CHAPTER 1

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 $\boxed{A\ \mathit{type}}$, 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 $\boxed{A = B\ \mathit{type}}$, 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 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.

[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, $\boxed{\Gamma\ \mathit{ctx}}$ ("Γ is a context") and $\boxed{x \mathrel{\#} \Gamma}$ ("x is fresh in Γ).

> To know $\Gamma\ \mathit{ctx}$ 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 $\Gamma' \gg A\ \mathit{type}$, and $x \mathrel{\#} \Gamma$.
>
> To know $x \mathrel{\#} \Gamma$ 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 $x \mathrel{\#} \Gamma'$.

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

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

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

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.

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

Next, we intend to make the judgement unit *type* evident; and this is done by defining what counts as a canonical verification of $\mathsf{unit}$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 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:

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

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

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. 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) $$\mathsf{void} \Rightarrow \mathsf{void}$$

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:

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

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

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).