Type theory and its meaning explanations

Jonathan Sterling

ABSTRACT. At the heart of intuitionistic type theory lies an intuitive semantics called the "meaning explanations"; crucially, when meaning explanations are taken as definitive for type theory, the core notion is no longer "proof" but "verification". We'll explore how type theories of this sort arise naturally as enrichments of logical theories with further judgements, and contrast this with modern proof-theoretic type theories which interpret the judgements and proofs of logics, not their propositions and verifications.

Contents

CHAPTER 1

Logical Theories

To start, we will consider the notion of a *logical theory*; in my mind, it begins with a species (or set) of judgements that can be proposed, asserted, and (if they are evident) known.

1. Judgements of a logical theory

The basic forms of judgement for a logical theory will be $\boxed{P\ prop}$ and $\boxed{P\ true}$; and what is P? It is a member of the species of terms, which are made meaningful in the course of making the judgement $P\ prop$ evident for a proposition P.

To each judgement is assigned a *meaning explanation*, which explicates the knowledge-theoretic content of the judgement. For a judgement $\mathcal{J}$, a meaning explanation should be in the form:

> To know $\mathcal{J}$ is to know...

The meaning of the judgement $P\ prop$ is, then, as follows:

> To know $P\ prop$ is to know that P is a proposition, which is to know what counts as a direct verification of P.

So if a symbol P is taken to denote a proposition, we must know *what sort of thing* is to be taken as a direct verification of P, and this is understood as part of the definition of P. A "direct verification" is understood in constrast with an "indirect verification", which is to be thought of as a means or plan for verifying the proposition. Now, the judgement $P\ true$ is only meaningful in case we know $P\ prop$ (this is called a presupposition). Then the meaning of $P\ true$ is as follows:

> To know $P\ true$ is to have a verification of P.

From the (implicit) presupposition $P\ prop$, we already know what counts as a verification, so the meaning explanation is well-defined. Note that it is the same to have a means or plan for verifying P as to have a (direct) verification; this follows from the fact that one may put into action a plan for verifying P and achieve such a verification, and likewise, it is possible to propound a plan of verification by appeal to an existing verification.

2. Higher-order judgements

The judgements we have described so far are "categorical" in the sense that they are made without assumption. We will need to define a further form of judgement, which is called "hypothetical", and this is the judgement under hypothesis $\boxed{\mathcal{J}\ (\mathcal{J}_1, \ldots, \mathcal{J}_n)}$.Its meaning explanation is as follows:

> To know the judgement $\mathcal{J}\ (\mathcal{J}_1, \ldots, \mathcal{J}_n)$ is to know the categorical
> judgement $\mathcal{J}$ assuming you know the judgements $\mathcal{J}_1, \ldots, \mathcal{J}_n$.

2.1. General judgement. Another kind of higher order judgement is "general judgement", which is judgement with respect to a collection of free variables, $\boxed{|_{\vec{x}}\,\mathcal{J}}$. The meaning explanation for this new judgement is as follows:

> To know the judgement $|_{\vec{x}}\,\mathcal{J}$ is, for any arbitrary collection of
> expressions $\vec{E}$ of the same length of $\vec{x}$, to know $[\vec{E}/\vec{x}]\mathcal{J}$ (i.e.
> the substitution of each expression in $\vec{E}$ for each variable in $\vec{x}$
> respectively).

As far as notation is concerned, the general judgement bar binds the least tightly of all the other notations we have considered.

2.2. Hypothetico-general judgement. When hypothetical judgment is used inside general judgement (as in $|_{x}\,A(x)\ true\ (B(x)\ true)$, we term the whole thing a "hypothetico-general" judgement. One thing bears clarifying, which is, why do we write $P\ true\ (P\ true)$ rather than $|_{P}\,P\ true\ (P\ true)$?

The former is formally not a single judgement, but rather a *scheme* for judgements, where P and Q are intended to be replaced with concrete expressions by the person making use of the inference. On the other hand, the latter is itself a single judgement which may be asserted all on its own.

3. Propositions and verifications

Now that we have propounded and explained the minimal system of judgements for a logical theory, let us populate it with propositions. First, we have falsity $\bot$, and we wish to make $\bot\ prop$ evident; to do this, we simply state what counts as a direct verification of $\bot$, which is that there is no direct verification of $\bot$.

The next basic proposition is trivial truth $\top$, and to make $\top\ prop$ evident, we state that a direct verification of $\top$ is trivial. The definition of $\top$ thus validates the judgement $\top\ true$.

Next, let us define conjunction; in doing so, we will make evident the hypothetical judgement $P \wedge Q\ prop\ (P\ prop, Q\ prop)$; equivalently, we can display this as a rule of inference:

$$\frac{P\ prop \quad Q\ prop}{P \wedge Q\ prop}$$

A direct verification of $P \wedge Q$ consists in a verification of P and a verification of Q; this validates the assertion of the judgement $P \wedge Q$ *true* (P *true*, Q *true*). Because it is a valid inference, we can write it as an inference rule:

$$\frac{P \; true \quad Q \; true}{P \wedge Q \; true}$$

A direct verification of $P \vee Q$ may be got either from a verification of P or one of Q. From this definition we know $P \vee Q$ *prop* (P *prop*, Q *prop*), or

$$\frac{P \; prop \quad Q \; prop}{P \vee Q \; prop}$$

The verification conditions of disjunction give rise to two evident judgements $P \vee Q$ *true* (P *true*) and $P \vee Q$ *true* (Q *true*), which we can write as inference rules:

$$\frac{P \; true}{P \vee Q \; true} \qquad \frac{Q \; true}{P \vee Q \; true}$$

Finally, we must define the circumstances under which $P \supset Q$ is a proposition (i.e. when $P \supset Q$ *prop* is evident). And we intend this to be under the circumstances that P is a proposition, and also that Q is a proposition assuming that P is true. In other words, $P \supset Q$ *prop* (P *prop*, Q *prop* (P *true*)), or

$$\frac{P \; prop \quad Q \; prop \; (P \; true)}{P \supset Q \; prop}$$

Now, to validate this judgement will be a bit more complicated than the previous ones. But by unfolding the meaning explanations for hypothetical judgement, proposition-hood and truth of a proposition, we arrive at the following explanation:

> To know $P \supset Q$ *prop* (P *prop*, Q *prop* (P *true*)) is to know what counts as a direct verification of $P \supset Q$ when one knows what counts as a direct verification of P, and, when one has such a verification, what counts as a direct verification of Q.

(Note that unless P *true*, it need not be evident that Q *prop*.) If this judgement is going to be made evident, then we must indeed come up with what should count as a direct verification of $P \supset Q$ under the assumptions described above.

And so to have a direct verification of $P \supset Q$ is to have a verification of Q assuming that one has one of P; this is the meaning of implication, and it validates the judgement $P \supset Q$ *true* (Q *true* (P *true*)) , and may be written as an inference rule as follows:

$$\frac{Q \; true \; (P \; true)}{P \supset Q \; true}$$

4. Judgements for verifications

So far, we have given judgements which define what it means to be a proposition, and thence for each proposition, we have by definition a notion of what should count as a verification of that proposition. And by definition, to assert the judgement P *true* is to assert that one has a verification of P, but we have not considered any judgements which actually deal with such verifications directly.

It is a hallmark of Martin-Löf's program to resolve the contradiction between syntax and semantics not by choosing symbols over meanings or meanings over symbols, but by endowing symbols with meaning in the course of knowing the evident judgements. As such, P is a symbol, but when we assert P *prop* we are saying that we know what proposition P denotes.

A similar thing can be done with verifications themselves, by placing them in the syntactic domain together with the propositional symbols. And then, we can consider a judgement such as "M is a verification of P", and in knowing that judgement, we know what verification M is meant to denote. In practice, this judgement has been written in several ways:

$M \in P$	M is an element of P
$M \Vdash P$	M realizes P
$P \llcorner \mathsf{ext}\, M \lrcorner$	P is witnessed by M

But they all mean the same thing, and so we will tentatively give the following meaning explanation to this new judgement:

> * To know $M \in P$ is to know that M is a verification of P.

But now that we have started to assign expressions to verifications, we must be more careful about differentiating *direct verifications* (which we will call "canonical") from *indirect verifications* (which we will call "non-canonical"). So the domain of expressions must itself be accorded with a notion of reduction to canonical form, and this corresponds with putting into action a plan of verification in order to get an actual verification.

> To know $M \Rightarrow M'$ is to know that M is an expression which reduces to a canonical form M'.

Now, we can correct the previous meaning explanation as follows:

> To know $M \in P$ is to know an M' such that $M \Rightarrow M'$ and M' is a canonical (direct) verification of P.

The meaning explanation for P *prop* must be accordingly modified to take into account the computational behavior of the expression domain:

> To know P *prop* is to know a P' such that $P \Rightarrow P'$ and P' is
> a canonical proposition, which is to say, that one knows what
> counts as a canonical verification for P'.

In practice, when it is clear that P is canonical, then we will simply say, "To know P *prop* is to know what counts as a canonical verification of P".

As an example, then, we will update the evidence of the assertion $P \supset Q$ *prop* (P *prop*, Q *prop* (P *prop*)). The meaning of this, expanded into spoken language, is as follows:

> To know $P \supset Q$ *prop* (P *prop*, Q *prop* (P *prop*)) is to know
> what counts as a canonical verification of $P \supset Q$ under the
> circumstances that $P \Rightarrow P'$, such one knows what counts as a
> canonical verification P', and, if one has such a verification, $Q \Rightarrow$
> Q' such that one knows what counts as a canonical verification
> of Q'.

And this is evident, since we will say that a canonical verification of $P \supset Q$ is an expression $\lambda x.E$ such that we know the hypothetico-general judgement $|_x E \in Q \ (x \in P)$. This validates the assertion $\lambda x.E \in P \supset Q \ (E \in Q \ (x \in P))$, written as an inference rule:

$$\frac{|_x E \in Q \ (x \in P)}{\lambda x.E \in P \supset Q}$$

By the addition of this judgement, we have graduated from a logical theory to a type theory, in the sense of *Constructive Mathematics and Computer Programming* (Martin-Löf, 1979). In fact, we may dispense with the original P *true* form of judgement by *defining* it in terms of the new $M \in P$ judgement as follows:

$$\frac{M \in P}{P \ true}$$

Further forms of judgement, such as assertions of equality between propositions and verifications, may be added, as Martin-Löf does. However, this is not strictly necessary as they too may be defined in terms of the $M \in P$ judgement in the presence of an *equality* proposition; this is what is done in Constable et al's Computational Type Theory, which has only one primitive form of judgement.

Computational Type Theories

As alluded to at the end of the previous chapter, we may add a judgement $M \in A$ which deals directly with the objects M which verify the propositions (or types) P. We will develop a full theory of dependent types in the sense of Martin-Löf 1979.

1. The judgments of a computational type theory

Because we will need to consider the introduction of types which do not have a trivial (intensional) equality relation, we must first amend the meaning explanations for some of our judgements, and add a few new forms of judgement. First, we will refer to *types* rather than *propositions* in order to emphasize the generality of the theory; in some places, the word *set* is used instead.

The meaning of hypothetical and general judgement are the same as in the previous chapter, and so we will not reproduce them here.

The first form of judgement is $\boxed{A\ type}$, and its meaning explanation is as follows:

> To know A *type* is to know what counts as a canonical verification
> of A and when two such verifications are equal.

The next form of judgement is $\boxed{M \in A}$, which remains the same as before:

> To know $M \in A$ (presupposing $A\ type$) is to know an M' such
> that $M \Rightarrow M'$ and M' is a canonical verification of A.

We'll need to add new judgements for equality (equality of types, and equality of verifications). For equality of types $\boxed{A = B\ type}$, there are a number of possible meaning explanations, but we'll use the one that Martin-Löf used starting in 1979[1]:

> To know $A = B\ type$ (presupposing $A\ type$ and $B\ type$) is to
> know $|_M M \in B$ ($M \in A$) and $|_M M \in A$ ($M \in B$), and more-
> over $|_{M,N} M = N \in B$ ($M = N \in A$) and $|_{M,N} M = N \in$
> A ($M = N \in A$).

In other words, two types are equal when they have the same canonical verifications, and moreover, the same equality relation over their canonical verifications.

---

[1]This meaning explanation was abandoned in Constable's Computational Type Theory for practical reasons.

Now, because we are allowing the definition of types with arbitrary equivalence relations, we cannot use plain hypothetico-general judgement in the course of defining our types. For instance, if we were going to try and define the function type $A \supset B$ in the same way as we did in the previous chapter, we would permit "functions" which are not in fact functional, i.e. they do not take equal inputs to equal outputs. As such, we will need to bake functionality (also called extensionality) into the definition of functions, and since we will need this in many other places, we elect to simplify our definitions by baking it into a single judgement which is meant to be used instead of hypothetico-general judgement.

The judgement which expresses simultaneously generality, hypothesis and functionality has been written in multiple ways. Martin-Löf has always written it as $\mathcal{J} \ (\Gamma)$, but this is a confusing notation because it appears as though it is merely a hypothetical judgement (but it is much more, as will be seen). Very frequently, it is written with a turnstile, $\Gamma \vdash \mathcal{J}$, and in the early literature surrounding Constable's Computational Type Theory and Nuprl, it was written $\boxed{\Gamma \gg \mathcal{J}}$; we choose this last option to avoid confusion with a similar judgement form which appears in proof-theoretic, intensional type theories; we'll call the judgement form a "sequent".

This judgement must be defined simultaneously with two other judgements, $\boxed{\Gamma \ ctx}$ ("Γ is a context") and $\boxed{x \ \# \ \Gamma}$ ("x is fresh in Γ).

> To know $\Gamma \ ctx$ is to know that $\Gamma \Rightarrow \cdot$, or it is to know a variable x and an expression A such that $\Gamma \Rightarrow \Gamma', x \in A$ and $\Gamma' \gg A \ type$, and $x \ \# \ \Gamma$.

> To know $x \ \# \ \Gamma$ is to know that $\Gamma \Rightarrow \cdot$, or it is to know a variable y (which is not x) and an expression A such that $\Gamma \Rightarrow \Gamma', x \in A$ and $x \ \# \ \Gamma'$.

In other words contexts are inductively generated by the following grammar:

$$\frac{}{\cdot \ ctx} \qquad \frac{\Gamma \ ctx \quad \Gamma \gg A \ type \quad x \ \# \ \Gamma}{\Gamma, x \in A \ ctx}$$

$$\frac{}{x \ \# \ \cdot} \qquad \frac{x \ \# \ \Gamma}{x \ \# \ \Gamma, y \in A}$$

We will say that $\Gamma \gg \mathcal{J}$ is only a judgement under the presuppositions that $\Gamma \ ctx$ and that $\mathcal{J}$ is a categorical judgement of the form $A \ type$ or $M \in A$. Its meaning explanation must be given separately for each kind of conclusion.

Preliminarily, for any context Γ, let $\vec{\Gamma}_0$ represent the sequence of its variables and let $\vec{\Gamma}_1$ represent the sequence of their types. Then, when we use a sequence in place of an expression in a judgement, it is meant as a shorthand for an iterated judgement over the entire sequence.

Now we may begin giving the meaning explanations for $\Gamma \gg \mathcal{J}$, starting with typehood with respect to a context:

> To know $\Gamma \gg A$ *type* is to know $|_{\vec{M}}\, [\vec{M}/\vec{\Gamma}_0]A$ *type* $(\vec{M} \in \vec{\Gamma}_1)$, and moreover, to know
>
> $$|_{\vec{M},\vec{N}}\, [\vec{M}/\vec{\Gamma}_0]A = [\vec{N}/\vec{\Gamma}_0]A \ type \ (\vec{M} = \vec{N} \in \vec{\Gamma}_1)$$

This is quite noisy, so let us expand it:

> To know $\Gamma \gg A$ *type* is to know that for any arbitrary sequence of values $\vec{M}$ such that you know $\vec{M} \in \vec{\Gamma}_1$, then you know that the substitution of $\vec{M}$ for the variables $\vec{\Gamma}_0$ in A is a type (i.e. it is a family of types defined over the types $\vec{\Gamma}_1$); moreover, that this family of types is functional with respect to Γ.

Then we can explain type equality with respect to a context:

> To know $\Gamma \gg A = B$ *type* is to know
>
> $$|_{\vec{M}}\, [\vec{M}/\vec{\Gamma}_0]A = [\vec{M}/\vec{\Gamma}_0]B \ type \ (\vec{M} \in \vec{\Gamma}_1)$$
>
> and moreover, to know
>
> $$|_{\vec{M},\vec{N}}\, [\vec{M}/\vec{\Gamma}_0]A = [\vec{N}/\vec{\Gamma}_0]B \ type \ (\vec{M} = \vec{N} \in \vec{\Gamma}_1)$$

Next, membership with respect to a context is explained:

> To know $\Gamma \gg L \in A$ is to know
>
> $$|_{\vec{M}}\, [\vec{M}/\vec{\Gamma}_0]L \in [\vec{M}/\vec{\Gamma}_0]A \ (\vec{M} \in \vec{\Gamma}_1)$$
>
> and moreover, to know
>
> $$|_{\vec{M},\vec{N}}\, [\vec{M}/\vec{\Gamma}_0]L = [\vec{N}/\vec{\Gamma}_0]L \in [\vec{M}/\vec{\Gamma}_0]A \ (\vec{M} = \vec{N} \in \vec{\Gamma}_1)$$

Finally, member equality with respect to a context has an analogous explanation:

> To know $\Gamma \gg L = L' \in A$ is to know
>
> $$|_{\vec{M}}\, [\vec{M}/\vec{\Gamma}_0]L = [\vec{M}/\vec{\Gamma}_0]L' \in [\vec{M}/\vec{\Gamma}_0]A \ (\vec{M} \in \vec{\Gamma}_1)$$
>
> and moreover, to know
>
> $$|_{\vec{M},\vec{N}}\, [\vec{M}/\vec{\Gamma}_0]L = [\vec{N}/\vec{\Gamma}_0]L' \in [\vec{M}/\vec{\Gamma}_0]A \ (\vec{M} = \vec{N} \in \vec{\Gamma}_1)$$

The simultaneous definition of multiple judgements may seem at first concerning, but it can be shown to be non-circular by induction on the length of the context Γ.

2. The definitions of types

We will now define the types of a simple computational type theory without universes. In the course of doing so, opportunities will arise for further clarifying the position of the judgements, meaning explanations and proofs on the one hand, and the propositions, definitions and verifications on the other hand.

2.1. The unit type. First, we introduce two canonical forms with trivial reduction rules:

(Canonical)   unit $\Rightarrow$ unit $\bullet \Rightarrow \bullet$

Next, we intend to make the judgement unit *type* evident; and this is done by defining what counts as a canonical verification of unitand when two such verifications are equal. To this end, we say that $\bullet$ is a canonical verification of unit, and that it is equal to itself. I wish to emphasize that this is the entire definition of the type: we have introduced syntax, and we have defined the canonical forms, and there is nothing more to be done.

In traditional treatments of type theory, a type is "defined" by writing out a bunch of inference rules, but in type theory, the definitions that we have given above are prior to the rules, which are justified in respect of the definitions and the meaning explanations for the judgements. For instance, based on the meaning of the various forms of sequent judgement, the following rules schemes are justified:

$$\overline{\Gamma \gg \text{unit } type} \qquad \overline{\Gamma \gg \text{unit} = \text{unit } type}$$

$$\overline{\Gamma \gg \bullet \in \text{unit}} \qquad \overline{\Gamma \gg \bullet = \bullet \in \text{unit}}$$

Each of the assertions above has evidence of a certain kind; since the justification of these rules with respect to the definitions of the logical constants and the meaning explanations of the judgements is largely self-evident, we omit it. It is just important to remember that it is not the rules which define the types; a type A is defined in the course of causing the judgement A *type* to become evident according to its meaning explanation. These rules merely codify standard patterns of use, nothing more, and they must each be justified.

2.2. The empty type. The empty type is similarly easy to define. First, we introduce a constant:

(Canonical)   void $\Rightarrow$ void

To make the judgement void *type* evident, we will say that there are no canonical verifications of void, and be done with it. This definition validates some further rules

schemes:

$$\overline{\Gamma \gg \mathsf{void}\ type} \qquad \overline{\Gamma \gg \mathsf{void} = \mathsf{void}\ type}$$

$$\frac{\Gamma \gg M \in \mathsf{void}}{\mathcal{J}}$$

The last rule simply says that if we have a verification of void, then we may conclude any judgement whatsoever. Remember that the inference rules are just notation for an *evident* hypothetical judgement, e.g. $\mathcal{J}$ ($\Gamma \gg M \in \mathsf{void}$).

Note that we did not introduce any special constant into the syntax/computation language to represent the elimination of a verification of void (in intensional type theories, this non-canonical form is usually called $\mathsf{abort}(R)$. This is because, computationally speaking, there is never any chance that we should ever have use for such a term, since we need only consider the evaluation of closed terms (which is guaranteed by the meaning explanations).

Proof Theories and Logical Frameworks

1. Proof-theoretic type theory

1.1. Analytic and synthetic Judgement. A synthetic judgement is one for which the experience of *coming to know it* necessarily results in the synthesis of some new information; on the other hand, to know an *analytic* judgement is to know it purely on the basis of the information contained inside it. So analytic judgements are decidable, since if they may become evident, it will be purely on the basis of their own content; whereas synthetic judgements become evident when one has acquired some particular evidence for them.

A logical theory has, then, both analytic and synthetic judgements; the judgement P *prop* is analytic, since its evidence follows from the definition of P, whereas the assertion of P *true* entails the knowledge of some extra information, namely a verification of P. When we have extended the logical theory to a type theory in the manner of the previous sections, the judgement $M \in P$ is also synthetic, since $M \in P$ is not self-evident in general.

But why is it not enough to assert that M verifies P to know $M \in P$? It suffices to define a P such that one cannot decide in general whether some term is a verification of it. Let us define the propositional symbol P, and we intend to know the judgement P *prop*, whose meaning is to be expanded as follows:

> To know P *prop* is to know counts as a canonical verification of P.

We will say, then, that $\bullet$ is a canonical verification of P just when Goldbach's conjecture is true. Then it comes immediately that the judgement $M \in \mathsf{P}$ may not be known or refuted on its own basis, nor even the judgement $\bullet \in \mathsf{P}$, since they depend on an proposition whose truth is not known:

> To know $M \in \mathsf{P}$ is to know that $M \Rightarrow M'$ to a canonical verification of P.
>
> $\Leftrightarrow$ To know $M \in \mathsf{P}$ is to know that $M \Rightarrow \bullet$ such that Goldbach's conjecture is true.

1.2. Proof of a judgement vs. verification of a proposition. Because the judgement $M \in P$ is synthetic, we cannot say that it gives rise to a proof

theory for the logic, since the core judgement of a proof theory $M : A$ must be analytic, in order to avoid the infinite regress of a proof theory requiring a proof theory requiring a proof theory, and so on.

The notion of verification of a proposition could never be the same as proof anyway, except in the most trivial circumstances, since a verification is meant to be an effective operation which realizes the truth of the proposition, and no constraints whatsoever (termination, totality, etc.) are placed on these operations except those which come from the meaning of the judgements (see Dummett).

So a proof theory is necessarily intensional, and its judgements are to be analytic/decidable. What is it, then, that we have considered so far which corresponds with a proof M such that $M : P$ in a proof theory? As discussed above, M is not merely a term such that $M \in P$, since this is not in general enough information to know whether M is a proof. In fact, M must comprise all the logical inferences which led to the knowledge that P is true, and so a meaning explanation for the judgement $M : P$ in a proof theory immediately suggests itself:

> To know $M : P$ is to know that M is evidence of the judgement
> P *true*.

And so the term domain of the proof theory is not the same as the one that we have considered so far; it must consist in terms which represent traces of the inferences made in the course of knowing the judgements of a logical theory. There is a sense in which one can consider the types of a proof theory to interpret the judgements of the logical theory, and this methodology is called "judgements as types" (and this implies "derivations as terms").

What I am calling a "proof-theoretic type theory" is a type theory of the sort used in the proof assistants Agda, Coq and Idris, whereas the kind of type theory that I have described in the previous sections, the one based on meaning explanations, underlies the proof assistant Nuprl.

The proof-theoretic type theories on the one hand are often called "intensional" and the meaning-theoretic type theories on the other hand are usually "extensional"; these characterizations are certainly true, though I fear that comparing one of the former with one of the latter is not quite fair, since there is not any clear analogy to be had. That is to say, the judgement $M \in P$ is a judgement which is added to a logical theory and its meaning is (briefly) "M evaluates to a canonical verification of P", whereas $M : P$ cannot be construed as a judgement added to a logical theory. Instead, it must be understood as part of a theory which is overlayed atop an existing logical theory; it is possible to understand the theory which contains the judgement $M : P$ to be a metatheory, or logical framework, for

the theory which contains the judgement P *true*, which can be construed as the "object language".

In short, the judgements $M \in P$ and $M : P$ are unrelated to each other in two respects: firstly, that they have different meanings, and secondly that the one is at the same level as the judgements of a logical theory, whereas the latter is a judgement in a theory which is defined over a logical theory.

2. Martin-Löf's Logical Framework

To make this more concrete, let us expound a proof theoretic type theory called **MLLF**, which stands for "Martin-Löf's Logical Framework"; in the course of introducing each type, we will specify which judgement of the underlying logical theory $\mathcal{L}$ it is meant to interpret. Since we are now beginning to refer to judgements in multiple theories, where there is ambiguity, we will use the notation "$\mathcal{T}$-judgement" to refer to a judgement in the theory $\mathcal{T}$, and $\mathcal{T} \vdash \mathcal{J}$ will be shorthand for "the $\mathcal{T}$-judgement $\mathcal{J}$ is evident".

We start with four categorical judgements:

Judgement	Pronunciation
$\alpha : type$	α is a type
$\alpha = \beta : type$	α and β are equal types
$M : \alpha$	M is of type α
$M = N : \alpha$	M and N are equal at type α

But we have not defined the meaning of the judgements; let us do so below:

> To know $\alpha : type$ is to know what counts as an object of type α,
> and when two such objects are equal.

For now, we'll leave the question of what is an "object" as abstract; in many cases, types will represent $\mathcal{L}$-judgements and objects will represent $\mathcal{L}$-derivations.

> To know $\alpha = \beta : type$ is to know that any object of type α is also an object of type β, and two equal objects of type α are equal as objects of type β (necessarily presupposing $\alpha : type$ and $\beta : type$).

> To know $M : \alpha$ is to know that M is an object of type α (necessarily presupposing $\alpha : type$).

> To know $M = N : \alpha$ is to know that M and N are equal objects of type α (necessarily presupposing $M : \alpha$ and $N : \alpha$).

Next, we will introduce hypothetical judgement in the logical framework; to avoid confusion, where $\mathcal{J}$ ($\mathcal{J}'$) is a hypothetical $\mathcal{L}$-judgement, we will use $\mathcal{J}$ $[\mathcal{J}']$ for hypothetical judgements in the logical framework. Then,

To know $\mathcal{J}\ [\mathcal{J}_1, \ldots, \mathcal{J}_n]$ is to know $\mathcal{J}$ when one knows $\mathcal{J}, \ldots, \mathcal{J}_n$.

At this point, we may begin adding types to the logical framework. In practice, most types which we will introduce in the logical framework will be defined in terms of a judgement of the logical theory $\mathcal{L}$ which lies below it. For instance, hypothetical judgement in $\mathcal{L}$ is represented by a function type in the logical framework, $(x : \alpha)\beta$, whose typehood is meant to be evident under the following circumstances

$$\frac{\alpha : type \quad \beta : type\ [x : \alpha]}{(x : \alpha)\beta : type}$$

Or as a hypothetical judgement, $(x : \alpha)\beta : type\ [\alpha : type, \beta : type\ [x : \alpha]]$.

Now, to know this judgement is to know that under the circumstances we know what is an object of type α and when two such objects are equal, and that if we have such an object x, we know what an object of type β is, and when two such objects are equal—then we know what an object of type $(x : \alpha)\beta$ is. To make this evident, then, we will say that under those circumstances an object of type $(x : \alpha)\beta$ is an object $[x]M$ such that one knows $M : \beta\ [x : \alpha]$ and $[y/x]M = [z/x]M\ [y = z : \alpha]$; furthermore, two such objects are equal just when they yield equal outputs for equal inputs. (The equality is extensional, but since it is with respect to the definitional equality, this does not break the decidability of the judgement)

Then, for each atomic $\mathcal{L}$-proposition P, we can easily define a type $\mathsf{Prf}(P)$, as follows. Under the circumstances that $\mathcal{L} \vdash P\ prop$, then $\mathbf{MLLF} \vdash \mathsf{Prf}(P) : type$, since we will define an object of type $\mathsf{Prf}(P)$ to be an $\mathcal{L}$-derivation of $P\ true$; beyond reflexivity, further definitional equalities can be added to reflect the harmony of introduction and elimination rules.

Now, the definitions we have given for the types above are "intuitively" correct, but they actually fail to satisfy the meaning explanation that we have given for $\alpha : type$, because they do not take into account neutral terms. In the following sections, we will investigate this problem in more detail and propose a solution.

2.1. What is an "object"? It is time to revisit what it means to be an "object" of a type in the proof-theoretic type theory; we must note how this will necessarily differ from what it meant to be a "verification" of a proposition in the previous sections. Namely, a verification of a proposition is either a *canonical verification* of that proposition (and what sort of thing this might be is known from the presupposition $P\ prop$), or it is a means of getting such a canonical verification (i.e. a term which evaluates to a canonical verification).

On the other hand, what we have called an "object" of type P is quite different, since in addition to the possibility that it is a canonical proof of the judgement $P\ true$, it may also be *neutral* (i.e. blocked by a variable); we will call this "normal" rather than "canonical". Why does this happen?

In order to keep the judgement $M : A$ analytic (decidable), its meaning expla-
nation can no longer be based on the idea of the computation of closed terms to
canonical form; instead, we will consider the computation of open terms (i.e. terms
with free variables) to *normal* form. The desire for $M : A$ to be analytic follows
from our intention that it characterize a *proof theory*: we must be able to recognize
a proof when we see one. But why are closed-term-based meaning explanations
incompatible with this goal? Consider briefly the following judgement:

$$M(n) \in P \ (n \in \mathbb{N})$$

To know this judgement is to know that $M(n)$ computes to a canonical ver-
ification of P whenever n is a natural number; when P's use of n is not trivial,
this amounts to testing an infinite domain (all of the natural numbers), probably
by means of mathematical induction. The judgement is then clearly synthetic:
to know it is, briefly, to have come up with an (inductive) argument that $M(N)$
computes to a canonical verification of P at each natural number n.

On the other hand, the judgement $M(n) : P \ [n : \mathbb{N}]$ must have a different
meaning, one which admits its evidence or refutation purely on syntactic/analytic
grounds. In essence, it is to know that $M(n)$ is a proof of P for any *arbitrary*
object/expression n such that $n : \mathbb{N}$ (i.e., the only thing we know about n is that it
is of type $\mathbb{N}$; we do not necessarily know that it is a numeral).

3. A critique of MLLF

The type theory which we constructed in the previous section is to be considered
a proof theory for a logic with the judgements P *prop*, P *true* and $\mathcal{J} \ (\mathcal{J}_1, \ldots, \mathcal{J}_n)$.
There are a few reasons to be dissatisfied with this state of affairs, which I shall
enumerate in this section.

3.1. Lack of computational content. Unlike the type theory in the first
chapter, there is no built-in computational content. In a computational type theory
which is defined by the verificationist meaning explanations, the computational
content of terms is understood immediately by means of the $M \Rightarrow M'$ relation; that
is, computation is prior to the main judgements because their meaning explanations
are defined in terms of evaluation to canonical form.

On the other hand, in the type theory above we did not give a primitive re-
duction relation; instead, we simply permitted the endowment of proofs with defi-
nitional equalities which reflect the harmony of introduction and elimination rules.
That is, if we have known the $\mathcal{L}$-judgement P *true* by means of an indirect argu-
ment (derivation), it must be the case that this derivation corresponds to a direct
one; we reflect this in the proof theory by defining the indirect derivation to be
definitionally equal to the direct one.

However, this does not amount to computational content being present in terms: only *post facto* may the definitional equality be construed as giving rise to computation, through a metamathematical argument which shows that the definitional equality is confluent and can be used to define a functional normalization relation.

And this is the reason for the peculiarity of the proof-theoretic meaning explanations, namely that they do not include phrases like "evaluates to a canonical...", since evaluation may only be understood after taking the meanings of the judgements ($\alpha : type$, $\alpha = \beta : type$, $M : \alpha$, $M = N : \alpha$) as giving rise to a closed formal system which is susceptible to metamathematical argument: to refer to evaluation in the meaning explanations for the core judgements, then, would be impredicative.

3.2. Modularity of definition. By the same token, the distinction between canonical (direct) and non-canonical (indirect) proof may not be understood as a core notion in the theory, but must be understood separately, secondarily. Why is this a problem? It means that the definition of each type must be made with the full knowledge of the definitions of every other type; in essence, the open-ended nature of type theory is obliterated and one is forced into a fixed formal system; this is in addition to the fact that it causes the epistemic content of $\alpha : type$ for any type α to be extremely complicated.

To illustrate, let us consider as an example a type theory which has four type-formers: trivial truth $\top$, trivial falsity $\bot$, implication $(\alpha)\beta$, and conjunction $\alpha\&\beta$; we will then introduce the following terms to represent proofs: the trivial element $\bullet$, *reductio ad absurdum* abort($\alpha; E$), abstraction $[x : \alpha]E$, application $E(E')$, pairing $\langle E, E' \rangle$, and projections fst(E), snd(E).

If we will try to make the judgement $\top : type$ evident, the deficiencies of the formulation will immediately present themselves.

> To know $\top : type$ is to know what counts as an object of type $\top$, and when two such objects are equal. An object of type $\top$, then, is either the expression $\bullet$, or an expression abort($\top; E$) such that we know $E : \bot$, or an expression $E(E')$ such that we know $E : (\alpha)\top$ and $E' : \alpha$, or an expression fst(E) such that we know $E : \top\&\beta$ for some β, or an expression snd(E) such that we know $E : \alpha\&\top$ for some α; and we additionally have that $\bullet$ is equal to $\bullet$, and ...

To save space, we elide the rest of the definition of equality for $\top$; what we have seen so far already suffices to bring to light a serious problem: the definition of any type requires knowledge of the entire syntax of the theory. The judgement $\alpha : type$ may never be made evident in isolation, but must be done with full understanding of all the other types and their definitions.

Furthermore, to extend an existing theory with a new type, the definitions of every other type must be rewritten to account for the elimination forms of the new type.

4. The logic of verifications and uses