Categorical Semantics of Dependent Type Theory

Daniel Mroz

July 18, 2019

Abstract

Homotopy type theory is built around a core of intensional Martin Löf dependent type theory, with a homotopical view of identity types. To understand the (higher) categorical semantics of HoTT, we must understand the categorical semantics of these two components.

The original contribution and the main focus of this dissertation is anundergraduate-level exposition of the categorical semantics of dependent types, with frequent reference to the well-understood Curry-Howard correspondence ("propositions as types"). We present two different sorts of models (categories with families, comprehension categories) of dependent types, and conclude that the motto is "types as fibrations".

Contents

Chapter 1

Introduction

There is a syntax/semantics duality between type theory and category theory. We can think of type theories as being a formal syntactic system for reasoning about categories, and in the other direction we can interpret types and typed terms as objects and morphisms in an appropriate category. There is a further connection between the study of logic and proofs, and type theory—the "propositions as proofs" paradigm, where we view propositions as types, and proofs as typed terms.

The fundamental example of this viewpoint is the Curry-Howard correspondence between intuitionistic logic and the simply typed lambda calculus, which was first presented explicitly in [How80]. This connection was further expanded by Lambek, who realized that cartesian closed categories (CCC) provide suitable semantics for the lambda calculus ([Lam85]). By this we mean that any type judgement $\Gamma \vdash s : A$ of the lambda calculus can be translated into an arrow $[\![s]\!] : [\![\Gamma]\!] \to [\![A]\!]$ in any CCC $\mathcal{C}$, and this translation is sound and complete.

Martin Löf's intuitionistic (or constructive) dependent type theory (ML) is a richer type theory than the lambda calculus where types may depend on other types. It was introduced in [Mar75; Mar82; MS84] as a formal system for constructive mathematics. One particularly interesting feature of ML is its identity types $\mathrm{Id}_A(a, b)$. Terms of this type can be regarded as "proofs that a and b are equal", or as "identifications of a as b", or as "paths from a to b". *Extensional* ML adds rules that make identity types sub-singletons: they are either uninhabited, or have only a single element. If we do not add this rule, we have an *intensional* type theory, where identity types may contain many (possibly unequal) proofs of equality. In this case, types may be interpreted as having a groupoid structure provided by their identity types, as noted in [HS95].

In fact, this groupoid structure should be higher dimensional, as from any terms $a, b : A$ and $p, q : \mathrm{Id}_A(a, b)$ we can form the higher identity type $\mathrm{Id}_{\mathrm{Id}_A(a,b)}(p, q)$. When we view identity terms as paths, higher identity types are the type of *paths between paths*, i.e. *homotopies*. Thus we get *homotopy type theory*, a homotopical view of type theory, which has been proposed as a formal foundation for mathematics, and as a syntax for doing homotopy theory ([Uni13]).

The goal of this dissertation is to explain the categorical semantics of ML type theory. In chapter 2 we describe the rules of ML, and provide some background and intuition for the homotopical viewpoint. In chapter 3 we show how to model dependent types categorically as categories with families, or more generally as comprehension categories (the "types as fibrations" paradigm).

Chapter 2

Background

2.1 Intuitionistic logic and constructivism

One of the axioms of classical logic is that for every proposition A, we may deduce $A \vee \neg A$—the *law of the excluded middle* (LEM). In particular this allows us to perform proofs by contradiction: to prove A assume $\neg A$, derive falsity, and conclude that A must hold. If A is a statement of the form "there exists an object x with property P", we might prove it by constructing some object x and proving that x does indeed satisfy P. On the other hand, a proof by contradiction may not be very satisfying as it doesn't actually construct a specific object with property P.

Constructivism is an approach to mathematics where proofs of existential statements must be explicit constructions. The logic of constructive mathematics is intuitionistic logic, which differs from classical logic in that LEM is not universally true.

In classical logic we can express implication as $A \supset B \equiv \neg A \vee B$. This is not so constructively—rather we think of proofs of $A \supset B$ as procedures that transform a proof of A into a proof of B.

2.2 Curry-Howard

For lack of space, we will not go into the details of correspondence between the simply typed lambda calculus and CCC, but we will refer to it frequently for motivation and context. A lucid presentation can be found in [AT10].The important things to note are that:

- The primitive objects of the simply typed lambda calculus are judgements of the form $x_1 : A_1, \dots x_n : A_n \vdash s : A$ where $x_1, \dots x_n$ are variables, s is a lambda term, and $A_1, \dots A_n, A$ are types. Such a judgement is translated into an arrow $[\![s]\!] : [\![A_1]\!] \times \cdots \times [\![A_n]\!] \to [\![A]\!]$

- This translation is inductive, and corresponds to the typing rules—the translation of a complex judgement $\Gamma \vdash s : A$ is built up from the translations of its components. Since the lambda calculus has a rule to form product types, we need to come up with a way to interpret product types, which turns out to be the categorical product.

Intuitionistic Logic	Simply Typed λ-calculus	Cartesian Closed Category
Proposition	Type	Object
Proof	Term	Morphism
Implication	Function type	Exponential object
Conjunction	Product type	Product object

2.3 Dependent Type Theory

The simply typed lambda calculus has type constructors that allow for the building of new types from previous ones. The product type $A \times B$ is composed from the types A and B. We may think of these type constructors as functions at the type level—they take types as arguments and return types as outputs. Dependent type theories add type constructors that may take values as inputs. In ML type theory there is one primitive type; the unit type $\mathbf{1}$, and three type constructors; the dependent product Π, the dependent sum Σ, and the identity type Id. We will briefly attempt to build some intuition about these.

For any type A and any two terms $a, b : A$ we can form the *identity type* $\mathrm{Id}_A(a, b)$. The terms of this type can be regarded as "proofs that a and b are equal", or as "identifications of a as b", or as "paths from a to b". Of course, if we have two terms $p, q : \mathrm{Id}_A(a, b)$ we may form the type $\mathrm{Id}_{\mathrm{Id}_A(a,b)}(p, q)$ whose elements are proofs that p and q are equal, and so on. If we have constructed a term $p : \mathrm{Id}_A(a, b)$ we say that a and b are *propositionaly equal*, and we may write this as $a \rightsquigarrow b$.

If A is a type and $B(x)$ is a type depending on A—that is for any a of type A, $B(a)$ is also a type—then we can form the *dependent product type* $\Pi_{x:A} B(x)$. The terms of this type are *dependent functions* f that for every input $a : A$ have an output $f(a) : B(a)$. We may also form the *dependent sum type* $\Sigma_{x:A} B(x)$, whose terms are (a, b) where $a : A$ and $b : B(a)$.

If x is not a free variable of B then for any $a : A$ $B(a)$ is just the type B. In this case we write the dependent product type $\Pi_{x:A} B$ as $A \to B$, and the dependent sum type $\Sigma_{x:A} B$ as $A \times B$.

The unit type $\mathbf{1}$ has a unique term $\star$, in the sense that if $a : \mathbf{1}$ we can construct a witness of equality between a and $\star$, a term $p : \mathrm{Id}_{\mathbf{1}}(a, \star)$.

Remark 2.3.1. One application of dependent types is in more powerful type systems for functional programming. For example, given a natural number n and a type A we can define a type $\mathrm{Vec}(n, A)$ of vectors of length n with elements in A by the rules

$$\frac{A \ \mathsf{Type}}{\mathrm{nil} : \mathrm{Vec}(0, A)}$$

$$\frac{x : A \qquad xs : \mathrm{Vec}(n, A)}{\mathrm{cons}(x, xs) : \mathrm{Vec}(n+1, A)}$$

These are inductive rules of the same sort as we have seen in the typed lambda calculus. The first rule, for example, should be interpreted as saying "if A is a type then there is a term nil of type $\mathrm{Vec}(0, A)$". Hence we could define a dependent function

$$\mathsf{add} : \Pi_{n:\mathbb{N}} \left(\mathrm{Vec}(n, \mathbb{N}) \to \mathrm{Vec}(n, \mathbb{N}) \to \mathrm{Vec}(n, \mathbb{N})\right)$$

that adds two vectors of natural numbers. Note that the type signature *requires* the two arguments have the same length, and *guarantees* that the result has the same length too.

Remark 2.3.2. Another reason dependent type theories are useful is that they model (intuitionistic) first order logic. Predicates correspond to Dependent types, universal quantification to dependent functions, and existential quantification to dependent sums.

We now give an informal overview of ML type theory, mostly following [Gar09] but with some differences in notation. For a precise treatment see [Hof97].

There are three sorts of familiar syntactic objects we will deal with: types, terms, contexts, and variables which may occur in both types and terms and represent types and terms.

Syntax	Notation
Types	$A, B, C, \ldots$
Terms	$a, b, c, \ldots$
Contexts	$\Gamma, \Delta, \Theta, \ldots$
Variables	$x, y, z \ldots$

A *context* is a list $\Gamma = x_1 : A_1, \ldots x_n : A_n$ associating to each variable x_i a type A_i. We require that the free variables of each type expression A_i are contained in $\{x_1, \ldots x_{i=1}\}$, and that each A_i *actually is a type* in the context $x_1 : A_1, \ldots x_{i-1} : A_{i-1}$ (that is, we can make the judgement $x_1 : A_1, \ldots x_{i-1} : A_{i-1} \vdash A_i\ \mathsf{Type}$). We denote the empty context by $[]$.

We will have four sorts of judgements:

Judgement	Meaning
$\Gamma \vdash A\ \mathsf{Type}$	A is a type
$\Gamma \vdash a : A$	a is a term of type A
$\Gamma \vdash a \equiv b : A$	a and b are *definitionaly equal* terms of type A
$\Gamma \vdash A \equiv B\ \mathsf{Type}$	A and B are *definitionaly equal* types

These have obvious well-formation requirements—to make the judgement $\Gamma \vdash a = b : A$ we need to have $\Gamma \vdash a : A$, $\Gamma \vdash b : A$, and $\Gamma \vdash A\ \mathsf{Type}$, and so on. We may omit the context if it is empty or implicit.

Remark 2.3.3. Note that it only makes sense to call an expression A a *type* if we have a context Γ and a judgement $\Gamma \vdash A\ \mathsf{Type}$, and likewise for terms. We will commit a slight abuse of notation and nonetheless refer to expressions that are syntactically well-formed as types and terms, even if we may not have an appropriate judgement. For example, we will call $\mathrm{Vec}(n, A)$ a type, and $\lambda x.y$ a term, even in the lack of a typing judgement.

Remark 2.3.4. Definitional equality, also known as *judgemental equality*, is a syntactic notion, for when two things are equal by definition. As an example, if we defined the functions $f, g : \mathbb{N} \to \mathbb{N}$ by $f(x) \equiv 2 + x, g(x) \equiv x + x$ then we could make the judgement $f(2) \equiv g(2) : \mathbb{N}$. Definitional equality is different from propositional equality, which is a semantic sort of equality inside the type theory. In particular we are not able to express notions like "if $a \equiv b$ then ..." inside the theory, though of course we can do so in the metatheory.

The next thing to do is specify the rules of inference. There are equality rules, which say that definitional equality is reflexive, symmetric, and transitive. Furthermore, definitional equality is a congruence. We also have the standard structural rules of context weakening, exchange, and contraction.

We will take substitution (on both terms and types) as a primitive notion in the metatheory, and denote it in the standard way; $A[B/x]$ is A but with every instance of the variable x replaced with B. If A is a type possibly containing a variable x, we may write it explicitly as $A(x)$, and denote the substitution $A[B/x]$ as $A(B)$ (and likewise for terms).

The rules for the unit type are:

$$\frac{}{\mathbf{1}\ \mathsf{Type}}\ \mathbf{1}\text{-FORM}$$

$$\frac{}{\star : \mathbf{1}}\ \mathbf{1}\text{-INTRO}$$

$$\frac{x : 1 \vdash C(x)\ \mathsf{Type} \qquad d : C(\star)}{x : 1 \vdash U_d(x) : C(x)}\ \mathbf{1}\text{-ELIM}$$

$$\frac{x : 1 \vdash C(x)\ \mathsf{Type} \qquad d : C(\star)}{U_d(x) \equiv d : C(\star)}\ \mathbf{1}\text{-COMP}$$

The first two rules assert the existence of the primitive type $\mathbf{1}$, and a *canonical term* $\star$.

The next two say that if we have a type $C(x)$ dependent on $\mathbf{1}$, and a term d of type $C(\star)$, we can form a term $U_d(x)$ of type $C(x)$ which is definitionaly equal to d when we take x to be canonical element $\star$ of $\mathbf{1}$.

The next types we introduce will also have formation and introduction rules that tell us how to create new types and canonical terms, and elimination and computation rules, that specify how to "consume" the canonical terms.

The rules for the dependent product type are:

$$\frac{A\ \mathsf{Type} \qquad x : A \vdash B(x)\ \mathsf{Type}}{\Pi_{x:A} B(x)\ \mathsf{Type}}\ \Pi\text{-FORM}$$

$$\frac{x : A \vdash f(x) : B(x)}{\lambda x.f(x) : \Pi_{x:A} B(x)}\ \Pi\text{-ABS}$$

$$\frac{f : \Pi_{x:A} B(x)}{fx : B(x)}\ \Pi\text{-APP}$$

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

The rules for the dependent sum type are:

$$\frac{A\ \mathsf{Type} \qquad x : A \vdash B(x)\ \mathsf{Type}}{\Sigma_{x:A} B(x)\ \mathsf{Type}}\ \Sigma\text{-FORM}$$

$$\frac{a : A \qquad b : B(a)}{\langle a, b\rangle : \Sigma_{x:A} B(x)}\ \Sigma\text{-INTRO}$$

$$\frac{z : \Sigma_{x:A} B(x) \vdash C(z)\ \mathsf{Type} \qquad x : A, y : B(x) \vdash d(x, y) : C(\langle x, y\rangle)}{z : \Sigma_{x:A} B(x) \vdash E_d : C(z)}\ \Sigma\text{-ELIM}$$

$$\frac{z : \Sigma_{x:A} B(x) \vdash C(z)\ \mathsf{Type} \qquad x : A, y : B(x) \vdash d(x, y) : C(\langle x, y\rangle)}{x : a, y : B(x) \vdash E_d(\langle x, y\rangle) \equiv d(x, y) : C(\langle x, y\rangle)}\ \Sigma\text{-COMP}$$

Finally, we have the rules for the identity type:

$$\frac{A\ \mathsf{Type} \qquad a, b : A}{\mathrm{Id}_A(a, b)\ \mathsf{Type}}\ \text{Id-FORM}$$

$$\frac{a : A}{r(a) : \mathrm{Id}_A(a, a)}\ \text{Id-INTRO}$$

$$\frac{x, y : A, p : \mathrm{Id}_A(x, y) \vdash C(x, y, p)\ \mathsf{Type} \qquad x : A \vdash d(x) : C(x, x, r(x))}{x, y : A, p : \mathrm{Id}_A(x, y) \vdash J_d(x, y, p) : C(x, y, p)}\ \text{Id-ELIM}$$

$$\frac{x, y : A, p : \mathrm{Id}_A(x, y) \vdash C(x, y, p)\ \mathsf{Type} \qquad x : A \vdash d(x) : C(x, x, r(x))}{x : A \vdash J_d(x, x, r(x)) \equiv d(x) : C(x, x, r(x))}\ \text{Id-COMP}$$

These rules comprise the intensional version of ML type theory. The extensional theory adds the following rules, which are not derivable from the previous rules:

$$\frac{a, b : A, p : \mathrm{Id}_A(a, b)}{p \equiv r(a) : \mathrm{Id}_A(a, b)}\ \text{Id-EXT}$$

$$\frac{f, g : \Pi_{x:A} B(x) \qquad x : A \vdash fx \equiv gx}{f \equiv g : \Pi_{x:A} B(x)}\ \Pi\text{-EXT}$$

Note that for the judgement $p \equiv r(a) : \mathrm{Id}_A(a,b)$ to be well formed we also require that $a \equiv b : A$. So the Id-EXT rule says that if the identity type $\mathrm{Id}_A(a,b)$ is inhabited by a term p, then in fact $a \equiv b$, and $p \equiv r(a)$. In this case the groupoid structure on types is trivial, there are no higher dimensional features. On one hand, this makes the type theory rather less interesting, but it has the benefit of simplifying the categorical semantics. Perhaps surprisingly, while intensional ML has decidable type-checking, extensional ML does not ([Hof95].

We can prove the claim mentioned earlier:

Proposition 2.3.1. *Every term $a : \mathbf{1}$ is (propositionaly) equal to $\star$.*

Proof. Translating this statement into type theory yields the type $\Pi_{x:\mathbf{1}}\mathrm{Id}_{\mathbf{1}}(x,\star)$. So to prove the claim it is sufficient to construct a term of the appropriate type, which we do below.

$$\dfrac{\dfrac{\dfrac{}{x : \mathbf{1} \vdash \mathrm{Id}_{\mathbf{1}}(x,\star)\ \mathsf{Type}} \qquad \dfrac{}{r(\star) : \mathrm{Id}_{\mathbf{1}}(\star,\star)}}{x : 1 \vdash U_{r(\star)}(x) : \mathrm{Id}_{\mathbf{1}}(x,\star)}}{\lambda x.U_{r(\star)}(x) : \Pi_{x:\mathbf{1}}\mathrm{Id}_{\mathbf{1}}(x,\star)}$$

□

We would also like to prove that propositional equality actually *acts like equality*, that is:

Proposition 2.3.2. *For any type A the following hold:*

(refl.) *For all $a : A$ we have $a \rightsquigarrow a$*

(sym. *If $a \rightsquigarrow b$ then $b \rightsquigarrow a$*

(trans.) *If $a \rightsquigarrow b$ and $b \rightsquigarrow c$ then $a \rightsquigarrow c$*

Proof.
Suppose A Type.

(refl.) We want a term $\mathsf{refl} : \Pi_{x:A}\mathrm{Id}_A(x,x)$. Take refl to be $\lambda x.r(x)$.

(sym.) We want a term $\mathsf{sym} : \Pi_{x:A}\Pi_{y:A}\Pi_{p:\mathrm{Id}_A(x,y)}\mathrm{Id}_A(y,x)$. Using Id-ELIM we can form the judgement $x, y : A, p : \mathrm{Id}_A(x,y) \vdash J_{r(x)}(x,y,p) : \mathrm{Id}_A(y,x)$. Then, if we take $\mathsf{sym} \equiv \lambda x, y, p.J_{r(x)}(x,y,p)$, repeated application of Π-ABS will give us the desired judgement.

Remark 2.3.5. Note that sym has the property that $\mathsf{sym}(x,x,r(x)) \equiv r(x)$. When $x, y : A$ are clear from context we will write $\mathsf{sym}(x,y,p)$ as p^{-1}.

Remark 2.3.6. What we've done in this proof is specify that if $x \equiv y$ and $p \equiv r(x) : \mathrm{Id}_A(x,y)$, we want $\mathsf{sym}(x,y,p) \equiv r(x) : \mathrm{Id}_A(y,x)$. The Id-ELIM rule provides us with a term $J_d(x,y,p)$ that extends sym. This is sometimes called the *path induction principle*—by specifying the value of a function on the canonical elements of $\mathrm{Id}_A(x,y)$ (the constant

reflexive paths $r(x) : \mathrm{Id}_A(x, x)$) we can define a function on all the paths in $\mathrm{Id}_A(x, y)$.

Compare with the familiar induction principle on $\mathbb{N}$. To define a function $f : \mathbb{N} \to X$ it is sufficient to specify a value for $f(0)$ and a way of generating $f(n + 1)$ from $f(n)$. This works because $\mathbb{N}$ is generated by 0 and the successor operation $+1$. Similarly, the path induction principle says that identity types are in some sense generated by the constant paths.

(trans.) We want a term $\mathsf{tran} : \Pi_{x:A}\Pi_{y:A}\Pi_{z:a}\Pi_{p:\mathrm{Id}_A(x,y)}\Pi_{q:\mathrm{Id}_A(y,z)}\mathrm{Id}_A(x, z)$. As $\mathrm{Id}_A(x, z)$ doesn't actually depend on p and q, we may also write this type as
$\Pi_{x:A}\Pi_{y:A}\Pi_{z:A}\left(\mathrm{Id}_A(x, y) \to \mathrm{Id}_A(y, z) \to \mathrm{Id}_A(x, z)\right)$. By the path induction principle, we may assume that $x \equiv y \equiv z$ and $p \equiv q \equiv r(x)$. Then we will take the value of $\mathsf{trans}(x, y, z, p, q)$ to be $r(x) : \mathrm{Id}_A(x, z)$.

Remark 2.3.7. Note also that trans has the property that
$\mathsf{trans}(x, x, x, r(x), r(x)) \equiv r(x)$. When $x, y, z : A$ are clear in context we will write $\mathsf{trans}(x, y, z, p, q)$ as $p \centerdot q$, to be regarded as the concatenation of the paths p and q.

□

2.4 Homotopy type theory

We will not go into the full details of homotopy type theory as presented by the Univalent Foundations Program in [Uni13]. In particular we will not concern ourselves with the univalence axiom, or higher inductive types. What we are primarily interested in is *homotopical interpretations of type theory*, like the groupoid interpretation of types.

We will first show that identity terms with "concatenation" really do behave like paths in a topological space. For example, we want $p : Id_A(x, y)$ concatenated with the constant path $r(y)$ to be just p. We don't mean that $p \centerdot r(y)$ is *definitionaly equal* to p, but that it is homotopic to p. In other words, that there is a term $q : \mathrm{Id}_{\mathrm{Id}_A(x,y)}(p \centerdot r(y), p)$. The term q is a homotopy between the paths $p \centerdot r(y)$ and p.

Proposition 2.4.1 (Lemma 2.1.4 in [Uni13])**.** *Let $x, y, z, w : A$ and $p : Id_A(x, y), q : Id_A(y, z), r : Id_A(z, w)$. Then we have that:*

1. $p \rightsquigarrow p \centerdot r(y)$ *and* $r(x) \centerdot p \rightsquigarrow p$.
2. $p^{-1} \centerdot p \rightsquigarrow r(y)$ *and* $p \centerdot p^{-1} \rightsquigarrow r(x)$
3. $(p^{-1})^{-1} \rightsquigarrow p$
4. $p \centerdot (q \centerdot r) \rightsquigarrow (p \centerdot q) \centerdot r$

Proof. Proven in [Uni13], using path induction. □

The groupoid interpretation was introduced by Striecher and Hofmann in [HS95]. They showed that the rule Id-EXT is not derivable from the other rules of intensional ML type theory by constructing a model where it does not hold.

In their interpretation a type A is regarded as a groupoid whose elements are terms $a : A$, and whose arrows $p : a \to b$ are terms $p : \mathrm{Id}_A(a, b)$ (modulo propositional equality). Identity arrows are $a \to a$ are given by $r(a)$, composition by $\cdot$, and inverses by p^{-1}. Proposition 2.4.1 shows that all the rules of a groupoid are indeed satisfied. In such a model the rule Id-EXT corresponds to the statement that every such groupoid has at most one arrow between any two objects. So to refute Id-EXT it is sufficient to construct "an interpretation of type theory in which types are interpreted as arbitrary groupoids, provided one succeeds in *ascribing appropriate meaning to the type ... formers* [emphasis added]".

Chapter 3

The semantics of dependent types

Recall (from [AT10]) how we form the *syntactic category* of the simply typed lambda calculus (STLC), or the *term model*. It is called the syntactic category because the structure corresponds exactly to the syntax of the lambda calculus. More precisely, for every type A we have an object $[A]$, and we add an extra object $\mathbf{1}$ (this will be terminal). The arrows are:

- For every equivalence class of terms t with $x : A \vdash t : B$ we have an arrow $[t] : [A] \to [B]$
- For every equivalence class of terms t with $\vdash t : [B]$ we have an arrow $[t] : \mathbf{1} \to B$
- For every type A there is a terminal arrow $! : [A] \to \mathbf{1}$

Composition is given by substitution, for if $x : A \vdash t : B$ and $y : B \vdash s : C$ then we may deduce $x : A \vdash s[t/y] : C$. It is not hard to verify that this is indeed a cartesian closed category, and two arrows $[t], [s] : [A] \to [B]$ are equal if and only if the representative terms t and s are equal.

We mentioned in the introduction that we consider objects of a CCC as contexts. Why do we now say they are types? It is because a context of the STLC $\Gamma = x_1 : A_1, \ldots x_n : A_n$ is really no more that the sum (or rather, the *product*) of its parts—we may equivalently consider the context $x : A_1 \times \cdots \times A_n$ (up to the appropriate bookkeeping).

In the theory of dependent types this is not the case; contexts have a more rigid structure. Let's try to recreate the substitution/composition analogy above in dependent type theory. That is, suppose we have a judgement

$$x : A \vdash t : B(x) \tag{3.1}$$

We want to compose this with a judgement of the form

$$y : B(x) \vdash s : C(y) \tag{3.2}$$

But already this doesn't make sense—$B(x)$ needn't be a type if we don't have $x : A$.

Let us try again with the judgement

$$x : A, y : B(x) \vdash s : C(x, y) \tag{3.3}$$

How can we compose the judgements 3.1 and 3.3 to get a judgement $x : A \vdash \cdots : C(x, \cdots)$? The same sort of substitution yields:

$$x : A \vdash s[t/y] : C(x, t) \tag{3.4}$$

Note that we are explicitly writing out the free variables of C here. We could have written judgement 3.3 as $x : A, y : B \vdash s : C$. Then we would write judgement 3.4 with an explicit type substitution:

$$x : A \vdash s[t/y] : C[t/y] \tag{3.5}$$

This style of notation shows what is going on more clearly. When we compose judgements we do so in a base context that stays the same. The actual composition is performed by substitution *in both types and terms*. At this point, one might imagine having, for every context Γ, a category of types and terms with respect to Γ. The questions now are:

- How do these categories interact?
- What happens when we extend a context Γ (with $\Gamma \vdash A$ Type) to the context $\Gamma, x : A$? Can we transfer judgements made in Γ to judgements in $\Gamma, x : A$?
- More generally, under what conditions can we translate a judgement made in one context to another context?

Remark 3.0.1. It is at this point that Hofmann introduces a notion of "pre-syntax" in [Hof97]. The idea is that pre-syntax includes terms, types, and contexts which are well-formed but possibly not well-typed. By remark 2.3.3 we call Hofmann's pre-terms just terms, and to specify that a term t is a term in the sense of Hofmann, we say that there is a judgement $\Gamma \vdash t : A$. We do however have the requirement that contexts be well-typed. In this section, we relax that requirement and allow for contexts that may not be well-typed, like $\Gamma = x : A(y)$ where A is a type that depends on y. Only when a context appears in a judgement will we require that it is well-typed.

3.1 More on contexts

To answer the questions above, we first define some rather technical notions relating to contexts.

Let $\Gamma = x_1 : A_1, x_2 : A_2(x_1), \ldots x_n : A_n(x_1, \ldots x_{n-1})$ and $\Delta = y_1 : B_1, \ldots y_m : B_m(y_1, \ldots y_{m-1})$ be two contexts.

Definition 3.1.1. A *context morphism* (also called a *substitution* or *interpretation* of variables) $\Gamma \to \Delta$ consists of m terms $t_1, \dots t_m$ such that we have the judgements:

$$\begin{aligned}
&\Gamma \vdash t_1 : B_1 \\
&\Gamma \vdash t_2 : B_2(t_1) \\
&\quad \dots \\
&\Gamma \vdash t_m : B_m(t_1, \dots t_{m-1})
\end{aligned}$$

We may write this as $(t_i) : \Gamma \to \Delta$. If we have two context morphisms $(t_i), (s_i) : \Gamma \to \Delta$ with

$$\begin{aligned}
&\Gamma \vdash t_1 \equiv s_2 : B_1 \\
&\Gamma \vdash t_2 \equiv s_2 : B_2(t_1) \\
&\quad \dots \\
&\Gamma \vdash t_m \equiv s_m : B_m(t_1, \dots t_{n-1})
\end{aligned}$$

we say that (t_i) and (s_i) are equivalent, and denote this as $(t_i) \equiv (s_i)$.

Remark 3.1.1. We may think of a context $\Gamma = x_1 : A_1, x_2 : A_2(x_1), \dots x_n : A_n(x_1, \dots x_{n-1})$ as a *list of assumptions*, that we have some terms x_i of type A_i. In this case, we can think of a context morphism $\Gamma \to \Delta$ as *satisfying the assumptions of* Δ*, given the assumptions of* Γ. That is, if we are given terms $x_i : A_i$, a context morphism tells us how we may transform these (i.e. substitute them into other terms) to produce terms $y_i : B_i$.

Now, suppose further that $n = m$. We may rename (α-convert) the variables of Δ to those of Γ, replacing each y_i with x_i.

Definition 3.1.2. Say that Γ and Δ are *equivalent* if we have the judgements:

$$\begin{aligned}
\vdash A_1 &\equiv B_1 \ \mathsf{Type} \\
x_1 : A_1 &\vdash A_2 \equiv B_2 \ \mathsf{Type} \\
&\dots \\
x_1 : A_1, \dots x_{n-1} : A_{n-1} &\vdash A_n \equiv B_n \ \mathsf{Type}
\end{aligned}$$

In this case we write $\Gamma \equiv \Delta$.

Remark 3.1.2. We could have included context equivalence as a judgement form in our type theory, in the way of [Str12], but as [Hof97] points out, it is equally valid to have equivalence of contexts as a derived notion.

We will now state some properties of context morphisms, but for brevity, omit the proofs—they are straightforward induction arguments, and most can be found in [Hof97].

Example 3.1.1.
1. For every context Γ there is a unique context morphism $() : \Gamma \to []$.

2. For every context $\Gamma = x_1 : A_1, \ldots x_n : A_n$ the list of terms (x_i) is a context morphism $\Gamma \to \Gamma$, which we denote as id_Γ. Let x be a fresh variable, and $\Gamma \vdash A$ Type. Then (x_i) can also be regarded as a context morphism $\Gamma, x : A \to \Gamma$. In this case we write it as $p(\Gamma, A)$.

3. Let $\Gamma = x_1 : A_1, \ldots x_n : A_n$ be a context, a be a term with $\Gamma \vdash a : A$, and x a fresh variable. We can form a context morphism $(\bar{a}) : \Gamma \to \Gamma, x : A$ by taking $(\bar{a}) = (x_1, \ldots x_n, a)$.

4. Let $\gamma = (t_i) : \Gamma \to \Delta$, where Δ has n variables $x_1, \ldots x_n$. Let $\Theta = y_1 : C_1, \ldots y_m : C_m$ and suppose Δ, Θ is a well-typed context. Then $\Gamma, \Theta[t_1/x_1, \ldots t_n/x_n]$ is also a well-typed context. Furthermore there is a context morphism $q(\gamma, \Theta) : \Gamma, \Theta[t_1/x_1, \ldots t_n/x_n] \to \Delta, \Theta$ given by $(t_1, \ldots t_n, y_1, \ldots y_m)$.

   In the case that $\Delta = \Gamma, x_n : A$ and $\gamma = p(\Gamma, A)$, the morphism $q(\gamma, \Theta)$ has the signature $\Gamma, x : A, \Theta \to \Gamma, \Theta$ and is given by $(x_1, \ldots x_{n-1}, y_1, \ldots y_m)$. We can think of this as implementing context weakening in the middle of the context, as opposed to $p(\Gamma, A)$ which only weakens at the end.

5. Let $\Gamma = x_1 : A_1, \ldots x_n : A_n$ be a context, a be a term with $\Gamma \vdash a : A$, and $\gamma = (t_i) : \Delta \to \Gamma$ a context morphism. We write $a[\gamma]$ for the term $a[t_1/x_1, \ldots t_n/x_n]$, and likewise $A[\gamma]$ for the type $A[t_1/x_1, \ldots t_n/x_n]$. We then have the judgement $\Delta \vdash a[\gamma] : A[\gamma]$.

Proposition 3.1.1. *1. Let $(t_i) : \Gamma \to \Delta, (s_j) : \Delta \to \Theta$. Suppose Δ has the form $y_1 : B_1, \ldots y_m : B_m$. We may define the composition $s \circ t : \Gamma \to \Theta$ by $(s \circ t)_j = s_j[t_1/y_1, \ldots, t_m/y_m]$.*

2. *Composition of context morphisms is associative, and id_Γ is its identity. This means that we can form a category of contexts and context morphisms, with a terminal object $[]$.*

3. *If $\Gamma \vdash a : A$ then $p(\Gamma, A) \circ \bar{a} = id_\Gamma$.*

4. *If $(t_1, \ldots t_n, t) : \Gamma \to (\Delta, x : A)$ then $p(\Gamma, A) \circ (t_1, \ldots t_n, t) \equiv (t_1, \ldots t_n)$ (as context morphisms $\Gamma \to \Delta$), and $x[(t_1, \ldots t_n, t)] \equiv t : A$ (as terms)*

5. *If $\gamma : \Gamma \to \Delta$ and $\Delta, x : A$ is a well-typed context, then $p(\Delta, A) \circ q(\gamma, x : A) \equiv \gamma \circ p(\Gamma, A[\gamma])$.*

6. *If $\gamma : \Gamma \to \Delta$ and $\Delta \vdash a : A$ then $\bar{a} \circ \gamma \equiv q(\gamma, x : A) \circ \overline{a[\gamma]}$.*

7. *If $\Gamma \vdash A$ Type and x is a fresh variable then $id_{\Gamma, x:A} \equiv (x_1, \ldots x_n, x)$ where $(x_i) = p(\Gamma, A) = id_\Gamma$.*

3.2 Modelling dependent types

Now we are ready to provide an answer to the questions of Section 3.1, and introduce the idea of a categories with families. Just as a CCC is a category

with extra structure suitable for Modelling the STLC, a category with families is a category with extra structure that models dependent types. There are several other approaches to Modelling dependent types (contextual categories, categories with attributes, and comprehension categories), but we will start with cwfs because of their intuitive nature, and close relation to the syntax of dependent types.

To motivate the definition of a cwf we will try once again to construct the syntactic category of ML type theory. We shall start with a base category of contexts and context morphisms. For every context Γ we will have a set of types in context Γ, denoted $\mathbf{Type}(\Gamma) = \{A \mid \Gamma \vdash A\ \mathsf{Type}\}$, and for every such type A a set of terms of type A in context Γ, denoted $\mathbf{Term}(\Gamma, A) = \{a \mid \Gamma \vdash a : A\}$.

Recall that every context morphism $\gamma : \Gamma \to \Delta$ induces a substitution map $-[\gamma]$ taking types and terms of Δ to types and terms of Γ. Furthermore, by Example 3.1.1, we may also regard a judgement $\Gamma \vdash a : A$ as a context morphism $(\bar{a}) : \Gamma \to \Gamma, x : A$. How can we reconcile these two views?

Let's return to the example of Section 3.1, where we have judgements:

$$\begin{aligned} x : A &\vdash t : B \\ x : A, y : B &\vdash s : C \\ x : A &\vdash s[t/y] : C[t/y] \end{aligned}$$

and corresponding context morphisms:

$$\begin{aligned} \bar{t} &= (x, t) : (x : A) \to (x : A, y : B) \\ \bar{s} &= (x, y, s) : (x : A, y : B) \to (x : A, y : B, z : C) \\ \overline{s[t/y]} &= (x, s[t/y]) : (x : A) \to (x : A, w : C[t/y]) \\ \bar{s} \circ \bar{t} &= (x, t, s[t/y]) : (x : A) \to (x : A, y : B, z : C) \end{aligned}$$

Now we may apply Proposition 3.1.1.6 to relate the morphisms $\bar{s} \circ \bar{t}$ and $\overline{s[t/y]}$.

$$\bar{s} \circ \bar{t} = q(\bar{t}, z : C[t/y]) \circ \overline{s[t/y]} : (x : A) \to (x : A, y : B, z : C)$$

Another question we may ask is: "what is the relationship between Γ and $\Gamma, x : A$?" We want to somehow characterize the property that $\Gamma, x : A$ is the unique context (up to renaming x) that extends Γ by A.

The key idea is that we can form the judgement $\Gamma, x : A \vdash x : A$, so we have a variable element $x \in \mathbf{Term}((\Gamma, x : A), A)$. By variable, we mean that any term can be substituted for it. More precisely, if we have $\Gamma \vdash a : A$, we can form the context morphism $\bar{a} : \Gamma \to (\Gamma, x : A)$. Applying the induced substitution to x will yield a:

$$x[\bar{a}] \equiv a$$

Just as every element of $\mathbf{Term}(\Gamma, A)$ induces a morphism $\bar{a} : \Gamma \to (\Gamma, x : A)$, so would we like the opposite to be true—the context morphisms should contain all the information about terms. The following result clarifies this idea.

Well, if we have a context morphism $\gamma : \Gamma \to (\Gamma, x : A)$ then we can form the judgement $\Gamma \vdash t : A[\gamma]$ where t is the last component of γ. Suppose further that $A[\gamma] \equiv A$. If we write $\Gamma = x_1 : A_1, \ldots x_n : A_n$ this means that $\gamma = (x_1, \ldots, x_n, t)$. So the context morphisms $\Gamma \to (\Gamma, x : A)$ that preserve A are in correspondence with terms $\Gamma \vdash t : A$.

Remark 3.2.1. Note that a context morphism preserves A if and only if it is a right section to $p(\Gamma, A)$. This characterization will allow us to generalize the above statement to any cwf.

3.3 Categories with families

This leads to the following definitions.

Definition 3.3.1. The *category of families of sets* is denoted **Fam**. The objects of **Fam** are families of sets $(B_x)_{x \in A}$. A morphism $(B_x)_{x \in A} \to (B'_{x'})_{x' \in A'}$ consists of a function $f : A \to A'$ and a family of functions $(g_x : A_x \to A'_{f(x)})_{x \in A}$.

Remark 3.3.1. Note that **Fam** is equivalent to the arrow category $\mathbf{Set}^{\to}$.

Definition 3.3.2 (Adapted from [Dyb95] and [Hof97]). A *category with families* is:

1. A base category $\mathcal{C}$. The objects of $\mathcal{C}$ are called (semantic) *contexts*, and the morphisms are called (semantic) *substitutions*.

2. A functor $T : C^{\mathrm{op}} \to \mathbf{Fam}$. For a context Γ in $\mathcal{C}$ we have a family of sets $T(\Gamma)$. We write:

$$T(\Gamma) = (\mathbf{Term}(\Gamma, A))_{A \in \mathbf{Type}(\Gamma)}$$

   The intuition is that from any semantic context Γ we get a set of semantic types $\mathbf{Type}(\Gamma)$, containing all the types A with $\Gamma \vdash A\ \mathsf{Type}$. Each component set $\mathbf{Term}(\Gamma, A)$ contains the semantic terms t with $\Gamma \vdash t : A$.

   If $\gamma : \Gamma \to \Delta$ is a morphism of $\mathcal{C}$ then $T(\gamma)$ consists of a function $\gamma_1 : \mathbf{Type}(\Delta) \to \mathbf{Type}(\Gamma)$ and for each $a \in \mathbf{Type}$ a function $\gamma_{2,A} : \mathbf{Term}(\Delta, A) \to \mathbf{Term}(\Gamma, \gamma_1(A))$. For a type $A \in \mathbf{Type}(\Delta)$ and a term $a \in \mathbf{Term}(\Delta, A)$ we will write

$$A\{\gamma\} \text{ for } \gamma_1(A) \in \mathbf{Type}(\Gamma)$$
$$a\{\gamma\} \text{ for } \gamma_{2,A}(a) \in \mathbf{Term}(\Gamma, A\{[\gamma\})$$

   We call these functions (semantic) *substitutions* on (semantic) types and (semantic) terms.

3. A terminal object $\diamond$ of $\mathcal{C}$, which we call the *empty (semantic) context*

   Such that, for every context Γ in $\mathcal{C}$ and every $A \in \mathbf{Type}(\Gamma)$ there is an *extended context* $\Gamma; A$ in $\mathcal{C}$, a *first projection* that is a morphism $p(A) : \Gamma; A \to \Gamma$ in $\mathcal{C}$, and a *second projection*, a term $v_A \in \mathbf{Term}(\Gamma; A, A\{p(A)\})$.

Remark 3.3.2. We call the term v rather than q, as in [Dyb95], to avoid confusion with the context morphism defined in Example 3.1.1.

These enjoy the universal property that for any morphism $\gamma : \Delta \to \Gamma$ of $\mathcal{C}$ and term $a \in \mathbf{Term}(\Delta, A\{\gamma\})$ there is a unique morphism $\langle \gamma, a \rangle : \Delta \to \Gamma; A$ with

$$p \circ \langle \gamma, a \rangle = \gamma \text{ and,}$$
$$v\{\langle \gamma, a \rangle\} = a$$

Now we are ready to describe the term model of ML type theory. The reason for our choice of notation in the definition of a cwf will become clear.

Definition 3.3.3. The *term model* of ML type theory is a category with families, where:

1. We take $\mathcal{C}$ to be the category of (equivalence classes of) well-typed contexts and (equivalence classes of) context morphisms. We will commit a mild abuse of notation and denote equivalence classes by their representatives.

2. The terminal object of $\mathcal{C}$ is the (equivalence class of) the empty context $[]$.

3. Now we describe the functor T. For a context Γ we take $T(\Gamma)$ to be $(\mathbf{Term}(\Gamma, A))_{A \in \mathbf{Type}(\Gamma)}$, where $\mathbf{Type}$ is the set of (definitional equivalence classes of) types A with $\Gamma \vdash A\ \mathsf{Type}$, and $\mathbf{Term}(\Gamma, A)$ is the set of (definitional equivalence classes of) terms a with $\Gamma \vdash a : A$.

   Let $\gamma : \Gamma \to \Delta$ be a context morphism. We describe $T(\gamma)$. First, we have an action on type-sets, i.e. a map $\mathbf{Type}(\Delta) \to \mathbf{Type}(\Gamma)$. We take this to be $A \mapsto A[\gamma]$. We also need, for each $A \in \mathbf{Type}(\Delta)$ a map $\mathbf{Term}(\Delta, A) \to \mathbf{Term}(\Gamma, A[\gamma])$. We take this to be the map $a \mapsto a[\gamma]$. That is, the semantic substitution $-\{\gamma\}$ is given by the syntactic substitution $-[\gamma]$.

   Now, If Γ is a context and A a type in $\mathbf{Type}(\Gamma)$ then we take the extended context $\Gamma; A$ to be $\Gamma, x : A$ (where x is fresh). The morphism $p : \Gamma; A \to \Gamma$ is the morphism $p(\Gamma, A)$ defined above. The term $v \in \mathbf{Term}(\Gamma; A, A[p])$ corresponds to the judgement $\Gamma, x : A \vdash x : A$. We verify that the universal property holds.

   So let $\gamma = (t_i) : \Delta \to \Gamma$ be a context morphism, and $a \in \mathbf{Term}(\Delta, A[\gamma])$, so that we have a judgement $\Delta \vdash a : A[\gamma]$. We take $\langle \gamma, a \rangle : \Delta \to \Gamma; A$ to correspond to the context morphism $(t_1, \ldots t_n, a) : \Delta \to \Gamma, x : A$.

   Now the equalities $p \circ \langle \gamma, a \rangle \equiv \gamma$ and $v[\langle \gamma, a \rangle] \equiv a$ follow immediately from Proposition 3.1.1.4.

   To see uniqueness, suppose $\theta : \Delta \to \Gamma; A$ is a context morphism, and write $\theta = (t_1, \ldots, t_n, t_{n+1})$. Suppose further that $p \circ \theta = \gamma$ and $v[\theta] = a$.

   The first equation implies that the first n components of θ equal those of γ, and the second implies that the last component of θ is a. So we see that θ must equal $\langle \gamma, a \rangle$.

When $(\mathcal{C}, T)$ is a cwf, we may refer to it as just $\mathcal{C}$. Many of the syntactic context morphisms we defined earlier can be defined as semantic context morphisms in a general cwf, such that the syntactic and semantic versions agree (up to equivalence) in the term model. We have already seen this agreement with $p(A)$ and $p(\Gamma, A)$.

Definition 3.3.4. Let $\mathcal{C}$ be a cwf.

1. Let $a \in \mathbf{Term}(\Gamma, A)$. Note that $A = A\{\mathrm{id}_\Gamma\}$ (set-theoretic equality), therefore $a \in \mathbf{Term}(\Gamma, A\{\mathrm{id}_\Gamma\})$ too. Hence define the semantic context morphism $\bar{a}$ by

   $$\bar{a} = \langle \mathrm{id}_\Gamma, a \rangle : \Gamma \to \Gamma; A$$

   If $\mathcal{C}$ is the term model, the semantic context morphism $\bar{a}$ corresponds to the syntactic morphism $\bar{a}$, for if Γ has variables $x_1, \ldots x_n$:

   $$\begin{aligned} \langle \mathrm{id}_\Gamma, a \rangle &= (x_1, \ldots x_n, a) \\ &= \bar{a} \end{aligned}$$

2. Let $\gamma : \Delta \to \Gamma$, and $A \in \mathbf{Type}(\Gamma)$. We may define a context morphism

   $$q(\gamma, A) = \langle \gamma \circ p(A\{\gamma\}), v_{A\{\gamma\}} \rangle : \Delta; A\{\gamma\} \to \Gamma; A$$

   In the term model, we have (writing $\Gamma = x_1, A_1, \ldots x_n, A_n$ and $\gamma = (t_1, \ldots t_n)$.

   $$\begin{aligned} q(\gamma, A) &= \langle \gamma \circ p(A\{\gamma\}, v_{A\{\gamma\}} \rangle : \Delta, x : A[\gamma] \to \Gamma, x : A \\ &= \langle \gamma \circ (x_1, \ldots x_n), x \rangle \\ &= (t_1, \ldots t_n, x) \\ &= q(\gamma, x : A) \end{aligned}$$

   Now we define a *weakening map* by induction as:

   - A projection $p(A) : \Gamma; A \to \Gamma$, or
   - A morphism $q(\delta, B)$ where δ is a weakening map.

   In the term model weakening maps have the form described in Example 3.1.1.

   If γ is a weakening map we may refer to $A\{\gamma\}$ and $a\{\gamma\}$ as A^+ and a^+ when there is no likelihood of confusion. If $\gamma : \Gamma \to \Delta$ is any context morphism we may refer to $q(\gamma, A)$ as γ^+.

Example 3.3.1. Suppose we have the judgement $\Gamma \vdash A\ \mathsf{Type}$. Then, $A \in \mathbf{Type}(\Gamma)$ (in the term model). We do, by weakening, also have the judgement $\Gamma, x : B \vdash A$, and so $A \in \mathbf{Type}(\Gamma, y : B)$. In arbitrary cwfs we will still want to allow such weakenings, but we won't be able to link semantic types and contexts to judgements.

So let's try a different approach. We have the weakening projection $p(B) : \Gamma, x : B) \to \Gamma$, hence the substitution map $-\{p(B)\}$ will take a type of Γ to a type of $\Gamma, x : B$. So we have $A\{p(B)\} \in \mathbf{Type}(\Gamma, x : B)$ (which by the notation convention we can write as A^+. In the term model we have $\Gamma \vdash A\{p(B)\} \equiv A\ \mathsf{Type}$, and in other models we will have similar behaviour.

Hence the intuition behind weakening substitutions is that they allow you to "cast" (in the programming sense) types A in context Γ, Δ to types in the extended context $\Gamma; A; \Delta$, and when both contexts are clear we just write the cast type as A^+.

3.4 Extra structure on cwfs

In this section we define some additional structure on top of a cwf that will model the features of ML–as the products and exponentials of a CCC are required to model product types and function types. We mostly follow [Hof97], filling in some gaps (and correcting some typos) along the way.

The process is straightforward: for every rule concerning type constructors of ML we add corresponding conditions on a cwf. Throughout this section, let $(\mathbf{C}, T)$ be a cwf.

Definition 3.4.1. We say that $\mathcal{C}$ *supports the unit type* if:

1. For any Γ there is a type $\mathbf{1} \in \mathbf{Type}(\Gamma)$ and a term $\star \in \mathbf{Term}(\Gamma, \mathbf{1})$.
2. For any $C \in \mathbf{Type}(\Gamma; \mathbf{1})$ and $d \in \mathbf{Term}(\Gamma, C\{\bar{\star}\})$ there is a term $U_C(d) \in \mathbf{Term}(\Gamma; \mathbf{1}, C)$

such that for any $\gamma : \Delta \to \Gamma$,

$$\begin{aligned} U_C(d)\{\bar{\star}\} &= d \\ \mathbf{1}\{\gamma\} &= \mathbf{1} \in \mathbf{Type}(\Delta) \\ \star\{\gamma\} &= \star \in \mathbf{Term}(\Delta, \mathbf{1}) \\ U_C(d)\{\gamma\} &= U_{C\{\gamma\}}(d\{\gamma\}) \end{aligned}$$

Remark 3.4.1. A good way to gain some intuition for this definition (and the subsequent ones) is to think about what the rules of type theory let you do in the term model, and then generalize to arbitrary cwfs.

Since for every syntactic context Γ we have $\Gamma \vdash \mathbf{1}\ \mathsf{Type}$, we should like to have $\mathbf{1}$ in the type-set $\mathbf{Type}(\Gamma)$. Similarly, we want a term $\star$ inside every term-set $\mathbf{Term}(\Gamma, \mathbf{1})$.

How do we handle the elimination rule? The first hypothesis is that $\Gamma, x : \mathbf{1} \vdash C\ \mathsf{Type}$, i.e. that we have a type C in the extend context $\Gamma, x : \mathbf{1}$. This corresponds to having a type $C \in \mathbf{Type}(\Gamma; \mathbf{1})$.

The second second hypothesis is that we have a judgement $\Gamma \vdash d : C[\star/x]$. First we note (writing $\Gamma = x_1 : A_1, \ldots x_n : A_n$) that the single substitution $C[\star/x]$ can be regarded as applying the context morphism $\bar{\star} = (x_1, \ldots x_n, \star) : \Gamma \to \Gamma, x : \mathbf{1}$ to C. So we this is equivalent to having a term $d \in \mathbf{Term}(\Gamma, C\{\bar{\star}\})$.

The conclusion then states that there is a term U_d with the judgement $\Gamma, x : \mathbf{1} \vdash U_d : C$, or in other words, a term in $\mathbf{Term}(\Gamma; \mathbf{1}, C)$, which we will call $U_C(d)$.

The computation rule is translated in a similar manner, and finally, we require that everything is well-behaved with respect to substitution.

It is clear that the term model supports the unit type.

Definition 3.4.2. We say that $\mathcal{C}$ *supports* Π*-types* if:

(Π-FORM) For any $A \in \mathbf{Type}(\Gamma)$ and $B \in \mathbf{Type}(\Gamma; A)$ there is a semantic type $\Pi(A, B) \in \mathbf{Type}(\Gamma)$.

(Π-ABS) For any $t \in \mathbf{Term}(\Gamma; A, B)$ there is a semantic term $\lambda_{A,B}(t) \in \mathbf{Term}(\Gamma, \Pi(A, B))$

(Π-APP) For any $f \in \mathbf{Term}(\Gamma, \Pi(A, B))$ and $a \in \mathbf{Term}(\Gamma, A)$ there is a semantic term $\mathrm{App}_{A,B}(f, a) \in \mathbf{Term}(\Gamma, B\{\bar{a}\})$

such that:

$$\begin{aligned}
\mathrm{App}_{A,B}(\lambda_{A,B}(t), a) &= t\{\bar{a}\} \\
\Pi(A, B)\{\gamma\} &= \Pi(A\{\gamma\}, B\{q(\gamma, A)\} \\
\lambda_{A,B}(f)\{\gamma\} &= \lambda_{A\{\gamma\},B\{q(\gamma,A)\}}(f\{q(\gamma, A)\} \\
\mathrm{App}_{A,B}(f, a)\{\gamma\} &= \mathrm{App}_{A\{\gamma\},B\{q(\gamma,A)\}}(f\{\gamma\}, a\{\gamma\})
\end{aligned}$$

The equations above are to hold for all types A, B, terms f, t, a and context morphisms $\gamma : \Delta \to \Gamma$.

Remark 3.4.2. The first equation corresponds to the Π-β rule. The other three specify that the new operators are stable under semantic substitution

Hofmann suggest another way to encode this:

Definition 3.4.3. The cwf $\mathcal{C}$ supports Π types if there are operations Π and λ as above, and for every $A \in \mathbf{Type}(\Gamma), B \in \mathbf{Type}(\Gamma; A)$ there is a context morphism

$$\mathrm{App}_{A,B} : \Gamma; A; \Pi(A, B)^+ \to \Gamma; A; B$$

such that

$$p(B) \circ \mathrm{App}_{A,B} = p(\Pi(A, B)^+) \tag{3.6}$$

and for every term $t \in \mathbf{Term}(\Gamma; A, B)$ we have

$$\mathrm{App}_{A,B} \circ \overline{\lambda_{A,B} t\{p(A)\}} = \bar{t} \tag{3.7}$$

and for every morphism $\gamma : \Delta \to \Gamma$ we have

$$\mathrm{App}_{A,B} \circ q(q(\gamma, A), \Pi(A, B)\{q(\gamma, A)\}) = q(q(\gamma, A), B) \circ \mathrm{App}_{A\{\gamma\},B\{q(\gamma,A\}} \tag{3.8}$$

To assist the reader, we will make the weakening explicit. As we have $\Pi(A,B) \in \mathbf{Type}(\Gamma)$, we weaken to $\Gamma; A$ by $p(A) : \Gamma; A \to \Gamma$ to get $\Pi(A,B)^+ = \Pi(A,B)\{p(A)\} \in \mathbf{Type}(\Gamma; A)$.

Remark 3.4.3. Hofmann's paper contains an error : the λ in equation 3.7 is missing.

Proposition 3.4.1. *These two definitions are equivalent.*

Proof. The proof is not very interesting and rather hard to decipher, so we omit it. The key idea, mentioned in the example below, is that the App morphism performs application on variables. □

Example 3.4.1. The term model supports Π-types in the obvious way. If we have $\Gamma, x : A \vdash B\ \mathsf{Type}$, then:

$$\begin{aligned}\Pi(A,B) &=_{\text{def}} \Pi_{x:A}(B)\\ \lambda_{A,B}(t) &=_{\text{def}} \lambda x.t\\ \text{App}_{f,a} &=_{\text{def}} fa\end{aligned}$$

Equation 3.6 holds because of the $\Pi - \beta$ rule. The other equations hold because semantic substitution corresponds to syntactic substitution, and we have rules specifying that substitution preserves judgements.

Note that (writing $\Gamma = x_1 : A_1, \ldots x_n : A_n$) the App morphism corresponds to the context morphism

$$\begin{aligned}&\text{App} : (\Gamma, x : A, y : \Pi_{x:A}B) \to (\Gamma, z : A, w : B)\\ &\text{App} = (x_1, \ldots x_n, x, yx)\end{aligned}$$

In other words, it performs application on variables only, and we recover application on all terms by using substitution. To make this precise, suppose that $\Gamma \vdash a : A, f : \Pi_{x:A}B$, so we can form the context morphism

$$\bar{f} \circ \bar{a} = (x_1, \ldots x_n, a, f) : \Gamma \to \Gamma, x : A, y : \Pi_{x:A}B$$

and then notice that

$$\begin{aligned}\text{App} \circ \bar{f} \circ \bar{a} &= (x_1, \ldots x_n, x, fa)\\ &= \overline{\text{App}_{A,B}(f,a)}\end{aligned}$$

Definition 3.4.4. We say that $\mathcal{C}$ *supports* Σ*-types* if:

> For any $A \in \mathbf{Type}(\Gamma)$ and $B \in \mathbf{Type}(\Gamma; B)$ there is a semantic type $\Sigma(A,B) \in \mathbf{Type}(\Gamma)$.
>
> There is a morphism $\text{Pair}_{A,B} : \Gamma; A; B \to \Gamma; \Sigma(A,B)$
>
> For any $C \in \mathbf{Type}(\Gamma; \Sigma(A,B))$ and any term $c \in \mathbf{Term}(\Gamma, C\{\text{Pair}_{A,B}\})$ there is a term $E_{A,B,C}(c) \in \mathbf{Term}(\Gamma; \Sigma(A,B), C)$

such that the following conditions hold:

$$p(\Sigma(A,B)) \circ \mathrm{Pair}_{A,B} = p(A) \circ p(B)$$
$$E_{A,B,C}(c)\{\mathrm{Pair}_{A,B}\} = c$$

, and all of these morphisms and operations are stable under substitution (from now on we will omit writing out these stability conditions, as the notation is dense and doesn't add much). The first equation says that Pair doesn't touch Γ, and the second models the Σ-ELIM rule.

Example 3.4.2. The term model supports Σ types, for we may take (writing $\Gamma = x_1, A_1, \ldots x_n, A_n$):

$$\Sigma(A,B) =_{\mathrm{def}} \Sigma_{x:A}(B)$$
$$\mathrm{Pair}_(A,B) =_{\mathrm{def}} (x_1, \ldots x_n, \langle x, y\rangle) : (\Gamma, x:A, y:B) \to (\Gamma, z : \Sigma_{x:A}B)$$
$$E_{A,B,C}(c) =_{\mathrm{def}} E_c$$

Note that, as before, the Pair morphism performs pairing on variables, and substitution can handle the other cases.

Definition 3.4.5. We say that $\mathcal{C}$ *supports intensional Id types* if

1. For each type $A \in \mathbf{Type}(\Gamma)$ there is a type $\mathrm{Id}_A \in \mathbf{Type}(\Gamma; A; A^+)$
2. There is a morphism $\mathrm{Refl}_A : \Gamma; A \to \Gamma; A; A^+; \mathrm{Id}_A$
3. For every $C \in \mathbf{Type}(\Gamma; A; A^+; \mathrm{Id}_A)$ and any term $d \in \mathbf{Term}(\Gamma; A, C\{\mathrm{Refl}_A\})$ there is a term $J_{A,C}(d) \in \mathbf{Term}(\Gamma; A; A^+; \mathrm{Id}_A, C)$

such that

$$(\mathrm{Id}_A) \circ \mathrm{Refl}_A = \overline{v_A} : \Gamma; A \to \Gamma; A; A^+$$
$$J_{A,C}(d)\{\mathrm{Refl}_A\} = d$$

and the same sort of stability under substitution conditions hold. The first rule ensures that Refl doesn't touch Γ and duplicates v_A, while the second says that the Id-ELIM rule holds.

Example 3.4.3. The term model supports intensional Id types, for if we write $\Gamma = x_1, A_1, \ldots x_n, A_n$ then:

$$\mathrm{Id}_A =_{\mathrm{def}} \mathrm{Id}_A(x,y) \in \mathbf{Type}(\Gamma, x:A, y:A)$$
$$\mathrm{Refl}_A =_{\mathrm{def}} (x_1, \ldots, x_n, x, x, r(x)) : (\Gamma, x:A) \to (\Gamma, y:A, z:A, p : \mathrm{Id}_A(y,z))$$
$$J_{A,C}(d) =_{\mathrm{def}} J_d$$

Definition 3.4.6. Say that $\mathcal{C}$ supports extensional Id types if it supports intensional Id types, and furthermore:

1. Whenever $A \in \mathbf{Type}(\Gamma)$ and $C \in \mathbf{Type}(\Gamma; A; A^+; \mathrm{Id}_A)$ and $c \in \mathbf{Term}(\Gamma; A; A^+; \mathrm{Id}_A, C)$ then $c = J_{A,B}(c\{\mathrm{Refl}_A\})$

The next proposition is not completely obvious, so we prove it.

Proposition 3.4.2. *The term model of extensional ML supports extensional Id types.*

Proof. Fix a context $\Gamma = x_1 : A_1, \ldots x_n : A_n$. Suppose $\Gamma \vdash A$ Type and $\Delta =_{\mathrm{def}} \Gamma, x : A, y : A, z : \mathrm{Id}_A(x, y) \vdash c : C$.

Consider the context morphism $\gamma =_{\mathrm{def}} p(A) \circ p(\mathrm{Id}_A(x, y) : \Delta \to \Gamma, x : A$, which we can concretely write as:

$$(x_1, \ldots x_n, x)$$

Now consider the composition

$$\begin{aligned} &\mathrm{Refl}_A \circ \gamma : \Delta \to \Delta \\ &\mathrm{Refl}_A \circ \gamma = (x_1, \ldots, x_n, x, x, r(x)) \end{aligned}$$

Note that we have the judgements

$$\begin{aligned} &\Delta \vdash z : \mathrm{Id}_A(x, y) \\ &\Delta \vdash x \equiv y : A \\ &\Delta \vdash z \equiv r(x) : \mathrm{Id}_A(x, y) \end{aligned}$$

by the extensionality rule.

Hence we may conclude a (syntactic, and therefore semantic in the term model) equivalence of context morphisms $\mathrm{Refl}_A \circ \gamma \equiv \mathrm{id}_\Delta$.

Now we may compute:

$$\begin{aligned} J_{A,C}(c\{\mathrm{Refl}_A\}) &= J_{A,C}(c\{\mathrm{Refl}_A\})\{\mathrm{Refl}_A \circ \gamma\} \\ &= (J_{A,C}(c\{\mathrm{Refl}_A\})\{\mathrm{Refl}_A\})\,\{\gamma\} \\ &= (c\{\mathrm{Refl}_A\})\,\{\gamma\} \\ &= c\{\mathrm{Refl}_A \circ \gamma\} \\ &= c \end{aligned}$$

So the extensional term model supports extensional identity types, as desired. □

Definition 3.4.7. Say that $\mathcal{C}$ *supports ML* if it supports unit types, Π types, Σ types, and Id types. If, in addition, it supports extensional Id types we say that $\mathcal{C}$ *supports extensional ML*.

3.5 An interpretation function

Let $\mathcal{C}$ be a cwf that supports ML. The goal of this section is to show how we might define a function $[\![-]\!]$ that will take:

Well-typed contexts Γ	to	semantic contexts $[\![\Gamma]\!]$ in $\mathcal{C}$
Judgements $\Gamma \vdash A\ \mathsf{Type}$	to	semantic types $[\![A]\!] \in \mathbf{Type}([\![\Gamma]\!])$
Judgements $\Gamma \vdash a : A$	to	semantic terms $[\![a]\!] \in \mathbf{Term}([\![\Gamma]\!], [\![A]\!])$

There are two possible approaches. The first and more standard, as seen in [AT10], is to define $[\![-]\!]$ by induction on derivations of judgements. The problem is that derivations of judgements need not be unique, so we will need to additionally prove that different derivations of the same judgement yield equal results.

Hofmann uses a different method, inductively defining a partial interpretation function $[\![-]\!]$ that sends (potentially non-well-typed):

Contexts Γ	to	$[\![\Gamma]\!]$ in $\mathcal{C}$
Contexts Γ and types A	to	$[\![A]\!] \in \mathbf{Type}([\![\Gamma]\!])$
Contexts Γ and terms a	to	$[\![a]\!] \in \mathbf{Term}([\![\Gamma]\!], A)$, for some $A \in \mathbf{Type}([\![\Gamma]\!])$.

Then it is shown that this interpretation function is defined on well-typed contexts, types, and terms. The proof of this is by induction on derivations of judgements.

The proof that the usual soundness and completeness properties are satisfied is straightforward but rather involved. For lack of space we omit it.

3.6 Fibrations and comprehension categories

Categories with families are simple and straightforward to understand, but the definition feels rather synthetic. In contrast, comprehension categories (introduced by Jacobs in [Jac93]) have a more minimal notation, and a cleaner categorical theory. We begin by recalling some definitions.

Definition 3.6.1. Let $p : \mathcal{E} \to \mathcal{B}$ be a functor. A morphism $f : D \to E$ in $\mathcal{E}$ is called *strong cartesian over the morphism* $u : A \to B$ in $\mathcal{B}$ if:

1. $pf = u$, and
2. For any $f' : D' \to E$ in $\mathcal{E}$ and $v : A' \to A$ in $\mathcal{B}$ with $pf' = u \circ v$ there is a unique $\phi : D' \to D$ with $f' = f \circ \phi$ and $p\phi = v$.

We call the functor p a *fibration* if for every object E in $\mathcal{E}$ and morphism $u : A \to pE$ in $\mathcal{B}$ there is a strong cartesian $f : D \to E$ over u.

Remark 3.6.1. This is a generalization of a fibration in topology, a map that satisfies the homotopy lifting property. Just as we can think of a topological fibration as consisting a fiber space that "varies" (or is indexed by) the base

space, so too can we think of a categorical fibration as a fiber category that varies over a base category.

If $p : \mathcal{E} \to \mathcal{B}$ is a fibration, then for any B in $\mathcal{B}$ we may define the fiber category $p^{-1}B$. The objects of $p^{-1}(B)$ are objects E in $\mathcal{E}$ with $pE = B$, and the arrows are $f : E \to E'$ in $\mathcal{E}$ with $pf = \mathrm{id}_B$.

In this way we can define a functor $\mathcal{B}^{\mathrm{op}} \to \mathbf{Cat}$ from any fibration (often called the *Grothendieck construction*), and in fact the converse is true too.

Definition 3.6.2. A *comprehension category* is a functor $\mathcal{P} : \mathcal{E} \to \mathcal{B}^{\to}$ such that:

1. $\mathrm{cod} \circ \mathcal{P} : \mathcal{E} \to \mathcal{B}$ is a fibration, and
2. Whenever f is a strong cartesian morphism in $\mathcal{E}$ the morphism $\mathcal{P}f$ in $\mathcal{B}$ is a pullback.

Recall that $\mathcal{B}^{\to}$ is the arrow category of $\mathcal{B}$. Its objects are arrows $f : A \to B$ in $\mathcal{B}$, and a morphism from $f : A \to B$ to $g : C \to D$ consists of two arrows $h : A \to C$ and $j : B \to D$ such that the square below commutes:

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ \downarrow{\scriptstyle h} & & \downarrow{\scriptstyle j} \\ C & \xrightarrow{g} & D \end{array}$$

The functor $\mathrm{cod} : \mathcal{B}^{\to} \to \mathcal{B}$ sends objects $f : A \to B$ to their codomain, B, and morphisms (h, j) to their second component.

Proposition 3.6.1. *Any cwf is also a comprehension category.*

Proof. Let $(\mathcal{C}, T)$ be a cwf. There is an evident functor $\mathbf{Type} : \mathcal{C}^{\mathrm{op}} \to \mathbf{Set}$ that sends contexts Γ to $\mathbf{Type}(\Gamma)$ and context morphisms $\gamma : \Gamma \to \Delta$ to the substitutions $-\{\gamma\} : \mathbf{Type}(\Delta) \to \mathbf{Type}(\Gamma)$.

We take $\mathcal{E}$ to be the category whose objects are pairs (Γ, A) with Γ a context of $\mathcal{C}$ and $A \in \mathbf{Type}(\Gamma)$. For every morphism $\gamma : \Delta \to \Gamma$ in $\mathcal{C}$ we have a corresponding morphism $\tilde{\gamma} : (\Delta, A\{\gamma\}) \to (\Gamma, A)$ in $\mathcal{E}$.

Now we take $\mathcal{P} : \mathcal{E} \to \mathcal{C}^{\to}$ to be the functor that sends

- Objects (Γ, A) to $p(A) : \Gamma; A \to \Gamma$, and
- Morphisms $\tilde{\gamma} : (\Delta, A\{\gamma\}) \to (\Gamma, A)$ to $(q(\gamma, A), \gamma)$.

To confirm that the action on morphisms is well defined, form the diagram

$$\begin{array}{ccc} \Delta; A\{\gamma\} & \xrightarrow{p(A\{\gamma\})} & \Delta \\ \downarrow{\scriptstyle q(\gamma,A)} & & \downarrow{\scriptstyle \gamma} \\ \Gamma; A & \xrightarrow{p(A)} & \Gamma \end{array}$$

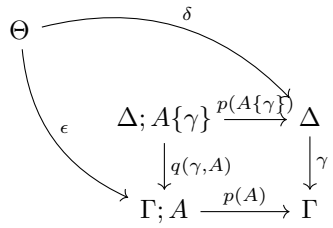
Noting that $q(\gamma, A) =_{\text{def}} \langle \gamma \circ p(A\{\gamma\}), v_{A\{\gamma\}} \rangle$ it follows that

$$\begin{aligned} p(A) \circ q(\gamma, A) &= p(A) \circ \langle \gamma \circ p(A\{\gamma\}), v_{A\{\gamma\}} \rangle \\ &= \gamma \circ p(A\{\gamma\}) \end{aligned}$$

so the diagram commutes, as desired.

Let $p = \text{cod} \circ \mathcal{P} : \mathcal{E} \to \mathcal{C}$. We want to show that p is a fibration. Let (Γ, A) be an object of $\mathcal{E}$, and $\gamma : \Delta \to \Gamma$. We will show that $\tilde{\gamma}$ is strongly cartesian over γ. Certainly $p\tilde{\gamma} = \gamma$. Furthermore, suppose $\tilde{\delta} : (\Theta, A\{\delta\} \to (\Gamma, A)$ is a morphism of $\mathcal{E}$, and $\epsilon : \Omega \to \Delta$ a morphism of $\mathcal{C}$ with $p\tilde{\delta} = \gamma \circ \epsilon$. As $p\tilde{\delta} = \delta = \delta$, we see that $\delta = \gamma \circ \epsilon$, and we may take $\phi : (\Theta, A\{\delta\}) \to (\Delta, A\{\gamma\})$ to be $\tilde{\epsilon}$. So it follows that p is indeed a fibration.

Now we need to confirm that the commutative square above is actually a pullback. So suppose we have a context Θ and morphisms δ, ϵ such that the diagram below commutes.



Note that $v_A \in \mathbf{Term}(\Gamma; A, A\{p(A)\})$, so therefore

$$\begin{aligned} v_A\{\epsilon\} &\in \mathbf{Term}(\Theta, A\{p(A) \circ \epsilon\}) \\ &= \mathbf{Term}(\Theta, A\{\gamma \circ \delta\}) \end{aligned}$$

Hence we may take $\phi : \Theta \to \Delta; A\{\gamma\}$ to be $\langle \delta, v_A\{\epsilon\} \rangle$. By the universal property, we have

$$\begin{aligned} p(A\{\gamma\}) \circ \phi &= p(A\{\gamma\}) \circ \langle \delta, v_A\{\epsilon\} \rangle \\ &= \delta \end{aligned}$$

So the top triangle commutes, as required.

We may also form the morphism

$$\begin{aligned} &\psi : \Theta \to \Gamma; A \\ &\psi = \langle \gamma \circ \delta, v_A\{\epsilon\} \rangle \end{aligned}$$

By the universal property, this is the unique morphism with

- $p(A) \circ \psi = \gamma \circ \delta$, and
- $v_A\{\psi\} = v_A\{\epsilon\}$

But note that ϵ and $q(\gamma, A) \circ \phi$ also satisfy these equations, for:

$$\begin{aligned} p(A) \circ \epsilon &= \gamma \circ \delta \\ p(A) \circ q(\gamma, A) \circ \phi &= p(A) \circ \langle \gamma \circ p(A\{\gamma\}), v_{A\{\gamma\}} \rangle \circ \phi \\ &= \gamma \circ p(A\{\gamma\}) \circ \langle \delta, v_A\{\epsilon\} \rangle \\ &= \gamma \circ \delta \end{aligned}$$

and

$$\begin{aligned} v_A\{\epsilon\} &= v_A\{\epsilon\} \\ v_A\{q(\gamma, A) \circ \phi\} &= v_A\{\langle \gamma \circ p(A\{\gamma\}), v_{A\{\gamma\}} \rangle\}\{\phi\} \\ &= v_{A\{\gamma\}}\{\langle \delta, v_A\{\epsilon\} \rangle\} \\ &= v_A\{\epsilon\} \end{aligned}$$

Hence it follows that $\epsilon = q(\gamma, A) \circ \phi$, and so the diagram commutes. Uniqueness of ϕ follows from the universal property of $\langle -, - \rangle$. $\square$

Chapter 4

Conclusion

We have introduced Martin Löf's dependent type theory, and its intensional and extensional variants. We have explored the three interesting type constructors, Π, Σ, and Id, and we have commented on how identity types allow for a rich groupoid structure on types—the "homotopical perspective".

In order to understand the categorical semantics of dependent types, we had to take a closer look at contexts, and define context morphisms. The key issue is that contexts of the simply typed lambda calculus really are just made up of their types, with no regard for order or any other structure, but in a dependent type theory, types in a context may depend on previous types. This means that types and terms aren't *absolute*, they only make sense with respect to a context.

Categories with families are an intuitive model of ML type theory. A cwf consists of a base category of contexts and context morphisms, and for each context, a set of terms in context indexed by the set of types in context. Just as a CCC is a category with extra structure to model the type constructors of the STLC, we defined notions of extra structure on cwfs that model the type constructors of ML. We defined the term model as a cwf, and showed that it supports this extra structure.

Finally, we take a step back and generalize the notion of a cwf to a comprehension category, a (perhaps) less intuitive model of dependent types, but with a cleaner categorical formulation.

References

[Mar75] Per Martin-Löf. "An intuitionistic theory of types: Predicative part". In: *Studies in Logic and the Foundations of Mathematics*. Vol. 80. Elsevier, 1975, pp. 73–118.

[How80] William A Howard. "The formulae-as-types notion of construction". In: *To HB Curry: essays on combinatory logic, lambda calculus and formalism*. Academic Press, 1980, pp. 479–490.

[Mar82] Per Martin-Löf. "Constructive mathematics and computer programming". In: *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences* 312.1522 (1982), pp. 501–518.

[MS84] Per Martin-Löf and Giovanni Sambin. *Intuitionistic type theory*. Vol. 9. 1984.

[Lam85] Joachim Lambek. "Cartesian closed categories and typed λ-calculi". In: *LITP Spring School on Theoretical Computer Science*. Springer. 1985, pp. 136–175.

[Car86] John Cartmell. "Generalised algebraic theories and contextual categories". In: *Annals of pure and applied logic* 32 (1986), pp. 209–243.

[Jac93] Bart Jacobs. "Comprehension categories and the semantics of type dependency". In: *Theoretical Computer Science* 107.2 (1993), pp. 169–207.

[Dyb95] Peter Dybjer. "Internal type theory". In: *International Workshop on Types for Proofs and Programs*. Springer. 1995, pp. 120–134.

[Hof95] Martin Hofmann. "Extensional concepts in intensional type theory". PhD thesis. University of Edinburgh, 1995.

[HS95] Martin Hofmann and Thomas Streicher. "The groupoid interpretation of type theory". In: *Twenty-five years of constructive type theory*. 1995.

[Hof97] Martin Hofmann. "Syntax and semantics of dependent types". In: *Extensional Constructs in Intensional Type Theory*. Springer, 1997, pp. 13–54.

[Gar09] Richard Garner. "Two-dimensional models of type theory". In: *Mathematical Structures in Computer Science* 19.4 (2009), pp. 687–736.

[AT10] Samson Abramsky and Nikos Tzevelekos. "Introduction to categories and categorical logic". In: *New structures for physics*. Springer, 2010, pp. 3–94.

[Str12] Thomas Streicher. *Semantics of type theory: correctness, completeness and independence results*. Springer Science & Business Media, 2012.

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

[CD14] Pierre Clairambault and Peter Dybjer. "The biequivalence of locally cartesian closed categories and Martin-Löf type theories". In: *Mathematical Structures in Computer Science* 24.6 (2014).