Type theory and the logic of toposes

QI, Xuanrui

A master's thesis submitted to the
Graduate School of Mathematics, Nagoya University

Advisor: Professor Jacques Garrigue

Contents

Introduction

The goal of this thesis is to provide an survey of categorical logic from the perspective of λ-calculi and type theory. Particularly, this thesis is motivated by a discussion on the Coq-Club mailing list [Sem21], asking whether the type theory underlying the proof assistant Coq forms a topos. Therefore, the running goal of this thesis is to find and construct a type theory that corresponds to the internal language of a topos.

A **type theory** is a computational version of logic based on a simple computational language called the *λ-calculus*. The observation that a proof is essentially a computation, and logical propositions can be interpreted as types of computations (e.g., $\forall x, P(x)$ means that given the input x, a proof of $P(x)$ can be computed) makes the λ-calculus a proper setting for logic and foundations of mathematics. Particularly, *Martin-Löf type theory*, which will be introduced in Chapter 3 of this thesis, is a simple setting for higher-order logic.

Here, we are interested in type theory as a model of higher-order logic: often, higher-order logic is presented in the style of a "type theory". Every type theory corresponds to a logical calculus; this is often called the "Curry-Howard correspondence". In our thesis, we consider three such equivalences: the simply typed λ-calculus to intuitionistic propositional logic, Martin-Löf type theory to constructive higher-order logic, and the extensional calculus of constructions to (roughly) a slightly weaker version of our "everyday mathematical language".

We begin with a brief overview of category theory, with many results and proofs coming from [MM92], and then proceed to consider the simplest case of λ-calculi and type theories: the simply typed λ-calculus. First, we introduce the computational rules and basic properties of the λ-calculus without introducing types, and then introduce *simple types* to make our λ-calculus into a type theory. Then, based on Lambek and Scott's results [LS86], we give a correspondence between the simply-typed λ-calculus and cartesian closed categories, a "nice" class of categories often used in mathematics. Particularly, we prove two theorems about this correspondence:

Theorem. *Any cartesian closed category is a model of the simply typed λ-calculus.*

Theorem. *The internal language of a cartesian closed category is simply typed λ-calculus.*

Chapter 3 is the technical core of this thesis. In chapter 3, we introduce Martin-Löf type theory, a higher-order type theory that is sufficiently expressive to develop much of mathematics. We then provide a semantics, or model, of MLTT, in categories of presheaves. This construction, due to Hofmann [Hof97], serves as the basis of interpreting type theory in categories, so we describe it in detail following

Huber's presentation [Hub16]. This construction can be stated in the form of the following theorem:

Theorem. *The extensional Martin-Löf type theory has a model in a category of presheaves of sets.*

However, there are some discrepancies between our presentation of MLTT (usually called "intensional type theory") and the behavior of the model (called "extensional type theory"), which leads to chapter 4, which explains these discrepancies as the difference between intensional and extensional type theory. We explain why the presheaf model supports only extensional type theory, and introduce the language of ∞-categories to describe informally a model for intensional type theory based on ∞-categories. Moreover, following [Uni13], we introduce a powerful axiom called univalence which holds in the ∞-category model.

Finally, in Chapter 5, we proceed to present the main goal of our thesis, which is to describe a type theory that is closely related to a topos. Toposes are a certain class of categories that can serve as a "universe" for mathematics. The archetypal example of a topos is **Set**, the category of sets, which is the setting for the usual kind of mathematics that we are familiar with; its internal language is the mathematical language we are used to. However, any topos allows a form of reasoning that is close to our mathematical intuition and which is strong enough to develop a large part of modern mathematics. Synthesizing prior work in [Pit00; Mai05], we show the following result, which is the main result of this thesis:

Theorem. *The internal type theory of an elementary topos is the extensional calculus of constructions, with a proof-irrevelant, impredicative universe of propositions.*

This thesis assumes that a reader has some knowledge of category theory, as covered in a textbook like [Mac98] or [Awo10]. It also requires the reader to have some familiarity with the theory of propositional and first-order logic. However, it requires no advanced knowledge of mathematical logic, or any knowledge of type theory.

Acknowledgements

I thank my advisor, Professor Jacques Garrigue, for supervising me and guiding me in writing this thesis. I also thank Dr. Takafumi Saikawa for helping me a lot with the technical contents, especially chapter 3, and Dr. Reynald Affeldt for helping find errors in this thesis and providing valuable feedback. Finally, I thank the members of Types and Category Theory Zulip forums for answering questions related to this thesis, and members of the Coq-Club mailing list, for discussions that motivated and guided the writing of this thesis.

Chapter 1

Categorical and topos-theoretic preliminaries

1.1 Notation

In this thesis, we use the calligraphic letters $\mathcal{C}$, $\mathcal{D}$, etc., for unspecified categories, which are usually small categories. The letters $\mathcal{E}$, $\mathcal{F}$ and $\mathcal{G}$ are reserved for toposes.

Objects of categories are denoted using capital letters, and $X : \mathcal{C}$ means that X is an object of $\mathcal{C}$; some authors prefer $\in$, but depending on the foundation chosen, $\mathcal{C}$ might not be a set (or there might be no membership relation $\in$ at all), therefore we use the foundation-neutral colon notation, reserving $X : U$ to the case when U is in fact a set.

Morphisms are denoted using small Latin letters f, g, h, etc., or small Greek letters τ, σ, etc. Functors are also denoted using capital letters, but we use F, G, H, etc., to avoid confusion with objects of categories. Sheaves and presheaves are denoted using the same letters[1].

$\mathrm{Hom}_{\mathcal{C}}(X, Y)$ denotes the hom-set of morphisms between X and Y in a category $\mathcal{C}$; we also write $\mathrm{Hom}(X, Y)$ if the category is clear. We write $\mathbf{PSh}(\mathcal{C})$ for the category of presheaves over $\mathcal{C}$, in the literature, sometimes $\hat{\mathcal{C}}$ is used.

In this thesis, the plural form of "topos" is always "toposes", although some authors and readers may prefer the Greek style "topoi".

1.2 The Yoneda lemma

We begin our review of categorical notions with a restatement of the Yoneda lemma, due to its central importance in category theory and in this thesis. It is often considered the "fundamental theorem" of category theory. Here, we recall the statement and fix some notations.

1.2.1 A note on set theory

The category **Set** will be used throughout the thesis. We give a definition below:

Definition 1.2.1 (category of sets)**. Set** is the category of sets. The objects are sets, and the morphisms are the functions/maps between sets.

[1] Because the French word for "sheaf" is "faisceau".

In Zermelo-Fraenkel set theory with or without choice (ZF/ZFC), the collection of all sets is not a set. Normally, one requires the collection of objects to be a *proper class*, and call categories where the collection of objects is not a set a *large category*, and otherwise a *small category*. If for any pair X, Y of objects, the collection of morphisms between them is a set, then the category is additionally said to be *locally small*. If one prefers to work in ZFC, then many constructions described in this thesis require the assumption that some "base" category is small, or at least locally small.

However, there are some problems with this approach. First of all, many interesting categories are not small. Moreover, it can be dangerous to work with proper classes, as they are not, and do not behave exactly like, sets; particularly, many oddities may happen if one considers categories that are not locally small. Finally, one runs into trouble defining categories like **Cat**, whose objects are categories: it could not be defined as the category of all categories, as the collection of all categories do not form a proper class; it could be defined as the category of small categories, but this will exclude from **Cat** many categories that one might want to consider.

Alternatively, one can realize **Set** as $\mathbf{Set}_\kappa$, the category of sets with cardinality up to κ, where κ is a limit cardinal, and **Cat** as $\mathbf{Cat}_{\kappa'}$, the category of categories for which the collection of items are sets of cardinality less than κ', where κ' is another limit cardinal larger than κ. A "small" category is a category for which the collection of elements is a set of cardinality less than κ, and so on. This solution avoids proper classes completely, but it requires some set-theoretic techniques which will unnecessarily complicate our discussion.

To fix these problems, one can alternatively allow a hierarchy of *Grothendieck universes* (for example, as done in Chapter I, Section 6 of [Mac98]). Then, one may fix some universe U and say that a category is U-small (or just "small") if both the collections of objects and all collections of morphisms are sets in U. Then, we can realize the category **Set** as the category $\mathbf{Set}_U$ of U-small sets, where U is a universe large enough to contain all "interesting" sets, and realize **Cat** as another category $\mathbf{Cat}_{U'}$ where U' is a larger universes enough to contain all "interesting" categories.

The use of Grothendieck universes allows a much cleaner solution to the size problem, and often we will prefer this as our mental model. However, no construction in this text *require* universes unless explicitly stated, so one could also use either ZF(C)-based solution we have proposed above. Nevertheless, later on in this thesis we will be transparent with regard to foundations, and use only the terminology "small", "locally small" and "large". If one prefers limits cardinals Grothendieck universes, the "smallness" criteria are with respect to an unspecified limit cardinal or Grothendieck universe.

After on a long digression on set theory, we shall return to the Yoneda lemma, which we state below:

Lemma 1.2.1 (Yoneda lemma)**.** *Let $\mathcal{C}$ be a locally small category. Then for any functor $F : \mathcal{C}^{\mathrm{op}} \to \mathbf{Set}$, the map*

$$(\varphi : \mathrm{Hom}_{\mathcal{C}}(-, c) \to F) \mapsto F(c),$$

assigning to the natural transformation $\varphi : \mathrm{Hom}_{\mathcal{C}}(-, c) \to F$ the object $F(c)$, is an bijection, for any object c of $\mathcal{C}$.

Proof. The proof of the Yoneda lemma could be found in texts about category theory, such as [Mac98] or [Awo10]. □

For each $c : \mathcal{C}$, there is a functor $\mathbf{y}(c) = \mathrm{Hom}_{\mathcal{C}}(-, c) : \mathcal{C}^{\mathrm{op}} \to \mathbf{Set}$, embedding an object into its (contravariant) c-hom-set called the **presheaf represented by** c (for reasons we will see later in this chapter). Moreover, $\mathbf{y} : \mathcal{C} \to [\mathcal{C}^{\mathrm{op}}; \mathbf{Set}]$ itself is also a functor, often called the **Yoneda embedding** functor.

1.3 Cartesian closed categories

This thesis focuses on the "computational trinity", or the correspondence between logic (and set theory), type theory, and category theory. Every category theory corresponds to a type theory and thus a logic, but not all of them are interesting enough. A *cartesian closed category* is simple enough as a category, but supports enough structure to interpret a large amount of logic in it. These are the first objects we will study in this thesis.

Cartesian closed categories have a terminal object and all binary products, and this allows for many basic constructions in the category. We also require another structure, the *exponential*, in the category, which, on the other hand, could be though of as an "internal" hom in the category. The precise definition of exponential objects follow:

Definition 1.3.1 (exponential objects)**.** Let $\mathcal{C}$ be a category, and X, Y objects of $\mathcal{C}$. Assume that $\mathcal{C}$ has all binary products with X. Then an **exponential object** is an object Y^X along with a universal morphism $\mathrm{ev} : Y^X \times X \to Y$, also called the **evaluation morphism**[2].

In other words, Y^X is an exponential object if for any object Z of $\mathcal{C}$ and morphism $f : Z \times X \to Y$, there is a unique morphism $g : Z \to Y^X$ such that the following diagram commutes:

$$\begin{array}{ccc} Z \times X & & \\ {\scriptstyle g\times \mathrm{id}}\downarrow & \searrow^{f} & \\ Y^X \times X & \xrightarrow[\mathrm{ev}]{} & Y. \end{array}$$

Observation 1.3.2. In the case that $\mathcal{C}$ is locally small, there is a canonical isomorphism $\mathrm{Hom}(Z \times X, Y) \cong \mathrm{Hom}(Z, Y^X)$, give by the map $f \mapsto g$, which is a bijection due to the unique existence of g for each f.

Proof. This follows directly from the unique existence of g for each f. □

Definition 1.3.3 (cartesian closed category)**.** A category $\mathcal{C}$ is a **cartesian closed category** (or often just CCC) if it has a terminal object, all binary products, and all exponentials.

Exponential objects can also be defined equationally ([Awo10]).

[2] or the evaluation "map", by an abuse of language.

Lemma 1.3.1 (equational definition of exponentials)**.** *A category $\mathcal{C}$ has exponentials if for each pair of objects X, Y, there is an object Y^X and a morphism* $\mathrm{ev} : Y^X \times X \to Y$ *such that for each morphism $f : Z \times X \to Y$, there is a morphism $\tilde{f} : Z \to Y^X$ (also called the transpose of f), such that the following equations hold:*

$$\mathrm{ev} \circ (\tilde{f} \times \mathsf{id}_X) = f$$
$$\widetilde{\mathrm{ev} \circ (g \times \mathsf{id}_X)} = g$$

Proof. Diagram chasing. □

Many interesting categories have exponential objects:

Example 1.3.4. In the category **Set**, the exponential object Y^X is the set of functions from X to Y. The evaluation morphism is the usual evaluation map: $\mathrm{ev}(f, x) = f(x)$.

Definition 1.3.5 (presheaf)**.** Let $\mathcal{C}$ be a category, usually small or locally small. Then the category $[\mathcal{C}^{\mathrm{op}}; \mathbf{Set}]$ of contravariant functors from $\mathcal{C}$ to **Set** is called the category of (set-valued) presheaves (or preshaf category) over $\mathcal{C}$, and often denoted $\mathbf{PSh}(\mathcal{C})$. A functor $\mathcal{C}^{\mathrm{op}} \to \mathbf{Set}$ is called a **presheaf** over $\mathcal{C}$.

Proposition 1.3.2. *Let $\mathcal{C}$ be a small category. The presheaf category $\mathbf{PSh}(\mathcal{C}) = [\mathcal{C}^{\mathrm{op}}; \mathbf{Set}]$ has exponential objects. (Proposition 1 of I.6, [MM92].)*

Proof. First, for the definition of exponentials to make sense, we need to have products in $\mathbf{PSh}(\mathcal{C})$. The binary products are defined pointwise i.e. $(F \times G)(C) = F(C) \times G(C)$, and it is routine to verify that this definition indeed yields a product.

We proceed by deriving the expression for exponentials in $\mathbf{PSh}(\mathcal{C})$ first, and then prove that it is indeed an exponential.

To derive the expression, we may assume that $\mathbf{PSh}(\mathcal{C})$ has exponentials. Then we have an isomorphism $\mathrm{Hom}(H \times F, G) \cong \mathrm{Hom}(H, G^F)$, where F, G and H are presheaves on $\mathcal{C}$.

Now, consider the case $H = \mathbf{y}(C) = \mathrm{Hom}_{\mathcal{C}}(-, C)$, where C is an object of C and $\mathbf{y}$ is the Yoneda embedding. By the Yoneda lemma one has: $G^F(C) = \mathrm{Hom}(\mathbf{y}(C), G^F) = \mathrm{Hom}(\mathbf{y}(C) \times F, G)$.

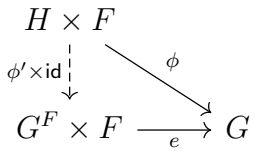
One could then define the exponential G^F as $c \mapsto \mathrm{Hom}(\mathbf{y}(C) \times F, G)$. Clearly, this is a functor $\mathcal{C}^{\mathrm{op}} \to \mathbf{Set}$, assigning to every object C the set of natural transformations between $\mathbf{y}(C) \times F$ and G. Note that this requires our assumption that $\mathcal{C}$ is small; otherwise, the collection of morphisms from $\mathbf{y}(C) \times F$ to G do not necessarily form a set.

We need to verify that this is indeed an exponential. Associate with it an evaluation map $\mathrm{ev} : G^F \times F \to G$. This is a natural transformation again, so one could define it by components:

$$\mathrm{ev}_C(\theta, y) = \theta_C(\mathsf{id}_C, y) \in G(C)$$

for $c : \mathcal{C}$, $\theta : \mathrm{Hom}_{\mathcal{C}}(-, C) \times F \to G$, and $y : F(C)$.

Moreover, for each any natural transformation $\phi : H \times F \to G$ one can find a (unique) $\phi : H \to G^F$ such that the diagram



commutes. For each $c : \mathcal{C}$ and $u \in H(C)$, we need tp find an element $\phi_c(u) \in G^F(C)$, i.e. a natural transformation $\phi'_c(u) : \mathrm{Hom}_{\mathcal{C}}(-, C) \times F \to G$ (see the isomorphism above). For any $f : D \to C$ and $x \in F(D)$, we can define: $(\phi'_C(u))_D : \mathrm{Hom}_{\mathcal{C}}(D, C) \times F(D) \to G(D)$, $(f, x) \mapsto \phi_D(u \circ f, x)$.

This is natural in D, and one can verify that $\mathrm{ev}_C(\phi'_C(u), y) = \phi_C(u, y)$, so the diagram above commutes. Therefore, the definition of ϕ' is completed, and thus G^F as defined above is indeed the exponential of G with F.

□

The previous examples show that many familiar categories are cartesian closed. All categorical products are defined up to isomorphism:

Example 1.3.6. Set is cartesian closed. The terminal object is the one-element set, the categorical product is the cartesian product, and S^T is the set of functions $T \to S$.

Example 1.3.7. FinSet, the category of finite sets, is cartesian closed. The constructions are exactly identical to those in **Set**.

Example 1.3.8. Cat, the category of categories and functors, is cartesian closed. The terminal object is the single-object category, and the categorical product is given by the product of categories. We ignore size issues, as well as the 2-category structure on **Cat**, for readers familiar with higher category theory.

Proposition 1.3.3. *Let $\mathcal{C}$ be a small category. The category $\mathbf{PSh}(\mathcal{C})$ is cartesian closed.*

Proof. Proposition 1.3.2 shows that $\mathbf{PSh}(\mathcal{C})$ has products and exponentials. The terminal object in $\mathbf{PSh}(\mathcal{C})$ is the constant presheaf that maps every object to the one-object set $\{*\}$. □

It is well known from category theory that any preordered set can be considered a category: the objects are the elements of the poset, and there is a morphism $x \to y$ if $x \leq y$. Here, we show that a certain class of posets are cartesian closed as categories.

Definition 1.3.9 (lattice)**.** A **lattice** is a poset L with a meet (or infimum) operator $\wedge$ and a join (or maximum) operator $\vee$, satisfying the following axioms:

- $x \wedge y \leq x$ and $x \wedge y \leq y$. Similarly, $x \leq x \vee y$ and $y \leq x \vee y$;
- if $z \leq x$ and $z \leq y$, then $z \leq x \wedge y$;
- similarly, if $x \leq z$ and $y \leq z$, then $x \vee y \leq z$.

Both $\wedge$ and $\vee$ need to be total operators, i.e., any pair of elements in L have a meet and a join.

It is easy to see that $\wedge$ and $\vee$ are "formally dual", if one considers L a category. In fact, the following fact is true for lattices.

Proposition 1.3.4. *A lattice has all binary products and all binary coproducts when considered as a category, with the binary product given by the meet and binary coproduct given by the join.*

Proof. Let $x, y \in L$ where L is a lattice. We have $x \times y = x \wedge y$; the canonical projections exist because $x \wedge y \leq x$ and $x \wedge y \leq y$. Moreover, for $z \in L$ such that $z \leq x$ $z \leq y$ (i.e., there are morphisms from z to x and from z to y), there is a morphism $z \to x \times y$, since $z \leq (x \wedge y)$ by definition. Note that in L, by definition, there is at most one morphism between any pair of objects, so the uniqueness condition is redundant.

The proof for $x \vee y$ is essentially the same. □

However, a lattice need not have initial and terminal objects, so it does not have *all* products and coproducts. We may fix this subtle problem by introducing a minimum and a maximum element.

Definition 1.3.10 (bounded lattice)**.** A **bounded lattice** is a lattice with a minimum object $\bot$ and a maximum object $\top$, such that $x \wedge \top = x$ and $x \vee \bot = x$.

Proposition 1.3.5. *A bounded lattice has all products and all coproduts when considered as a category. The initial object is $\bot$ and the terminal object is $\top$.*

Proof. A lattice already has all non-nullary products and coproducts (by the totality of $\wedge$ and $\vee$). The nullary product is just the terminal object, which is $\top$ since $x \leq \top$ for any x. Similarly, the nullary coproduct is the initial object, which is $\bot$. □

Now, we can finally define the class of "nice" posets we want. Readers familiar with denotational semantics or constructive logic will be familiar with the following definition.

Definition 1.3.11 (Heyting algebra)**.** A **Heyting algebra** is a bounded lattice with an operator $\implies$ (sometimes called the *implication*) operator, such that $x \wedge a \leq b$ if and only if $x \leq a \implies b$.

Proposition 1.3.6. *A Heyting algebra, when considered as a category, has exponential objects. The exponential of two objects x, y is given by $x \implies y$.*

Proof. The evaluation morphism is $(x \implies y) \wedge x \leq y$. The transpose of a morphism $x \wedge y \leq z$ is $x \leq y \implies z$. The two equations satisfied by exponentials can then be verified. □

Corollary 1.3.7. *A Heyting algebra is a CCC when considered as a category.*

Heyting algebras are closely connected to topological spaces. In other words, topology is a rich source of Heyting algebras.

Example 1.3.12. Let X be a topological space. Then, the collection of open sets of X, ordered by inclusion, forms a Heyting algebra.

Proof. As one expects, the meet is intersection, the join is union, the mimimum object is $\varnothing$, and the maximum object is X. The fact that $O(X)$, the collection of open sets of X, forms a bounded lattice follows directly from the axioms of a topological space.

The definition of the implication operator is more subtle. It is given by $U \implies V = ((X \backslash U) \cup V)^\circ$, where $-^\circ$ is the interior operator. One can verify that this satisfies the laws of a Heyting algebra. □

1.4 Toposes

A topos is a category that behaves like a mathematical universe, in some senses. Before giving a rigorous definition, we will first show some examples:

Example 1.4.1. The category **Set** of sets. The objects are sets (according to some set theory), and the morphisms are functions between sets. Morphism composition is just the composition of functions.

Example 1.4.2. The category **FinSet** of finite sets. The objects are finite sets, the morphisms are functions between sets, and composition is again just function composition. This is a full subcategory of **Set**.

We recall the definition of a full subcategory:

Definition 1.4.3 (full subcategory)**.** A full subcategory $\mathcal{D}$ of $\mathcal{C}$ is a subcategory such that for each pair of objects $X, Y : \mathcal{D}$, we have $\mathrm{Hom}_{\mathcal{D}}(X, Y) = \mathrm{Hom}_{\mathcal{C}}(X, Y)$.

Example 1.4.4. Given any small category[3] $\mathcal{C}$, the presheaf category $\mathbf{PSh}(\mathcal{C}) = [\mathcal{C}^{\mathrm{op}}; \mathbf{Set}]$ is a topos.

Example 1.4.5. The category $\mathbf{Sh}(X)$ of sheaves on a topological space X, or more generally the category $\mathbf{Sh}(\mathcal{C})$ on a site $\mathcal{C}$.

We give the definition of sheaves in the simple case, i.e., a sheaf on a topological case:

Definition 1.4.6 (sheaf)**.** A sheaf F of sets over a space X is a presheaf of sets on the category of opens of X, $\mathcal{O}(X)$, such that for each open covering $U = \bigcup_{i \in I} U_i$ of a open set $U \subseteq X$, the diagram

$$F(U) \xrightarrow{e} \prod_i F(U_i) \underset{q}{\overset{p}{\rightrightarrows}} \prod_{i,j} F(U_i \cap U_j)$$

is an equalizer diagram.

The maps e, p and q are defined as following. For any $t \in F(U)$, $e(t) = \{t|_{U_i} \mid i \in I\}$, and for any family $t_i \in F(U_i)$ indexed by $i \in I$, $p(t_i) = \{t_i|_{U_i \cap U_j}\}$ and $q(t_j) = \{t_j|_{U_i \cap U_j}\}$.

The definition of a sheaf of groups, rings, etc., can be obtained by replacing "sets" with "groups", "rings", etc.

[3] or, if using Grothendieck universes, U-small for some universe U. Similarly, "locally small" would mean U-locally small.

More generally, sheaves can be defined over any site, or category with a *Grothendieck topology*. We will not discuss this case here, and refer the interested reader to Chapter III of [MM92]. Sheaves are ubiquitous in mathematics: for example, both manifolds in differential/complex geometry, and schemes in algebraic geometry, can be defined as "spaces associated with a particular sheaf". Therefore, sheaves are an abundant source of toposes.

Next, we shall present the formal definition of a topos. To be precise, there are two basic notions of toposes: **elementary toposes**, which originate in logic, and **Grothendieck toposes**, which originated in Grothendieck's algebraic geometry. When we say only "topos", we mean an elementary topos. Here, we will focus on the notion of an elementary topos. Roughly speaking, an elementary topos is a category that behaves closely enough to **Set** so that mathematical language could be used in the category.

The most essential structure that makes toposes powerful is that of the *subobject classifier*:

Definition 1.4.7 (subobject classifier)**.** Let $\mathcal{C}$ be a category with finite limits. A **subobject classifier** is a monomorpshism $\text{true} : \mathbf{1} \to \Omega$ into a fixed object Ω, where $\mathbf{1}$ is a terminal object in $\mathcal{C}$, such that for any monic $S \to X$ in $\mathcal{C}$ there is a unique morphism $\phi : X \to \Omega$, such that

$$\begin{array}{ccc} S & \longrightarrow & \mathbf{1} \\ \downarrow & & \downarrow{\scriptstyle \text{true}} \\ X & \xrightarrow[\phi]{} & \Omega \end{array}$$

is a pullback square.

The object Ω is sometimes called the **object of truth values**.

Intuitively speaking, the subobject classifier provides a notion of "truth values" and "inclusion" in a category, enabling one to consider the objects of the category to be set-like.

Then, we can formally define the notion of elementary topos:

Definition 1.4.8 (elementary topos)**.** An elementary topos is a category with:

1. all finite limits and colimits, i.e. any diagram $\mathcal{J} \to \mathcal{C}$, where $\mathcal{J}$ is finite, has a limit/colimit;
2. exponential objects;
3. a subobject classifier.

An elementary topos is apparently also a CCC:

Proposition 1.4.1. *Any topos is cartesian closed.*

Proof. A topos has a terminal object and all binary products since both are finite limits. It also has all exponential objects by definition. □

Set is the canonical example of a topos. Particularly, we show that it has a subobject classifier:

Example 1.4.9. In **Set**, (roughly speaking) there are two truth values, "true" and "false". Therefore, the set of truth values is $\mathbf{2} = \{0, 1\}$. The one-element set, $\mathbf{1} = \{*\}$, clearly embeds into $\mathbf{2}$. Moreover, any inclusion $S \subseteq X$ could be considered as a function $X \to \mathbf{2}$:

$$\phi_S(x) = \begin{cases} 0, & x \notin S \\ 1, & x \in S. \end{cases}$$

One can verify by a simple diagram chase that this makes the following diagram a pullback square:

$$\begin{array}{ccc} S & \longrightarrow & \mathbf{1} \\ {\scriptstyle \iota}\downarrow & & \downarrow{\scriptstyle \text{true}} \\ X & \xrightarrow[\phi_S]{} & \mathbf{2} \end{array}$$

where true is defined as the embedding $* \mapsto 1$.

A category of presheaves on a small category also has a subobject classifier:

Proposition 1.4.2. *Let $\mathcal{C}$ be a small category. The presheaf category $\mathbf{PSh}(\mathcal{C})$ has a subobject classifier.*

Proof. The subobject classifer is the functor $\Omega : \mathcal{C}^{\text{op}} \to \mathbf{Set}$ which sends $C : \mathcal{C}$ to the collection of subfunctors of $\mathbf{y}(C)$, i.e. the functors/presheaves $S : \mathcal{C}^{\text{op}} \to \mathbf{Set}$ such that there is a monic $S \hookrightarrow \mathbf{y}(C)$. For details, see Chapter I, Section 4 of [MM92]. □

Furthermore, both **Set** and the $\mathbf{PSh}(\mathcal{C})$ (where $\mathcal{C}$ is small) are toposes.

Proposition 1.4.3. Set *is an elementary topos.*

Proof. A category is complete (i.e., has all small limits) if it has all equalizers and all small products (Corollary 2, Section 2, Chapter V, [Mac98]). **Set** has all equalizers and all finite products, so it is complete. Similarly, **Set** has all coequalizers and all finite coproducts, so by duality it is also cocomplete. The existence of exponentials and a subobject classifier have both been proven in this thesis. □

Proposition 1.4.4. *Let $\mathcal{C}$ be a small category.* $\mathbf{PSh}(\mathcal{C})$ *is an elementary topos.*

Proof. In $\mathbf{PSh}(\mathcal{C})$, all finite limits and colimits can be computed pointwise (Section 3, Chapter V, [Mac98]), so all finite limits and colimits exist since **Set** has both. By Proposition 1.3.2 and Proposition 1.4.2 $\mathbf{PSh}(\mathcal{C})$ has exponentials and a subobject classifier, so $\mathbf{PSh}(\mathcal{C})$ is a topos. □

1.5 Geometric morphisms

A notion of toposes is not complete without the notion of *morphisms* between them. Naturally, one may think that a morphism between toposes is one that "preserves structure":

Definition 1.5.1 (logical functor)**.** Let $\mathcal{E}$, $\mathcal{F}$ be elemetary toposes. A functor $F : \mathcal{E} \to \mathcal{F}$ is called a **logical functor** if it satisfies the following conditions:

1. F preserves all finite limits;

2. the canonical morphism $\phi_A : F(\Omega^A) \to \Omega^{F(A)}$ is an isomorphism.

This is a valid definition and has some applications in topos theory. However, we are in general more interested in another notion of toposes, the notion of a **geometric morphism**.

First, we recall the definition of adjoint functors:

Definition 1.5.2 (adjoint functor)**.** Let $\mathcal{C}$, $\mathcal{D}$ be categories, and $F : \mathcal{C} \to \mathcal{D}$, $G : \mathcal{D} \to \mathcal{C}$ functors. Then we say F and G are **adjoint functors** if there is a natural bijection between $\mathrm{Hom}(F(X), Y)$ and $\mathrm{Hom}(X, G(Y))$ for every $X : \mathcal{C}$ and $Y : \mathcal{D}$.

We write $F \dashv G$ if F and G are adjoint. F is called the **left adjoint**, and G is called the **right adjoint**.

The definition of geometric morphisms follows:

Definition 1.5.3 (geometric morphism)**.** Let $\mathcal{E}$, $\mathcal{F}$ be toposes. A **geometric morphism** between the pair of toposes, $f : \mathcal{E} \to \mathcal{F}$ consists of a pair of adjoint functors: the *direct image* $f_* : \mathcal{E} \to \mathcal{F}$, and the *inverse image* $f^* : \mathcal{F} \to \mathcal{E}$, such that the left adjoint f^* preserves finite limits.

Chapter 2

The simply-typed λ-calculus and cartesian closed categories

2.1 The simply-typed λ-calculus

A λ-calculus is a syntactic system for reasoning and computation. It consists of a number of *variables* and *terms*, as well as *reduction rules* that govern how terms reduce, i.e. compute. We will first give a simple example, the untyped λ-calculus.

2.1.1 The untyped λ-calculus

The untyped λ-calculus has only three basic forms: variable, abstraction and application. A variable is nothing more than a name, and we allow for a countably infinite number of names of variables: x, y, z, and so on.

$$\begin{array}{rr} M ::= x & \text{variable} \\ \mid \lambda x.M & \text{abstraction} \\ \mid M\ M & \text{application} \end{array}$$

Figure 2.1: terms of the untyped λ-calculus

An *abstraction* is like a function (and often called a "function") as it could be applied in an *application*. A variable that has been abstracted by a λ-abstraction (e.g., x in $\lambda x.M$) is called **bound**, and otherwise it is said to be **unbound**. A variable is said to be **free** in a term if it appears unbound in the term. A term without free variables is said to be **closed**.

Example 2.1.1. In the term $M = \lambda x.x\ y$, x is a bound variable and y is a free variable. M is not a closed term.

Since λ-abstractions are like functions, they could be expected to behave similarly to mathematical functions. Bound variables can be freely renamed; for example $\lambda x.x$ and $\lambda y.y$ denote the same term, just as $f(x) = x$ and $f(y) = y$ are the same function. This is called the **α-equivalence** of terms. We will always

consider terms up to α-equivalence, i.e., consider α-equivalent terms to be exactly the same.

When an abstraction is applied to a term, we substitute the term to the bound variables into the term, e.g. $(\lambda x.x\ z)\ y = y\ z$. We can also evaluate a term partially, just like when we evaluate mathematical functions at a certain point. This process is called **reduction**.

First, we need to define the concept of **substitution** in the form of $M[x := t]$, which means to "substitute all occurences of x in M with t". However, there might be instances where M and t have common variables, and naïve substitution will give unexpected results. Therefore, we must perform substitution in a *capture-avoiding* manner, defined by the following rules (here, M, N and T are terms and x, y variables):

$$
\begin{gathered}
x[x := M] = M \\
x[y := M] = x \\
(\lambda x.M)[x := N] = \lambda x.M \\
(\lambda x.M)[y := N] = \lambda x.(M[y := N]), \text{ if } x \neq y \text{ and } x \text{ is not free in } N \\
(M\ N)[x := T] = (M[x := T])\ (N[x := T])
\end{gathered}
$$

Figure 2.2: capture-free substitution of terms

Now, we can finally define precisely reduction in the untyped λ-calculus.

Definition 2.1.2 (β-reduction)**.** The (β-)reduction relation $\to_\beta$ is the smallest relation satisfying the following rules (see Figure 2.3).

$$
\textsc{Red-Beta}\ \frac{}{(\lambda x.M)N \to_\beta M[x := N]} \qquad \textsc{Red-Abs}\ \frac{M \to_\beta M'}{\lambda x.M \to_\beta \lambda x.M'} \qquad \textsc{Red-App-1}\ \frac{M \to_\beta M'}{M\ N \to_\beta M'\ N}
$$

$$
\textsc{Red-App-2}\ \frac{N \to_\beta N'}{M\ N \to_\beta M\ N'}
$$

Figure 2.3: β-reduction for untyped λ-calculus

Definition 2.1.3 (multi-step reduction)**.** The smallest multi-step closure of $\to_\beta$ relation is called the multi-step reduction relation and is denoted $\to_\beta^*$. In other words, $M \to_\beta^* N$ if there is a (finite) sequence S, T, ... of terms, such that $M \to_\beta S \to_\beta T \to_\beta ... \to_\beta N$.

Definition 2.1.4 (β-equivalence)**.** The smallest transitive, reflexive and symmetric closure of $\to_\beta$ is called the β-equivalence relation, or $\equiv_\beta$.

A term that cannot be further β-reduced is called **β-normal**, and if $M \to^* N$ where N is β-normal, N is called the **β-normal form** of M.

β-equivalence has the following important property, first proven by Church and Rosser:

Theorem 2.1.1 (Church-Rosser)**.** *If $M \equiv_\beta N$, then there is a term T such that $M \to^* T$ and $N \to^* T$.*

Proof. This is corollary 1.4.8 in [SU06]. Note that it is slightly different from the original Church-Rosser theorem (which is theorem 1.4.7 in the same book), but for practical purposes we will prefer this form of the Church-Rosser theorem. □

The Church-Rosser property has the following consequences:

Corollary 2.1.2. *If $M \equiv_\beta N$ and N is β-normal, then $M \to^* N$.*

Corollary 2.1.3. *If both M and N are in β-normal form and $M \equiv_\beta N$, then $M = N$.*

However, here is one more reduction relation on the untyped λ-calculus, usually denoted by the Greek letter η. This reduction stems from the observation that, e.g., in set-theoretic mathematics,$x \mapsto f(x)$ and f are considered identical. However, this is not covered by β-reductions, as $\lambda x.M\ x$ does not β-reduce to M. η-reduction is closely related to the notion of functional extentionality in mathematics, as we shall see later.

η-reduction ($\to_\eta$) is defined by the rules in Figure 2.4. $\to_\eta^*$ and $\equiv_\eta$ are defined analogously to $\to_\beta^*$ and $\equiv_\beta$.

$$\textsc{Eta}\ \frac{x \text{ is free in } M}{\lambda x.M\ x \to_\eta M} \qquad \textsc{Eta-Abs}\ \frac{M \to_\eta M'}{\lambda x.M \to \lambda x.M'} \qquad \textsc{Eta-App-1}\ \frac{M \to_\eta M'}{M\ N \to_\eta M'\ N} \qquad \textsc{Eta-App-2}\ \frac{N \to_\eta N'}{M\ N \to_\eta M\ N'}$$

Figure 2.4: η-reduction for the untyped λ-calculus

We say that $M \to_{\beta\eta} N$ if M β- or η-reduces to N. We define the multi-step $\beta\eta$-reduction $\to_{\beta\eta}^*$ and $\beta\eta$-equivalence $\equiv_{\beta\eta}$ analogously, and so do we the notion of $\beta\eta$-normal form. $\beta\eta$-reduction retains many of the properties of β-reduction: most notably, the Church-Rosser theorem and its consequences still hold with β replaced by $\beta\eta$.

Particularly, we will often use the notion of $\beta\eta$-equivalence in this thesis; when we write $M \equiv N$, it should be always understood as $M \equiv_{\beta\eta} N$ unless otherwise stated. $\to$ and $\to^*$, however may refer to just β-reduction or $\beta\eta$-reduction, depending on the context.

2.1.2 Simple types

Not all terms in untyped λ-calculus have normal forms. For example, the term $(\lambda x.x\ x)\ (\lambda x.x\ x)$ reduces to itself, and thus does not have a normal form. To solve this problem, we introduce **types** to classify terms and define the valid usages of terms. For example, only a function could be applied. The **simply-typed λ-calculus** (STLC) is the simplest of type systems for λ-calculi; it is also the simplest example of a **type theory**, or a logical theory of types and terms (usually λ-calculus terms), which is the central topic of this thesis.

In STLC, we allow for the following types:

$$
\begin{array}{rr}
\tau ::= \mathbf{1} & \text{base type} \\
\mid \tau \to \tau & \text{function type} \\
\mid \tau \times \tau & \text{product type}
\end{array}
$$

Figure 2.5: types of STLC

We extend the untyped λ-calculus with four new terms, (M, N), $\mathsf{fst}(M)$, $\mathsf{snd}(M)$ and tt, to accomodate the new constructions. The reduction rules for these constructions are as following (tt is normal and does not further reduce). Other β-reduction rules remain the same, and η-reductions behave as expected.

$$
\textsc{Red-Pair-1}\ \frac{M \to M'}{(M, N) \to (M', N)} \qquad
\textsc{Red-Pair-2}\ \frac{N \to N'}{(M, N) \to (M, N')} \qquad
\textsc{Red-Fst-In}\ \frac{M \to M'}{\mathsf{fst}(M) \to \mathsf{fst}(M')}
$$

$$
\textsc{Red-Fst}\ \frac{}{\mathsf{fst}(M, N) \to M} \qquad
\textsc{Red-Snd-In}\ \frac{M \to M'}{\mathsf{snd}(M) \to \mathsf{snd}(M')} \qquad
\textsc{Red-Snd}\ \frac{}{\mathsf{snd}(M, N) \to N}
$$

Figure 2.6: extra reduction rules for STLC

Hereafter, when we introduce new constructors like $(-, -)$ and fst, we always assume that reduction can occur "under" the constructor, analogous to Red-Abs and Red-Pair-1.

Function types behave like functions, and product types behave like cartesian products in set theory, or rather categorical products in category theory. The type $\mathbf{1}$ behaves similarly to the one-element set in set theory, or the terminal object in category. Later, we will also define and prove these similarities formally.

In STLC, we consider only well-typed terms; terms that could not be typed are not considered STLC terms. We write $M : \tau$ if the closed term M has the type τ. Non-closed terms can also be typed, given that we know the type of all free variables in the term. For this purpose, we introduce the notion of a (typing) **context**, which is simply a mapping from variables to types. Then, under an appropriate context Γ, any closed or open term in STLC can be given a type. We write $\Gamma \vdash M : \tau$ if M has type τ under the context Γ. Particularly, if M is closed, we often write $\vdash M : \tau$ instead of just $M : \tau$, to emphasize that M is closed, and can be typed without an external context.

The typing relation is formally defined by the following rules (in Figure 2.7). Note that the abstraction form λ has been slightly modified to include a type annotation, i.e. information of the type of x[1]. The reduction rules remain unchanged.

[1]Strictly speaking, this isn't necessary. Our presentation of STLC is sometimes called "Church-style". If we use unannotated λs, we run into the problem of well-typed and ill-typed terms, so we try to avoid this presentation.

$$\textsc{Typ-Var}\ \frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau} \qquad \textsc{Typ-TT}\ \frac{}{\Gamma \vdash \mathsf{tt} : \mathbf{1}} \qquad \textsc{Typ-Abs}\ \frac{\Gamma, x : \tau \vdash M : \tau'}{\Gamma \vdash \lambda(x : \tau).M : \tau \to \tau'}$$

$$\textsc{Typ-App}\ \frac{\Gamma \vdash M : \tau \to \tau' \qquad \Gamma \vdash N : \tau}{\Gamma \vdash M\ N : \tau'} \qquad \textsc{Typ-Pair}\ \frac{\Gamma \vdash M : \tau \qquad \Gamma \vdash N : \tau'}{\Gamma \vdash (M, N) : \tau \times \tau'} \qquad \textsc{Typ-Fst}\ \frac{\Gamma \vdash M : \tau \times \tau'}{\Gamma \vdash \mathsf{fst}(M) : \tau}$$

$$\textsc{Typ-Snd}\ \frac{\Gamma \vdash M : \tau \times \tau'}{\Gamma \vdash \mathsf{snd}(M) : \tau'}$$

Figure 2.7: typing rules for STLC

A proof of $\Gamma \vdash M : \tau$ is called a *typing derivation*, and usually proceed in the style of natural deduction. We present an example below:

Example 2.1.5. The type of the term $\lambda(x : \tau \times \tau').\mathsf{fst}(x)$ is $(\tau \times \tau') \to \tau$.

Proof. Natural deduction proof. See derivation (Figure 2.8). □

$$\textsc{Typ-Abs}\ \frac{\textsc{Typ-Fst}\ \dfrac{\dfrac{}{x : \tau \times \tau' \vdash x : \tau \times \tau'}}{x : \tau \times \tau' \vdash \mathsf{fst}(x) : \tau}}{\vdash \lambda(x : \tau \times \tau').\mathsf{fst}(x) : (\tau \times \tau') \to \tau}$$

Figure 2.8: proof of Example 2.1.5

Additionally, the typing derivations of STLC satisfy three *structural* rules (see Figure 2.9).

$$\textsc{Weakening}\ \frac{\Gamma \vdash M : \tau}{\Gamma; x : \sigma \vdash M : \tau} \qquad \textsc{Contraction}\ \frac{\Gamma; x : \sigma; y : \sigma \vdash M : \tau}{\Gamma; x : \sigma \vdash M[y := x] : \tau} \qquad \textsc{Exchange}\ \frac{\Gamma; x : \sigma; y : \varphi; \Sigma \vdash M : \tau}{\Gamma; y : \varphi; x : \sigma; \Sigma \vdash M : \tau}$$

Figure 2.9: Structural rules for STLC derivations

All type theories in the rest of this thesis satisfy all three structural rules of weakening, contraction and exchange, unless otherwise stated.

Having defined the notion of types, we would like to first ascertain that typing is well-defined. More specifically, we would like to prove the following theorem:

Theorem 2.1.4 (uniqueness of typing)**.** *If* $\Gamma \vdash M : \tau$ *and* $\Gamma \vdash M : \sigma$*, then* $\tau = \sigma$*.*

To prove this theorem, we use a technique called *structural induction*, which will be used throughout this thesis:

Proof. We perform (structural) induction on the structure of M.

- $M = x$. This is equivalent to showing that Γ defines a function on variables. Assuming that Γ is not ill-formed (i.e. contain two entries of x), one can perform (regular mathematical) induction on the size of Γ, and apply the structural rules of weakening and exchange, to show this;
- $M = \lambda(x : \alpha).N$. Necessarily, $\tau = \alpha \to \beta$ and $\Gamma; x : \alpha \vdash N : \beta$ for some type β. By the inductive hypothesis, we may assume that if $\Gamma; x : \alpha \vdash N : \beta'$, then necessarily $\beta = \beta'$. By the typing rule Typ-Abs, we know that if $\Gamma \vdash M : \sigma$, then $\sigma = \alpha \to \beta'$ for some β', where $\Gamma; x : \alpha \vdash N : \beta'$. However, by the IH, we know that $\beta = \beta'$. Therefore $\tau = \sigma$.

The induction for other cases proceed similarly. □

At the beginning of this section, we have mentioned that untyped λ-calculus has the problem that not all terms have a normal form. Introducing typing was our proposed solution, and as we have now defined the theory of STLC, we can prove that this is indeed a solution to the problem.

Theorem 2.1.5 ((strong) normalization of STLC)**.** *Every well-typed closed term in STLC has a $\beta\eta$-normal form. In other words, if $\vdash M : \tau$, then there is a $\beta\eta$-normal term N such that $M \to^* N$.*

Proof. See [SU06], section 4.4. □

Moreover, the Church-Rosser property and all of its corollaries continue to hold for the STLC, in both the β and $\beta\eta$ cases.

The theory of simply typed λ-calculus is much more complex, although most of the theory is out of the scope of this thesis. The interested reader is referred to a text on λ-calculi, such as [SU06]. We will, however, prove one result that is essential to the categorical theory of STLC, which we will develop in the next section.

Theorem 2.1.6 ($\beta\eta$-equivalence respects typing)**.** *Equivalent terms have the same type. In other words, if $\vdash M : \tau$ and $M \equiv N$, then $\vdash N : \tau$.*

To prove this theorem, we first prove the following lemmas:

Lemma 2.1.7 (subject reduction)**.** *If $\vdash M : \tau$ and $M \to_\beta N$, then $\vdash N : \tau$.*

Proof. Induction on the structure of the typing derivation $\Gamma \vdash M : \tau$ (where $\Gamma = \{\}$).

- Typ-Var and Typ-TT. The derivation could not be of these forms because neither a variable nor `tt` could further reduce.
- Typ-Abs. If the last rule applied is Typ-Abs, then $M = \lambda(x : \sigma).S$, and $\tau = \sigma \to \delta$. By induction principle, if $S \to S'$, then $x : \sigma \vdash S' : \delta$. Since the only possible reduction for an abstraction form is Red-Abs, we know that necessarily $N = \lambda(x : \sigma).S'$. Then by applying the rule Typ-Abs and the induction principle, we know immediately that $\vdash \lambda(x : \sigma).S' : \sigma \to \delta$ i.e. $\vdash N : \tau$.

- Typ-App. If the last rule applied is Typ-App, then it must be the case that $M = S\ T$, where $\vdash S : \sigma \to \tau$ and $\vdash T : \sigma$. Here, there are three possible reduction paths for M.

  1. M reduces by the rule Red-App-1, i.e. $N = S'\ T$. By induction principle, we know that if $S \to S'$, then $\vdash S' : \sigma \to \tau$. By the rule Typ-App we know that $\vdash S'\ T : \tau$ i.e. $\vdash N : \tau$.
  2. M reduces by the rule Red-App-2. The proof is essentially the same as the previous case.
  3. M reduces by the rule Red-Beta. We know that necessarily $S = \lambda(x : \sigma).R$, and $N = R[x := T]$. This is the interesting case, and we pose the general case of this as the next lemma.

- Typ-Pair, Typ-Fst and Typ-Snd. The proof is essentially same as the proof for Typ-Abs and the first two cases in Typ-App.

□

Lemma 2.1.8. *If* $\Gamma; x : \sigma \vdash M : \tau$ *and* $\vdash N : \sigma$*, then* $\Gamma \vdash M[x := N] : \tau$*.*

Proof. Induction on the structure of M. □

Corollary 2.1.9. *If* $\vdash M : \tau$ *and* $M \to_\beta^* N$*, then* $\vdash N : \tau$*.*

Proof. By the transitivity of $\to_\beta^*$ and Lemma 2.1.7. □

Corollary 2.1.10. *If* $M \equiv_\beta N$ *and* $\vdash M : \tau$*, then* $\vdash N : \tau$*.*

Proof. If $M \equiv_\beta N$, then by the Church-Rosser property of STLC, there is a term T such that $M \to_\beta^* T$ and $N \to_\beta^* T$, and by Corollary 2.1.9, $\vdash T : \tau$. Suppose $\vdash N : \tau'$, then $\vdash T : \tau'$. By the uniqueness of typing, $\tau = \tau'$, so $\vdash N : \tau$. □

We have essentially proven Theorem 2.1.6 for β-reductions. We will follow a similar procedure for the η case.

Lemma 2.1.11. *If* $\vdash M : \tau$ *and* $M \to_\eta N$*, then* $\vdash N : \tau$*.*

Proof. By induction on the typing judgment $\Gamma \vdash M : \tau$; the only non-trivial case is Typ-Abs combined with Eta. Here, $M = \lambda(x : \sigma).T\ x$, and $\tau = \sigma \to \delta$. By the rule Typ-Abs, we know that $\Gamma; x : \sigma \vdash T\ x : \delta$. By the rule Typ-App, we have $\Gamma; x : \sigma \vdash T : \sigma \to \delta$. Since x is free in T, by the contraction rule we have $\Gamma \vdash T : \sigma \to \delta$, i.e., $\Gamma \vdash T : \tau$. □

Corollary 2.1.12. *If* $M \equiv_\eta N$ *and* $\vdash M : \tau$*, then* $\vdash N : \tau$*.*

Proof. Identical to the proof for Corollary 2.1.10. □

Proof of Theorem 2.1.4. Since $\equiv_{\beta\eta}$ is just the union of $\equiv_\beta$ and $\equiv_\eta$, this follows by Corollary 2.1.10 and Corollary 2.1.12. □

2.1.3 The Curry-Howard correspondence

The simply typed λ-calculus is not only a computational system[2] but also a proof system. There are some evident similarities between STLC and natural deduction based proof systems. Consider the following example of a natural deduction proof of $\varphi \wedge \psi \Rightarrow \varphi$ in intuitionistic propositional logic[3] (hereafter also constructive propositional logic):

$$\Rightarrow\text{-}\textsc{Intro}\ \dfrac{\wedge\text{-}\textsc{ElimL}\ \dfrac{\varphi \wedge \psi \vdash \varphi \wedge \psi}{\varphi \wedge \psi \vdash \varphi}}{\vdash \varphi \wedge \psi \Rightarrow \varphi}$$

Figure 2.10: Natural deduction proof of $\varphi \wedge \psi \Rightarrow \varphi$

This derivation is basically identical to the typing derivation for $\lambda(x : \tau \times \tau').\mathsf{fst}(x)$. We may think of a type of STLC as a proposition in intuitionistic propositional logic with $\wedge$ and $\top$ (IPL), and a typing derivation as a natural deduction proof of the proposition. The terms in STLC, thus, can be thought of proofs of propositions. This identification is called the proposition-as-types interpretation of logic, or the **Curry-Howard correspondence**. We identify STLC types and IPL propositions according to the following table:

STLC type	IPL proposition
$\rightarrow$	$\Rightarrow$
$\times$	$\wedge$
$\mathbf{1}$	$\top$

The difference between STLC types and IPL propositions, thus, can be considered purely notational. In the rest of this thesis, we will not distinguish between STLC types and IPL propositions. The Curry-Howard correspondence for STLC can be formalized as the following theorem:

Theorem 2.1.13 (Curry-Howard correspondence)**.** *Let Γ be a context in STLC, Δ a context (or set of premises) in IPL, M a term in STLC, and φ a type in STLC (or equivalently its interpretation in IPL). Then:*

- *if $\Gamma \vdash M : \varphi$, then $\mathrm{cod}(\Gamma) \vdash \varphi$;*
- *if $\Delta \vdash \varphi$, then there is a term M such that $\Gamma \vdash M : \varphi$, where $\mathrm{cod}(\Gamma) = \Delta$.*

Proof. By structural induction on typing derivations. See also Chapter 5 of [SU06] for details. □

From Theorem 2.1.13, one can see that for logicians (particularly categorical logicians), λ-calculus could be thought as "another way to do logic". The difference

[2] to be precise, it is a term rewriting system (see [BN98] for a precise definition).

[3] Intuitionistic propositional logic is propositional logic without double-negation elimination or any laws equivalent to it, including (but not limited to) the law of excluded middle.

between STLC and IPL can be considered mainly differences in style, but an important difference is that in λ-calculi, one does logic in a *proof-relevant* way. Proofs — or terms — are a primary notion in λ-calculi, as opposed to proofs in logical calculi which are secondary in that they are not part of the typical presentation of logical calculi.

More complex type theories also have Curry-Howard correspondences, as we shall see in the following sections. It is important to note, however, that all logics we consider in this thesis are intuitionistic/constructive (that is, they do not validate double-negation elimination $\neg\neg\varphi \Rightarrow \varphi$ or the law of excluded middle $\varphi \vee \neg\varphi$), although we might work in a classical (i.e., non-constructive) metatheory. This can be explained by the fact that the use of double-negation elimination or the law of excluded middle necessarily gives non-constructive proofs, and cannot be expressed as terms in the λ-calculus because non-constructive proofs, by definition, could not be computed[4].

We do not introduce the proof theory of propositional, first-order and/or higher-order logical calculi, as this is out of the focus of this thesis. Whenever we use results relating to these logical calculi, we will cite the appropriate references.

2.2 STLC and cartesian closed categories

2.2.1 Syntactic categories

Every λ-calculus gives rise to a category called its *syntactic* category, since the category is based on the syntax of λ-calculus:

Definition 2.2.1 (syntactic category (Section 1.11, [LS86]))**.** Let $\mathcal{L}$ be a type theory, i.e. theory of a certain typed λ-calculus. The following data form a category $\mathbf{Syn}(\mathcal{L})$, called the **syntactic category** of $\mathcal{L}$:

- the objects of $\mathbf{Syn}(\mathcal{L})$ are the types of $\mathcal{L}$;
- the morphisms $\tau \to \sigma$ are sequents of the form $x : \tau \vdash M(x) : \sigma$, up to $\alpha\beta\eta$-equivalence in $M(x)$;
- the identity morphism at each object τ is the sequent $x : \tau \vdash x : \tau$;
- the composition of morphisms is given by the composition of λ-terms. Given two sequents i.e. morphisms $x : \tau \vdash M(x) : \sigma$ and $y : \sigma \vdash N(y) : \delta$, their composition is $x : \tau \vdash N(M(x))$.

Frequently, we will also identify the sequent $x : \tau \vdash M(x) : \sigma$ with the λ-term $\lambda(x : \tau).M(x)$; the two styles are essentially equivalent. Hereafter, we sometimes drop the annotation τ on x, if the annotation can be easily inferred. It is evident

[4]In this thesis, by "computation", we mean pure computation, which means that we compute using only mathematical functions that depend only on the value of their arguments. Some non-constructive proofs can actually be encoded in the λ-calculus if we allow impure computation, i.e. when the value of computation depends not only on the inputs. A reader familiar with programming will realize that this is actually closer to the usual conception of a program, which will almost inevitably interact with the external environment by e.g. reading from a file, taking input, etc.

that the given data form a category. We sometimes write $[\tau]$ if we want to emphasize that we are working in the syntactic category, and considering τ as an object of this category.

The subject of the previous section is the simply typed λ-calculus, and naturally one wants to consider the $\mathbf{Syn}(\lambda_\to)$, where $\lambda_\to$ is the theory of STLC (with $\mathbf{1}$ and $\times$). It turns out that $\mathbf{Syn}(\lambda_\to)$ is a quite nice category; namely, it is cartesian closed.

Theorem 2.2.1. *The syntactic category of STLC,* $\mathbf{Syn}(\lambda_\to)$*, is cartesian closed.*

Proof (Section 1.11, [LS86]). There are three requirements for a category to be a CCC: a terminal object, all binary products, and all exponentials. We construct them in $\mathbf{Syn}(\lambda_\to)$ as following:

- the terminal object is $[\mathbf{1}]$, and the unique map into the terminal object is given by $x : \tau \vdash \mathsf{tt} : \mathbf{1}$;
- the binary product $[\tau] \times [\sigma]$ is $[\tau \times \sigma]$, with the two canonical projections given by $p : \tau \times \sigma \vdash \mathsf{fst}(p)$ and $p : \tau \times \sigma \vdash \mathsf{snd}(p)$ respectively;
- the exponential object $[\sigma]^{[\tau]}$ is $[\tau \to \sigma]$;
- the evaluation morphism $[\sigma]^{[\tau]} \times \tau \to \sigma$ is given by $(f, x) : \tau \to \sigma \times \tau \vdash f\ x$.

$\mathbf{1}$ is indeed terminal since tt is its only constructor, and the rules for fst and snd guarantee that the type-theoretic product is indeed the categorical product. It is, however, tedious to directly verify the universal property of exponentials; we can instead use the equational definition given in Lemma 1.3.1.

To simplify calculations, we identify sequents with λ-terms (this is no more than a notational preference). The evaluation morphism ev is identified with the term $\lambda z.\mathsf{fst}(z)\ \mathsf{snd}(z)$, and for any $f : \delta \times \tau \to \sigma$, we define the transpose $\tilde{f} : \delta \times \sigma^\tau$ as the term $\lambda z.\lambda x.f\ (z, x)$. Both equations can be proven by $\beta\eta$-normalizing the left hand side of the equation. □

2.2.2 Interpreting STLC in CCCs

We have just proven that STLC naturally gives rise to a CCC. Now, we will go a step further and show that STLC could be interpreted in any CCC. However, we first need to define precisely the notion of "interpretation".

In the rest of this thesis, when we use STLC, we will allow a number of base types, and a number of uninterpreted constants belonging to each type. These constants are terms, but there are no reduction rules for them (hence the name "constants"); the base types can form products and functions like other types, but there are no new rules added to the calculus.

Definition 2.2.2 (model of a type theory)**.** A model $\mathcal{M}$ of a type theory $\mathcal{L}$ consists of the following data (see, e.g., [Hu20]):

- a collection called the *domain* of the model; here, we consider only the case where the domain forms a category $\mathcal{C}$;
- an assignment $[\![-]\!]$ from types of $\mathcal{L}$ to the objects of $\mathcal{C}$;

- an assignment $[\![-]\!]$ from contexts, satisfying $[\![\cdot]\!] = 1$ and $[\![\Gamma; x : \tau]\!] = [\![\Gamma]\!] \times [\![\tau]\!]$;
- an assignment $[\![\Gamma \vdash M : \tau]\!]$ for every valid typing judgment from the terms of type τ to morphisms $[\![\Gamma]\!] \to [\![\tau]\!]$. Sometimes, when Γ and τ are understood, we also write $[\![M]\!]$.

A model is also called a *denotational semantics*, particularly in computer science. Here, we stick to the term "model", or sometimes "semantics".

Example 2.2.3 (set-theoretic model of STLC)**.** The construction is primarily known in mathematical folklore, but a written description can be found in the Chapter 10 of [Sel13]. Take the category **Set**. We can show that **Set** is a model of $\lambda_{\to}$. The types are interpreted as follows:

$$\begin{aligned} [\![\mathbf{1}]\!] &= \{*\} \\ [\![\sigma \to \tau]\!] &= [\![\sigma]\!] \to [\![\tau]\!] \\ [\![\sigma \times \tau]\!] &= [\![\sigma]\!] \times [\![\tau]\!]. \end{aligned}$$

Note the abuse of notation here $\to$ means function type on the left-hand side, and function space (the set of all functions) on the right-hand side. Similarly, $\times$ means product type on the left-hand side, and the cartesian product on the right-hand side.

Next, we interpret the terms of STLC. We first consider the basic case of a variable; for the sake of simplicity, consider the case where $\Gamma = x_1 : \tau_1; ...; x_n : \tau_n$, and the judgment $x_1 : \tau_1; ...; x_n : \tau_n \vdash x_i : \tau_i$. Here, $[\![\Gamma]\!] = [\![\tau_1]\!] \times ... \times [\![\tau_n]\!]$, and naturally we define $[\![x_i]\!] = \pi_i$ the projection function.

The case of λ-abstraction is similarly simple. For $[\![\Gamma \vdash \lambda(x : \sigma).M : \sigma \to \tau]\!]$, we consider $[\![\Gamma; x : \sigma \vdash M : \tau]\!] = f : [\![\Gamma]\!] \times [\![\sigma]\!] \to [\![\tau]\!]$. Naturally, we define $[\![\Gamma \vdash \lambda(x : \sigma).M : \sigma \to \tau]\!] = \tilde{f} : [\![\Gamma]\!] \to ([\![\sigma]\!] \to [\![\tau]\!])$.

The other cases are interpreted as follows:

$$\begin{aligned} [\![\Gamma \vdash M\ N : \tau]\!] &= \mathrm{ev} \circ ([\![\Gamma \vdash M : \sigma \to \tau]\!], [\![\Gamma \vdash N : \sigma]\!]) \\ [\![\Gamma \vdash (M, N) : \sigma \times \tau]\!] &= ([\![\Gamma \vdash M : \sigma]\!], [\![\Gamma \vdash N : \tau]\!]) \\ [\![\Gamma \vdash \mathsf{fst}(M) : \sigma]\!] &= \pi_1 \circ [\![\Gamma \vdash M : \sigma \times \tau]\!] \\ [\![\Gamma \vdash \mathsf{snd}(M) : \tau]\!] &= \pi_2 \circ [\![\Gamma \vdash M : \sigma \times \tau]\!] \\ [\![\Gamma \vdash \mathsf{tt} : \mathbf{1}]\!] &= - \mapsto *. \end{aligned}$$

It should be noted that the construction of a set-theoretic model of STLC above uses only standard constructions in CCCs, namely products and functions (i.e. exponentials). Thus, even though the construction is for **Set**, the same construction can be used for any CCC. This proves the following main theorem:

Theorem 2.2.2. *Any CCC is a model of STLC.*

Proof. Types are interpreted as follows:

$$\begin{aligned} [\![\mathbf{1}]\!] &= 1 \\ [\![\sigma \to \tau]\!] &= [\![\tau]\!]^{[\![\sigma]\!]} \\ [\![\sigma \times \tau]\!] &= [\![\sigma]\!] \times [\![\tau]\!]. \end{aligned}$$

Interpretation of terms are the same as in Example 2.2.3, replacing function spaces with exponentials and cartesian products with categorical products. tt is interpreted as the unique morphism into 1.

This construction is well-known in the folklore among the researchers of λ-calculi and categorical logic, but without explicit written references. □

However, in the previous construction, we were considering terms, not equivalence classes of terms, i.e., terms up to $\beta\eta$-equivalence. We need to make sure that our construction respects the equivalence of terms.

Definition 2.2.4 (soundness)**.** A model $\mathcal{M}$ for $\mathcal{L}$ is sound if for any $M \equiv N : \tau$, we have $[\![M]\!] = [\![N]\!]$.

Theorem 2.2.3. *The CCC model of STLC is sound.*

Lemma 2.2.4. *If* $[\![\Gamma; x : \sigma \vdash M : \tau]\!] = f : [\![\Gamma]\!] \times [\![\sigma]\!] \to [\![\tau]\!]$ *and* $[\![\Gamma \vdash N : \sigma]\!] = g : [\![\Gamma]\!] \to [\![\sigma]\!]$*, then* $[\![\Gamma \vdash M[x := N] : \tau]\!] = f \circ (\mathsf{id}, g) : [\![\Gamma]\!] \to [\![\tau]\!]$.

Proof. Induction on the structure of M. □

Proof of Theorem 2.2.3. Induction on the structure of the derivation $\Gamma \vdash M : \tau$. We need only to consider the case where $M \to_{\beta\eta} N$; the other cases follow by transitivity and Church-Rosser.

Here we prove only one case (where $M \to_{\beta\eta} N$ by Red-Beta):

$$\begin{aligned}
[\![\Gamma \vdash (\lambda(x : \sigma).S)\ T : \tau]\!] &= \mathrm{ev} \circ (\widetilde{[\![\Gamma; x : \sigma \vdash S : \tau]\!]}, [\![\Gamma \vdash T : \sigma]\!]) \\
&= \mathrm{ev} \circ (\widetilde{[\![\Gamma; x : \sigma \vdash S : \tau]\!]}, \mathsf{id}) \circ (\mathsf{id}, [\![\Gamma \vdash T : \sigma]\!]) \\
&= [\![\Gamma; x : \sigma \vdash S : \tau]\!] \circ (\mathsf{id}, [\![\Gamma \vdash T : \sigma]\!]) \\
&= [\![\Gamma \vdash S[x := T] : \tau]\!].
\end{aligned}$$

Other cases can be proven similarly. □

Corollary 2.2.5 (functoriality of $[\![-]\!]$)**.** *Let $\mathcal{C}$ be any CCC. Then $[\![-]\!]$ defines a functor $\mathbf{Syn}(\lambda_\to) \to \mathcal{C}$ described as follows:*

- *any object i.e. type τ in $\mathbf{Syn}(\lambda_\to)$ is mapped to the object $[\![\tau]\!] : \mathcal{C}$;*
- *any sequent $x : \sigma \vdash M(x) : \tau$ is mapped to the morphism $[\![x : \sigma \vdash M(x) : \tau]\!] : [\![\sigma]\!] \to [\![\tau]\!]$.*

Proof. Clearly $[\![x : \sigma \vdash x : \sigma]\!] = \mathsf{id}$ (the product of one object is itself, and the canonical projection is just the identity morphism).

We need to show that $[\![-]\!]$ respects composition of morphisms. Given two morphisms $x : \sigma \vdash M(x) : \tau$ and $y : \tau \vdash N(y) : \delta$, the composition of them is $x : \sigma \vdash N(M(x)) : \delta$. To show that $[\![x : \sigma \vdash N(M(x)) : \delta]\!] = [\![y : \tau \vdash N(y) : \delta]\!] \circ [\![x : \sigma \vdash M(x) : \tau]\!]$, we can proceed by a simple induction on M and N. □

The familiar Heyting algebra semantics of STLC is just a special case of the interpretation of STLC by CCCs.

Example 2.2.5 (Heyting algebra semantics of STLC)**.** A Heyting algebra H is a sound model of STLC.

Proof. A Heyting algebra is a CCC if considered as a category, so any Heyting algebra is a sound model of STLC. □

2.2.3 The internal language of a category

In the previous part, we have shown how to construct a CCC from STLC. We may naturally ask if, in the reverse direction, we could generate a language based on a CCC. This is in fact possible, and the generated language is called the *internal language* of a CCC. In general, any category with enough structure has an internal language.

We define the internal language of a CCC (the definition is adapted from Section 5, Chapter 6 of [MM92]). Let $\mathcal{C}$ be a CCC, then its internal language $\mathcal{L}(\mathcal{C})$ is defined as following:

- the types are the objects of $\mathcal{C}$. Here, we will not distinguish between objects X of $\mathcal{C}$ and types X of $\mathcal{L}(\mathcal{C})$;
- each variable $x : X$ is a term of type X, interpreted as the identity morphism id_X;
- a term $\phi(u, v, ...)$ of type X, containing free variables $u : U$, $v : V$, etc., will be interpreted as a morphism $U \times V \times ... \to X$[5];
- for two terms $M : X$ and $N : Y$, interpreted by the arrows $f : U \to X$ and $g : V \to Y$ respectively, there is a term (M, N) which is the product of M and N, interpreted by the categorical product (f, g);
- for a term $M : X$ and $f(x) : Y$, interepreted by the arrows $\sigma : U \to X$ and $f : X \to Y$ respectively, there is a term $f \circ M$, interpreted by the categorical composite $f \circ \sigma$;
- for a term $f : X \to Y$ and $M : X$, interpreted by the arrows $\theta : U \to Y^X$ and $\sigma : V \to X$, there is a term $f(M) : Y$ interpreted by evaluation:

$$U \times V \xrightarrow{(\theta,\sigma)} Y^X \times X \xrightarrow{\text{ev}} Y;$$

- for a variable $x : X$ and a term M interpreted by the morphism $\sigma : X \times U \to Y$, there is a term $\lambda x.M : X \to Y$ (or perhaps $x \mapsto M$), interpreted by the transpose of σ, $\tilde{\sigma} : U \to Y^X$.

One can see that taking the internal language of a CCC is essentially the inverse operation of taking the semantics of lambda calculus in the such CCC. We have the following result:

Proposition 2.2.6. *The internal language of* $\mathbf{Syn}(\lambda_\to)$ *is* $\lambda_\to$.

Proof. By verifying each construction. □

Essentially, the internal language of a CCC is STLC. Therefore, we often say, informally, that STLC is "equivalent" to (the category of) CCCs.

[5] if a term is closed, then of course the morphism will come from the nullary product, i.e., 1.

Chapter 3

Dependent type theory and presheaf semantics

3.1 Martin-Löf type theory

The simply-typed λ-calculus corresponds to (intuitionistic) propositional logic, which is a simple logical system. However, IPL is far from sufficient for doing mathematics. For example, it lacks quantifiers ($\forall$ and $\exists$) and equalities, making it impossible for complex mathematics to be developed. In fact, in IPL it is not even possible to develop Peano arithmetic, so a stronger system is obligatory for substantial mathematics.

Martin-Löf proposed a system [Mar72], nowadays often called Martin-Löf type theory (or intuitionistic/constructive type theory), that serves as a constructive foundation for first order and higher-order reasoning and computation. Here, we present Martin-Löf's type theory as an example of **dependent type theory**, or a type theory where types could depend on terms, allowing quantifiers to be encoded.

We begin with the syntax of MLTT. In MLTT, there is no syntactic distinction between terms and types; types are just special terms. We allow all terms and types of STLC in MLTT, but we add two more type-terms (see Figure 3.1). λ and $(-,-)$ are the introduction forms for Π and Σ, respectively, and $\to$ and $\times$ could be thought of as special cases of Π and Σ, in which M does not actually contain x. In the rest of this thesis, we use the notation $\to$ and $\times$ when appropriate, but do not treat them separately.

$M ::= ...$	all terms/types in STLC
$\mid \Pi(x : M).M$	Π-/dependent function types
$\mid \Sigma(x : M).M$	Σ-/dependent product types
$\mid \mathcal{U}$	universe of types

Figure 3.1: terms of MLTT

Rather confusingly, in the literature, Π and Σ are often called "dependent product" and "dependent sum" types, respectively. However, we believe such

naming is not only perplexing, but also distracting as they do not conform to the actual behavior of the types. Therefore, we will never use this alternate set of notations in our thesis.

Next, we define the typing rules for MLTT. The typing rules of MLTT are significantly more involved than those for STLC, for various reasons. There is no syntactic-level distinction between types and terms, but not all terms are semantically types. Therefore, we first need a judgment $\Sigma \vdash \tau\ \mathsf{type}$, meaning that "$\tau$ is a type". Furthermore, we need that all contexts are well-formed, in that all contexts indeed map variables to types; we write $\Gamma\ \mathsf{ctx}$ for the judgment "Γ is a well-formed context". The rules of type and context well-formedness is defined in Figure 3.2.

Hereafter, we will generally use the convention that an upper case denotes a type, and a lower case letter denotes a term that does not form a type.

$$\textsc{Wf-Var}\ \frac{\Gamma\ \mathsf{ctx} \qquad \Gamma(x) = \tau \qquad \Gamma \vdash \tau\ \mathsf{type}}{\Gamma \vdash x\ \mathsf{type}} \qquad \textsc{Wf-}\Pi\ \frac{\Gamma \vdash A\ \mathsf{type} \qquad \Gamma; x : A \vdash B\ \mathsf{type}}{\Gamma \vdash \Pi(x : A).B\ \mathsf{type}}$$

$$\textsc{Wf-}\Sigma\ \frac{\Gamma \vdash A\ \mathsf{type} \qquad \Gamma; x : A \vdash B\ \mathsf{type}}{\Gamma \vdash \Sigma(x : A).B\ \mathsf{type}} \qquad \textsc{Wf-}\mathbf{1}\ \frac{\Gamma\ \mathsf{ctx}}{\vdash \mathbf{1}\ \mathsf{type}} \qquad \textsc{Wf-Empty}\ \frac{}{\vdash \{\}\ \mathsf{ctx}}$$

$$\textsc{Wf-Ctx}\ \frac{x_1 : A_1; ...; x_{n-1} : A_{n-1} \vdash A_n\ \mathsf{type}}{\{x_1 : A_1; ...; x_n : A_n\}\ \mathsf{ctx}}$$

Figure 3.2: type and context well-formedness for MLTT

Then, we can define the typing rules for MLTT (see Figure 3.3). We follow mainly Appendix A.2 of [Uni13] in presenting the typing rules for MLTT.

Since in MLTT, types may consist of variables and/or terms, we must have a systematic way of determining if two types are the same. If two types are "the same", then they may be converted seamlessly. We call this relation **definitional equality/equivalence** (or *judgmental equality*, or *intensional equivalence/equality*), meaning that the two terms are *defined* to be equivalent. This is different from the $\beta\eta$-equivalence, or *observational equality*, that we have defined in the previous chapter.

Definitional equality, unlike observational equality, is a *typed* relation, meaning that the equality of two terms is associated with a specific type. We write $\Gamma \vdash a \equiv b : A$ if a and b are definitionally equal, both having type A. Definitional equality is defined by the following rules (Figure 3.4, again, following mainly Appendix A.2 of [Uni13]). The rule Typ-Conv explains how definitional equality is essential to type checking: since types can contain arbitrary terms, it is sometimes necessary to convert across equivalences to properly give types to terms.

We also assume that definitional equality is a *congruence*, meaning that constructors preserve equality. For example, if $\Gamma \vdash b \equiv b' : B$, then we also have $\Gamma \vdash \lambda(x : A).b \equiv \lambda(x : A).b' : \Pi(x : A).B$, et cetera. We do not write down explicitly these rules, because there are numerous of them. However, whenever we

$$\textsc{Typ-Type}\ \frac{\Gamma \vdash A\ \mathsf{type}}{\Gamma \vdash A : \mathcal{U}} \qquad \textsc{Typ-Var}\ \frac{\Gamma(x) = A}{\Gamma \vdash x : A} \qquad \textsc{Typ-Abs}\ \frac{\Gamma \vdash A\ \mathsf{type} \qquad \Gamma; x : A \vdash b : B}{\Gamma \vdash \lambda(x : A).b : \Pi(x : A).B}$$

$$\textsc{Typ-App}\ \frac{\Gamma \vdash f : \Pi(x : A).B \qquad \Gamma \vdash a : A}{\Gamma \vdash f\ a : B[x := N]}$$

$$\textsc{Typ-Pair}\ \frac{\Gamma; a : A \vdash B\ \mathsf{type} \qquad \Gamma \vdash a : A \qquad \Gamma \vdash b : B[x := a]}{\Gamma \vdash (a, b) : \Sigma(x : A).B} \qquad \textsc{Typ-Fst}\ \frac{\Gamma \vdash p : \Sigma(x : A).B}{\Gamma \vdash \mathsf{fst}(p) : A}$$

$$\textsc{Typ-Snd}\ \frac{\Gamma \vdash p : \Sigma(x : A).B}{\Gamma \vdash \mathsf{snd}(p) : B[x := \mathsf{fst}(p)]} \qquad \textsc{Typ-TT}\ \frac{}{\Gamma \vdash \mathsf{tt} : \mathbf{1}}$$

Figure 3.3: typing for MLTT

introduce a new constructor (like λ and $(-,-)$), we always implcitly assume that it has the property of respecting definitional equality.

$$\textsc{Eq-Refl}\ \frac{\Gamma \vdash a : A}{\Gamma \vdash a \equiv b : A} \qquad \textsc{Eq-Sym}\ \frac{\Gamma \vdash a \equiv b : A}{\Gamma \vdash b \equiv a : A} \qquad \textsc{Eq-Trans}\ \frac{\Gamma \vdash a \equiv b : A \qquad \Gamma \vdash b \equiv c : A}{\Gamma \vdash a \equiv c : A}$$

$$\textsc{Ty-Conv}\ \frac{\Gamma \vdash a : A \qquad \Gamma \vdash A \equiv B : \mathcal{U}}{\Gamma \vdash a : B} \qquad \textsc{Eq-Conv}\ \frac{\Gamma \vdash a \equiv b : A \Gamma \vdash A \equiv B : \mathcal{U}}{\Gamma \vdash a \equiv b : B}$$

$$\textsc{Eq-Beta}\ \frac{\Gamma; x : A \vdash b : B \qquad \Gamma \vdash a : A}{\Gamma \vdash (\lambda(x : A).b)\ a \equiv b[x := a] : B[x := a]}$$

$$\textsc{Eq-Fst}\ \frac{\Gamma; a : A \vdash B\ \mathsf{type} \qquad \Gamma \vdash a : A \qquad \Gamma \vdash b : B[x := a]}{\Gamma \vdash \mathsf{fst}(a, b) \equiv a : A}$$

$$\textsc{Eq-Snd}\ \frac{\Gamma; a : A \vdash B\ \mathsf{type} \qquad \Gamma \vdash a : A \qquad \Gamma \vdash b : B[x := a]}{\Gamma \vdash \mathsf{snd}(a, b) \equiv b : B[x := a]}$$

Figure 3.4: Definitional equality for MLTT

The definition of reduction for MLTT is essentially the same as that of untyped λ-calculus and STLC, and so is the definition of observational, i.e., $\beta\eta$-, equivalence. The proof of strong normalization and of Church-Rosser for MLTT

is notoriously complicated, but we should note here that MLTT is a strongly normalizing system.

It is easy to see that if two terms are definitionally equal, then they are necessarily observationally equal. However, two terms that are observationally equal need not be definitionally equal. We will see the distinction between the two concepts in the following sections.

3.1.1 Curry-Howard correspondence for MLTT

Just like STLC, MLTT also corresponds to a logical calculus. As we have mentioned in the beginning of the section, MLTT corresponds to first-order and higher-order logic. The formalization we currently have allows quantification over terms which are not types, and corresponds to constructive first-order/predicate logic. The Curry-Howard correspondence for MLTT is given by the following table:

MLTT type	FOL proposition
$\to$	$\Rightarrow$
$\Pi(x : A).B$	$\forall(x : A), B$
$\Sigma(x : A), B$	$\exists(x : A), B$
$\mathbf{1}$	$\top$

One can prove a "correspondence" theorem similar to Theorem 2.1.13 for MLTT, using a more complicated structural induction proof.

Sometimes, in MLTT, we also use the notation $\forall$ and $\exists$ for Π and Σ, respectively, when we want to emphasize the fact that a type represents a logical proposition.

3.1.2 Universes

Now, we could no longer hide the fact that the formulation of MLTT that we have just described is in fact slightly flawed. Most notably, we did not allow forming types $\Pi(x : A).B$ and $\Sigma(x : A).B$ where x is in fact a type, since $\mathcal{U}$ is not a "proper" type in its own right.

Naturally, we might ask why not simply add the following rule and make $\mathcal{U}$ a type in its own right:

$$\frac{}{\Gamma \vdash \mathcal{U}\ \mathsf{type}}\ \textsc{Wf-Univ}$$

which has the consequence $\Gamma \vdash \mathcal{U} : \mathcal{U}$.

However, this is in fact very dangerous, as Girard has shown in [Gir72], since it allows a term of any type to be constructed, which is equivalent to being able to prove any proposition, making the type theory inconsistent as a logic. This is analogous to the well-known Russell's paradox in set theory, which states that non-well-founded sets (and thus sets that contain themselves) leads to inconsistency.

But it does not mean that we are out of luck. There is a safe alternative: allow a *countably infinite hierarchy* of universes, indexed by natural numbers $i = 0, 1, ...$, by introducing either set of rules given below (Figure 3.5 or Figure 3.6). Moreover, all occurences of $\mathcal{U}$ in previously introduced rules are to be replaced by $\mathcal{U}_i$, and

A type to be replaced by A type_i, meaning that "A is a type at universe level i". When we are not concerned with the particular universe level, we sometimes drop the level index.

We call the former style (Figure 3.5) **universes à la Russell** (or *cumulative universes*) and the later (Figure 3.6) **universes à la Tarski**. In the latter style, there are two operators on universes and types, the quoting operator $\ulcorner - \urcorner$ and the interpretation (unquoting) operator $\mathsf{El}(-)$, which are inverses of each other.

One can see that the two styles are essentially equivalent. Universes à la Russell, however, are much simpler, so there is the natural question of why introducing universes à la Tarski at all; however, later we will see cumulativity in à la Russell universes are quite difficult to model, and that à la Tarski universes correspond better to models of MLTT, so we will prefer to use universes à la Tarski when working with models of type theory. However, we will often use the à la Russell style merely as a shorthand; one can recover the à la Tarski style by adding quoting and unquoting operators as appropriate.

$$\textsc{Wf-Univ}\ \frac{}{\Gamma \vdash \mathcal{U}_i\ \mathsf{type}_{i+1}} \qquad \textsc{Typ-Univ}\ \frac{\Gamma \vdash A\ \mathsf{type}_i}{\Gamma \vdash A : \mathcal{U}_i} \qquad \textsc{Typ-Cumul}\ \frac{\Gamma \vdash A : \mathcal{U}_i}{\Gamma \vdash A : \mathcal{U}_{i+1}}$$

Figure 3.5: Universes à la Russell

$$\textsc{Wf-Univ}\ \frac{}{\Gamma \vdash \mathcal{U}_i\ \mathsf{type}_{i+1}} \qquad \textsc{Typ-Univ}\ \frac{\Gamma \vdash A\ \mathsf{type}_i}{\Gamma \vdash \ulcorner A \urcorner : \mathcal{U}_i} \qquad \textsc{Typ-Interp}\ \frac{\Gamma \vdash A : \mathcal{U}_i}{\Gamma \vdash \mathsf{El}(A)\ \mathsf{type}_i}$$

Figure 3.6: Universes à la Tarski

With universes, we can define type-valued functions, i.e., terms with type $\Pi(x : A).\mathcal{U}_i$. We also call such functions **type families**.

3.1.3 The identity type

In MLTT, sometimes we also include a type $a =_A b$, called the **identity type**. This type encodes the concept that two terms are (observationally) equal, and is different from the definitional concept of equality, which is only meta-theoretic and is not a type. We call this notion of equality **propositional equality**, as it corresponds to the equality proposition $a = b$ in predicate logic.

For each term $a : A$, there is a term $\mathsf{refl}_a : a =_A a$. This is the only way to construct a term of type $a =_A b$. For each type $a =_A b$, there is a term $\mathcal{J}_A$, called the *eliminator* of $a =_A b$, which we define below. The typing, reduction and equivalence rules for the identity type are given below, in Figure 3.7. The rule by which $\mathcal{J}_A$ reduces is also called *rule J* or *axiom J*; the exotic naming is due to Martin-Löf [Mar72].

$$\textsc{Wf-}=_A\quad \frac{\Gamma \vdash A\ \mathsf{type} \qquad \Gamma \vdash a : A \qquad \Gamma \vdash b : A}{\Gamma \vdash a =_A b\ \mathsf{type}} \qquad \textsc{Typ-Refl}\quad \frac{\Gamma \vdash A\ \mathsf{type} \qquad \Gamma \vdash a : A}{\Gamma \vdash \mathsf{refl}_a : a =_A a}$$

$$\textsc{Typ-J}\quad \frac{\Gamma; x, y : A; p : x =_A y \vdash C\ \mathsf{type} \qquad \Gamma; z : A \vdash C[x := z, y := z, p := \mathsf{refl}_z] \qquad \Gamma \vdash a : A \qquad \Gamma \vdash b : A \qquad \Gamma \vdash p' : a =_A b}{\Gamma \vdash \mathcal{J}_A(x.y.p.C, z.c, a, b, p') \vdash C[x := a, y := b, p := p']}$$

$$\textsc{Red-J}\quad \frac{}{\mathcal{J}_A(x.y.p.C, z.c, a, b, \mathsf{refl}_a) \to c[z := a]}$$

$$\textsc{Eq-J}\quad \frac{\Gamma; x, y : A; p : x =_A y \vdash C\ \mathsf{type} \qquad \Gamma; z : A \vdash C[x := z, y := z, p := \mathsf{refl}_z] \qquad \Gamma \vdash a : A}{\Gamma \vdash \mathcal{J}_A(x.y.p.C, z.c, a, a, \mathsf{refl}_a) \equiv c[z := a] : C[x := a, y := a, p := \mathsf{refl}_a]}$$

Figure 3.7: Formation, typing, reduction and equality rules for the identity type $a =_A b$

It might not be obvious what role $\mathcal{J}_A$ serves. We can illustrate the role of $\mathcal{J}_A$ by constructing from each $p : x =_A y$ a term $\mathsf{transport}_P(p, -) : P(x) \to P(y)$, where $P : A \to \mathcal{U}$ is a type family over A. This means we can *transport* a term/proof along an equality, to obtain a new proof "for free". In other words, it manifests Leibniz's principle of "indiscernibility of identicals": if x has the property P and $x = y$, then y must have also have the property P.

3.2 Presheaf semantics for MLTT

Just like that STLC is closely related to CCCs, MLTTs are also related to a class of categories. Here, we will focus on the presheaf semantics of MLTT, giving MLTT a model in a particular kind of topos, or a topos of presheaves over a small category.

3.2.1 Categories with families

Before formally introducing the presheaf model, we introduce **categories with families** (CwFs), which are a type of categories closely related to the syntax of MLTT. CwFs can be thought of as a *family* of models for MLTT, unifying several interesting models of MLTT. Using CwFs as a framework for the semantics of type theory was first proposed in [Dyb96], but here we follow the presentation in [CCD20].

Here, **Fam** is the category of *family of sets*. Its elements are families $\{U_x\}_{x \in X}$ of sets, and a morphism $\{U_x\}_{x \in X} \to \{V_y\}_{y \in Y}$ consists of a reindexing function $f : X \to Y$, and a family of functions $\{g_x\}_{x \in X}$, $g_x : U_x \to V_{f(x)}$. Equivalently, **Fam** is just the arrow category/morphism category $\mathbf{Mor}(\mathbf{Set})$ of **Set**.

Definition 3.2.1 (category with families (CwF))**.** A category with families $\mathcal{C}$ consists of the following data:

- a category $\mathcal{C}$ (by abuse of notation), often called the "category of contexts", with a terminal object. We use the capital Greek letters Γ, Δ, etc., to range over the objects of $\mathcal{C}$, and lower-case Greek letters σ, τ, etc., to range over the morphisms of $\mathcal{C}$;

- a **Fam**-valued presheaf $T : \mathcal{C}^{\mathrm{op}} \to \mathbf{Fam}$. Particularly, we define two operators Ty and Tm as follows: we write $\mathrm{Ty}(\Gamma) = X$ if $T(\Gamma) = \{U_x\}_{x \in X}$, and for $A \in X = \mathrm{Ty}(\Gamma)$, we write $\mathrm{Tm}(U; A) = U_A$. The two roughly corresponds to the "types in context Γ" and the "terms of type A, interpreted under context Γ", hence the notation.

  For a morphism $\gamma : \Delta \to \Gamma$, we have a map of families $T(\gamma) : \mathrm{Tm}(\Gamma, A)_{A \in \mathrm{Ty}(\Gamma)} \to \mathrm{Tm}(\Delta, B)_{B \in \mathrm{Ty}(\Delta)}$, which consists of a function $-[\gamma] : \mathrm{Ty}(\Gamma) \to \mathrm{Ty}(\Delta)$, and for each $A \in \mathrm{Ty}(\Gamma)$ a function $-[\gamma] : \mathrm{Tm}(\Gamma; A) \to \mathrm{Tm}(\Delta, A[\gamma])$. They are called "substitution of types" and "substitution of terms", respectively;

- a *comprehension* operation, assigning to each $\Gamma : \mathcal{C}$ and type $A \in \mathrm{Ty}(\Gamma)$ a context $\Gamma.A : \mathcal{C}$, a projection:

  $$\mathbf{p}(\Gamma; A) : \Gamma.A \to \Gamma$$

  and a term (in the CwF sense):

  $$\mathbf{q}(\Gamma; A) \in \mathrm{Tm}(\Gamma.A; A[\mathbf{p}(\Gamma; A)])$$

  satisfying the following universal property: for any $\gamma : \Delta \to \Gamma$ and all $a \in \mathrm{Tm}(\Delta, A[\gamma])$, there is a unique $(\gamma; a) : \Delta \to \Gamma.A$ such that $\mathbf{p}(\Gamma; A) \circ (\gamma, a) = \gamma$, and $\mathbf{q}(\Gamma; A)[(\gamma, a)] = a$.

  Particularly, there is a morphism $-[a] = -[(\mathsf{id}, a)] : \Gamma \to \Gamma.A$ on types and terms.

  Often, when there is no risk of confusion, we drop $(\Gamma; A)$ and write just $\mathbf{p}$ and $\mathbf{q}$.

Alternatively, the definition of a CwF can be given with Ty and Tm as primitives (see Section 3.1 of [Hof97]). One can see that the definition of a CwF is rather syntactical, following closely the syntax of MLTT. As one may expect, $[\![\Gamma]\!] = \Gamma$ (again, by a abuse of notation) and particularly $[\![\cdot]\!] = \mathbf{1}$ (the terminal object of $\mathcal{C}$. Moreover, $[\![\Gamma \vdash A\ \mathsf{type}]\!] \in \mathrm{Ty}(\Gamma)$, and $[\![\Gamma \vdash a : A]\!] \in \mathrm{Tm}(\Gamma; A)$, i.e. $\mathrm{Ty}(\Gamma)$ and $\mathrm{Tm}(\Gamma; A)$ are the collections of types A and terms a such that $\Gamma \vdash A\ \mathsf{type}_i$ and $\Gamma \vdash a : A$, respectively. Substitution on terms and substitution on types in CwFs interpret substitutions on terms and types, respectively.

One can define CwFs by giving $\mathcal{C}$, $\mathrm{Ty}(\Gamma)$ and $\mathrm{Tm}(\Gamma; A)$. For example, a set-theoretic model of MLTT can be constructed by taking $\mathcal{C} = \mathbf{Set}$.

Example 3.2.2 (set-theoretic model of MLTT ([Hof97], [Hub16]))**.** The set-theoretical model is simply the CwF model with $\mathcal{C} = \mathbf{Set}$. One could set $\mathrm{Ty}(\Gamma) = \{\sigma_\gamma\}$ to be a family of sets indexed by $\gamma \in \Gamma$, and $\mathrm{Tm}(\Gamma; A)$ an element $a_\gamma \in \{\sigma_\gamma\}$. Context extensions are defined as:

$$\Gamma.A = \{(\gamma, a) \mid \gamma \in \Gamma, a \in A_\gamma\}$$

with $\mathbf{p}(\gamma, a) = a$ and $\mathbf{q}(\gamma, a) = a$.

Furthermore, CwFs, with some extra axioms, are expressive enough to interpret dependent functions and dependent products (Π and Σ), but here we skip the more generic description, and leave the description to the following part, where we discuss the presheaf model of MLTT as a special case of the CwF model.

3.2.2 The presheaf model

Essentially, the presheaf model is a special case of the CwF model where Γ is a presheaf category. We work out this special case in full.

Let $\mathcal{C}$ be a category, usually small. A presheaf over $\mathcal{C}$, one recalls, is the category $\mathbf{PSh}(\mathcal{C}) = [\mathcal{C}^{\mathrm{op}}; \mathbf{Set}]$. Then, one can construct a model of MLTT in $\mathbf{PSh}(\mathcal{C})$. This was first proposed by Hofmann in [Hof97] using the language of CwFs. Here, we follow the exposition in [Hub16].

A context can be interpreted by a presheaf $\Gamma : \mathcal{C}^{\mathrm{op}} \to \mathbf{Set}$. What are the morphisms between presheaves? Naturally, one might think of "morphisms of contexts", which are called *substitutions*, or context maps. We make an interlude here to define substitutions of substitutions of contexts:

Definition 3.2.3 (context substitution/morphism)**.** Let $\Gamma = x_1 : A_1; ..., x_n : A_n$ and $\Delta = y_1 : B_1; ...; y_m : B_m$ be contexts. Then a substitution $\rho : \Gamma \to \Delta$ is a sequence of substitutions $y_1 := a_1(x_1, ..., x_n), .., y_m := a_m(x_1, ..., x_n)$, where $a_1, ..., a_n$ are terms, such that $\Gamma \vdash a_i : B_i$ for each a_i.

Particularly, for any $\Gamma \vdash A$ type, one can apply a context substitution $\sigma : \Delta \to \Gamma$ to A, obtaining a type $A\sigma$ by applying the sequence of substitutions in σ. A similar operation can be defined on terms in Γ.

Now, one can say that each context Γ is interepreted by a presheaf Γ, and each substitution $\sigma : \Gamma \to \Delta$ is interpreted by a morphism of presheaves/natural transformation $\sigma : \Gamma \to \Delta$ (again, an abuse of notation). By definition, for each $I : \mathcal{C}$, there is a set $\Gamma(I)$, and for $f : I \to J$ in $\mathcal{C}$, there is a function $\Gamma_f : \Gamma(I) \to \Gamma(J)$, $\rho \mapsto \rho f$. Often, an element of $\Gamma(I)$ is denoted as $\rho \in \Gamma(J)$. σ is a natural transformation, so for each $I : \mathcal{C}$ there is a component σ_I of σ; however, we will often drop the subscript. For example, $(\sigma_I(\rho))f = \sigma_j(\rho f)$ by definition of a natural transformation, but we write simply $(\sigma\rho)f = \sigma(\rho f)$.

Next, one defines the interpretation of types A, or precisely the judgments $\Gamma \vdash A$ type, as types might not make sense without a particular context because they may contain free variables. For each $I : \mathcal{C}$ and $\rho \in \Gamma(I)$, one requires a set $A\rho$, and for each $f : J \to I$, a function $A\rho \to A(\rho f)$, often written as $a \mapsto af$, such that $a\mathsf{id} = a$ and $(af)g = a(fg)$ for $g : K \to J$. Substitutions $\Delta \vdash A\sigma$ type with $\sigma : \Delta \to \Gamma$ are given by $(A\sigma)\rho = A(\sigma\rho)$ for $\rho \in \Delta(I)$, along with the induced map

$$(A\sigma)\rho = A(\sigma\rho) \to A((\sigma\rho)f) = (A\sigma)(\rho f).$$

More abstractly, one can define $\mathrm{Ty}(\Gamma)$ in terms of the **Grothendieck construction** on Γ. The eponymous construction is defined in general by Grothendieck, in [Gro71] [1], but here we define only a special case due to Yoneda and Mac Lane, following I.6 of [MM92].

[1] For readers familiar with K-theory, this is unrelated to the Grothendieck construction in K-theory, which constructs a group from an abelian semigroup.

Definition 3.2.4 (Grothendieck construction/category of elements of a presheaf)**.** Let $F : \mathcal{C}^{\mathrm{op}} \to \mathbf{Set}$ be a presheaf on $\mathcal{C}$. The Grothendieck construction on F, or the category of elements of F, often denoted by $\int_{\mathcal{C}} F$ or just $\int F$, is defined by the following data:

- the objects are all pairs (X, p) where $X : \mathcal{C}$ and $p \in F(X)$;
- the morphisms $(X, p) \to (Y, q)$ are the morphisms $f : X \to Y$ such that $p = qf$; the composition of these morphisms are given by the composition of morphisms in the underlying category.

We write $\pi_F : \int_{\mathcal{C}} F \to \mathcal{C}$ for the projection functor $(X, p) \mapsto X$. The following diagrams commute:

$$\begin{array}{ccc} \int_{\mathcal{C}} F & & \\ {\scriptstyle \pi_F}\downarrow & & \\ \mathcal{C} & \xrightarrow{\quad F \quad} & \mathbf{Set} \end{array} \qquad\qquad \begin{array}{ccc} p & \dashrightarrow & q \in F(Y) \\ \top\,\vdots\downarrow & & \top\,\vdots\downarrow \\ X & \xrightarrow[\quad f \quad]{} & Y. \end{array}$$

An intuitive way to understand the Grothendieck construction is that "the Grothendieck construction takes structured, boxed-up data and flattens it by throwing it all into one big space. The projection functor is then tasked with remembering which box each datum originally came from" [Spi14].

Note that the Grothendieck construction yields a functor $\int_{\mathcal{C}} : \mathbf{PSh}(\mathcal{C}) \to \mathbf{Cat}$ from the category of presheaves on $\mathcal{C}$ to the category of categories. A morphism of presheaves/natural transformation $\tau : F \to G$ induces a functor $\int_{\mathcal{C}} \tau : \int_{\mathcal{C}} F \to \int_{\mathcal{C}} G$ in the way one would expect.

There is an alternative definition of $\mathrm{Ty}(\Gamma)$ as $\mathrm{Ty}(\Gamma) = \mathbf{PSh}(\int_{\mathcal{C}} \Gamma)$, i.e., a type is interpreted by a presheaf on the category of elements of Γ. In other words, A is a presheaf $\int_{\mathcal{C}} \Gamma^{\mathrm{op}} \to \mathbf{Set}$, so sometimes we also write $A(I, \rho)$ instead of $A\rho$, if it is intended to emphasize I. Given a context substitution interpreted as a morphism $\sigma : \Delta \to \Gamma$, the application of the substitution σ on a type $\Gamma \vdash A\ \mathsf{type}$, or $A[\sigma]$, is simply interpreted as precomposing $[\![A]\!]$ with $\int_{\mathcal{C}} \sigma : \int_{\mathcal{C}} \Delta \to \int_{\mathcal{C}} \Gamma$.

Finally, one defines the interpretation of terms (i.e., their typing judgments) $\Gamma \vdash a : A$. Given $\Gamma \vdash A\ \mathsf{type}$ and $[\![A]\!] = A\rho$, the interpretation of a is a family of elements $a\rho \in A\rho$ for each $I : \mathcal{C}$ and $\rho \in \Gamma(I)$, satisfying $a\rho f = a(\rho f)$ for all $f : J \to I$.

For $A \in \mathrm{Ty}(\Gamma)$, one defines the context extension $\Gamma.A$ by: $(\Gamma.A))(I) = \{(\rho, u) \mid \rho \in \Gamma(I), u \in A\rho\}$ for any $I : \mathcal{C}$, and $(\rho, u)f = (\rho f, uf)$ for $f : J \to I$. The canonical projections are given by $\mathbf{p}(\rho, u) = \rho$, and $\mathbf{q}(\rho, u) = u$. One could check that these definitions indeed satisfy conditions required by the definition of a CwF.

3.2.3 Interpreting dependent types

With the basic framework set up, we now proceed to find interpretations for Π and Σ.

Given $\Gamma \vdash A\ \mathsf{type}$ and $\Gamma; x : A \vdash B\ \mathsf{type}$, one can find an interpretation for $\Gamma \vdash \Pi(x : A).B$, i.e. the set $(\Pi(x : A).B)\rho$ for $I : \mathcal{C}$, $\rho \in \Gamma(I)$. Often, we will simply write ΠAB for $\Pi(x : A).B$ (and similarly for Σ), as names are unessential in our discussion; this can be formalized as a nameless syntax [Bru72].

Roughly speaking, the nameless syntax (often called **de Bruijn indices**) is no more than replacing variable names with positional indices. For example, one could consider the term $\lambda x.\lambda y.x\ y$; to convert this term into nameless/de Bruijn form, one simply replaces the first occuring variable (x) with 0, the second (y) with 1, and so on. Then, one obtains the nameless form of this term: $\lambda\lambda 0\ 1$.

Let $\rho \in \Gamma(I)$. The elements $w \in (\Pi AB)\rho$ are families of functions $w = \{w_f\}_{f:J\to I}$, indexed by functions $f : J \to I$, where $J : \mathcal{C}$, such that $w_f u = w_f(u) \in B(\rho f, u)$ for each $u \in A(\rho f)$, where $J : \mathcal{C}$ and $f : J \to I$, and such that $(w_f u)g = w_{fg}(ug)$ for any $g : K \to J$.

For such a family $w \in (\Pi AB)\rho$, define $wf \in (\Pi AB)(\rho f)$, $f : J \to I$ to be

$$(wf)_g u = w_{fg} u \in B(\rho fg, u)$$

for any $J : K \to J$, $u \in A(\rho fg)$. One can verify that $w\mathsf{id} = w$ and $wfg = w(fg)$.

Next, one needs to define the interpretation for the constructor for Π, λ. Again, using the nameless syntax, given $\Gamma; a \vdash b : B$, one would like to define the interpretation of $\Gamma \vdash \lambda b : \Pi AB$, i.e. the element $(\lambda b)\rho \in (\Pi AB)\rho$ for $\rho \in \Gamma(I)$. $(\lambda b)\rho$, like $(\Pi AB)\rho$, is also defined as a family of functions indexed by $f : J \to I$. For $u \in A(\rho f)$, there is

$$((\lambda b)\rho)_f u = b(\rho f, u) \in B(\rho f, u).$$

This well-defined as

$$(((\lambda b)\rho)f)_g u = ((\lambda b)\rho)_{fg} u = b(\rho fg, u) = ((\lambda b)(\rho f))_g u$$

using the identity

$$(((\lambda b)\rho)_f u)g = (b(\rho f, u))g = b(\rho(fg), ug) = ((\lambda b)\rho)_{fg}(ug)$$

for any $g : K \to J$.

Finally, one defines the semantics of application. Given $\Gamma \vdash u : \Pi AB$ and $\Gamma \vdash v : A$, one defines $(u\ v)\rho = (u\rho)_{\mathsf{id}}(v\rho) \in B(\rho, v\rho)$. Note that $B(\rho, v\rho) = B(\mathsf{id}, v)\rho = B[v]\rho$, meaning that this definition respects substitution. Moreover, there is

$$((\lambda b)\ v)\rho = ((\lambda b)\rho)_{\mathsf{id}}(v\rho) = b(\rho, v\rho) = b[v]\rho$$

meaning that our definitions respect β-reduction. Similarly, one can also check the rules respect η using the commutativity conditions given previously.

Σ-types are much easier to interpret; their interpretation is based on the existence of products in our semantic category (a category of presheaves, which is a CCC). For $\Gamma \vdash A\ \mathsf{type}$ and $\Gamma; A \vdash B\ \mathsf{type}$, one defines define

$$(\Sigma AB)\rho = \{(a, b) | a \in A\rho, b \in B(\rho, a)\}$$

where $\rho \in \Gamma(I)$, $I : \mathcal{C}$. For $f : J \to I$, we define $(a, b)f = (af, bf)$.

The interpretation of the type former $(-,-)$ is also defined elementwise. If $\Gamma \vdash a : A$ and $\Gamma; A \vdash b : B$, we define $(a, b)\rho = (a\rho, b\rho)$. Similarly, for fst and snd, we define $(\mathsf{fst}(p))\rho = a$ and $(\mathsf{snd}(p))\rho = b$ if $p\rho = (a, b)$. One can verify that this validates the necessary reduction rules by calculation.

3.2.4 Universes

The presheaf model validates universes à la Tarski, but not cumulative universes à la Russell. The interpretation of the universe hierarchy requires a countable hierarchy of universes in the underlying set theory; one can achieve this using, e.g., Grothendieck universes. We refrain from discussing the fine points of set theory, but refer the interested reader to [Shu08] for a discussion of the potential set-theoretic issues.

Let us we have a set-theoretic hierarchy of universes, $U_0 \subseteq U_1 \subseteq U_2 \subseteq ...$, which might be Grothendieck universes or universes defined using other set-theoretic techniques. This approach of interpreting type-theoretic universes by "lifting" set-theoretic universes was proposed by Hofmann and Streicher [HS97]; here we follow Huber's exposition [Hub16] again.

We begin with the small types, i.e., the types A such that $\Gamma \vdash A\ \mathsf{type}_0$. We interpret this using small sets, i.e., members of U_0: that is, if $\Gamma \vdash A\ \mathsf{type}_0$, then we interpret it as $A\rho \in U_0$. Next we interpret the universe of small types, $\mathcal{U}_0$ in the empty context; this is equivalent to giving a context U which interprets the context $\{\mathcal{U}_0\}$.

We can define this using the Yoneda embedding: for each $I : \mathcal{C}$, we define $U(I)$ to be $\mathrm{Ty}(\mathbf{y}I)$, such that the members are small. Alternatively, it is the collection of U_0-valued (i.e., small-set valued) presheaves on $\int_{\mathcal{C}} \mathbf{y}I$, which is in turn equivalent to $\mathcal{C}/I$ [nLa21a]. In other words, $U(I)$ consists of the small types A, each represented by a collection $\{A_f\}$ indexed by $f : J \to I$, $J : \mathcal{C}$.

For each small type $\Gamma \vdash A\ \mathsf{type}_0$, we need to have a term $\Gamma \vdash \ulcorner A \urcorner : \mathcal{U}_0$, which is the *code* of A, and for each $\Gamma \vdash T : \mathcal{U}_0$, we need to have a (small) type $\Gamma \vdash \mathsf{El}(T)\ \mathsf{type}_0$, such that $\mathsf{El}(-)$ and $\ulcorner - \urcorner$ are inverses of each other.

First, we define the interpretation of $\mathsf{El}(-)$. Given $\Gamma \vdash T : \mathcal{U}_0$, we have a small type $\Gamma \vdash \mathsf{El}(T)\ \mathsf{type}_0$. For any $\rho \in \Gamma(I)$, we have $T\rho \in U(I)$, and we define $(\mathsf{El}(T))\rho = (T\rho)_{\mathsf{id}_I}$, which is a small set. For any $f : J \to I$, we define the map $(\mathsf{El}(T))\rho \to (\mathsf{El}(T))\rho f$ as the map $(T\rho)_{\mathsf{id}} \to (T\rho)_f$.

Then we define the interpretation of the quoting operator, $\Gamma \vdash \ulcorner A \urcorner : \mathcal{U}_0$, given a small type $\Gamma \vdash A\ \mathsf{type}_0$. For any $\rho \in \Gamma(I)$, we define $\ulcorner A \urcorner \rho \in U(I)$ as: $(\ulcorner A \urcorner \rho)_f = A(\rho f)$ for any $f : J \to I$. One can verify the axioms for quoting and unquoting via calculation.

We have demonstrated the construction of the model for universes at level 0; inductively applying this procedure yields models for the entire universe hierarchy.

So far, we have given Π types, Σ types and universes interpretations in the presheaf model for MLTT. However, one thing is still missing: identity types. The problem here is that although one could interpret identity types in the presheaf model, the behavior of this interpretation is somehow different from what we have promised: the presheaf model validates *extensional*, rather than *intensional* identity types as defined in subsection 3.1.3, a distinction which we will explain in the next chapter. The presheaf axioms also validates some axioms that are not provable from the axioms of the identity type as we have define. The attempts to deal with these problems gave rise to **homotopy type theory** [Uni13], an approach to type theory based on methods originating in algebraic topology and higher category theory.

Finally, we note that the presheaf model is a sound model of MLTT, so it is a "safe" model to use. However, the proof is long and technical, so we omit it here

and refer to the interested reader to [Hof97] for details.

3.3 Other models of MLTT

There are many other models of MLTT. Perhaps one of the earliest models of MLTT was the model in **locally cartesian closed categories**, which we define below:

Definition 3.3.1 (locally cartesian closed categories)**.** A category $\mathcal{C}$ is locally cartesian closed (LCCC) if it has finite limits, and for each object $C : \mathcal{C}$, the slice category $\mathcal{C}/C$ is cartesian closed.

Seely showed in [See84] that models of MLTT (minus universes) can be constructed in any LCCC. Particularly, every topos is LCCC, since the slice categories of toposes are again toposes, which are CCCs. Thus, the presheaf model given above could be considered a special case of the LCCC model.

However, the LCCC model given by Seely suffers from a coherence problem, which resulted in other models, particularly those based on CwFs, to be preferred over the LCCC model. However, LCCCs are the "simplest" models of MLTT, in the sense that there is a biequivalence between the category of Martin-Löf type theories and the (2-)category of LCCCs [CD11].

Awodey and Warren proposed a model of MLTT in Quillen-style *model categories* [AW09], which are a class of categories used in homotopy theory. Since a model category can be considered as the presentation of an $(\infty, 1)$-category, this suggests a connection between higher category theory and MLTT, which will be further explored in the next chapter. For the theory of model categories, we refer the interested reader to Appendix A of [Lur09].

Hofmann and Streicher gave a model of MLTT in the category of *groupoids* [HS95], or categories in which every morphism is invertible. This is essentially another instance of the interpretation of MLTT in CwFs, with $\mathcal{C}$ chosen to be the category **Grpd** of groupoids. However, this model is of particular interest, for reasons we will see in the coming chapter.

As our model of MLTT was based on presheaves, the natural question to ask is whether this model can be extended to categories of sheaves. The answer is positive [Coq12], and such a sheaf model was used effectively to prove some independence results in type theory [Man16]. However, this model has problems with universes; as a solution, one could replace sheaves with stacks, or 2-sheaves [CMR17].

Chapter 4

The identity type and the intensional-extensional dichotomy

4.1 Intensional and extensional type theory

In the previous chapter, we have introduced the identity type in MLTT via the refl constructor and the $\mathcal{J}$ rule. The identity type, however, was taken to be the same as any other type: for example, there might be many terms of type $a =_A b$, just like there may be many functions from a type A to another type B.

From a constructivist perspective, this can be desirable, as one might want to discern between different proofs of the same proposition. However, usually in mathematics, equality is a unitary concept; there isn't a notion of a term equal to another term in "many different ways". Moreover, the concept of identity is mathematically "opaque"; from the perspective of a mathematician using a proof of equality, nothing is changed if one replaces a proof with another proof. In other words, all proofs are observationally equivalent. Therefore, one may want to add a (type-theoretic) axiom, i.e. a primitive term, stating that all proofs of identity are indeed equal. This axiom is called **uniqueness of identity proofs**, often abbreviated as UIP:

$$\mathsf{UIP} : \forall(A : \mathcal{U}_i).\forall(x, y : A).\forall(p, q : x =_A y).p =_{x=_A y} q.$$

This is also often stated in an equivalent form, which is due to Streicher [Str93], also called "axiom K":

$$\mathcal{K} : \forall(A : \mathcal{U}_i).\forall(x : A).\forall(P : x =_A x \to \mathcal{U}_i).(P(\mathsf{refl}_x) \to \forall(h : x =_A x).P(h)).$$

For those familiar with computer science, $\mathcal{K}$ also has a computational interpretation as the basis for pattern matching on dependent types [GMM06], while UIP is non-computational.

The absence of UIP can have some rather unexpected results. Consider the following variant of the equality type, often called *heterogeneous equality*:

$$\frac{\text{Wf-HEq}\quad \Gamma \vdash a : A \qquad \Gamma \vdash b : B}{\Gamma \vdash a \approx_{A,B} b\ \mathsf{type}}$$

with the same constructor refl and eliminator $\mathcal{J}$. Naturally, one may think that this is equivalent to normal equality, as it would be impossible to construct a term

of $a \approx_{A,B} b$ when A and B are not in fact equal types. However, rather surprisingly, it is impossible to prove that $a \approx b$ implies $a = b$ without invoking UIP or $\mathcal{K}$.

Of course, one may also desire the principle of *functional extensionality*, meaning that two functions are equal if they are equal at any points, which holds by definition in set theory but cannot be proven from the rules of MLTT:

$$\mathsf{funext} : \forall(X, Y : \mathcal{U}_i).\forall(f, g : X \to Y).(\forall(x : X).fx =_Y gx) \to f =_{X \to Y} g.$$

Another concern with MLTT as presented in the previous chapter is that the judgmental equality is "too weak"; it is not the case that all terms that are propositionally equal are definitionally equal. In other words, it might be the case that there is a term p such that $\Gamma \vdash p : x =_A y$, while it is not true that $\Gamma \vdash x \equiv y : A$. In other words, there are terms that are "in fact" equal which our system does not treat as being equal. A natural solution is to add the following rule, often called **equality reflection** (Figure 4.1). Using the equality reflection rule, it is possible to prove UIP as a theorem [Hof95].

$$\textsc{Eq-Ref}\quad \frac{\Gamma \vdash p : x =_A y}{\Gamma \vdash x \equiv y : A}$$

Figure 4.1: Equality reflection

Adding the equality reflection rules makes MLTT **extensional**, meaning that observationally equal terms are considered fully equal. Unlike MLTT as presented in the previous chapter, which has a straightforward decision procedure, typing of terms in extensional MLTT is no longer decidable [Hof95], although it is still possible to check for the validity of a given proof tree. On the contrary, a type theory without such a rule is said to be **intensional**, meaning that intensional/definitional equality is considered separate from observational/external equality.

Adding UIP or axiom K as well as functional extensionality gives an intensional type theory some properties akin to extensional type theory while typing remains decidable. Sometimes, type theories with UIP and functional extensionality are called *propositionally extensional*, although these type theories are technically still intensional.

4.2 The identity type in the presheaf model

We did not give the construction of the identity type in the presheaf model in the previous chapter, since the behavior of the identity type "is not as expected". In fact, the presheaf model validates extensional equality, i.e., the interpretation of the identity type satisfies

For $\Gamma \vdash A\ \mathsf{type}$, $\Gamma \vdash x : A$ and $\Gamma \vdash y : A$, we need to define the set $(x =_A y)\rho$. We define this by passing to set-theoretic equality, by setting $(x =_A y)\rho = \{*\}$ the singleton set when $x\rho = y\rho$, and the empty set $\varnothing$ otherwise. Then, for any $\Gamma \vdash x : A$, we can define $(\mathsf{refl}_x)\rho = * \in (x =_A y)\rho$. We don't give the interpretation for $\mathcal{J}$ here, but it can also be given by a similar approach.

Clearly, our presheaf model validates UIP, because the interpretation of any term $\Gamma \vdash p : x =_A y$ must be $p\rho = * \in (x =_A y)\rho$. It also validates equality reflection, because we define the interpretation of $(x =_A y)$ using the interpretations $x\rho$ and $y\rho$, and thus observational equality is equivalent to definitional equality, which is exactly what equality reflection proposes.

4.3 Towards homotopy type theory

The presheaf model is a model for extensional type theory, so we naturally wonder if there are models for intensional type theory. However, most models for type theory interpret extensional type theory: just as in the presheaf model, propositional equality can be considered essentially an instance of the equalizer construction, while definitional equality is given by the equality of the underlying set/morphism [Mai05]. The universal property for the equalizer construction essentially guarantees uniqueness of identity proofs, and, furthermore, equality reflection. The first model of MLTT given by Seely [See84] was extensional; essentially all models based on LCCCs/toposes are extensional.

The first model of MLTT which only validates intensional identity is the groupoid model of Hofmann and Streicher [HS95]. In this model, contexts are interpreted as groupoids, or categories where all morphisms are invertible, and types as groupoid-valued functors. It is possible to prove that UIP is not validated by this model, which was also the first proof that UIP is independent from the rules of intensional type theory [HS95].

The other models for intensional type theory all have origins in algebraic topology, requiring a new way to think about identity types. Next, we introduce this novel approach to type theory, homotopy type theory (HoTT), which also highlights why intensional type theory is desirable in its own.

4.3.1 Intensional type theory and higher categorical structure

To motivate homotopy type theory, we briefly introduce the language of higher categories. Higher category theory begins with the following observation from algebraic topology. Consider a topological space X, and we can construct its fundamental groupoid $\Pi_0(X)$: the objects are the points of X, and the morphisms between p and q are the paths between p and q.

A path between p and q is just a continuous function $f : [0, 1] \to X$ such that $f(0) = p$ and $f(1) = q$. It is well-known in topology that one can equip the set of such continuous functions via the compact-open topology; therefore we can consider *paths of paths*, and *paths of paths of paths*, and so on[1]. This construction can be repeated indefinitely, so we obtain a groupoid where we have morphisms, morphisms of morphisms (2-morphisms), morphisms of 2-morphisms (3-morphisms), etc., culminating in an ∞-groupoid. We can also generalize this to general categories, i.e. 2-categories, 3-categories, and higher categories. We don't give the precise definitions of 2-groupoids, 2-categories, and so on here, but the interested reader can refer to [Bae97].

[1] In fact there are some coherence issues which we will not describe here.

Many categories have canonical higher category structures. One example is the fundamental groupoid of a space, as we have mentioned above. Another example is **Cat**, the category of categories, which is a 2-category: a 1-morphism is a functor, and a 2-morphism between 1-morphisms is a natural transformation.

However, there are some issues with the naïve definition. We can make the definition precise by using some insights from algebraic topology, following [Lur09]. Let Δ be the *simplex category*, or the category of abstract simplicial complexes and simplicial maps. More directly, the objects of Δ are the totally ordered sets $[n]$, and the morphisms are weakly order-preserving maps. A **simplicial set** is a presheaf on Δ. To any category $\mathcal{C}$, we can obtain a simplicial set $N(\mathcal{C})$ called the *nerve* of $\mathcal{C}$:

Definition 4.3.1 (the nerve of a category (section 1.1.2, [Lur09], [nLa21b]))**.** Note first that Δ is a full subcategory of **Cat**, as a simplex, considered as a poset, is a category. The **nerve** of a small category $\mathcal{C}$ is the simplicial set given by:

$$N(\mathcal{C}) : \Delta^{\mathrm{op}} \hookrightarrow \mathbf{Cat}^{\mathrm{op}} \xrightarrow{\mathrm{Hom}(-,\mathcal{C})} \mathbf{Set}.$$

$N(-)$ defines a functor (called the *nerve functor*) from the category **Cat** of (small) categories to the category of simplicial sets. Particularly, the nerve functor defines a fully faithful inclusion from **Cat** to $\mathbf{Cat}_\infty$ (the "category" of ∞-categories), which also has a left adjoint which sends every ∞-category to its *fundamental category* (cf. proposition 1.2.3.1, [Lur09]).

We have the following result on the relationship between nerves of categories and simplicial sets:

Proposition 4.3.1. *Let K be a simplicial set. Then the following conditions are equivalent:*

(1) there exists a small category $\mathcal{C}$ and an isomorphism $K \cong N(\mathcal{C})$;

(2) for each $0 < i < n$ and each diagram

$$\begin{array}{ccc} \Lambda^n_i & \longrightarrow & K \\ \downarrow & \nearrow & \\ \Delta^n & & \end{array}$$

there is a unique dotted arrow such that the diagram commutes. Here, Δ^n is the standard n-simplex, and Λ^n_i is the ith horn, obtained from Δ^n by deleting the interior and the face opposite the ith vertex.

Proof. Proposition 1.1.2.2 in [Lur09]. □

The second condition in Proposition 4.3.1 above is called the *extension condition*, and the morphism $\Delta^n \to K$ is called the extension of $\Lambda^n_i \hookrightarrow \Delta^n$. We can then define an ∞-category as (Definition 1.1.2.4 in [Lur09]):

Definition 4.3.2 (∞-category)**.** An ∞-category, or precisely $(\infty, 1)$-category, is a simplicial set K such that for any $0 < i < n$, any map $f_0 : \Lambda^n_i \hookrightarrow \Delta^n$ admits an extension $f : \Delta^n \to K$.

It is possible to define many concepts and constructions in ∞-categories analogous to regular categories; for details, see chapter 1.2 of [Lur09].

Homotopy type theory begins with the observation that in intensional type theory, there is an ∞-groupoid structure on any type generated by the identity type [Uni13]. Given a type A and $x, y : A$, we have a groupoid structure on A where the morphisms are given by terms of type $x =_A y$. Furthermore, for any $p, q : x =_A y$, we can form the 2-morphisms, which are terms of type $p =_{x=_A y} q$; repeating this construction gives us an ∞-groupoid structure on A. A recent result [AFS21] shows that this correspondence is indeed an isomorphism, in that the universe of types is isomorphic to the type of ∞-groupoids as defined internally in intensional type theory.

This is another way to understand why conventional categorical models of type theory support only extensional identity types: there is not enough structure to support this extra groupoid structure in a 1-category; **Grpd** is a 2-category and is enough to support part of this groupoid structure on the types (we usually say that it is "2-truncated") and refute UIP.

4.3.2 The univalence axiom

As the name "homotopy type theory" suggests, another view of intensional type theory is from the perspective of homotopy theory. Besides considering a type as an ∞-groupoid, we can also consider types and spaces, terms as points, and values of $x =_A y$ as paths. Then, the values of type $p =_{x=_A y} q$ are 2-paths, and so on. Indeed, Grothendieck has conjectured [Gro83] that the ∞-groupoids are equivalent to the topological spaces, for some well-behaved notation of ∞-groupoid. (Kapranov and Voevodsky have "proved" this hypothesis for some particular notion of ∞-groupoid [KV91], but their proof was later found to be incorrect. This became a motivation for the development of HoTT.) We follow chapter 2 of [Uni13] in this exposition leading to the definition of the univalence axiom.

Let $P : A \to \mathcal{U}$ be a type family, and $f, g : \Pi(x : A).P(x)$. Then, we define a homotopy as a term of the type

$$(f \sim g) := \Pi(x : A).(f(x) = g(x)).$$

Just as in homotopy theory, we consider things "up to coherent higher homotopies". That is, given a function $f : A \to B$, we consider its inverses "up to coherent higher homotopies", or *quasi-inverses*. This corresponds to the concept of "weak ∞-groupoid", in which composition of morphisms satisfy associativity only up to some notion of equivalence; an example would be the composition of paths in topological spaces. We say that f is a weak equivalence if there are functions $g, h : B \to A$ such that $f \circ g \sim \mathsf{id}_B$ and $h \circ f \sim \mathsf{id}_A$. In the language of type theory, we have a term of type

$$\mathsf{isequiv}(f) := \big(\Sigma(g : B \to A).(f \circ g \sim \mathsf{id}_B)\big) \times \big(\Sigma(h : B \to A).(h \circ f \sim \mathsf{id}_A)\big)$$

and we write $A \cong B$ if there is an equivalence f, i.e., $A \cong B := \Sigma(f : A \to B).\mathsf{isequiv}(f)$.

The **axiom of univalence** states that *equivalent terms may be identified*, or in type-theoretic language:

$$\mathsf{univalent}_{\mathcal{U}} : (A =_{\mathcal{U}} B) \cong (A \cong B).$$

We say a universe $\mathcal{U}$ is *univalent* if it satisfies this axiom. This allows us to, for example, identify isomorphic structures (such as groups) and to *transport* properties across the equivalence. This is a powerful axiom which corresponds to our intuition in subjects like algebra, but it is not compatible with UIP (Theorem 7.2.1 in [Uni13]). Usually, by "homotopy type theory", we mean intensional type theory where the universe of types is univalent.

4.3.3 A word on models

Just as intuition provides, homotopy type theory (and indeed intensional type theory itself) corresponds to $(\infty, 1)$-categories, and more precisely $(\infty, 1)$-toposes [Lur09]. Shulman showed that all Lurie-style $(\infty, 1)$-toposes indeed support a univalent universe [Shu19]. The model of HoTT itself, however, is given by a 1-category which *presents* an ∞-category in some way. The earliest model was given by Voevodsky and was based on simplicial sets (an overview was given in [KL12]). A constructive model based on *cubical sets* was given by Bezem, Coquand, and Huber [BCH14]. Both models are based on the classic presheaf model of type theory, but essentially limit the interpretation of types to certain presheaves satisfying some condition.

Chapter 5

Elementary toposes and the Calculus of Constructions

5.1 The universe of propositions

In the previous chapter, it is showed that intensional type theory is a setting for proof-relevant mathematics, and adding either UIP/$\mathcal{K}$ or equality reflection makes identity types proof-irrelevant. However, identity propositions are not the only kind of propositions; there may be other propositions which may be of any type. Of course, one cannot introduce UIP-like axioms for any type, as this will be absurd, so a natural thought would be to separate types that represent logical propositions from other types. This could be formalized in a universe separate from $\mathcal{U}_i$, which is usually called Prop, lying in the universe hierarchy at the same level as $\mathcal{U}_0$, that is to say $\mathsf{Prop} : \mathcal{U}_1$ (in Russell-style notation). Since Props represent logical propositions, hereafter we will use the convention that $\forall$ and $\exists$ are used for types in Prop, while Π and Σ are used for regular types (or when the type is in either Prop or $\mathcal{U}_i$.

One might ask why there isn't a countable hierarchy of Prop universes, just like the $\mathcal{U}_i$ hierarchy. The answer is that Prop is usually taken to be **impredicative**, i.e. a quantification over Prop is still a Prop. Consider, e.g. the type $\Pi(T : \mathcal{U}_i).T$; this type has type $\mathcal{U}_{i+1}$, although it deals only with types of level $\leq i$. One could, however, not allow this type to have type $\mathcal{U}_i$, as one will run into Girard's paradox as alluded to before. On the other hand, it is permissible that $(\forall(T : \mathsf{Prop}).T) : \mathsf{Prop}$, as long as a value of a type in Prop is not used to compute a value of a type in the normal universe hierarchy. That is to say, we allow rules like

$$\frac{\begin{array}{l}\textsc{W}_{\textsc{F}}\text{-}\Pi\text{-}* \\ \Gamma \vdash A\ \mathsf{type}_i^* \qquad \Gamma; x : A \vdash B\ \mathsf{type}_i^*\end{array}}{\Gamma \vdash \Pi(x : A).B : \mathsf{Prop}}$$

where $\Gamma \vdash A\ \mathsf{type}_i^*$ means that A is either a type at universe level i, or a proposition, under Γ. That is to say, if a proposition is used to construct a type, then the constructed type can only be a proposition.

The impredicative universe Prop is proposed by Coquand and Huet [CH88] to add some form of impredicativity to MLTT. Originally, Coquand and Huet's Prop universe was not proof-irrelevant, however in this thesis we always take Prop to be proof-irrelevant, that is, Prop satisfies the following axiom (called either *proof*

irrelevance or *propositional extensionality* in the literature):

$$\mathsf{proof\_irrelevance} : \forall(P : \mathsf{Prop}).\forall(p, q : P).p = q.$$

We can go further by supporting *definitional proof irrelevance*, which amounts to adding the following rule:

$$\frac{\textsc{Prop-Irrel}\quad \Gamma \vdash P : \mathsf{Prop} \qquad \Gamma \vdash p, q : P}{\Gamma \vdash p \equiv q : P}.$$

Of course, in extensional type theory, propositional and definitional proof-irrelevance are equivalent. Since we are working with extensional type theory in the rest of this chapter, we don't distinguish between propositional and definitional proof-irrelevance.

Note that different authors mean different things by Prop: an impredicative universe, as in [CH88], a proof-irrelevant (but predicative) universe, or both. In this thesis, Prop is always both impredicative and proof-irrelevant. MLTT with universes, identity types, and Prop is often called the **calculus of constructions** (CoC), a name first proposed in [CH88].

5.2 The correspondence between calculus of constructions and elementary toposes

In this section, we will prove what could be considered the central result of this thesis: a elementary topos corresponds to the extensional calculus of constructions with an impredicative, proof-irrelevant Prop. This section, and in fact the entire thesis, can be seen as an extended answer to a discussion on the Coq-Club mailing list [Sem21], although all concepts we use are known in the literature.

We proceed by showing that a CwF can be constructed from any (elementary) topos.

Proposition 5.2.1. *Any elementary topos $\mathcal{E}$ supports a CwF structure.*

Proof. The construction is based on example 6.14 in [Pit00], but the original exposition constructs a category with attributes, which is an older formalism equivalent to that of CwFs. Here, we translate the construction to the modern language of CwFs. The translation method is due to Hofmann (definition 3.10 in [Hof97]), but the details are original.

Let $\Gamma : \mathcal{E}$ be an object of $\mathcal{E}$, i.e. a context. We define $\mathrm{Ty}(\Gamma)$ as the collection of all pairs $A = (A, a)$, where $A : \mathcal{E}$ and $a : \Gamma \times A \to \Omega$. Here, Ω denotes the object of truth values in $\mathcal{E}$, i.e. the codomain of the subobject classifier. $\Gamma.A$ is defined as the pullback of a along the subobject classifier $1 \rightarrowtail \Omega$:

$$\begin{array}{ccc} \Gamma.A & \longrightarrow & \Gamma \times A \\ \downarrow & & \downarrow{\scriptstyle a} \\ \mathbf{1} & \rightarrowtail & \Omega \end{array}$$

is a pullback square.

The pullback diagram above gives us a morphism $\Gamma.A \to \Gamma \times A$. We compose this with fst to obtain the canonical projection $\mathbf{p}(\Gamma; A) : \Gamma.A \to \Gamma$. For any morphism $f : \Delta \to \Gamma$, we define $(A, a)[f] = (A, a \circ (f \times \mathsf{id}_A))$.

$\mathrm{Tm}(\Gamma; A)$ is the collection of sections (i.e. right inverses) of $\mathbf{p}(\Gamma; A)$, i.e. the collection of morphisms $g : \Gamma \to \Gamma.A$ such that $\mathbf{p} \circ g = \mathsf{id}_\Gamma$.

The only missing part is the construction of the term $\mathbf{q}(\Gamma; A) \in \mathrm{Tm}(\Gamma.A; A[\mathbf{p}(\Gamma; A)]$, which is a morphism $\Gamma.A \to \Gamma.A.A[\mathbf{p}]$ that is a section of $\mathbf{p} : \Gamma.A.A[\mathbf{p}] \to \Gamma.A$. This is a bit involved, but we may use the fact that the following diagram commutes and is a pullback square (proposition 3.9 in [Hof97]):

$$\begin{array}{ccc} \Gamma.A.A[\mathbf{p}] & \xrightarrow{\mathbf{p}} & \Gamma.A \\ {\scriptstyle \mathbf{p}}\downarrow & \nearrow & \downarrow{\scriptstyle \mathbf{p}(\Sigma;A)} \\ \Gamma.A & \xrightarrow[\mathbf{p}(\Sigma;A)]{} & \Gamma \end{array},$$

and the fact that a diagonal map gives rise to a section. □

Next, we informally explain the interpretation of type constructors in the topos model, roughly following [Mai05] The semantics of Π, Σ and universes follow the generic CwF model construction and is similar to the corresponding constructions in the presheaf model, and we give only a brief description here. The Σ and identity types have simple interpretations: ΣAB is interpreted as the categorical product $A \times B$, and $x =_A y$ can be interpreted as the equalizer of the morphisms that represent x and y. The required maps can be obtained by factoring through the pullback diagram above. Both could be interpreted in any topos, as a topos has all finite limits.

Π-types are much more involved, as they are interpreted by the *right adjoint to the pullback functor* [Mai05]. Consider any pullback square in a topos $\mathcal{C}$:

$$\begin{array}{ccc} Y & \dashrightarrow & X \\ {\scriptstyle f'}\vdots & & \downarrow{\scriptstyle f} \\ B & \xrightarrow[g]{} & A \end{array}$$

the pullback along g is functorial; it induces a functor $f^* : \mathcal{C}/A \to \mathcal{C}/B$ called the *pullback functor* (or the *base change functor*, often preferred by algebraic geometers). If this functor has a right adjoint $\Pi_f : \mathcal{C}/B \to \mathcal{C}/A$, then the category can interpret Π-types. Indeed, in every topos, any pullback functor has a right adjoint (theorem 2 in Section IV.7, [MM92]). Therefore, any topos can interpret the Π-types.

The universe hierarchy actually requires more structure in the topos; we need the notion of *universes in a topos*. This is discussed in [Str05], and we refer interested readers to that article. However, we note that many toposes, e.g. **Set** and presheaf/sheaf toposes, do have universes inherited from the underlying set theory (e.g. Grothendieck universes).

The impredicative, proof-irrelevant universe Prop is interpreted by Ω, as [Mai05] shows. A note on Prop's is that we need this universe to be Tarski-style as well, so we need quoting and unquoting as well.

Another way to give semantics to Prop using the subobject classifier is to do so via the **subset types**, which one could understand as the a modification of the

dependent products $\Pi(x : A).P(x)$, where $P : A \to \mathsf{Prop}$ is a family of propositions, and where snd is not defined. We often write this type as $\{x : A \mid P(x)\}$, just like subsets in set theory. The existence of subset types is equivalent to the existence of proof-irrelevant Prop, as one can always form the subset type where A is the one-element type. We can consider these types as subobjects in the topos-theoretic sense. That is, given A, $X = \{x : A \mid P(x)\}$ can be interpreted as a subobject $X \rightarrowtail A$ of A, with the associated morphism $x : \Gamma \times X \to \Omega$ defined by composing the subobject morphism with a. By calculation, it is possible to show that the subobject classifier gives an interpretation of subset types, which in turn gives a semantics to Prop.

Bibliography

[AFS21] Antoine Allioux, Eric Finster, and Matthieu Sozeau. "Types are internal ∞-groupoids". In: *36th Anual ACM/IEEE Symposium on Logic in Computer Science (LICS 2021)*. Ed. by Daniele Gorla and Leonid Libkin. IEEE Computer Society, 2021. URL: https://hal.inria.fr/hal-03133144/document.

[AW09] Steve Awodey and Michael A. Warren. "Homotopy theoretic models of identity types". In: *Mathematical Proceedings of the Cambridge Philosophical Society* 146 (2009).

[Awo10] Steve Awodey. *Category Theory*. 2nd ed. Oxford University Press, 2010.

[Bae97] John C. Baez. "An introduction to n-categories". In: *Category Theory and Computer Science*. Ed. by Eugenio Moggi and Giuseppe Rosolini. Springer-Verlag, 1997.

[BCH14] Marc Bezem, Thierry Coquand, and Simon Huber. "A Model of Type Theory in Cubical Sets". In: *19th International Conference on Types for Proofs and Programs (TYPES 2013)*. Ed. by Ralph Matthes and Aleksy Schubert. Vol. 26. Leibniz International Proceedings in Informatics (LIPIcs). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2014, pp. 107–128.

[BN98] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[Bru72] N.G. de Bruijn. "Lambda Calculus Notation with Nameless Dummies: A Tool for Automatic Formula Manipulation, with Application to the Church-Rosser Theorem". In: *Indigationes Mathematicae* 34 (1972).

[CCD20] Simon Castellan, Pierre Clairambault, and Peter Dybjer. *Categories with Families: Unityped, Simply Typed, and Dependently Typed*. 2020. arXiv: 1904.00827 [cs.LO]. URL: https://arxiv.org/abs/1904.00827.

[CD11] Pierre Clairambault and Peter Dybjer. "The Biequivalence of Locally Cartesian Closed Categories and Martin-Löf Type Theories". In: *TLCA 2011: Typed Lambda Calculi and Applications*. Ed. by Luke Ong. Springer-Verlag, 2011.

[CH88] Thierry Coquand and Gérard Huet. "The Calculus of Constructions". In: *Information and Computation* 76 (1988).

[CMR17] Thierry Coquand, Bassel Mannaa, and Fabian Ruch. "Stack semantics of type theory". In: *32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2017)*. IEEE Computer Society, 2017.

[Coq12] Thierry Coquand. *Sheaf model of type theory*. 2012. URL: `http://www.cse.chalmers.se/~coquand/sheaf.pdf`.

[Dyb96] Peter Dybjer. "Internal type theory". In: *Types for Proofs and Programs*. TYPES '95. Springer, 1996.

[Gir72] Jean-Yves Girard. "Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur". Thèse d'État. 1972.

[GMM06] Healfdene Goguen, Conor McBride, and James McKinna. "Eliminating Dependent Pattern Matching". In: *Algebra, Meaning, and Computation: Essays dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday*. Vol. 4060. Lecture Notes in Computer Science. Springer-Verlag, 2006.

[Gro71] Alexander Grothendieck. "Catégories fibrées et descente". In: *Revêtements étales et groupe fondamental (SGA 1)*. Vol. 224. Lecture Notes in Mathematics. Springer-Verlag, 1971.

[Gro83] Alexander Grothendieck. *Pursuing Stacks*. Unpublished drafts, personally communicated to Ronald Brown. 1983. URL: `https://thescrivener.github.io/PursuingStacks/ps-online.pdf`.

[Hof95] Martin Hofmann. "Extensional Concepts in Intensional Type Theory". PhD thesis. University of Edinburgh, 1995.

[Hof97] Martin Hofmann. "Syntax and Semantics of Dependent Types". In: *Semantics and Logics of Computation*. Ed. by Andrew M. Pitts and Peter Dybjer. Cambridge University Press, 1997. Chap. 3.

[HS95] Martin Hofmann and Thomas Streicher. "The groupoid interpretation of type theory". In: *Twenty-five years of constructive type theory*. Vol. 36. Oxford Logic Guides. Oxford Univ. Press, 1995.

[HS97] Martin Hofmann and Thomas Streicher. *Lifting Grothendieck Universes*. Unpublished note. 1997. URL: `https://www2.mathematik.tu-darmstadt.de/~streicher/NOTES/lift.pdf`.

[Hu20] Jason Z. S. Hu. *Categorical Semantics for Type Theories*. 2020. URL: `https://hustmphrrr.github.io/asset/pdf/comp-exam.pdf`.

[Hub16] Simon Huber. "Cubical Interpretations of Type Theory". PhD thesis. University of Gothenburg, 2016.

[KL12] Chris Kapulkin and Peter LeFanu Lumsdaine. *The Simplicial Model of Univalent Foundations (after Voevodsky)*. 2012. URL: `http://arxiv.org/abs/1211.2851`.

[KV91] Mikhail M. Kapranov and Vladimir A. Voevodsky. "∞-groupoids and homotopy types". In: *Cahiers de topologie et géométrie différentielle catégoriques* 32 (1 1991).

[LS86] J. Lambek and P. J. Scott. *Introduction to Higher Categorical Logic*. Cambridge University Press, 1986.

[Lur09] Jacob Lurie. *Higher Topos Theory*. Princeton University Press, 2009.

[Mac98] Saunders Mac Lane. *Categories for the Working Mathematician*. 2nd ed. Vol. 5. Graduate Texts in Mathematics. Springer-Verlag, 1998.

[Mai05] Maria Emilia Maietti. "Modular correspondence between dependent type theories and categories including pretopoi and topoi". In: *Mathematical Structures in Computer Science* 15 (6 2005).

[Man16] Bassel Mannaa. "Sheaf Semantics in Constructive Algebra and Type Theory". PhD thesis. University of Gothenburg, 2016.

[Mar72] Per Martin-Löf. *An intuitionistic theory of types*. Unpublished preprint. 1972. URL: https://archive-pml.github.io/martin-lof/pdfs/An-Intuitionistic-Theory-of-Types-1972.pdf.

[MM92] Saunders Mac Lane and Ieke Moerdijk. *Sheaves in Geometry and Logic*. Universitext. Springer-Verlag, 1992.

[nLa21a] nLab authors. *category of elements*. Revision 34. July 2021. URL: http://ncatlab.org/nlab/show/category%5C%20of%5C%20elements.

[nLa21b] nLab authors. *nerve*. Revision 70. July 2021. URL: http://ncatlab.org/nlab/show/nerve.

[Pit00] Andrew M. Pitts. "Categorical Logic". In: *Handbook of Logic in Computer Science, Volume 5*. Ed. by Samson Abramsky, Dov M. Gabbay, and Tom S. E. Maibaum. Oxford University Press, 2000.

[See84] R.A.G. Seely. "Locally cartesian closed categories and type theory". In: *Mathematical Proceedings of the Cambridge Philosophical Society* 95 (1 1984).

[Sel13] Peter Selinger. *Lecture notes on the lambda calculus*. 2013. arXiv: 0804.3434 [cs.LO].

[Sem21] Vincent Semeria. *Is Coq a topos?* Message to the Coq-Club mailing list. 2021. URL: https://sympa.inria.fr/sympa/arc/coq-club/2021-03/msg00013.html.

[Shu08] Michael A. Shulman. *Set theory for category theory*. 2008. arXiv: 0810.1279 [math.CT].

[Shu19] Michael Shulman. *All* $(\infty, 1)$*-toposes have strict univalent universes*. 2019. arXiv: 1904.07004 [math.AT].

[Spi14] David I. Spivak. *Category Theory for the Sciences*. The MIT Press, 2014.

[Str05] Thomas Streicher. "Universes in Toposes". In: *From Sets and Types to Topology and Analysis: Towards practicable foundations for constructive mathematics*. Ed. by Laura Crosilla and Peter Schuster. Oxford University Press, 2005.

[Str93] Thomas Streicher. "Investigations into Intensional Type Theory". Habilitationsschrift. Ludwig Maximilian University of Munich, 1993.

[SU06] Morton Heine B. Sørensen and Paweł Urzyczyn. *Lectures on the Curry-Howard Correspondence*. Vol. 149. Studies in Logic and the Foundations of Mathematics. Elsevier Science, 2006.

[Uni13] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study: `https://homotopytypetheory.org/book`, 2013.