category

[10]The formalization of this theorem can be found at `https://github.com/UniMath/UniMath/blob/0e1bd59/UniMath/CategoryTheory/categories/CartesianCubicalSets.v#L178-L183`

to form a model of HoTT. In order to fully formalize that cartesian cubical sets form a model by this axiomatization one needs to also formalize that the other four axioms hold. It would be interesting to use such a formalization to see what conditions on the underlying category and/or presheaves occurring in the axioms that are necessary for the formalization to go through, as was done for axiom (B3) in this thesis. It might also be interesting to formalize other cube categories that are used to give constructive models of the univalence axiom and prove that they satisfy Coquand's axioms [5]. Such a formalization of some other cube category could build upon the one of cartesian cubical sets done here in order to make the formalization a bit simpler and faster.

Proof assistants provide many new possibilities for the future. One is for mathematicians to verify their results using computers and thus remove the component of human error. Another is the possibility to have computer aid in finding new results, as was done for Theorem 4.7 in this thesis. Although this generalization of the proof in Licata et al. [9, p. 10] could easily have been found without the aid of a computer, it was very convenient to simply use the formalization to find the most general conditions under which the theorem holds. It is not hard to believe that this type of workflow, where the formalization is done first and the theoretical proof second, may occur many more times in the future if mathematicians form the habit of formalizing the work they do. Maybe we will discover results with the aid of computers that we would not otherwise have discovered, or that would have taken us much longer to discover without computer aid? On the other hand, we may perhaps, in the project of formalizing all current mathematical knowledge, find that some accepted proofs contain errors that we would not have otherwise found. In this way, computer formalization of mathematics has the potential to bring about a fundamental change of the whole subject. Where this can take us, only the future will tell!

References

[1] Carlo Angiuli, Guillaume Brunerie, Thierry Coquand, Kuen-Bang Hou (Favonia), Robert Harper, and Daniel R. Licata. Syntax and models of cartesian cubical type theory. Draft available at `https://github.com/dlicata335/cart-cube/blob/master/cart-cube.pdf`, 2017.

[2] Marc Bezem, Thierry Coquand, and Simon Huber. The univalence axiom in cubical sets, 2017.

[3] Evan Cavallo, Anders Mörtberg, and Andrew W. Swan. Unifying cubical models of univalent type theory. Preprint available at `http://staff.math.su.se/anders.mortberg/papers/unifying.pdf`, 2019.

[4] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical type theory: a constructive interpretation of the univalence axiom. *CoRR*, abs/1611.02108, 2016.

[5] Thierry Coquand. A survey of constructive presheaf models of univalence. *ACM SIGLOG News*, 5(3):54–65, July 2018.

[6] Chris Kapulkin and Peter LeFanu Lumsdaine. The simplicial model of univalent foundations (after Voevodsky), 2012.

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

[8] Saunders Mac Lane and Ieke Moerdijk. *Sheaves in Geometry and Logic: A First Introduction to Topos Theory.* Universitext. Springer New York, 1994.

[9] Daniel R. Licata, Ian Orton, Andrew M. Pitts, and Bas Spitters. Internal universes in models of homotopy type theory. *CoRR*, abs/1801.07664, 2018.

[10] Peter Lumsdaine. Weak omega-categories from intensional type theory. *Logical Methods in Computer Science*, 6(3), Sep 2010.

[11] Per Martin-Löf and Giovanni Sambin. *Intuitionistic type theory.* Napoli: Bibliapolis, 1984.

[12] Ian Orton and Andrew M. Pitts. Axioms for modelling cubical type theory in a topos. *CoRR*, abs/1712.04864, 2017.

[13] Giovanni Sambin and Jan M. Smith. *Twenty Five Years of Constructive Type Theory.* Oxford Logic Guides. Clarendon Press, 1998.

[14] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics.* `https://homotopytypetheory.org/book`, Institute for Advanced Study, 2013.

[15] Vladimir Voevodsky. An experimental library of formalized mathematics based on the univalent foundations. *Mathematical Structures in Computer Science*, 25(5):1278–1294, 2015.