Appendix B

Type theory

This Appendix is meant as a brief introduction to Martin-Löf type theory. Although the relevance of the theory of ΠW-pretoposes to type theory is not what will be pursued in this thesis (it is applications to set theory that will be worked out), type theory is "ideologically" important. It is certainly arguable, but in my opinion Martin-Löf type theory is the paradigmatic constructive-predicative theory. It is not the absence of the Law of Excluded Middle or the powerset axiom that vindicates the constructive and predicative status of the set theory **CZF**, but its interpretation in type theory. In the same vein I feel that the example of a ΠW-pretopos definable from type theory, proves that a ΠW-pretopos is a constructive and predicative structure (this example is discussed in the Chapter 2 of this thesis). For this reason I think it is important to introduce Martin-Löf type theory. But I should say right away that I realise that what I tell here can in no way compete with the book-length expositions by the experts (see [57] and [63], and also [64]).

There is immediately one complication: there exist different versions of this theory, extensional and intensional, polymorphic and monomorphic. Following [63], I have made the choice to introduce the polymorphic version, which I feel is easier to motivate, but I am not unaware of the advantages of the monomorphic version (for that, see [64]). Comments on these issues I have relegated to the footnotes. For the polymorphic version, I will discuss both the intensional and extensional versions.

Per Martin-Löf created the type theory that bears his name in order to clarify constructive mathematics. And, in fact, I believe that it is as an attempt to analyse the practice of the constructive mathematician that it can best be introduced.

An exposition of type theory should be preceeded by a discussion of Martin-Löf's theory of expressions. I wish to treat this issue very quickly, referring to Chapter 3 of [63] for more details. The expression

$$y + \sin y$$

is analysed as the *application* of a binary operation $+$ to a variable y and an expression consisting of the unary function sin applied to that same variable y. Using brackets

for application, this expression should be written as:

$$+(y, \sin(y)).$$

In the expression

$$\int_1^x y + \sin y \, dy$$

the variable y is considered to be a *dummy*. Nothing depends essentially on it being y, rather than z or α. For that reason, the integral $\int$ is thought of as working on an expression, the integrand $y + \sin y$, in which y does not occur freely, but has rather been abstracted away. This abstraction is denoted by

$$(y) \; + (y, \sin(y)).$$

Certain expressions involving application and abstraction are considered equal (*definitionally equal*, denoted by $\equiv$), like:

$$((x)\, e)(x) \equiv e,$$

or

$$(x)\, b \equiv (y)\, b'$$

if b' is b, with all free occurences of y replaced by x. In fact, as explained in [63], application and abstraction satisfy the rules of the typed lambda calculus (with the α, β and η-rule).

After the development of the theory of expressions, the first step in the analysis of mathematics is that all the expressions (terms, like constants, variables etc.) are always of a certain *type*. A mathematician who, in the course of an argument, introduces a certain variable x never assumes that x is just an arbitrary set, but always assumes that x is a mathematical object of a particular kind, like a natural number, a 2 by 2 matrix, an element of a group etc. These mathematical kinds are called *types* and a basic assumption in type theory is that all mathematical objects are always presupposed to be of a certain type. That x is of type A is usually abbreviated as $x \in A$ and that A is a type as $A\,\mathrm{Type}$, the fact that A and A' are equal types as $A = A'$. An additional aspect in the analysis is that mathematical objects can only be compared as elements of the same type. In type theory, it makes no sense to wonder whether x, which is of type natural number, and y, which is of type 2 by 2 matrix, are identical or not identical. More controversially perhaps, the question whether the real number 2 is the same as the integer 2 is regarded as ill-posed. When a and b are equal objects of type A, this is written as $a = b \in A$. I have now enumerated all the *judgement forms*, which are

$$\begin{aligned} &A\,\mathrm{Type} \\ &A = A' \\ &a \in A \\ &a = b \in A. \end{aligned}$$

These are the kind of statements that are recognised in Martin-Löf type theory.

Types have structure and the next two steps analyse this structure. The second step is the recognition of *dependent types*: types can depend on the value of a term x of another type. An example is $\mathbb{R}^n$, which is a type dependent on the value of $n \in \mathbb{N}$. Another example is from category theory, where the homset

$$\mathrm{Hom}(A, B)$$

depends on the values of A and B of type "object in the category $\mathcal{C}$". The recognition of dependent types is the main cause of the technical difficulty of the theory.

A third step is the recognition of *type constructors*. There are ways of building new types from old types. For example, when σ and τ are types, there is the type

$$\sigma \times \tau$$

of pairs whose first element is of type σ and second element is of type τ. For example, one could form the type $\mathbb{N} \times \mathbb{N}$ of pairs of natural numbers. In this example, $\times$ is the type constructor, but there are also type constructors acting on dependent types, as in the next example. When $B(a)$ is a type dependent on $a \in A$, there is the type

$$\Sigma a \in A.\, B(a)$$

of pairs (a, b) where $a \in A$ and $b \in B(a)$. In set theory, one usually writes something like $\coprod_{a \in A} B_a$. This allows one to build the type

$$\Sigma n \in \mathbb{N}.\, \mathbb{R}^n$$

which is the type of finite sequences of reals together with their length.

In the final (fourth) step, there is the rôle of *contexts*. In the course of an argument, when the mathematician has introduced all kinds of variables x, y ..., and she is reasoning about them, there is always implicit the typing information, which gives the right types for all the variables she is working with. In a formal system like Martin-Löf type theory, all these assumptions are required to be made fully explicit. Therefore judgements like

$$a \in \sigma$$

are always made within a context Γ which gives explicitly all the types of the free variables occurring in a and σ. For example, the statement that $\mathbb{R}^n$ is a type can only be made within the context $n \in \mathbb{N}$:

$$n \in \mathbb{N} \vdash \mathbb{R}^n \,\mathrm{Type}.$$

Therefore the statements that are premisses or conclusions in an argument are of the following shapes:

$$\begin{aligned}
&\Gamma \vdash A \,\mathrm{Type}\\
&\Gamma \vdash A = A'\\
&\Gamma \vdash a \in A\\
&\Gamma \vdash a = b \in A,
\end{aligned}$$

where Γ is a context.

The general form of a context is the following:

$$\Gamma = [a_1 \in \sigma_1, a_2 \in \sigma_2(a_1), \ldots, a_n \in \sigma_n(a_1, a_2, \ldots, a_{n-1})],$$

where the a_i are distinct variables of the appropriate types. This includes the *empty context* [] for $n = 0$. The presupposition here is that

$$a_1 \in \sigma_1, \ldots, a_i \in \sigma_i(a_1, \ldots a_{i-1}) \vdash \sigma_{i+1}(a_1, \ldots, a_i)\,\mathrm{Type}$$

for all $i < n$.

A brief remark about these "presuppositions": what they amount to in this case is that Γ can never be a context occurring in a statement, without these presuppositions having been derived before. Typically in logic, the well-formed statements are delineated, before formulating rules circumscribing which of those are provable. Here, both processes occur simultaneously: well-formedness of types, for example, is a property that has to be derived within the system (this is why there is a judgement form $A\,\mathrm{Type}$). This is also why the axiom $A = A$ has as a premiss $A\,\mathrm{Type}$, because otherwise it could possibly not be well-formed.

Martin-Löf type theory is organized as follows: it is a system like natural deduction, with two sets of rules. First, there is a basic set of axioms, that essentially regulates the use of $=$: it is an equivalence relation allowing substitution. Then there are the rules for the different type constructors, four for each.

The rules for $=$ are:

$$\frac{a \in A}{a = a \in A} \quad \frac{a = b \in A}{b = a \in A} \quad \frac{a = b \in A \quad b = c \in A}{a = c \in A}$$

$$\frac{A\,\mathrm{Type}}{A = A} \qquad \frac{A = B}{B = A} \qquad \frac{A = B \quad B = C}{A = C}$$

$$\frac{a \in A \quad A = B}{a \in B} \quad \frac{a = b \in A \quad A = B}{a = b \in B}$$

These rules have to be read with the following convention in mind: when formulating a rule, the context that is shared by all the premisses and conclusion is omitted. This means that the first rule in this list is really:

$$\frac{\Gamma \vdash a \in A}{\Gamma \vdash a = a \in A}$$

for any context Γ.

The substitution rules are as follows:

$$\frac{x \in A \vdash C(x)\,\mathrm{Type} \quad a \in A}{C(a)\,\mathrm{Type}}$$

$$\frac{x \in A \vdash C(x) \quad a = b \in A}{C(a) = C(b)}$$

$$\frac{x \in A \vdash c(x) \in C(x) \quad a \in A}{c(a) \in C(a)}$$

$$\frac{x \in A \vdash c(x) \in C(x) \quad a = b \in A}{c(a) = c(b) \in C(a)}$$

$$\frac{x \in A \vdash C(x) = D(x) \quad a \in A}{C(a) = D(a)}$$

$$\frac{x \in A \vdash c(x) = d(x) \in C \quad a \in A}{c(a) = d(a) \in C}$$

And finally there is the following assumption rule:

$$\frac{A\,\mathrm{Type}}{x \in A \vdash x \in A}$$

The second set of rules consists of four rules for every type constructor. I start with Π. It is analogous to the construction of the set $\Pi_{i \in I} A_i$ for an indexed family $(A_i)_{i \in I}$ in set theory: it is the set of functions that chooses for each $i \in I$ an element in the corresponding A_i.

First there is the *formation rule*:

$$\frac{A\,\mathrm{Type} \quad x \in A \vdash B\,\mathrm{Type}}{\Pi x \in A.\, B(x)\,\mathrm{Type}}$$

The *introduction rule*:

$$\frac{x \in A \vdash b(x) \in B(x)}{\lambda(b) \in \Pi x \in A.\, B(x).}$$

The *elimination rule*:

$$\frac{f \in \Pi x \in A.\, B(x) \quad a \in A}{\mathrm{apply}(f, a) \in B(a),}$$

and the *equality rule*:

$$\frac{x \in A \vdash b(x) \in B(x) \quad a \in A}{\mathrm{apply}(\lambda(b), a) = b(a) \in B(a).}$$

In case B does not contain x, one usually writes $A \to B$ instead of $\Pi x \in A.\, B(x)$. These rules are secretly accompanied by rules for judgemental equality ($=$) like the following:

$$\frac{x \in A \vdash b(x) = c(x) \in B(x)}{\lambda(b) = \lambda(c) \in \Pi x \in A.\, B(x).}$$

But these will be omitted in the sequel.[1]

The other types are thought of as being inductively generated.[2] The general pattern can be observed from the rules for $\times$, the (binary) product type. First, there is the formation rule:

$$\frac{A\,\mathrm{Type} \quad B\,\mathrm{Type}}{A \times B\,\mathrm{Type}}$$

The introduction rule is:

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

If one thinks of types as boxes, this rule tells us that there is a canonical way of putting something into the box $A \times B$: take elements $a \in A$ and $b \in B$ and pair them (elements of the form $\mathrm{pair}(a, b)$ are therefore called canonical elements). The elimination rule expresses that such elements exhaust the product type in the form of an associated induction principle:

$$\frac{p \in A \times B \quad v \in A \times B \vdash C(v) \quad x \in A, y \in B \vdash e(x, y) \in C(\mathrm{pair(a,b)})}{\mathrm{split}(p, e) \in C(p).}$$

What this says, in terms of boxes, is that in case I am given a family of boxes $C(v)$ labelled by elements v in the type $A \times B$ and that I am given a way of putting elements into boxes for every box labelled by a canonical element (i.e. into $C(\mathrm{pair}(x, y))$ for every $x \in A$ and $y \in B$), I have a way of putting elements in every box. The associated equality rule says that this way agrees with (extends) the given method for the canonical elements:

$$\frac{a \in A \quad b \in B \quad x \in A, y \in B \vdash e(x, y) \in C(\mathrm{pair}(x, y))}{\mathrm{split}(\mathrm{pair}(a, b), e) = e(a, b) \in C(\mathrm{pair}(a, b)).}$$

The rules for all the type constructors follow this pattern. The rules for the remaining type constructors will be given at the end of this Appendix.

So far any mathematician, even the classical one, may sympathise with the development of the theory. It is by taking the next step that the system becomes essentially constructive. As one may have the feeling that I have only explained the set-theoretic part of the system, one may wonder how logic is incorporated in it. This is done

[1] When the monomorphic version is formulated in terms of a logical framework, as is customary, it is not necessary to add these rules.

[2] It is possible to formulate the monomorphic version in such a way that the elements of the Π-types are also inductively generated.

following the propositions-as-types interpretation: a proposition is interpreted as the type of its proofs. The type constructors correspond to the various logical constants and a proposition is considered true, when its corresponding type of proofs is inhabited, i.e. there is a term of the appropriate type. This in itself might not make the system constructive, but the type-theoretic understanding of what a proof is, does. The type-theoretic interpretation (which follows the BHK-interpretation) of a proof of an existential proposition $\exists a \in A.\, B(a)$ is as the Σ-type $\Sigma a \in A.\, B(a)$, which therefore means that implicit in a proof of this proposition is an $a \in A$ which has the desired property $B(a)$. Likewise, the Π-type interprets the universal quantifier $\forall$. One can see that the natural deduction rules for $\forall$ are derived rules for the system:

$$\frac{x \in A \vdash B(x)\,\mathrm{True}}{\forall x \in A.\, B(x)\,\mathrm{True}} \qquad \frac{\forall x \in A.\, B(x)\,\mathrm{True} \quad a \in A}{B(a)\,\mathrm{True}.}$$

The $\times$-type interprets conjunction $\wedge$ and the following are also derived rules:

$$\frac{A\,\mathrm{True} \quad B\,\mathrm{True}}{A \wedge B\,\mathrm{True}} \qquad \frac{A \wedge B\,\mathrm{True}}{A\,\mathrm{True}} \qquad \frac{A \wedge B\,\mathrm{True}}{B\,\mathrm{True}.}$$

In order to have a complete translation from first-order intuitionistic logic into type theory, one needs to have identity types. In the course of history, two sets of rules have been formulated, an intensional version Id and an extensional version Eq, resulting in two different type theories: intensional and a stronger extensional type theory.[3]

Id-type (intensional) Intuitive description: set of proofs of an identity statement.

Formation rule
$$\frac{A\,\mathrm{Type} \quad a \in A \quad b \in A}{\mathrm{Id}(A, a, b)\,\mathrm{Type}}$$

Introduction rule
$$\frac{a \in A}{\mathrm{r}(a) \in \mathrm{Id}(A, a, a)}$$

Elimination rule
$$\frac{\begin{array}{c} a \in A \\ b \in A \\ c \in \mathrm{Id}(A, a, b) \\ x \in A, y \in A, z \in \mathrm{Id}(A, x, y) \vdash C(x, y, z)\,\mathrm{Type} \\ x \in A \vdash d(x) \in C(x, x, \mathrm{r}(x)) \end{array}}{\mathrm{J}(c, d) \in C(a, b, c)}$$

Equality rule
$$\frac{\begin{array}{c} a \in A \\ x \in A, y \in A, z \in \mathrm{Id}(A, x, y) \vdash C(x, y, z)\,\mathrm{Type} \\ x \in A \vdash d(x) \in C(x, x, \mathrm{r}(x)) \end{array}}{\mathrm{J}(\mathrm{r}(a), d) = d(a) \in C(a, a, \mathrm{r}(a))}$$

[3] The extensional identity type does not fit into the monomorphic version of type theory as formulated in [64].

Eq-type (extensional) Intuitive description: set of proofs of an identity statement.

$$\text{Formation rule} \qquad \frac{A\,\mathrm{Type} \quad a \in A \quad b \in A}{\mathrm{Eq}(A, a, b)\,\mathrm{Type}}$$

$$\text{Introduction rule} \qquad \frac{a \in A}{\mathsf{r} \in \mathrm{Eq}(A, a, a)}$$

$$\text{Elimination rule} \qquad \frac{c \in \mathrm{Eq}(A, a, b)}{a = b \in A}$$

$$\text{Equality rule} \qquad \frac{c \in \mathrm{Eq}(A, a, b)}{c = \mathsf{r} \in \mathrm{Eq}(A, a, b)}$$

One can see that the intensional version fits with the philosophy that types are inductively generated sets and therefore with the general pattern, but the extensional version is closer to category theory. Another important difference is that in type theory with the extensional identity types, the judgemental equality $=$ and the propositional equality Eq collapse, thereby making judgmental equality and type checking undecidable, while in a type theory with intensional equality types, the different equalities are kept apart and judgmental equality and type checking remain decidable.

Also because of this, Martin-Löf considers the intensional version the right one, and although I appreciate the philosophical and computer-scientific reasons for this, the category theorist in me is dismayed, as it makes the categorical properties of the system much more akward. Also it makes the theory of ML-categories and ΠW-pretoposes less relevant to the study of Martin-Löf type theory.

From the syntax of type theory, whether intensional or extensional, one can build a category in the following way. Objects are types (within the empty context) modulo the judgemental equality $=$, while morphisms from a type A to a type B are terms of type $A \to B$, again modulo the equality $=$. The fact that one so obtains a category, is entirely trivial.

It is an (extension of a) result by Seely [79] that for extensional type theory this gives an ML-category.[4] It would be very convenient if one could prove that this was the initial ML-category, but, as people discovered, this overlooks subtle coherence problems related to substitution. This is connected to the general problem of interpreting extensional type theories in ML-categories.[5]

But solutions to the latter problem have been found: one can use the theory of fibrations, see [36] and [41], or change the type theory by introducing explicit substitution operators, see [25]. In either way, one can consider ML-categories and ΠW-pretoposes as models of extensional type theory. It is on this fact that the

[4] But in this connection universes are essential to show that the sums are disjoint. Moreover, an extension of the type theory with quotient types should yield a ΠW-pretopos.

[5] Strictly speaking, a categorical semantics has only been worked out for the monomorphic version.

relevance of the theory of ML-categories and ΠW-pretoposes is based, and this is what is behind the PER models or ω-set models of type theory. There are still some obscurities. For example, what remains unclear to me is to what extend an analysis of the initial ML-category can throw light on extensional type theory, but this is manifestly a fruitful approach.

When one turns to intensional type theory, matters become very opaque. If one performs the same construction as above starting from intensional type theory, the structure of the category will be much less nice, but it will be something like a weak ΠW-pretopos, a notion introduced in Chapter 3. To get a "decent" category from intensional type theory, one should perform the setoids construction explained in Chapter 2. In this way, one obtains a ΠW-pretopos. The importance of this result is of "ideological" importance in the sense explained at the beginning of this Appendix, but it does not make clear how ΠW-pretoposes help to understand intensional type theory.

I will end by formulating the rules for the remaining type constructors in Martin-Löf type theory.

0-type Intuitive description: the empty set.

Formation rule	$\dfrac{}{0\,\mathrm{Type}}$
Introduction rule	None.
Elimination rule	$\dfrac{a \in 0 \quad x \in 0 \vdash C(x)\,\mathrm{Type}}{\mathrm{case}(a) \in C(a)}$
Equality rule	None.

1-type Intuitive description: the one-point set.

Formation rule	$\dfrac{}{1\,\mathrm{Type}}$
Introduction rule	$\dfrac{}{* \in 1}$
Elimination rule	$\dfrac{a \in 1 \quad x \in 1 \vdash C(x)\,\mathrm{Type} \quad b \in C(*)}{\mathrm{case}(a, b) \in C(a)}$
Equality rule	$\dfrac{x \in 1 \vdash C(x)\,\mathrm{Type} \quad b \in C(*)}{\mathrm{case}(*, b) = b \in C(*)}$

+-type Intuitive description: disjoint union of two sets.

Formation rule
$$\frac{A\,\mathrm{Type} \quad B\,\mathrm{Type}}{A + B\,\mathrm{Type}}$$

Introduction rules
$$\frac{a \in A \quad B\,\mathrm{Type}}{\mathsf{inl}(a) \in A + B} \qquad \frac{A\,\mathrm{Type} \quad b \in B}{\mathsf{inr}(b) \in A + B}$$

Elimination rule
$$\frac{\begin{array}{c} c \in A + B \\ v \in A + B \vdash C(v)\,\mathrm{Type} \\ x \in A \vdash d(x) \in C(\mathsf{inl}(x)) \\ y \in B \vdash e(y) \in C(\mathsf{inr}(y)) \end{array}}{\mathsf{when}(c, d, e) \in C(c)}$$

Equality rules
$$\frac{\begin{array}{c} a \in A \\ v \in A + B \vdash C(v)\,\mathrm{Type} \\ x \in A \vdash d(x) \in C(\mathsf{inl}(x)) \\ y \in B \vdash e(y) \in C(\mathsf{inr}(y)) \end{array}}{\mathsf{when}(\mathsf{inl}(a), d, e) = d(a) \in C(\mathsf{inl}(a))}$$

$$\frac{\begin{array}{c} b \in B \\ v \in A + B \vdash C(v)\,\mathrm{Type} \\ x \in A \vdash d(x) \in C(\mathsf{inl}(x)) \\ y \in B \vdash e(y) \in C(\mathsf{inr}(y)) \end{array}}{\mathsf{when}(\mathsf{inr}(b), d, e) = e(b) \in C(\mathsf{inr}(b))}$$

Σ-type Intuitive description: disjoint union of a family of sets.

Formation rule
$$\frac{A\,\mathrm{Type} \quad x \in A \vdash B(x)\,\mathrm{Type}}{\Sigma x \in A.\, B(x)\,\mathrm{Type}}$$

Introduction rule
$$\frac{a \in A \quad x \in A \vdash B(x)\,\mathrm{Type} \quad b \in B(a)}{\langle a, b\rangle \in \Sigma x \in A.\, B(x)}$$

Elimination rule
$$\frac{\begin{array}{c} c \in \Sigma x \in A.\, B(x) \\ v \in \Sigma x \in A.\, B(x) \vdash C(v)\,\mathrm{Type} \\ x \in A, y \in B(x) \vdash d(x, y) \in C(\langle x, y\rangle) \end{array}}{\mathsf{split}(c, d) \in C(c)}$$

Equality rule
$$\frac{\begin{array}{c} a \in A \\ b \in B \\ v \in \Sigma x \in A.\, B(x) \vdash C(v)\,\mathrm{Type} \\ x \in A, y \in B(x) \vdash d(x, y) \in C(\langle x, y\rangle) \end{array}}{\mathsf{split}(\langle a, b\rangle, d) = d(a, b) \in C(\langle a, b\rangle)}$$

W-type Intuitive description: set of well-founded trees with fixed branching type (see Chapter 2).

Formation rule

$$\frac{A\,\mathrm{Type} \quad x \in A \vdash B(x)\,\mathrm{Type}}{\mathrm{W}x \in A.\, B(x)\,\mathrm{Type}}$$

Introduction rule

$$\frac{a \in A \quad t \in B(a) \to \mathrm{W}x \in A.\, B(x)}{\sup(a, t) \in \mathrm{W}x \in A.\, B(x)}$$

Elimination rule

$$\frac{\begin{array}{c} a \in \mathrm{W}x \in A.\, B(x) \\ v \in \mathrm{W}x \in A.\, B(x) \vdash C(v)\,\mathrm{Type} \\ y \in A, z \in B(y) \to \mathrm{W}x \in A.\, B(x), \\ u \in \Pi x \in B(y).\, C(z(x)) \vdash b(y, z, u) \in C(\sup(y, z)) \end{array}}{\mathrm{wrec}(a, b) \in C(a)}$$

Equality rule

$$\frac{\begin{array}{c} d \in A \\ t \in B(d) \vdash e(t) \in \mathrm{W}x \in A.\, B(x) \\ v \in \mathrm{W}x \in A.\, B(x) \vdash C(v)\,\mathrm{Type} \\ y \in A, z \in B(y) \to \mathrm{W}x \in A.\, B(x), \\ u \in \Pi x \in B(y).\, C(z(x)) \vdash b(y, z, u) \in C(\sup(y, z)) \end{array}}{\mathrm{wrec}(\sup(d, e), b) = b(d, e, \lambda((t)\,\mathrm{wrec}(e(t), b))) \in C(\sup(d, e)}$$