Type theory and its meaning explanations

Jonathan Sterling

Thanks to Bob Harper, Peter Dybjer, Bengt Norström, Carlo Angiuli and Darin Morrison for invaluable conversations which helped sharpen my view of the meaning explanations for computational and intensional type theories.

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 $\boxed{P\ \mathit{prop}}$ and $\boxed{P\ \mathit{true}}$; 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* ("P is a proposition") evident for a proposition P.

The forms of judgement may be construed as containing *inputs* and *outputs*; an *input* is something which is inspected in the course of knowing a judgement, whereas an *output* is something which is synthesized (or created) in the course of knowing a judgement. The positions of *inputs* and *outputs* in constitute for a judgement form what is called its *mode*, and we color-code it in this presentation for clarity.[1]

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 would count 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; these distinctions will be explained in more detail later on. Now,

[1]We will not see any judgements with *outputs* at first, but it will become necessary as soon as we consider judgements about computation, where the reduction of a term is synthesized from the redex. Modes may be used to construe a judgement as having algorithmic content.

the judgement P *true* ("P is 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 having a means or plan for verifying P is equivalent to having 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 or generality.

2.1. Hypothetical judgement.

2.1. Hypothetical judgement. We will need to define a further form of judgement, which is called "hypothetical", and this is the judgement under hypothesis $\boxed{\mathcal{J}\ (\mathcal{J}')}$, pronounced "$\mathcal{J}$ under the assumption $\mathcal{J}'$".Its meaning explanation is as follows:

> To know the judgement $\mathcal{J}\ (\mathcal{J}')$ is to know the categorical judgement $\mathcal{J}$ assuming you know the judgement $\mathcal{J}'$.

Hypothetical judgement may be iterated, and $\mathcal{J}\ (\mathcal{J}_1, \mathcal{J}_2)$ will be used as notation for for $\mathcal{J}\ (\mathcal{J}_2)\ (\mathcal{J}_1)$.

2.2. General judgement. Another kind of higher order judgement is "general judgement", which is judgement with respect to a variable, $\boxed{|_x\, \mathcal{J}}$, pronounced "for an arbitrary x, $\mathcal{J}$". The meaning explanation for this new judgement is as follows:

> To know the judgement $|_x\, \mathcal{J}$ is, to know $[E/x]\mathcal{J}$ (i.e. the substitution of E for x in the expression $\mathcal{J}$) for any arbitrary expression E, [2]

As far as notation is concerned, the bar symbol binds the least tightly of all the other notations we have considered. Likewise, general judgement may be iterated, and the notation $|_{x,y}\, \mathcal{J}$ will be used as notation for $|_x\, |_y\, \mathcal{J}$.

[2]Technically, E is qualified as being of the same valence as x, but because we have not developed a formal theory of expressions in this presentation, I choose to ignore this issue.

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

The former is really not a single judgement, but rather a *scheme* for judgements, where P is intended to be replaced with a concrete expression by the person asserting the judgements. 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 $\bot\ \textit{prop}$ evident; to do this, we simply state what counts as a direct verification of $\bot$: there is no direct verification of $\bot$.

The next basic proposition is trivial truth $\top$, and to make $\top\ \textit{prop}$ evident, we state that a direct verification of $\top$ is trivial. The definition of $\top$ thus validates the judgement $\top\ \textit{true}$ (i.e. that we have a verification of $\top$; this is immediate).

Next, let us define conjunction; in doing so, we will make evident the hypothetical judgement $P \wedge Q\ \textit{prop}\ (P\ \textit{prop}, Q\ \textit{prop})$; equivalently, we can display this as a rule of inference:[3]

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

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 $P \wedge Q\ \textit{true}\ (P\ \textit{true}, Q\ \textit{true})$. Because it is a valid inference, we can write it as an inference rule:

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

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 $P \vee Q\ \textit{prop}\ (P\ \textit{prop}, Q\ \textit{prop})$, or

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

[3]Evident hypothetical judgements are often written as rules, i.e.

$$\frac{\textit{premise}}{\textit{conclusion}}$$

rather than *conclusion* (*premise*). It must be stressed that only *evident*/*known* judgements may be written in this way.

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:

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

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

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

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.[4]

If the judgement $P \supset Q$ *prop* (P *prop*, Q *prop* (P *true*)) is going to be made evident, then we must 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:

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

4. Judgements for verifications

So far, we have given judgements which define what it means to be a proposition, namely P *prop*, and thence for each proposition, we have by definition a notion of what should count as a verification of that proposition. And we have a judgement P *true*, which in its assertion means that one has (a way to obtain) such a verification of P, but we have not considered any judgements which actually refer to the verifications themselves symbolically.

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

[4]Note that unless P *true*, it need not be evident that Q *prop*; in other words, Q only has to be a proposition if P is true. It would also be acceptable to give a stronger definition to implication, but this is the one accepted by Martin-Löf.

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 representing them with symbols in the same way we have done for the propositions. 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:

Notation	Pronunciation
$M \in P$	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, so we will choose the notation $\boxed{M \in P}$ and pronounce it "M verifies P". Tentatively, the following defective meaning explanation could be given:

* 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 a direct (canonical) verification; reduction to canonical form will be represented by a judgement $\boxed{M \Rightarrow M'}$, pronounced "M evaluates to M'".

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

An example of an evident reduction judgement in elementary mathematics would be $3 + 4 \Rightarrow 7$; note that $3 + 4 \Rightarrow 1 + 6$ is, on the other hand, not evident, since this judgement describes reduction to *canonical* form, whereas $1 + 6$ is not a canonical number.

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.

If it is not yet clear why it would have been a mistake to fail to use the notion of reduction to canonical form in the above meaning explanation, consider that each time a proposition is defined, it should be possible to do so without knowing what other propositions exist in the theory. But if we consider non-canonical forms (as would be necessary if we omitted the $M \Rightarrow M'$ premise), then we would have to fix in advance all the possible non-canonical forms in the computation system in the course of defining each proposition. As such, the open-ended nature of the logic would be destroyed; in a later chapter, the seriousness of this problem will be made even more clear.

The meaning explanation for $P\ \mathit{prop}$ must be accordingly modified to take into account the computational behavior of expressions:

> To know $P\ \mathit{prop}$ 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 $P\ \mathit{prop}$ is to know what counts as a canonical verification of P". As an example, then, we will update the evidence of the following assertion:

$$P \supset Q\ \mathit{prop}\ (P\ \mathit{prop}, Q\ \mathit{prop}\ (P\ \mathit{prop}))$$

The meaning of this, expanded into spoken language, is as follows:

> To know $P \supset Q\ \mathit{prop}\ (P\ \mathit{prop}, Q\ \mathit{prop}\ (P\ \mathit{prop}))$ is to know what counts as a canonical (direct) verification of $P \supset Q$ under the circumstances that $P \Rightarrow P'$, such that 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 the above judgement 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 $\lambda x.E \in P \supset Q\ (|_x\ E \in Q\ (x \in P))$, or, 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 $P\ \mathit{true}$ form of judgement by *defining* it in terms of the new $M \in P$ judgement as follows:

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

CHAPTER 2

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.[1]

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 presentations, 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 $\boxed{A\ \mathit{type}}$, and its meaning explanation is as follows:

> To know A *type* is to know an A' such that $A \Rightarrow A'$, and you 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 verifications, and equality of types respectively). First, equality of verifications is written $\boxed{M = N \in A}$, and means the following:

> To know $M = N \in A$ (presupposing $M \in A$ and $N \in A$, and thence A *type*) is to know that the values of M and N are equal as canonical verifications of A.

[1]Please note that the judgements given here, and their meaning explanations, are *not* the same as those used in Constable et al's "Computational Type Theory" and Nuprl. In this chapter, we use the term "computational type theory" in a general sense to characterize a family of type theories which have their origin in Martin-Löf's 1979 paper *Constructive Mathematics and Computer Programming*.

The fact that M and N reduce to canonical values which verify A is known from the presuppositions of the judgement; and what it means for them to be equal as such is known from the evidence of the presupposition A *type* which is obtained from the other presuppositions.

For equality of types $\boxed{A = B \textit{ type}}$, there are a number of possible meaning explanations, but we'll use the one that Martin-Löf used starting in 1979:

> 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 moreover $|_{M,N}\, M = N \in B\ (M = N \in A)$ 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. Note that there are other possible explanations for type equality, including more intensional ones that appeal to the syntactic structure of type expressions, and these turn out to be more useful in proof assistants for practical reasons. However, the extensional equality that we have expounded is the easiest and most obvious one to formulate, so we will use it here.

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 plain 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".

First, we will add the syntax of contexts into the computation system:

(Canonical)
$$\cdot \Rightarrow \cdot$$
$$(\Gamma, x : A) \Rightarrow (\Gamma, x : A)$$

The sequent judgements will be defined simultaneously with two other judgements, $\boxed{\Gamma\ \mathit{ctx}}$ ("Γ is a context") and $\boxed{x \# \Gamma}$ ("x is fresh in Γ"). Their meaning explanations are as follows:

> To know $\Gamma\ \mathit{ctx}$ is to know that $\Gamma \Rightarrow \cdot$, or it is to know a variable x and expressions Δ, A such that $\Gamma \Rightarrow \Delta, x : A$ and $\Delta \gg A\ \mathit{type}$, and $x \# \Delta$.
>
> To know $x \# \Gamma$ is to know that $\Gamma \Rightarrow \cdot$, or, if $\Gamma \Rightarrow \Delta, y : A$ such that x is not y and $x \# \Delta$.

In other words, the well-formed contexts are inductively generated by the following grammar:

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

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

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

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

> To know $\Gamma \gg A\ \mathit{type}$ is to know, if $\Gamma \Rightarrow \cdot$, then $A\ \mathit{type}$; otherwise, if $\Gamma \Rightarrow \Delta, x : B$, that $|_M\, \Delta \gg [M/x]A\ \mathit{type}\ (\Delta \gg M : B)$ and moreover $|_{M,N}\, \Delta \gg [M/x]A = [N/x]A\ \ \mathit{type}\ (\Delta \gg M = N \in B)$.

In this way, we have defined the sequent judgement by structural recursion on the context. We can explain type equality sequents in a similar way:

> To know $\Gamma \gg A = B\ \mathit{type}$ is to know, if $\Gamma \Rightarrow \cdot$, that $A = B\ \mathit{type}$; otherwise, if $\Gamma \Rightarrow \Delta, x : C$, it is to know
>
> $$|_M\, \Delta \gg [M/x]A = [M/x]B\ \mathit{type}\ (\Delta \gg M \in C)$$
>
> and moreover, to know
>
> $$|_{M,N}\, \Delta \gg [M/x]A = [N/x]B\ \mathit{type}\ (\Delta \gg M = N \in C)$$

Next, the meaning of membership sequents is explained:

> To know $\Gamma \gg M \in A$ is to know, if $\Gamma \Rightarrow \cdot$, that $M \in A$; otherwise, if $\Gamma \Rightarrow \Delta, x : B$, it is to know
>
> $$|_N\, \Delta \gg [N/x]M \in [N/x]A\ (\Delta \gg N \in B)$$

and moreover, to know

$$|_{L,N}\ \Delta \gg [L/x]M = [N/x]M \in A\ (\Delta \gg L = N \in B)$$

Finally, member equality sequents have an analogous explanation:

To know $\Gamma \gg M = M' \in A$ is to know, if $\Gamma \Rightarrow \cdot$, that $M = M' \in A$; otherwise, if $\Gamma \Rightarrow \Delta, x : B$, it is to know

$$|_{N}\ \Delta \gg [N/x]M = [N/x]M' \in A\ (\Delta \gg N \in B)$$

and moreover, to know

$$|_{L,N}\ \Delta \gg [L/x]M = [N/x]M' \in A\ (\Delta \gg L = N \in B)$$

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) $\qquad \mathsf{unit} \Rightarrow \mathsf{unit} \qquad \bullet \Rightarrow \bullet$

Next, we intend to make the judgement unit *type* evident; and this is done by defining what counts as a canonical verification of unit and 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 the presentations of type theory which are currently in vogue, 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 of the judgements. For instance, based on the meaning of the various forms of sequent judgement, the following rule schemes are justified:

$$\frac{}{\Gamma \gg \mathsf{unit}\ \mathit{type}} \qquad \frac{}{\Gamma \gg \mathsf{unit} = \mathsf{unit}\ \mathit{type}}$$

$$\frac{}{\Gamma \gg \bullet \in \mathsf{unit}} \qquad \frac{}{\Gamma \gg \bullet = \bullet \in \mathsf{unit}}$$

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 in nearly all cases. 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. 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) $\mathsf{void} \Rightarrow \mathsf{void}$

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:

$$\frac{}{\Gamma \gg \mathsf{void}\ \mathit{type}} \qquad \frac{}{\Gamma \gg \mathsf{void} = \mathsf{void}\ \mathit{type}}$$

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

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. $\Gamma \gg \mathcal{J}\ (\Gamma \gg M \in \mathsf{void})$.

Note that we did not introduce any special constant into the computation system to represent the elimination of a verification of void (in proof-theoretic 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), and by its very definition, there can never be a closed verification of void.

2.3. The cartesian product of a family of types. This will be our first dependent type, and it will likewise be the first example of a non-trivial addition to the computation system. First, let us add our canonical and non-canonical forms and their reduction rules:

(Canonical) $\mathsf{\Pi}(A; x.B) \Rightarrow \mathsf{\Pi}(A; x.B) \qquad \lambda(x.E) \Rightarrow \lambda(x.E)$

(Non-canonical) $$\frac{[M/x]E \Rightarrow N}{\mathsf{ap}(\lambda(x.E); M) \Rightarrow N}$$

This is the first time in this chapter that we have introduced a term former with binding structure; it should be noted that the intensional equality of expressions is up to alpha equivalence, and we will not pay attention to issues of variable renaming in our presentation.

We will make evident the following judgement scheme:

$$\frac{A\ \mathit{type} \quad x : A \gg B\ \mathit{type}}{\mathsf{\Pi}(A; x.B)\ \mathit{type}}$$

Or, written as a hypothetical judgement:

$$\mathsf{\Pi}(A; x.B)\ \mathit{type}\ (A\ \mathit{type},\ x : A \gg B\ \mathit{type})$$

This is to say, under the stated assumptions, we know what counts as a canonical verification of $\mathsf{\Pi}(A; x.B)$ and when two such verifications are equal. We will say that $\lambda(x.E)$ is a canonical verification of $\mathsf{\Pi}(A; x.B)$ just when we know $x : A \gg E \in B$; moreover, that two verifications $\lambda(x.E)$ and $\lambda(y.E')$ are equal just when $z \in A \gg [z/x]E = [z/y]E' \in [z/x]B$.

By the meaning of the sequent judgement, this is to say that a lambda expression must be functional with respect to its domain (i.e. it must take equals to equals). We did not need to hypothesize directly two elements of the domain and their equality because this is part of the meaning explanation for the sequent judgement already. Likewise, two lambda expressions are equal when equal inputs yield equal results in both.

The familiar inference rules, which codify the standard mode of use for the family cartesian product, are justified by this definition:

$$\frac{\Gamma \gg A\ \mathit{type} \quad \Gamma, x : A \gg B\ \mathit{type}}{\Gamma \gg \mathsf{\Pi}(A; x.B)\ \mathit{type}} \qquad \frac{\Gamma \gg A = A'\ \mathit{type} \quad \Gamma, z : A \gg [z/x]B = [z/y]B'\ \mathit{type}}{\Gamma \gg \mathsf{\Pi}(A; x.B) = \mathsf{\Pi}(A'; y.B')\ \mathit{type}}$$

$$\frac{\Gamma, x : A \gg E \in B}{\Gamma \gg \lambda(x.E) \in \mathsf{\Pi}(A; x.B)} \qquad \frac{\Gamma, z : A \gg [z/x]E = [z/y]E' \in [z/x]B}{\Gamma \gg \lambda(x.E) = \lambda(y.E') \in \mathsf{\Pi}(A; x.B)}$$

$$\frac{\Gamma \gg M \in \mathsf{\Pi}(A; x.B) \quad \Gamma \gg N \in A}{\Gamma \gg \mathsf{ap}(M; N) \in [N/x]B} \qquad \frac{\Gamma \gg M = M' \in \mathsf{\Pi}(A; x.B) \quad \Gamma \gg N = N' \in A'}{\Gamma \gg \mathsf{ap}(M; N) = \mathsf{ap}(M'; N') \in [N/x]B}$$

Note that the type equality rule scheme that we gave above is structural; it is validated by the meaning explanations, but it is by no means the full totality of possible equalities between family cartesian product types, which is extensional in this theory.

It will be instructive to explicitly justify the application rule above with respect to the meaning explanations, since I have claimed that such rules are posterior to the definitions we expounded prior to giving these rules.

Proof. It suffices to consider the case that $\Gamma \Rightarrow \cdot$, because hypotheses may always be added to the context (this is called weakening). And so, by the meaning of the sequent judgement at the empty context, the rule amounts to the assertion

$$\mathsf{ap}(M; N) \in [N/x]B \ (M \in \Pi(A; x.B),\ N \in A)$$

By the meaning explanation for hypothetical judgement, and the definition of the family cartesian product type, we know that $M \Rightarrow \lambda(x.E)$ for some E such that we know $x : A \gg E \in B$; from the meaning of the sequent judgement, we can conclude $|_L\, [L/x]E \in [L/x]B\ (L \in A)$. On the other hand, from the computation rules, we know that if for some particular E', $[N/x]E \Rightarrow E'$, then we know $\mathsf{ap}(\lambda(x.E); N) \Rightarrow E'$; to demonstrate the evidence of the premise, we may instantiate L at N to know $[N/x]E \in [N/x]B$, whence by the meaning of membership, we know that there exists some canonical E' such that $[N/x] \Rightarrow E'$. $\square$