Computer Science
University of Manchester

Categorical Properties of Logical Frameworks

Yuxi Fu

Technical Report UMCS–93–6–3

Categorical Properties of Logical Frameworks

Yuxi Fu

Department of Computer Science
University of Manchester
Oxford Rd., Manchester, U.K.

Abstract

In this paper we give a new presentation of *ELF* which is well-suited for semantic analysis. We introduce the notions of internal codability, internal definability, internal typed calculi and frame languages. These notions are central to our perspective of logical frameworks. We will argue that a logical framework is a typed calculus which formalizes the relationship between internal typed languages and frame languages. In the second half of the paper, we demonstrate the advantage of our logical framework by showing some categorical properties of it and of encodings in it. By doing so we hope to indicate a sensible model theory of encodings.

1 Introduction

The notion of logical frameworks first appeared in Martin-Löf's work ([NPS90]). It was meant to be a simple language with which his set theory can be precisely defined. The idea is picked up and extended in [HHP87, HHP92]. According to the authors, logical frameworks are languages in which logics can be adequately defined. The good properties of the meta-languages then guarantee that all manipulations of the defined logics are correct and easy (or even decidable). The language in [HHP87, HHP92], now known as *ELF*, consists of kinds, constructors and objects. Types are constructors of kind *Type*. In addition to the usual type theoretical features, it has *signatures* which declare constant terms. In other words, signatures are constant contexts except that one can define constant constructors in a signature. When coding up an object language, one first defines an appropriate signature so that logical operators become constant objects or constructors. An important feature about *ELF* is that *rules* in object languages are presented as objects in the framework. Derivations then amount to function applications. To guarantee the correctness of the encoding, an adequacy theorem must be proved each time a logic is defined in the framework. The adequacy theorem basically says that what has been formulated in *ELF* is sound and faithful with respect to the object language of interest.

A prominent question a designer of a logical framework must address is this: How are the variables in an object language represented in the framework? *ELF*'s answer is to *identify* the variables in an object language with those of the framework itself. Leaving the philosophical consequence of this attitude aside, such a convention works most of the time. Here are some other arguments for the convention: (i) The only binding operation in a logical framework is required to code various bindings in many different logics; (ii) There doesn't seem to be any other way. One important observation in this paper is that this variable-identification has serious impact on model theory.

A selling point of *ELF*, and logical frameworks in general, is that non-constructive logics can also be treated. The point is that whatever the object logic is, be it classical or intuitionistic, any assertion that holds in the logic must be constructively verifiable. In the light of type theory, an assertion forms a range of significance. So it can be represented as a type and its inhabitants are its verifiers. This is the so-called "judgement-as-type" principle.

The core of Martin-Löf's logical framework and that of *ELF* are the same. But their emphasis is different. Martin-Löf sees his logical framework as a foundational language upon which his constructive set theory is built. For that to be possible, he needs to talk about definitional equality for the object languages. His solution is to use equational theories of his framework. So to be precise, Martin-Löf's monomorphic set theories are equational theories in the underlying calculus. On the other hand, the motivation in [HHP87, HHP92] is to design a meta-language once and for all, so that logics can be represented and proof-checking be mechanized. To keep the calculus decidable, one has to settle for the internal representation of the definitional equality in object languages. Now $=$, like $\in$, plays a special role in proof theory. But *ELF* treats $x = y$ as a judgement just like other judgements. This is a price one has to pay to retain decidability.

The belief that the legitimacy of a proof should be effectively recognizable leads to the principle that languages supporting proof-checking should be decidable. Apart from *ELF*, this property holds of several other systems. The historically first among them is de Bruijn's *Automath* family ([dB80]) whose main goal is to carry out some mathematical arguments on computers. The intensional set theory of Martin-Löf ([NPS90]) is also decidable. Its grand purpose is to provide a new foundation for constructive mathematics. The construction-like calculi ([CH88, Luo90, Luo91]) are the blend of F_ω and constructive set theory. They are perhaps the strongest logical systems we can ever have. Other type theories have too sophisticated type constructors to be decidable. One example is the *NuPRL* system ([ea86]). The logical theory of constructions described in [MA88] has altogether a different flavour. Most of the languages we just mentioned do not purport to be logical frameworks.

Analogous to the compile/run-time dichotomy in functional programming languages viewed as indexed categories ([Mog89, HMM90, Mog91]), the relationship between the meta-language and object languages is similar. Our view of what constitute a model for an encoding in a logical framework will make the comparison clearer. On one hand, the meta-theory of the framework should be well understood. On the other hand, the need for good properties of an object language depends on a specific application we have in mind. At another level, the formulated object calculi are compile-time objects; their well-formedness must be checked by machine before they are used and it is done once for all. The run-time objects are the terms constructed in terms of the formulated language.

Another advantage in insisting on such a dichotomy is shown in [Luo92]. There the author takes the view that, as far as programmes are concerned, the natural notion for computational equality is the intensional one. But when reasoning about programs, say in a correctness proof, we would very much like the extensional equality to hold *logically*. This can be achieved only if the logical framework is extensional (externally). Extensionality is essential for concise representations of object languages; it also simplifies proofs of adequacy theorem. In this paper we insist that all logical frameworks should be extensional.

The proof theoretical and pragmatical aspects of logical frameworks have been extensively studied ([HP91, HPJ91]). The model theory of them however has not yet been paid enough attention it deserves. This paper sets out to remedy this. But first, we need to make clear what *we* mean by a logical framework in this work. *A logical framework is a typed calculus upon (or in) which typed calculi are defined. Putting differently, the theory of logical frameworks is about internal type theory.* This unifies the views taken in [HHP87, HHP92] and [NPS90] and, arguably, also covers other more complex languages. The various well-known coding techniques ([BB85, Wra89]) belong to internal type theory. As a matter of fact, one uses one or more of them whenever one designs a language in a logical framework. This definition is semantically motivated. What internal categories to an ambient category, object languages are to a logical framework.

In this paper we introduce the notions of internal definability, internal codability, internal typed calculi and frame languages. An example is given to show the potential usefulness of this definition. We propose an alternative presentation of Edinburgh *LF* based on the idea of internal typed calculi and frame languages. The system is analyzed in terms of internal codability and definability. We show that the new presentation makes it convenient for an algebraic investigation. We use three examples to illustrate some new observations about logical frameworks. Category theory is used to relate the model theory of the logical framework to those of the represented languages.

Section 2 fixes some notation that will be employed later. Section 3 gives the definitions central to later development. Section 4 is an undemanding introduction to our logical framework λ_{TT}. Some sample encodings are given in section 5. Section 6 details the fact that λ_{TT} has built-in mechanism to generate internal full subcategories, whose relationship with the variable convention is shown next. Section 8 and 9 reveals some further categorical properties of λ_{TT}. Section 10 outlines the idea of model theory for encodings.

2 Preliminaries

The technical definitions given in this section are necessary to describe the categorical properties of λ_{TT}. The material is standard; see [B̀85, Pav90, Jac91] for more on the theory of fibrations and [HRR90, Pit87] for relevant internal category theory.

Suppose $p : \mathcal{F} \longrightarrow \mathcal{B}$ is a functor. $Y \xrightarrow{f} X$ in $\mathcal{F}$ is said to be a *cartesian lifting* of $J \xrightarrow{u} I$ in $\mathcal{B}$ if (i) $pf = u$ and (ii) for any $Z \xrightarrow{g} X$ in $\mathcal{F}$ and $pZ \xrightarrow{m} J$ satisfying $pg = m; u$, there is a unique morphism $Z \xrightarrow{\phi} Y$ with $p\phi = m$ and $g = \phi; f$. p is a *fibration* if for every morphism $J \xrightarrow{u} I$ in the *base* category $\mathcal{B}$ and every object X in the *fibre* category $\mathcal{F}$ with $pX = I$, there is a cartesian lifting $u^{\star}X \xrightarrow{\overline{u}} X$ of u. A *cloven* fibration is a fibration equipped with a *cleavage* which is a choice $\{\overline{u}\}_{J \xrightarrow{u} I \in \mathcal{B}}$ of cartesian liftings. A cloven fibration is *split* if $\overline{Id} = Id$ and $\overline{v}; \overline{u} = \overline{v; u}$ for every pair of composable morphisms in $\mathcal{B}$. When this is the case, p can be presented as an *indexed category* $\mathcal{P} : \mathcal{B} \longrightarrow \mathbf{Cat}$ which is a 'functor' from $\mathcal{B}$ to the 'category' of categories. A *cartesian functor* H from $p : \mathcal{F} \longrightarrow \mathcal{B}$ to $q : \mathcal{E} \longrightarrow \mathcal{B}$ is a functor $H : \mathcal{F} \longrightarrow \mathcal{E}$ such that $p = H; q$ and H sends cartesian morphisms to cartesian morphisms. A cartesian functor is full (faithful) if and only if its fibrewise functors are full (faithful).

Let $\mathcal{F}$ and $\mathcal{B}$ be two categories. A *D-category* ([Ehr88]) is given by three functors $p, G : \mathcal{F} \longrightarrow \mathcal{B}$ and $I : \mathcal{B} \longrightarrow \mathcal{F}$ such that (i) $p \dashv I \dashv G$; (ii) the counit of the first adjunction (and therefore the unit of the second adjunction) is an iso (i.e. I is full and faithful) and (iii) p is a fibration. We say that the D-category is split if p is split.

An *internal category* $\mathbb{C}$ in $\mathcal{B}$ consists of objects $\mathbb{C}_0$, $\mathbb{C}_1$, $\mathbb{C}_2$ and morphisms as shown in the diagram below

$$\begin{tikzcd}
\mathbb{C}_2 \arrow[r, bend left, "\Pi_0"] \arrow[r, "\gamma"] \arrow[r, bend right, "\Pi_1"'] & \mathbb{C}_1 \arrow[r, bend left, "d_0"] \arrow[r, bend right, "d_1"'] & \mathbb{C}_0 \arrow[l, "ID"']
\end{tikzcd}$$

These morphisms in $\mathcal{B}$ must satisfy the well-known conditions. The *externalization* of $\mathbb{C}$

is a split fibration $dom_{\mathbb{C}} : [\mathbb{C}] \longrightarrow \mathcal{B}$. An object in $[\mathbb{C}]$ is a morphism $I \xrightarrow{\alpha} \mathbb{C}_0$. A map from $(I \xrightarrow{\alpha} \mathbb{C}_0)$ to $(J \xrightarrow{\beta} \mathbb{C}_0)$ is a pair $(I \xrightarrow{f} J, I \xrightarrow{m} \mathbb{C}_1)$ such that the two diagrams below commute.

$$
\begin{tikzcd}
I \arrow[r, "m"] \arrow[dr, "\alpha"'] & \mathbb{C}_1 \arrow[d, "d_0"] \\
& \mathbb{C}_0
\end{tikzcd}
\qquad
\begin{tikzcd}
I \arrow[r, "m"] \arrow[d, "f"'] & \mathbb{C}_1 \arrow[d, "d_1"] \\
J \arrow[r, "\beta"'] & \mathbb{C}_0
\end{tikzcd}
$$

$dom_{\mathbb{C}}$ sends $(I \xrightarrow{f} J, I \xrightarrow{m} \mathbb{C}_1)$ onto $I \xrightarrow{f} J$. The corresponding indexed category is denoted by $[\_, \mathbb{C}] : \mathcal{B} \longrightarrow \mathbf{Cat}$. A morphism $(I \xrightarrow{\alpha} \mathbb{C}_0) \longrightarrow (I \xrightarrow{\beta} \mathbb{C}_0)$ in $[I, \mathbb{C}]$ is a morphism $I \xrightarrow{m} \mathbb{C}_1$ such that the following diagram commutes.

$$
\begin{tikzcd}
& I \arrow[dl, "\alpha"'] \arrow[d, "m"] \arrow[dr, "\beta"] & \\
\mathbb{C}_0 & \mathbb{C}_1 \arrow[l, "d_0"] \arrow[r, "d_1"'] & \mathbb{C}_0
\end{tikzcd}
$$

Let $\mathbf{D}$ be a collection of morphisms, called *display maps* ([Tay86, HP89]), of $\mathcal{B}$ such that for any $\bullet \xrightarrow{d} X$ in $\mathbf{D}$ and any $Y \xrightarrow{f} X$, there is a (unique) morphism $\bullet \xrightarrow{c} Y$ in $\mathbf{D}$ rendering the diagram

$$
\begin{tikzcd}
\bullet \arrow[r] \arrow[d, "c"'] & \bullet \arrow[d, "d"] \\
Y \arrow[r, "f"'] & X
\end{tikzcd}
$$

a pullback. Let $\mathcal{B}/\mathbf{D}$ be the full subcategory of the arrow category $\mathcal{B}^{\rightarrow}$ determined by $\mathbf{D}$. The definition of $\mathbf{D}$ simply says that $\mathcal{B}/\mathbf{D} \xrightarrow{cod} \mathcal{B}$ is a (split) fibration.

Suppose $\mathcal{B}$ has finite products and $\mathbf{D}$ is a family of display maps. Let $V \xrightarrow{G} \mathbb{C}_0$ be in $\mathbf{D}$. We can transplant the well-known construction of an internal full subcategory in a locally cartesian closed category to $\mathcal{B}$ if there are enough exponential structures in $\mathcal{B}/\mathbf{D} \xrightarrow{cod} \mathcal{B}$ (notice that $\mathcal{B}$ already has enough pullback structures guaranteed by $\mathbf{D}$). We will call the resulting $\mathbb{C}$ the internal category *induced* by $V \xrightarrow{G} \mathbb{C}_0$. For any $I \xrightarrow{a} \mathbb{C}_0$, we have a display map $G[a]$ that is the pulling-back of G along a. We say the G-induced small category $\mathbb{C}$ is an *internal full subcategory* of $\mathcal{B}$ relative to $\mathbf{D}$ if

$$\begin{array}{rcl} (I \xrightarrow{a} \mathbb{C}_0) & \longmapsto & G[a] \\ (I \xrightarrow{a} \mathbb{C}_0) \xrightarrow{(\alpha,m)} (J \xrightarrow{b} \mathbb{C}_0) & \longmapsto & (\alpha, m^\star) : G[a] \to G[b] \end{array}$$

establishes a full and faithful cartesian functor from $dom_{\mathbb{C}} : [\mathbb{C}] \longrightarrow \mathcal{B}$ to $\mathcal{B}/\mathbf{D} \xrightarrow{cod} \mathcal{B}$.

We say the internal category $\mathbb{C}_0$ has *explicit cartesian closed structures* if there are morphisms $\mathbf{1} \xrightarrow{\dot{\top}} \mathbb{C}_0$, $\mathbb{C}_0 \times \mathbb{C}_0 \xrightarrow{\dot{\times}} \mathbb{C}_0$ and $\mathbb{C}_0 \times \mathbb{C}_0 \xrightarrow{\dot{\to}} \mathbb{C}_0$ such that we have pullbacks as follows:

$$\begin{tikzcd} \mathbf{1} \arrow[r] \arrow[d] & V \arrow[d, "G"] \\ \mathbf{1} \arrow[r, "\dot{\top}"'] & \mathbb{C}_0 \end{tikzcd} \qquad \begin{tikzcd} \bullet \arrow[r] \arrow[d, "G[1] \times_{\mathbb{C}_0^2} G[2]"] & V \arrow[d, "G"] \\ \mathbb{C}_0 \times \mathbb{C}_0 \arrow[r, "\dot{\times}"'] & \mathbb{C}_0 \end{tikzcd} \qquad \begin{tikzcd} \bullet \arrow[r] \arrow[d, "G[1] \to_{\mathbb{C}_0^2} G[2]"] & V \arrow[d, "G"] \\ \mathbb{C}_0 \times \mathbb{C}_0 \arrow[r, "\dot{\to}"'] & \mathbb{C}_0 \end{tikzcd}$$

where $G[i]$ is the pulling-back of G along $\mathbb{C}_0 \times \mathbb{C}_0 \xrightarrow{\pi_i} \mathbb{C}_0$ and $\times_{\mathbb{C}_0^2}$ ($\to_{\mathbb{C}_0^2}$) is the product (exponential) functor on $(\mathcal{B}/\mathcal{D})^{-1}(\mathbb{C}_0 \times \mathbb{C}_0)$. We say $\mathbb{C}$ has *explicit A-indexed product* if there is a morphism $(A \to \mathbb{C}_0) \xrightarrow{\dot{\Pi}} \mathbb{C}_0$ such that we have a pullback diagram as follows:

$$\begin{tikzcd} \bullet \arrow[r] \arrow[d, "\Pi_A(ev^\star(G))"'] & V \arrow[d, "G"] \\ A \to \mathbb{C}_0 \arrow[r, "\dot{\Pi}"'] & \mathbb{C}_0 \end{tikzcd}$$

where $ev^\star(G)$ is the pulling-back of G along $(A \to \mathbb{C}_0) \times A \xrightarrow{ev} \mathbb{C}_0$ and Π_A is the right adjoint to the weakening functor over $(A \to \mathbb{C}_0) \times A \xrightarrow{\pi_1} (A \to \mathbb{C}_0)$.

If $\mathbb{C}$ has explicit cartesian closed structures (A-indexed products), then the externalization of $\mathbb{C}$ has fibrewise cartesian closed structures (right adjoints to weakening functors over morphisms $I \times A \xrightarrow{\pi_1} I$).

3 Internal Typed Calculi and Frame Languages

Definition 3.1 *Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two typed calculi. Suppose the term models of $\mathcal{L}_1$ and $\mathcal{L}_2$ are fibrations $\mathcal{L}^1 : \mathcal{T}_1 \longrightarrow \mathcal{C}_1$ and $\mathcal{L}^2 : \mathcal{T}_2 \longrightarrow \mathcal{C}_2$ respectively. $\mathcal{L}_1$ is said to be* internally codable *in $\mathcal{L}_2$ (notation $\mathcal{L}_1 \blacktriangleleft \mathcal{L}_2$) if (i) there is an embedding functor $\mathcal{I} : \mathcal{C}_1 \longrightarrow \mathcal{T}_2$ and (ii) there is an internal category $\mathbb{L}$ in $\mathcal{T}_2$ such that the left diagram below commutes,* *where the square is a pullback and H is a full and faithful cartesian functor preserving all the necessary structures. We say $\mathcal{L}_1$ is* internally definable *in $\mathcal{L}_2$ (notation $\mathcal{L}_1 \circ\!\!\blacktriangleleft \mathcal{L}_2$) if H is an iso; that is if the right diagram below is a pullback.*

$$
\begin{tikzcd}
\mathcal{T}_1 \arrow[r, hook, "H"] \arrow[dr, "\mathcal{L}^1"'] & {[\mathbb{L}]_{\mathcal{C}_1}} \arrow[r, hook] \arrow[d, "dom_{\mathbb{L}} \restriction \mathcal{C}_1"] & {[\mathbb{L}]} \arrow[d, "dom_{\mathbb{L}}"] \\
 & \mathcal{C}_1 \arrow[r, hook, "\mathcal{I}"'] & \mathcal{T}_2
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathcal{T}_1 \arrow[r, hook] \arrow[d, "\mathcal{L}^1"'] & {[\mathbb{L}]} \arrow[d, "dom_{\mathbb{L}}"] \\
\mathcal{C}_1 \arrow[r, hook, "\mathcal{I}"'] & \mathcal{T}_2
\end{tikzcd}
$$

When $\mathcal{L}_1 \blacktriangleleft (\circ\!\!\blacktriangleleft)\mathcal{L}_2$, we say that $\mathcal{L}_1$ is the internal typed calculus *coded (defined) by $\mathbb{L}$ in $\mathcal{L}_2$ via $\mathcal{I}$. $\mathcal{L}_2$ is said to be the* frame calculus *for $\mathcal{L}_1$. Sometimes we also say that $\mathcal{L}^1$ is internally codable (definable) in $\mathcal{L}^2$ by $\mathbb{L}$ via $\mathcal{I}$.*

In the above two diagrams, the category $\mathcal{C}_2$ does not appear. This shouldn't be seen as a drawback of our definition. All the information about $\mathcal{C}_2$ is already contained in $\mathcal{T}_2$.

Intuitively, $\mathcal{L}_1 \blacktriangleleft \mathcal{L}_2$ means that the role of $\mathcal{L}_1$ can be imitated within $\mathcal{L}_2$. The precise meaning of $\mathcal{L}_1 \blacktriangleleft (\circ\!\!\blacktriangleleft)\mathcal{L}_2$ depends on $\mathcal{I}$, which is not specified in the above definition. For some particular embedding $\mathcal{I}$, $\mathcal{L}_1 \blacktriangleleft \mathcal{L}_2$ means that $\mathcal{L}_2$ is a conservative extension of $\mathcal{L}_1$.

It is possible to extend the above definition to higher order cases. But in this paper definition 3.1 suffices for our purpose.

3.1 Internal Codability in System F

This section takes a look at an example. We shall make a few observations about the family of system F in the perspective of definition 3.1. In order to avoid the messy syntactical account, we shall give a model-theoretic presentation. The notation to be introduced below are borrowed from [Pit87]. The results in this section also relies heavily on those in *loc.cit.*. We can not afford a complete account as it will occupy too much space and distract our attention from the main subject.

Let $\mathcal{F}_0$ denote the simply typed λ-calculus, $\mathcal{F}_1$ the simply typed λ-calculus with free type variables, $\mathcal{F}_2$ the second order polymorphism, $\mathcal{F}_\omega$ the higher order polymorphism, and $\mathcal{F}_i$ the i-th order polymorphism. Each F_i is required to have finite product types.

A term model of $\mathcal{F}_i$ is an indexed category $P_i : \mathcal{C}_i^{op} \longrightarrow \mathbf{Ccc}$ such that

- $\mathbf{Ccc}$ is the category of small cartesian closed categories.
- Substitution functors preserve cartesian closed structures on the nose.
- For $i > 0$, there is an object $U \in \mathcal{C}_i$ such that $\mathcal{C}_i^{op} \xrightarrow{P_i} \mathbf{Ccc} \xrightarrow{Obj} \mathbf{Set}$ is the representable functor $\mathcal{C}_i[\_, U]$.

- $\{\mathcal{C}_i\}_{i\leq\omega}$ can be classified into three groups:
  - $\mathcal{C}_0$ is the terminal category $\star$.
  - $\mathcal{C}_1$ has finite products such that every object of it is U^n, a product, for some $n \in \omega$.
  - For $i > 1$, $\mathcal{C}_i$ has finite products and exponentials whose order degree is less than i. Here is the definition of the order degree, $Od(X)$, of an object X: (i) the order degree of the terminal object is 0; (ii) $Od(U) = 1$; (iii) $Od(W \to V) = 1 + \max\{Od(W), Od(V)\}$ if both $Od(W)$ and $Od(V)$ are defined; (iv) the order degree of any other object is undefined. $\mathcal{C}_i$ must also satisfy the following condition: every object of $\mathcal{C}_i$ is a finite product each of its components has an order degree.
- If $i > 1$, then for any two objects X and I in $\mathcal{C}$ such that X has an order degree, there exists a functor $\Pi_I : P_i(I\times X) \longrightarrow P_i(I)$ right adjoint to the weakening functor $\pi^\star : P_i(I) \longrightarrow P_i(I \times X)$, such that these functors are natural in I, that is for any $I \xrightarrow{\alpha} J$, $\Pi_J; \alpha^\star = (\alpha \times 1_X)^\star; \Pi_I$.

Let $\mathcal{F}_i : \mathcal{T}_i \longrightarrow \mathcal{C}_i$ be the corresponding fibration[1]. The category $\mathcal{T}_i$ has as objects arrows $I \xrightarrow{A} U$. A morphism from $I \xrightarrow{A} U$ to $J \xrightarrow{B} U$ is a pair of arrows (α, f) such that $\alpha : I \longrightarrow J$ and $f : A \longrightarrow \alpha; B$ is a morphism in $\mathcal{F}_i^{-1}(I)$. We write $\top^i_I$ for the terminal object in $\mathcal{F}_i^{-1}(I)$, $\times^i_I$ and $\to^i_I$ for the product and exponential functors on $\mathcal{F}_i^{-1}(I)\times\mathcal{F}_i^{-1}(I)$ and $\mathcal{F}_i^{-1}(I)^{op}\times\mathcal{F}_i^{-1}(I)$ respectively. $!_A : A \longrightarrow \top^i_I$ is the unique morphism from A to the terminal object $\top^i_I$; we will omit the subscript A whenever it is clear from context.

Lemma 3.2 *For each $i > 0$, the following data form an internal category in $\mathcal{T}_i$. It is generated from $(1_U, !) : (U \xrightarrow{1_U} U) \longrightarrow (U \xrightarrow{\top^i_U} U)$.*

$$
\begin{array}{rcll}
\mathbb{D}^i_0 & \stackrel{\text{def}}{=} & U \xrightarrow{\top_U} U & \text{the terminal object in } \mathcal{F}_i^{-1}(U) \\
\mathbb{D}^i_1 & \stackrel{\text{def}}{=} & U^2 \xrightarrow{\pi_1 \to_{U^2} \pi_2} U & \\
\mathbb{D}^i_2 & \stackrel{\text{def}}{=} & U^3 \xrightarrow{(\pi_1 \to_{U^3} \pi_2) \times_{U^3} (\pi_2 \to_{U^3} \pi_3)} U & \\
d^i_0 & \stackrel{\text{def}}{=} & (\pi_1, !) & ! : (\pi_1 \to_{U^2} \pi_2) \longrightarrow \top_{U^2} \\
d^i_1 & \stackrel{\text{def}}{=} & (\pi_2, !) & ! : (\pi_1 \to_{U^2} \pi_2) \longrightarrow \top_{U^2} \\
\Pi^i_0 & \stackrel{\text{def}}{=} & (\langle \pi_1, \pi_2 \rangle, p_1) & (\pi_1 \to_{U^3} \pi_2) \times_{U^3} (\pi_2 \to_{U^3} \pi_3) \xrightarrow{p_1} (\pi_1 \to_{U^3} \pi_2) \\
\Pi^i_1 & \stackrel{\text{def}}{=} & (\langle \pi_2, \pi_3 \rangle, p_2) & (\pi_1 \to_{U^3} \pi_2) \times_{U^3} (\pi_2 \to_{U^3} \pi_3) \xrightarrow{p_2} (\pi_2 \to_{U^3} \pi_3) \\
ID^i & \stackrel{\text{def}}{=} & (\langle 1_U, 1_U \rangle, i) & i \text{ is the transpose of } 1_U \times \top_U \xrightarrow{\pi_1} 1_U \\
\gamma^i & \stackrel{\text{def}}{=} & (\langle \pi_1, \pi_3 \rangle, \overline{ev}) & \overline{ev} \text{ is the internal composition.}
\end{array}
$$

The reader is referred to [Pit87] for a proof.

Theorem 3.3 *Suppose that for $0 \leq i < j \leq \omega$, $\mathcal{F}_i$ and $\mathcal{F}_j$ have the same constant types and objects. Then (i) for any $0 \leq i < j \leq \omega$, $\mathcal{F}_i \blacktriangleleft \mathcal{F}_j$; (ii) $\mathcal{F}_\omega \circ\!\!\blacktriangleleft \mathcal{F}_\omega$.*

[1]In the sequel, we will not bother to differentiate the notation for a language and that for its term model.

We first mention two technical lemmas, whose proofs can be read off from [Pit87].

Lemma 3.4 *$\mathbb{U}_{i+1}$ is equipped with explicit internal cartesian closed structures. For any projection $I\times X \xrightarrow{\pi_I} I \in \mathcal{C}_{i+1}$ such that the order degree of X is less than i, the weakening functor $\pi_I^\star : [I, \mathbb{U}_{i+1}] \longrightarrow [I\times X, \mathbb{U}_{i+1}]$ has a right adjoint that is natural in I.*

Lemma 3.5 *Assume the premise of theorem 3.3. The following diagram is a pullback.*

$$\begin{tikzcd}
\mathcal{T}_i \arrow[r, hook, "H"] \arrow[d, "\mathcal{F}_i"'] & {[\mathbb{D}^i]} \arrow[d, "dom_{\mathbb{D}^i}"] \\
\mathcal{C}_i \arrow[r, hook, "\top_{(_)}"'] & \mathcal{T}_i
\end{tikzcd}$$

Here $\top_{(\_)}$ sends $J \xrightarrow{\alpha} I$ onto $(\alpha, !) : (J \xrightarrow{\top_J} U) \longrightarrow (I \xrightarrow{\top_I} U)$. It is obviously a full embedding. H preserves all the cartesian closed structures; it also preserves indexed products along projections $I \times X \xrightarrow{\pi_I} I$ where the order degree of X is less than i. So $dom_{\mathbb{D}^i}$ is almost a model of $\mathcal{F}_i$ except that it lacks certain right adjoints. But when $i = \omega$, $dom_{\mathbb{D}^i}$ has all the necessary right adjoints. So $\mathcal{F}_\omega \circ\!\blacktriangleleft \mathcal{F}_\omega$.

Proof of theorem 3.3 (sketch) In the following diagram, the right square is a pullback by lemma 3.5. The left square commutes by assumption. By the conservativity of $\mathcal{F}_{i+1}$ over $\mathcal{F}_i$, the embedding $\mathcal{F}_i \hookrightarrow \mathcal{F}_{i+1}$ is full and preserves all the relevant structures. So the composite $\mathcal{F}_i \hookrightarrow \mathcal{F}_{i+1} \to [\mathbb{D}^{i+1}]$ preserves all the structures and right adjoints ($dom_{\mathbb{D}^{i+1}}$ has just enough right adjoints to be a model of $\mathcal{F}_i$). So $\mathcal{F}_i \blacktriangleleft \mathcal{F}_{i+1}$.

$$\begin{tikzcd}
\mathcal{T}_i \arrow[r, hook, "Id"] \arrow[d, "\mathcal{F}_i"'] & \mathcal{T}_{i+1} \arrow[r, hook, "H"] \arrow[d, "\mathcal{F}_{i+1}"] & {[\mathbb{D}^{i+1}]} \arrow[d, "dom_{\mathbb{D}^{i+1}}"] \\
\mathcal{C}_i \arrow[r, hook, "Id"'] & \mathcal{C}_{i+1} \arrow[r, hook, "\top_{(_)}"'] & \mathcal{T}_{i+1}
\end{tikzcd}$$

Notice that the left square is not a pullback. For instance, $[\![\forall X.X]\!]$ is an object over $\mathbf{1} \in \mathcal{C}_2$ but not an object over $\mathbf{1} \in \mathcal{C}_1$. ■

As can be seen from the above proof, the main proof-theoretical fact that underlies theorem 3.3 is the conservativity of F_{i+1} over F_i under the condition stated in the theorem.

4 Introducing λ_{TT}

The purpose of the first two subsections is to refresh our memory on *ELF* and Martin-Löf's logical framework. In order to compare our logical framework to these two languages, we will give a complete presentation of each of the languages. It is our hope that by analysing all the three calculi at an elementary level, we can bring out the point and the advantage of λ_{TT}. Having reviewed the two languages, we will then introduce our logical framework λ_{TT}.

4.1 Martin-Löf's Logical Framework

There are two ways of presenting Martin-Löf's type theory. One approach is to define the type theory in a classical logic. Typed calculi so defined are *polymorphic* in the sense that type constructors in them are not fully specified. For instance in $\lambda x{:}A.b$, b could be of any type. The other approach is to define the type theory in Martin-Löf's logical framework. In contrast to the polymorphic case, terms in languages defined in Martin-Löf's logical framework are fully specified; these languages are therefore called *monomorphic*. To some extent the monomorphic Martin-Löf's type theory is guaranteed to be well-behaved. A well-known fact is that the 'extensional identity types' are not definable in monomorphic type theory. We should point out that these remarks apply to logical frameworks in general.

Martin-Löf's logical framework has following rules with the definitional equality being extensional:

Context

$$\frac{}{\langle\rangle\ \mathbf{valid}}\textsf{Empty Context} \qquad \frac{\Gamma \vdash A\ \mathbf{type} \quad x \text{ fresh}}{\Gamma, x : A\ \mathbf{valid}}\textsf{Context Extension}$$

Assumption

$$\frac{\Gamma, x : A, \Gamma'\ \mathbf{valid}}{\Gamma, x : A, \Gamma' \vdash x : A}\textsf{Variable}$$

Type

$$\frac{\Gamma \vdash A\ \mathbf{type} \quad \Gamma, x : A \vdash B\ \mathbf{type}}{\Gamma \vdash (x{:}A)B\ \mathbf{type}}\textsf{Prod} \qquad \frac{\Gamma \vdash A\ \mathbf{type} \quad \Gamma \vdash a : A \quad \Gamma \vdash A = B}{\Gamma \vdash a : B}\textsf{Tran}$$

Construction

$$\frac{\Gamma \vdash A\ \mathbf{type} \quad \Gamma, x : A \vdash B\ \mathbf{type} \quad \Gamma, x : A \vdash b : B}{\Gamma \vdash \lambda x{:}A.b : (x{:}A)B}\textsf{Abs}$$

$$\frac{\Gamma \vdash A\ \mathbf{type} \quad \Gamma, x : A \vdash B\ \mathbf{type} \quad \Gamma \vdash f : (x{:}A)B \quad \Gamma \vdash a : A}{\Gamma \vdash f(a) : B[a]}\textsf{App}$$

Set

$$\frac{\Gamma\ \textbf{valid}}{\Gamma \vdash Set\ \textbf{type}}\textsf{Set} \qquad \frac{\Gamma \vdash S : Set}{\Gamma \vdash El(S)\ \textbf{type}}\textsf{Refl}$$

Using this framework, the Π-types in Martin-Löf's monomorphic type theory can be defined as follows:

$$\begin{array}{lcl}
\Pi & : & (X{:}Set)(Y{:}(x{:}El(X))Set)Set \\
\Lambda & : & (X{:}Set)(Y{:}(x{:}El(X))Set)(f{:}(x{:}El(X))El(Y(x)))El(\Pi(X,Y)) \\
\bullet & : & (X{:}Set)(Y{:}(x{:}El(X))Set)(f{:}El(\Pi(X,Y)))(x{:}El(X))El(Y(x)).
\end{array}$$

The computational rule can be described as

$$\bullet(A, B, \Lambda(A, B, f), a) = f(a) : El(B(a)).$$

We finish this section by pointing out an important feature of Martin-Löf's framework:

> In Martin-Löf's logical framework, the definitional equalities of type theories (object languages) are *identified* with the definitional equality of the framework.

We will call this the *equality convention.*

4.2 The Edinburgh *LF*

ELF is based on the idea of Martin-Löf's logical framework. Its purpose is to code up a whole range of typed languages instead of just one group of them as in the case of Martin-Löf's framework. The system is structured into three levels: objects, (families of) types and kinds. The top level kinds provide the mechanism to introduce different universes. In any particular application, a fixed number of constants are introduced; they form a signature. These are the rules of *ELF.*

Valid Signatures

$$\frac{}{\langle\rangle}\textsf{Empty Sig} \qquad \frac{\Sigma\ \textbf{Sig} \quad \vdash_\Sigma K \quad a \text{ fresh}}{\Sigma, a : K\ \textbf{Sig}}\textsf{Kind Sig}$$

$$\frac{\Sigma\ \textbf{Sig} \quad \vdash_\Sigma A : Type \quad c \text{ fresh}}{\Sigma, c : A\ \textbf{Sig}}\textsf{Type Sig}$$

Valid Contexts

$$\frac{\Sigma\ \textbf{Sig}}{\vdash_\Sigma \langle\rangle}\textsf{Empty Ctxt} \qquad \frac{\vdash_\Sigma \Gamma \quad \Gamma \vdash_\Sigma A : Type \quad x \text{ fresh}}{\vdash_\Sigma \Gamma, x : A}\textsf{Type Ctxt}$$

Valid Kinds

$$\frac{\vdash_\Sigma \Gamma}{\Gamma \vdash_\Sigma \mathit{Type}}\textsf{Type Kind} \qquad \frac{\Gamma, x : A \vdash_\Sigma K}{\Gamma \vdash_\Sigma \Pi x{:}A.K}\textsf{Pi Kind}$$

Valid Families

$$\frac{\vdash_\Sigma \Gamma \quad c : K \in \Sigma}{\Gamma \vdash_\Sigma c : K}\textsf{Const Fam}$$

$$\frac{\Gamma, x : A \vdash_\Sigma B : \mathit{Type}}{\Gamma \vdash_\Sigma \Pi x{:}A.B : \mathit{Type}}\textsf{Pi Fam}$$

$$\frac{\Gamma, x : A \vdash_\Sigma B : K}{\Gamma \vdash_\Sigma \lambda x{:}A.B : \Pi x{:}A.K}\textsf{Abs Fam}$$

$$\frac{\Gamma \vdash_\Sigma A : \Pi x{:}B.K \quad \Gamma \vdash_\Sigma M : B}{\Gamma \vdash_\Sigma AM : K[M/x]}\textsf{App Fam}$$

$$\frac{\Gamma \vdash_\Sigma A : K \quad \Gamma \vdash_\Sigma K' \quad \Gamma \vdash_\Sigma K \simeq K'}{\Gamma \vdash_\Sigma A : K'}\textsf{Conv Fam}$$

Valid Objects

$$\frac{\vdash_\Sigma \Gamma \quad c : A \in \Sigma}{\Gamma \vdash_\Sigma c : A}\textsf{Const Obj}$$

$$\frac{\vdash_\Sigma \Gamma \quad x : A \in \Gamma}{\Gamma \vdash_\Sigma x : A}\textsf{Var Obj}$$

$$\frac{\Gamma, x : A \vdash_\Sigma M : B}{\Gamma \vdash_\Sigma \lambda x{:}A.M : \Pi x{:}A.B}\textsf{Abs Fam}$$

$$\frac{\Gamma \vdash_\Sigma M : \Pi x{:}A.B \quad \Gamma \vdash_\Sigma N : A}{\Gamma \vdash_\Sigma MN : B[N/x]}\textsf{App Obj}$$

$$\frac{\Gamma \vdash_\Sigma M : A \quad \Gamma \vdash_\Sigma A' \quad \Gamma \vdash_\Sigma A \simeq A'}{\Gamma \vdash_\Sigma M : A'}\textsf{Conv Obj}$$

To give an idea of how encodings in *ELF* look like, we present a fragmentary encoding of simply typed λ-calculus. A complete formalization should include encodings of the equation rules for lambda abstraction and application. The signature should contain at least the following constants:

$$\begin{array}{rcl}
U & : & \mathit{Type} \\
T & : & U \rightarrow \mathit{Type} \\
\doteq & : & \Pi\sigma{:}U.T(\sigma)\rightarrow T(\sigma)\rightarrow \mathit{Type} \\
\Rightarrow & : & U\rightarrow U\rightarrow U \\
\mathsf{abs} & : & \Pi\sigma{:}U.\Pi\tau{:}U.[T(\sigma)\rightarrow T(\tau)]\rightarrow T(\sigma\Rightarrow\tau) \\
\mathsf{app} & : & \Pi\sigma{:}U.\Pi\tau{:}U.T(\sigma\Rightarrow\tau)\rightarrow T(\sigma)\rightarrow T(\tau) \\
\beta & : & \Pi\sigma{:}U.\Pi\tau{:}U.\Pi f{:}T(\sigma)\rightarrow T(\tau).\Pi x{:}T(\sigma).[\mathsf{app}_{\sigma,\tau}(\mathsf{abs}_{\sigma,\tau}(f),x)\doteq_\tau fx].
\end{array}$$

We should remark that in *ELF* equalities in object languages are 'internalized' by appropriate families of types. Following Martin-Löf's logical framework, *ELF* employs the so-called *variable convention*:

Variables of object languages are identified with the variables in the logical framework.

4.3 λ_{TT}: A Framework for Defining Type Theories

A typed calculus can only be a framework for a certain collection of typed calculi[2]. A well-known example is that in *ELF*, we are not able to code up relevant logic. Moving to a stronger type system is not necessarily a good idea. As far as logical frameworks are concerned, the priority should always be with simplicity and good meta-theoretical properties. It should be emphasized that λ_{TT} is not meant to be a stronger logical framework. It is a manifestation of our view that a logical framework is a setting for defining frame languages.

As long as one's logical framework is a typed calculus, what one can code up in his framework are some other typed calculi; but that is all. Some logics can be faithfully represented in such a framework because they have type theoretical formulations. Some other logics can be mimicked in the framework because some aspects of them can be captured type theoretically and these aspects are considered to be important. It is for this reason that we think of λ_{TT} as a framework for defining *t*ype *t*heories.

The language λ_{TT} is designed with these goals in mind:

- It should unify *ELF* and Martin-Löf's logical framework.
- It should simplify model theoretical investigations.
- It should have a built-in mechanism that formalizes processes to define internally codable (definable) typed calculi.

There are several design decisions one has to make. Let's mention two of them:

- We decided that λ_{TT} should have finite product types.
- We decided that λ_{TT} should be an explicit language.

Both decisions are in line with the second goal mentioned above. As for pragmatics, we adopt the

- well-established variable convention and the
- well-behaved equality convention.

[2]We only consider languages where there is a notion of variables.

In true type theoretical spirit, λ_{TT} is an extension of Martin-Löf's logical framework. The additional features are product types and universe declaration[3]. The universe declaration is the best way of getting rid of higher order operators like those in *ELF* while retaining all the expressive power. The product types are a compromise between having Σ-types, which is troublesome, and having only Π-types, which sometimes does not produce good enough encodings (the problem is even more serious when we have universe declaration). For instance, in *ELF* the plus operator $+ : N \rightarrow N \rightarrow N$ has a drawback: $+(n)$ usually does not correspond to anything in object language. In λ_{TT} we can declare $+$ to be of type $N \times N \rightarrow N$. The problem now disappears.

λ_{TT} has the following assertions:

- Ω **Uni**—Ω is a list of special types called *universes*. A universe is a type whose inhabitants can be lifted to types.
- $\vdash_\Omega \Sigma$ **Sig**—Σ is a list of constant types and/or constant objects whose types may contain universes declared in Ω.
- $\vdash_{\Omega;\Sigma} \Gamma$ **Con**—Γ is a well-formed context under Ω and Σ.
- $\vdash_{\Omega;\Sigma;\Gamma} Th$ **Theory**—Th is a finite set of definitional equations under Ω, Σ and Γ.
- The other assertions are obvious.

The basic rules of λ_{TT} are as follows:

Universe

$$\frac{}{\vdash \langle\rangle\ \mathbf{Uni}}\textsf{Empty Universe} \qquad \frac{\vdash \Omega\ \mathbf{Uni} \quad U \text{ fresh}}{\vdash \Omega, U\ \mathbf{Uni}}\textsf{Universe Intro}$$

Signature

$$\frac{\vdash \Omega\ \mathbf{Uni}}{\vdash_\Omega \langle\rangle\ \mathbf{Sig}}\textsf{Empty Sig} \qquad \frac{\vdash_\Omega \Sigma\ \mathbf{Sig} \quad C \text{ fresh}}{\vdash_\Omega \Sigma, C\ \mathbf{Sig}}\textsf{Sig}_1 \qquad \frac{[] \vdash_{\Omega;\Sigma} A\ \mathbf{Type} \quad c \text{ fresh}}{\vdash_\Omega \Sigma, c : A\ \mathbf{Sig}}\textsf{Sig}_2$$

Context

$$\frac{\vdash_\Omega \Sigma\ \mathbf{Sig}}{\vdash_{\Omega;\Sigma} []\ \mathbf{Con}}\textsf{Empty Context} \qquad \frac{\Gamma \vdash_{\Omega;\Sigma} A\ \mathbf{Type}}{\vdash_{\Omega;\Sigma} \Gamma, x : A\ \mathbf{Con}}\textsf{Context Intro}$$

Type

$$\frac{\vdash_{\Omega;\Sigma} \Gamma\ \mathbf{Con} \quad U \text{ in } \Omega}{\Gamma \vdash_{\Omega;\Sigma} U\ \mathbf{Type}}\textsf{Var}_1 \qquad \frac{\vdash_{\Omega;\Sigma} \Gamma\ \mathbf{Con} \quad C \text{ in } \Sigma}{\Gamma \vdash_{\Omega;\Sigma} C\ \mathbf{Type}}\textsf{Var}_2$$

$$\frac{\Gamma \vdash_{\Omega;\Sigma} A\ \mathbf{Type} \quad \Gamma \vdash_{\Omega;\Sigma} B\ \mathbf{Type}}{\Gamma \vdash_{\Omega;\Sigma} A \times B\ \mathbf{Type}}\textsf{prod} \qquad \frac{\Gamma, x : A \vdash_{\Omega;\Sigma} B\ \mathbf{Type}}{\Gamma \vdash_{\Omega;\Sigma} \Pi x{:}A.B\ \mathbf{Type}}\textsf{Prod}$$

[3]As we have seen, Martin-Löf deals exclusively with only one universe *Set*.

$$\frac{\vdash_{\Omega;\Sigma} \Gamma\ \mathbf{Con}}{\Gamma \vdash_{\Omega;\Sigma} \mathbf{1}\ \mathbf{Type}}\mathsf{Unit} \qquad \frac{\Gamma \vdash_{\Omega;\Sigma} a : U \quad U \text{ in } \Omega}{\Gamma \vdash_{\Omega;\Sigma} \mathbf{t}_U(a)\ \mathbf{Type}}\mathsf{Reflection}$$

Object

$$\frac{\vdash_{\Omega;\Sigma} \Gamma\ \mathbf{Con} \quad c : C \text{ in } \Sigma}{\Gamma \vdash_{\Omega;\Sigma} c : C}\mathsf{Var}_s \qquad \frac{\vdash_{\Omega;\Sigma} \Gamma\ \mathbf{Con} \quad x : A \text{ in } \Gamma}{\Gamma \vdash_{\Omega;\Sigma} x : A}\mathsf{Var}_c$$

$$\frac{\Gamma, x : A \vdash_{\Omega;\Sigma} M : B}{\Gamma \vdash_{\Omega;\Sigma} \lambda x{:}A.M : \Pi x{:}A.B}\mathsf{Abs} \qquad \frac{\Gamma \vdash_{\Omega;\Sigma} M : \Pi x{:}A.B \quad \Gamma \vdash_{\Omega;\Sigma} N : A}{\Gamma \vdash_{\Omega;\Sigma} MN : B[N/x]}\mathsf{App}$$

$$\frac{\Gamma \vdash_{\Omega;\Sigma} M : A \quad \Gamma \vdash_{\Omega;\Sigma} N : B}{\Gamma \vdash_{\Omega;\Sigma} \langle M, N\rangle : A\times B}\mathsf{pair} \qquad \frac{\Gamma \vdash_{\Omega;\Sigma} M : A\times B}{\Gamma \vdash_{\Omega;\Sigma} \pi_1 M : A}\mathsf{Proj}_1$$

$$\frac{\Gamma \vdash_{\Omega;\Sigma} M : A\times B}{\Gamma \vdash_{\Omega;\Sigma} \pi_2 M : B}\mathsf{Proj}_2 \qquad \frac{\vdash_{\Omega;\Sigma} \Gamma\ \mathbf{Con}}{\Gamma \vdash_{\Omega;\Sigma} \star : \mathbf{1}}\mathsf{unit}$$

Transitivity

$$\frac{\Gamma \vdash_{\Omega;\Sigma} M : A \quad \Gamma \vdash_{\Omega;\Sigma} B\ \mathbf{Type} \quad A \simeq B}{\Gamma \vdash_{\Omega;\Sigma} M : B}\mathsf{Tran}$$

Here $\simeq$ is the extensional conversion. The conversion of types is induced by the conversion on terms through the lifting operator $\mathbf{t}$. The language described so far is the essential part of λ_{TT}, which we will call λ_{LF}. It is hoped that λ_{LF} has good meta-properties and is decidable. In order to have an external view on equations in object languages we need equational contexts:

$$\frac{\vdash_{\Omega} \Sigma\ \mathbf{Sig}}{\vdash_{\Omega;\Sigma;\Gamma} \langle\rangle\ \mathbf{Theory}}\mathsf{Empty\ Theory}$$

$$\frac{\Gamma \vdash_{\Omega;\Sigma} M : A \quad \Gamma \vdash_{\Omega;\Sigma} N : A \quad \vdash_{\Omega;\Sigma} Th\ \mathbf{Theory}}{\vdash_{\Omega;\Sigma} Th[\Gamma \vdash M = N : A]\ \mathbf{Theory}}\mathsf{Theory\ Introduction}$$

An equational context is just a finite list of definitional equations. A careful reader must have noticed that by eliminating the higher order structure in *ELF* we have collapsed rules about valid families and those about valid objects into one group.

There are at least three questions concerned with λ_{TT}:

1. Is the sublanguage λ_{LF} confluent and strongly normalizing?

2. How do we carry out meta-theoretical investigations into a typed calculus formulated in λ_{TT}?

3. What is the proper notion of semantics of λ_{TT}?

The answer to 1 is believed to be yes. But so far its combinatorial complexity has defeated all attempts to prove it. The problem is that the proof of subject reduction and that of Church-Rosser property are heavily interwined. No trick has been invented to break the cycle.

Question 2 is harder. It can not be tackled before the proof theory of λ_{TT} has been fully understood. This problem is closely related to that of compositional understanding of type theory—an issue that has not yet been properly addressed.

This paper attempts to give an answer to question 3.

5 Formulating Object Languages

Some encodings given in this section may look more mathematical than proof-theoretical. We have tailored the encodings in such a way as to make the subsequent discussions easier. The reader is encouraged to compare examples 5.1 and 5.3 to those given in sections 4.1 and 4.2. In sequel, we often use subscript for application. For instance, $\mathsf{app}_{\sigma,\tau}$ stands for $\mathsf{app}(\sigma)(\tau)$. We will not bother to differentiate the universe declaration Ω and signature Σ notationally in the following examples.

Before giving the actual encodings, it is perhaps helpful to explain how *ELF*, a higher order language, is modelled by λ_{TT}, a first order language. In *ELF* we have families of types such as $\delta \to Type$, where $\delta : Type$. A term $true$ of that kind associates with each term a of δ a type $true(a)$. In λ_{TT} we can do the same by replacing $true$ by a special type, a universe U. The declaration Ω contains a list of universes needed in formulating an object language. A closed term a of U is the name of the type $\mathbf{t}_U(a)$. The fact that in λ_{TT} one can not declare a family of types in a context means that in any event, a term of $\delta \to Type$ only uses a fragment of $Type$. For that reason there is no loss of generality when replacing $true$ by a universe.

Example 5.1 *Throughout the rest of this paper, we fix a simply typed λ-calculus with a finite number of constant types and objects. The following is a formulation of this calculus in λ_{TT}.*

$$
\begin{array}{rcl}
\Omega & : & U \\
 & & \begin{array}{rcl}
\wedge & : & U \times U \to U \\
\Rightarrow & : & U \times U \to U \\
\Diamond & : & U \\
\mathsf{v} & : & \mathbf{1} \to \mathbf{t}_U(\Diamond)
\end{array} \\
\Sigma & : & \begin{array}{rcl}
\mathsf{abs} & : & \Pi\sigma{:}U.\Pi\tau{:}U.[\mathbf{t}_U(\sigma) \to \mathbf{t}_U(\tau)] \to \mathbf{t}_U(\sigma \Rightarrow \tau) \\
\mathsf{app} & : & \Pi\sigma{:}U.\Pi\tau{:}U.\mathbf{t}_U(\sigma \Rightarrow \tau) \times \mathbf{t}_U(\sigma) \to \mathbf{t}_U(\tau) \\
\mathsf{pair} & : & \Pi\sigma{:}U.\Pi\tau{:}U.\mathbf{t}_U(\sigma) \times \mathbf{t}_U(\tau) \to \mathbf{t}_U(\sigma \wedge \tau) \\
\mathsf{pr}_1 & : & \Pi\sigma{:}U.\Pi\tau{:}U.\mathbf{t}_U(\sigma \wedge \tau) \to \mathbf{t}_U(\sigma) \\
\mathsf{pr}_2 & : & \Pi\sigma{:}U.\Pi\tau{:}U.\mathbf{t}_U(\sigma \wedge \tau) \to \mathbf{t}_U(\tau)
\end{array} \\
Th_\lambda & : & \begin{array}{l}
\vdash_{\Omega;\Sigma} [\sigma : U, \tau : U, f : \mathbf{t}_U(\sigma) \to \mathbf{t}_U(\tau), x : \mathbf{t}_U(\sigma) \vdash \mathsf{app}_{\sigma,\tau}(\mathsf{abs}_{\sigma,\tau}(f), x) = fx] \\
\vdash_{\Omega;\Sigma} [\sigma : U, \tau : U, f : \mathbf{t}_U(\sigma \Rightarrow \tau) \vdash \mathsf{abs}_{\sigma,\tau}(\lambda x{:}\mathbf{t}_U(\sigma).\mathsf{app}_{\sigma,\tau}(f, x)) = f] \\
\vdash_{\Omega;\Sigma} [\sigma : U, \tau : U, x : \mathbf{t}_U(\sigma), y : \mathbf{t}_U(\tau) \vdash (\mathsf{pr}_1)_{\sigma,\tau}(\mathsf{pair}_{\sigma,\tau}(x, y)) = x] \\
\vdash_{\Omega;\Sigma} [\sigma : U, \tau : U, x : \mathbf{t}_U(\sigma), y : \mathbf{t}_U(\tau) \vdash (\mathsf{pr}_2)_{\sigma,\tau}(\mathsf{pair}_{\sigma,\tau}(x, y)) = y] \\
\vdash_{\Omega;\Sigma} [\sigma : U, \tau : U, z : \mathbf{t}_U(\sigma \wedge \tau) \vdash \mathsf{pair}_{\sigma,\tau}((\mathsf{pr}_1)_{\sigma,\tau}z, (\mathsf{pr}_2)_{\sigma,\tau}z) = z]
\end{array}
\end{array}
$$

$$\vdash_{\Omega;\Sigma} [x : \mathbf{t}_U(\Diamond) \vdash \mathsf{v}(\star) = x]$$

$\mathcal{E}_\lambda$ will stand for this encoding. We have not bothered to spell out the encoding for the constants as it is obvious how it can be done. The same remark also applies to the following examples.

Example 5.2 *Again fix a higher order polymorphic λ-calculus with a finite number of constants. To formulate the language in λ_{TT}, one must have an operator that transforms higher order functionals to types. So in addition to the encoding for the simply typed λ-calculus, we need to add constants and equations that deal with kinds and constructors.*

$$\Omega \;:\; U, K$$

$$\Sigma \;:\; \begin{array}{rcl} -- & & \text{those in the signature of 5.1} \\ \dot{\wedge} & : & K \times K \to K \\ \dot{\Rightarrow} & : & K \times K \to K \\ \dot{\Diamond} & : & K \\ \dot{\mathsf{v}} & : & \mathbf{1} \to \mathbf{t}_K(\dot{\Diamond}) \\ \dot{\mathsf{abs}} & : & \Pi\kappa{:}K.\Pi\varrho{:}K.[\mathbf{t}_K(\kappa) \to \mathbf{t}_K(\varrho)] \to \mathbf{t}_K(\kappa \dot{\Rightarrow} \varrho) \\ \dot{\mathsf{app}} & : & \Pi\kappa{:}K.\Pi\varrho{:}K.\mathbf{t}_K(\kappa \dot{\Rightarrow} \varrho) \times \mathbf{t}_K(\kappa) \to \mathbf{t}_K(\varrho) \\ \dot{\mathsf{pair}} & : & \Pi\kappa{:}K.\Pi\varrho{:}K.\mathbf{t}_K(\kappa) \times \mathbf{t}_K(\varrho) \to \mathbf{t}_K(\kappa \dot{\wedge} \varrho) \\ \dot{\mathsf{pr}}_1 & : & \Pi\kappa{:}K.\Pi\varrho{:}K.\mathbf{t}_K(\kappa \dot{\wedge} \varrho) \to \mathbf{t}_K(\kappa) \\ \dot{\mathsf{pr}}_2 & : & \Pi\kappa{:}K.\Pi\varrho{:}K.\mathbf{t}_K(\kappa \dot{\wedge} \varrho) \to \mathbf{t}_K(\varrho) \\ T & : & K \\ \mathsf{in} & : & \mathbf{t}_K(T) \to U \\ \mathsf{out} & : & U \to \mathbf{t}_K(T) \\ \dot{\forall} & : & \Pi\kappa{:}K.(\mathbf{t}_K(\kappa) \to U) \to U \\ \mathsf{ABS} & : & \Pi\kappa{:}K.\Pi F{:}\mathbf{t}_K(\kappa) \to U.(\Pi\sigma{:}\mathbf{t}_K(\kappa).\mathbf{t}_U(F\sigma)) \to \mathbf{t}_U(\dot{\forall}_\kappa(F)) \\ \mathsf{APP} & : & \Pi\kappa{:}K.\Pi F{:}\mathbf{t}_K(\kappa) \to U.\mathbf{t}_U(\dot{\forall}_\kappa(F)) \to \Pi\sigma{:}\mathbf{t}_K(\kappa).\mathbf{t}_U(F\sigma) \end{array}$$

Th_{PL} : those in the equational theory in 5.1

$$\vdash_{\Omega;\Sigma} [\kappa : K, \varrho : K, f : \mathbf{t}_K(\kappa) \to \mathbf{t}_K(\varrho), x : \mathbf{t}_K(\kappa) \vdash \dot{\mathsf{app}}_{\kappa,\varrho}(\dot{\mathsf{abs}}_{\kappa,\varrho}(f), x) = fx]$$

$$\vdash_{\Omega;\Sigma} [\kappa : K, \varrho : K, f : \mathbf{t}_K(\kappa \dot{\Rightarrow} \varrho) \vdash \dot{\mathsf{abs}}_{\kappa,\varrho}(\lambda x{:}\mathbf{t}_K(\kappa).\dot{\mathsf{app}}_{\kappa,\varrho}(f, x)) = f]$$

$$\vdash_{\Omega;\Sigma} [\kappa : K, \varrho : K, x : \mathbf{t}_K(\kappa), y : \mathbf{t}_K(\varrho) \vdash (\dot{\mathsf{pr}}_1)_{\sigma,\tau}(\dot{\mathsf{pair}}_{\sigma,\tau}(x, y)) = x]$$

$$\vdash_{\Omega;\Sigma} [\kappa : K, \varrho : K, x : \mathbf{t}_K(\kappa), y : \mathbf{t}_K(\varrho) \vdash (\dot{\mathsf{pr}}_2)_{\sigma,\tau}(\dot{\mathsf{pair}}_{\sigma,\tau}(x, y)) = y]$$

$$\vdash_{\Omega;\Sigma} [\kappa : K, \varrho : K, z : \mathbf{t}_K(\kappa \dot{\wedge} \varrho) \vdash \dot{\mathsf{pair}}_{\sigma,\tau}((\dot{\mathsf{pr}}_1)_{\sigma,\tau} z, (\dot{\mathsf{pr}}_2)_{\sigma,\tau} z) = z]$$

$$\vdash_{\Omega;\Sigma} [x : \mathbf{t}_K(\dot{\Diamond}) \vdash \dot{\mathsf{v}}(\star) = x]$$

$$\vdash_{\Omega;\Sigma} [x : \mathbf{t}_K(T) \vdash \mathsf{out}(\mathsf{in}(x)) = x]$$

$$\vdash_{\Omega;\Sigma} [x : U \vdash \mathsf{in}(\mathsf{out}(x)) = x]$$

$$\vdash_{\Omega;\Sigma} [\kappa : K, F : \mathbf{t}_K(\kappa) \to U, x : (\Pi\sigma{:}\mathbf{t}_K(\kappa).\mathbf{t}_U(F\sigma)) \vdash \mathsf{APP}_{\kappa,F}(\mathsf{ABS}_{\kappa,F}(x)) = x]$$

$$\vdash_{\Omega;\Sigma} [\kappa : K, F : \mathbf{t}_K(\kappa) \to U, x : \mathbf{t}_U(\dot{\forall}_\kappa(F)) \vdash \mathsf{ABS}_{\kappa,F}(\mathsf{APP}_{\kappa,F}(x)) = x]$$

Intuitively K is the universe of kinds and T is the kind of all types. We will refer to the entire encoding of higher order polymorphism as $\mathcal{E}_{PL}$.

Example 5.3 *We now give an encoding of Martin-Löf's type theory built upon a finite number of constants. The encoding should look familiar. In this example, we leave out the subscript in* $\mathbf{t}_{Set}$.

$$\begin{array}{rrcl}
\Omega & : & Set & \\
& \pi & : & \Pi X{:}Set.(\mathbf{t}(X)\rightarrow Set)\rightarrow Set \\
& \sigma & : & \Pi X{:}Set.(\mathbf{t}(X)\rightarrow Set)\rightarrow Set \\
& \Lambda & : & \Pi X{:}Set.\Pi Y{:}\mathbf{t}(X)\rightarrow Set.\Pi f{:}(\Pi x{:}\mathbf{t}(X).\mathbf{t}(Y(x))).\mathbf{t}(\Pi(X,Y)) \\
\Sigma\ : & \bullet & : & \Pi X{:}Set.\Pi Y{:}\mathbf{t}(X)\rightarrow Set.\Pi f{:}\mathbf{t}(\pi(X,Y)).\Pi x{:}\mathbf{t}(X).\mathbf{t}(Y(x)) \\
& \mathbf{pair} & : & \Pi X{:}Set.\Pi Y{:}\mathbf{t}(X)\rightarrow Set.\Pi x{:}\mathbf{t}(X).\Pi y{:}\mathbf{t}(Y(x)).\mathbf{t}(\sigma(X,Y)) \\
& \mathbf{P}_1 & : & \Pi X{:}Set.\Pi Y{:}\mathbf{t}(X)\rightarrow Set.\Pi f{:}\mathbf{t}(\sigma(X,Y)).\mathbf{t}(X) \\
& \mathbf{P}_2 & : & \Pi X{:}Set.\Pi Y{:}\mathbf{t}(X)\rightarrow Set.\Pi f{:}\mathbf{t}(\sigma(X,Y)).Y(\mathbf{P}_1(X,Y,f))
\end{array}$$

$$\begin{array}{rcl}
Th_{ML} & : & \vdash_{\Omega;\Sigma} [A : Set, B : \mathbf{t}(A)\rightarrow Set, f : (\Pi x{:}\mathbf{t}(A).\mathbf{t}(B(x))), a : \mathbf{t}(A) \\
& & \vdash \bullet(A,B,\Lambda(A,B,f),a) = f(a) : \mathbf{t}(B(a))] \\
& & \vdash_{\Omega;\Sigma} [A : Set, B : \mathbf{t}(A)\rightarrow Set, f : \mathbf{t}(\pi(A,B)) \\
& & \vdash \Lambda(A,B,\lambda x{:}\mathbf{t}(A).\bullet(A,B,f,x)) = f : \mathbf{t}(\sigma(A,B))] \\
& & \vdash_{\Omega;\Sigma} [A : Set, B : \mathbf{t}(A)\rightarrow Set, a : \mathbf{t}(A), b : \mathbf{t}(B(a)) \\
& & \vdash \mathbf{P}_1(A,B,\mathbf{pair}(A,B,a,b)) = a : \mathbf{t}(A)] \\
& & \vdash_{\Omega;\Sigma} [A : Set, B : \mathbf{t}(A)\rightarrow Set, a : \mathbf{t}(A), b : \mathbf{t}(B(a)) \\
& & \vdash \mathbf{P}_2(A,B,\mathbf{pair}(A,B,a,b)) = b : \mathbf{t}(B(a))] \\
& & \vdash_{\Omega;\Sigma} [A : Set, B : \mathbf{t}(A)\rightarrow Set, f : \mathbf{t}(\sigma(A,B)) \\
& & \vdash \mathbf{pair}(A,B,\mathbf{P}_1(A,B,f),\mathbf{P}_2(A,B,f)) = f : \mathbf{t}(\sigma(A,B))]
\end{array}$$

We denote this encoding by $\mathcal{E}_{ML}$.

6 Small Categories in λ_{TT}

How do we code up an object language? Since the term model of λ_{TT} is a fibration, we can rephrase the question as this: how do we say "coding-up" in the language of fibrations? Our view, which is motivated by the example given in section 3, is that a coding-up process amounts to defining an internal category. It follows that in λ_{TT} there should be a routine way of constructing internal categories. It is the purpose of this section to show that λ_{TT} has a built-in mechanism for defining internal categories.

Let $\mathcal{E}$ be an encoding in λ_{TT}. We can build a term model $(\lambda_{TT},\mathcal{E})$ from the constants in the universe declaration and the signature of $\mathcal{E}$ using the equational contexts in $\mathcal{E}$. It is common knowledge that the term model forms a fibration $(\lambda_{TT},\mathcal{E}) : \mathcal{T} \longrightarrow \mathcal{C}$. $\mathcal{C}$ is the category of contexts and contexts realizations. Here a context realization $r : \Gamma \longrightarrow [x_1 : A_1,\ldots,x_n : A_n]$ is a tuple $((\cdots(a_1,a_2),\ldots,a_{n-1}),a_n)$ such that

$$\begin{array}{rcl}
\Gamma & \vdash & a_1 : A_1 \\
\Gamma & \vdash & a_2 : A_2[a_1/x_1] \\
& \vdots &
\end{array}$$

$$\Gamma \;\vdash\; a_n : A_n[a_1/x_1, \ldots, a_{n-1}/x_{n-1}].$$

The objects of $\mathcal{T}$ are types; for instance $\Gamma \vdash A$ is an object in $\mathcal{T}$ (We have left out the meta-symbol **type**). A morphism from $\Gamma \vdash A$ to $\Delta \vdash B$ is a pair (r, f) where $\Gamma \vdash r : \Delta$ is a context realization and $\Gamma \vdash f : A \rightarrow B[r]$. Here $B[r]$ is obtained from B by replacing all free variables by the corresponding objects in r. Notice that strictly speaking f should be the equivalence class $[f]$. We will however always confuse an equivalence class with one of its representatives. The functor $(\lambda_{TT}, \mathcal{E}) : \mathcal{T} \longrightarrow \mathcal{C}$ takes $\Gamma \vdash A$ to Γ and (r, f) to r.

There is a collection $\mathcal{D}$ of display maps in $\mathcal{C}$. A realization $\Gamma \xrightarrow{r} \Delta$ is in $\mathcal{D}$ iff it is of the form $[x_1 : A_1, \ldots, x_n : A_n, x_{n+1} : A_{n+1}] \xrightarrow{(x_1,\ldots,x_n)} [x_1 : A_1, \ldots, x_n : A_n]$. It is readily seen that $\mathcal{T} \xrightarrow{(\lambda_{TT},\mathcal{E})} \mathcal{C}$ is isomorphic to the fibration $\mathcal{C}/\mathcal{D} \xrightarrow{cod} \mathcal{C}$.

In category theory, there is a well-known technique of constructing an internal category from a generic morphism $T \xrightarrow{G} U$. In type theory this generic morphism becomes a generic type judgement $x : U \vdash G$ **type**. Not every type U has this property. We have called universes those which have this property. Suppose $\mathcal{U}$ is an encoding whose sole function is to declare a universe U. A distinguished feature of λ_{TT} is that $\mathcal{T}$ contains an internal category $\mathbb{U}$ constructed from U. This is made possible by the lifting operator $\mathbf{t}_U$. Using de Bruijn's notation, this small category has the following components:

$$
\begin{array}{rcll}
\mathbb{U}_0 & : & U \vdash \mathbf{1} & \\
\mathbb{U}_1 & : & U^2 \vdash \mathbf{t}_U(1) \rightarrow \mathbf{t}_U(2) & \\
\mathbb{U}_2 & : & U^3 \vdash [\mathbf{t}_U(1) \rightarrow \mathbf{t}_U(2)] \times [\mathbf{t}_U(2) \rightarrow \mathbf{t}_U(3)] & \\
d_0 & : & (\pi_1, U^2 \vdash \lambda w{:}[\mathbf{t}_U(1) \rightarrow \mathbf{t}_U(2)].\star & \pi_1 \text{ is } [x : U, y : U] \xrightarrow{(x)} [x : U] \\
d_1 & : & (\pi_2, U^2 \vdash \lambda w{:}[\mathbf{t}_U(1) \rightarrow \mathbf{t}_U(2)].\star) & \pi_2 \text{ is } [x : U, y : U] \xrightarrow{(y)} [y : U] \\
\Pi_0 & : & ((\pi_1, \pi_2), U^3 \vdash \lambda w{:}[\mathbf{t}_U(1) \rightarrow \mathbf{t}_U(2)] \times [\mathbf{t}_U(2) \rightarrow \mathbf{t}_U(3)].\pi_1 w) & \\
\Pi_1 & : & ((\pi_2, \pi_3), U^3 \vdash \lambda w{:}[\mathbf{t}_U(1) \rightarrow \mathbf{t}_U(2)] \times [\mathbf{t}_U(2) \rightarrow \mathbf{t}_U(3)].\pi_2 w) & \\
ID & : & (\delta_U, U \vdash \lambda w{:}\mathbf{1}.\lambda v{:}\mathbf{t}_U(1).v) & \delta_U \text{ is } [x : U] \xrightarrow{(x,x)} [x_1 : U, x_2 : U] \\
\gamma & : & ((\pi_1, \pi_3), U^3 \vdash \lambda w{:}[\mathbf{t}_U(1) \rightarrow \mathbf{t}_U(2)] \times [\mathbf{t}_U(2) \rightarrow \mathbf{t}_U(3)].\lambda v{:}\mathbf{t}_U(1).(\pi_2 w)((\pi_1 w)v)). &
\end{array}
$$

In the above definition, U^2 is the context $[x_1 : U, x_2 : U]$; (π_1, π_2) is the context realization $[x_1 : U, x_2 : U, x_3 : U] \xrightarrow{(x_1,x_2)} [x : U, y : U]$ etc.. The composition γ can be depicted as follows:

$$
\begin{array}{ll}
 & U^3 \vdash [\mathbf{t}_U(1) \rightarrow \mathbf{t}_U(2)] \times [\mathbf{t}_U(2) \rightarrow \mathbf{t}_U(3)] \\
(\pi_1, \pi_2) \Big\downarrow & \qquad \Downarrow U^3 \vdash \lambda w{:}[\mathbf{t}_U(1) \rightarrow \mathbf{t}_U(2)] \times [\mathbf{t}_U(2) \rightarrow \mathbf{t}_U(3)].\lambda v{:}\mathbf{t}_U(1).(\pi_2 w)((\pi_1 w)v) \\
 & U^2 \vdash \mathbf{t}_U(1) \rightarrow \mathbf{t}_U(2)
\end{array}
$$

Notice that $U^3 \vdash (\mathbf{t}_U(1) \rightarrow \mathbf{t}_U(2))[(\pi_1, \pi_3)] = \mathbf{t}_U(1) \rightarrow \mathbf{t}_U(3)$.

Lemma 6.1 *The above data form an internal category in $\mathcal{T}$.*

Proof: Routine. Let's just mention two things. First it can be easily checked that the following diagram is a pullback. Here ! is $\lambda x{:}A.\star$ for appropriate A; p_1 and p_2 are $U^3 \vdash \lambda w{:}[\mathbf{t}_U(1)\to\mathbf{t}_U(2)]\times[\mathbf{t}_U(2)\to\mathbf{t}_U(3)].\pi_1 w$ and $U^3 \vdash \lambda w{:}[\mathbf{t}_U(1)\to\mathbf{t}_U(2)]\times[\mathbf{t}_U(2)\to\mathbf{t}_U(3)].\pi_2 w$ respectively.

$$\begin{tikzcd}
U^3 \vdash [\mathbf{t}_U(1)\to\mathbf{t}_U(2)]\times[\mathbf{t}_U(2)\to\mathbf{t}_U(3)] \arrow[r, "((\pi_2,\pi_3),p_2)"] \arrow[d, "((\pi_1,\pi_2),p_1)"'] & U^2 \vdash \mathbf{t}_U(1)\to\mathbf{t}_U(2) \arrow[d, "(\pi_1,!)"] \\
U^2 \vdash \mathbf{t}_U(1)\to\mathbf{t}_U(2) \arrow[r, "(\pi_2,!)"'] & U \vdash \mathbf{1}
\end{tikzcd}$$

Second, to prove that compositions are associative one needs to form an object $\mathbb{U}_3$. It is the type $U^4 \vdash ([\mathbf{t}_U(1)\to\mathbf{t}_U(2)]\times[\mathbf{t}_U(2)\to\mathbf{t}_U(3)])\times[\mathbf{t}_U(3)\to\mathbf{t}_U(4)]$ of course. ■

Apart from $(\lambda_{TT},\mathcal{U})$, there are another two obvious functors. $\{\_\vdash\mathbf{1}\} : \mathcal{C}\longrightarrow\mathcal{T}$ embedds $\mathcal{C}$ in $\mathcal{T}$ by sending a context Γ to $\Gamma\vdash\mathbf{1}$. The functor $\{\_[\_]\}$ on the other hand is basically the context extension:

$$\begin{array}{rcl}
\mathcal{T} & \xrightarrow{\{_[_]\}} & \mathcal{C} \\
\Gamma\vdash A & \longmapsto & \Gamma, x:A \\
(r,f):(\Gamma\vdash A)\to(\Delta\vdash B) & \longmapsto & (r,fx):(\Gamma,x:A)\to(\Delta,y:B)
\end{array}$$

Lemma 6.2 $(\lambda_{TT},\mathcal{U}) \dashv \{\_\vdash\mathbf{1}\} \dashv \{\_[\_]\}$ *form a D-category.*

It is a direct consequence of lemma 6.2 that if we apply the functor $\{\_[\_]\}$ to $\mathbb{U}$ we get an internal category $\dot{\mathbb{U}}$ in $\mathcal{C}$.

Proposition 6.3 *In the diagram below, we have (i)* $\{\_\vdash\mathbf{1}\};(\lambda_{TT},\mathcal{U})$ *is the identity functor on* $\mathcal{C}$*; (ii) the small square is a pullback; (iii)* H *is a cartesian full and faithful functor.*

$$\begin{tikzcd}
{[\dot{\mathbb{U}}]} \arrow[r, hook] \arrow[d, "dom_{\dot{\mathbb{U}}}"'] \arrow[rr, bend left, "H"] & {[\mathbb{U}]} \arrow[d, "dom_{\mathbb{U}}"] & \mathcal{T} \arrow[d, "(\lambda_{TT},\mathcal{U})"] \\
\mathcal{C} \arrow[r, hook, "_\vdash\mathbf{1}"'] & \mathcal{T} \arrow[r, "(\lambda_{TT},\mathcal{U})"'] & \mathcal{C}
\end{tikzcd}$$

Proof: (i) is obvious. (ii) Suppose $\Gamma\in\mathcal{C}$. Objects in $[\Gamma,\dot{\mathbb{U}}]$ are realizations $\Gamma\to[x:U,y:\mathbf{1}]$, which are essentially objects $\Gamma\vdash a:U$; morphisms from $(a,\star):\Gamma\to[x:U,y:\mathbf{1}]$ to $(b,\star):\Gamma\to[x:U,y:\mathbf{1}]$ are essentially sequents $\Gamma\vdash f:\mathbf{t}_U(a)\to\mathbf{t}_U(b)$. On the other hand, objects in $[\Gamma \vdash \mathbf{1}, \mathbb{U}]$ are $(r, f) : (\Gamma \vdash \mathbf{1}) \to (U \vdash \mathbf{1})$, which are also essentially sequents $\Gamma \vdash a : U$; similarly morphisms in $[\Gamma \vdash \mathbf{1}, \mathbb{U}]$ correspond to $\Gamma \vdash f : \mathbf{t}_U(a) \to \mathbf{t}_U(b)$ as well. (iii) The functor H sends an object $(a, \star) : \Gamma \to [x : U, y : \mathbf{1}]$ onto $\Gamma \vdash \mathbf{t}_U(a)$ in $(\lambda_{TT}, \mathcal{U})^{-1}(\Gamma)$. It is clearly full and faithful. ■

Corollary 6.4 $\dot{\mathbb{U}}$ *is an internal full subcategory of* $\mathcal{C}$*, relative to* $\mathcal{D}$*, induced by the display map* $[x : U, y : \mathbf{t}_U(x)] \xrightarrow{(x)} [x : U]$.

The proof is again routine. Since $\mathcal{C}/\mathcal{D} \xrightarrow{cod} \mathcal{C}$ is isomorphic to $\mathcal{T} \xrightarrow{(\lambda_{TT}, \mathcal{U})} \mathcal{C}$, it makes sense to say that $\dot{\mathbb{U}}$ is an internal full subcategory of $\mathcal{T} \xrightarrow{(\lambda_{TT}, \mathcal{U})} \mathcal{C}$.

$$\begin{tikzcd}
{[\dot{\mathbb{U}}]} \arrow[r, hook] \arrow[dr, "dom_{\dot{\mathbb{U}}}"'] & \mathcal{C}/\mathcal{D} \arrow[r, "\simeq"] \arrow[d, "cod"'] & \mathcal{T} \arrow[dl, "{(\lambda_{TT}, \mathcal{U})}"] \\
 & \mathcal{C} &
\end{tikzcd}$$

7 The Variable Convention and Internal Codability

One of the immediate questions one has to address in formulating an object language in *any* logical framework is how to treat the variables in an encoding of the object language. In both Martin-Löf's logical framework and *ELF*, as well as in λ_{TT}, those variables are identified with the variables in the frameworks. There doesn't seem to be any other way. But how do we justify it? Even if we believe that the question is not entirely relevant, we should however at least face these two questions: (i) what is the model theoretical implication of the variable convention? and (ii) is our logical framework good with respect to the answer we give to (i)?

Suppose $\mathcal{E}$ is an encoding of $\mathcal{L}$ in λ_{TT} and U is a universe declared in $\mathcal{E}$. Suppose further that $\mathcal{E}$ constitutes an internal category of $\mathcal{C}$, the base category of $(\lambda_{TT}, \mathcal{E})$. If $\Gamma \vdash_{\Omega,\Sigma} a : U$, then a 'variable' of a is an indeterminant of an object in the fibre $[\Gamma, \mathbb{U}]$. If we are to identify a variable in $[\Gamma, \mathbb{U}]$ with a variable in λ_{TT}, then $[\Gamma, \mathbb{U}]$ should be a *full* subcategory of $(\lambda_{TT}, \mathcal{E})^{-1}(\Gamma)$. We therefore conclude that the categorical counterpart of the variable convention is this: if an object language is represented by a small category in $\mathcal{C}$, then it is an *internal full subcategory*. This means that there is a full and faithful cartesian functor H as shown in the diagram below.

$$\begin{tikzcd}
{[\dot{\mathbb{U}}]} \arrow[r, "H"] \arrow[dr, "dom_{\dot{\mathbb{U}}}"'] & \mathcal{T} \arrow[d, "{(\lambda_{TT}, \mathcal{E})}"] \\
 & \mathcal{C}
\end{tikzcd}$$

So to some extent the variable convention corresponds to internal codability. Type theoretically, this phenomenon can be interpreted as saying that the variable convention (and the equality convention) forces a kind of conservative extensionality relation between $\mathcal{E}$ and λ_{TT}.

The result in section 6 can now be interpreted as saying that in λ_{TT} the universe declaration mechanism conforms to the variable convention.

8 Structures of Universes

The internal categories discussed in section 6 are the backbones. When defining a typed calculus in λ_{TT}, we force those internal categories to have specific structures. Let's look at the first two examples given in section 5.

Lemma 8.1 *The equations in Th_λ simply says that the following diagrams are pullbacks in the fibre category of the fibration $(\lambda_{TT}, \mathcal{E}_\lambda)$, where $\otimes$ is for $\Rightarrow$ (or $\wedge$) and $\dot{\otimes}$ is $\rightarrow$ (or $\times$); $!$ is $\Gamma \vdash \lambda x{:}A.\star : A \rightarrow \mathbf{1}$ for appropriate Γ and A; m is $\lambda w{:}\mathbf{t}_U(1) \rightarrow \mathbf{t}_U(2).\mathsf{abs}(1)(2)w$ when $\otimes$ is $\Rightarrow$; it is $\lambda w{:}\mathbf{t}_U(1) \times \mathbf{t}_U(2).\mathsf{pair}(1)(2)w$ when $\otimes$ is $\wedge$.*

$$\begin{tikzcd}
\vdash \mathbf{t}_U(\Diamond) \arrow[r, "(\Diamond, Id)"] \arrow[d, "((),!)"'] & U \vdash \mathbf{t}_U(1) \arrow[d, "(1_U,!)"] \\
\vdash \mathbf{1} \arrow[r, "(\Diamond,!)"'] & U \vdash \mathbf{1}
\end{tikzcd}
\qquad
\begin{tikzcd}
U^2 \vdash \mathbf{t}_U(1) \dot{\otimes} \mathbf{t}_U(2) \arrow[r, "(\otimes, m)"] \arrow[d, "(1_{U^2},!)"'] & U \vdash \mathbf{t}_U(1) \arrow[d, "(1_U,!)"] \\
U^2 \vdash \mathbf{1} \arrow[r, "(\otimes,!)"'] & U \vdash \mathbf{1}
\end{tikzcd}$$

Proof: Routine. Suppose we have a commuting diagram like this

$$\begin{tikzcd}
\Gamma \vdash A \arrow[r, "(c, f)"] \arrow[d, "((a,b),!)"'] & U \vdash \mathbf{t}_U(1) \arrow[d, "(1_U,!)"] \\
U^2 \vdash \mathbf{1} \arrow[r, "(\Rightarrow,!)"'] & U \vdash \mathbf{1}
\end{tikzcd}$$

Clearly $c = (a, b)$ and $\Gamma \vdash f : A \rightarrow [\mathbf{t}_U(a) \rightarrow \mathbf{t}_U(b)]$. Let $((a,b), ?)$ be the mediating morphism. Then

$$\mathsf{abs}_{a,b}(?(x)) = f(x).$$

Therefore

$$?xz = \mathsf{app}_{a,b}(\mathsf{abs}_{a,b}(?x, z)) = \mathsf{app}_{a,b}(fx, z).$$

It follows that

$$? = \lambda x{:}A.\lambda z{:}\mathbf{t}_U(a).\mathsf{app}_{a,b}(fx, z).$$

The proofs for the other cases are similar. ■

The categorical implication of lemma 8.1 is obvious once we notice the following facts: (i) $\mathbb{U}$ is the internal category induced by $(U \vdash \mathbf{t}_U(1)) \xrightarrow{(1_U,!)} (U \vdash \mathbf{1})$; (ii) both $\vdash \mathbf{t}_U(\Diamond)$ and $\vdash \mathbf{1}$ are terminal object; (iii) $U^2 \vdash \mathbf{1}$ is the product of $U \vdash \mathbf{1}$ and $U \vdash \mathbf{1}$; (iv) $(U^2 \vdash \mathbf{t}_U(i)) \xrightarrow{(1_{U^2},!)} (U^2 \vdash \mathbf{1})$ is the pulling-back of $(U \vdash \mathbf{t}_U(1)) \xrightarrow{(1_U,!)} (U \vdash \mathbf{1})$ along the projection $(U^2 \vdash \mathbf{1}) \xrightarrow{\pi_i} (U \vdash \mathbf{1})$ and (v) $U^2 \vdash \mathbf{t}_U(1) \times \mathbf{t}_U(2)$ (or $U^2 \vdash \mathbf{t}_U(1) \to \mathbf{t}_U(2)$) is the product (or exponential) over U^2. Transport these results over to $\mathcal{C}$ using lemma 6.2, we conclude that the internal full subcategory $\dot{\mathbb{U}}$ has explicit cartesian closed structures.

In the case of $\mathcal{E}_{PL}$, this property holds for both $\mathbb{U}$ and $\mathbb{K}$. In addition we have

Lemma 8.2 *In the encoding $\mathcal{E}_{PL}$ of higher order λ-calculus, the following is a pullback in the category $\mathcal{T}_{PL}$ for any $\vdash_{\Omega;\Sigma} \kappa : K$.*

$$\begin{tikzcd}
\mathbf{t}_K(\kappa) \to U \vdash \Pi x{:}\mathbf{t}_K(\kappa).\mathbf{t}_U(1(x)) \arrow[r, "(\dot{\forall}_{\kappa,1}, \mathsf{ABS}_{\kappa,1})"] \arrow[d, "(Id,!)"'] & U \vdash \mathbf{t}_U(1) \arrow[d, "(1_U,!)"] \\
\mathbf{t}_K(\kappa) \to U \vdash \mathbf{1} \arrow[r, "(\dot{\forall}_{\kappa,1},!)"'] & U \vdash \mathbf{1}
\end{tikzcd}$$

Proof: Suppose the following diagram commutes.

$$\begin{tikzcd}
\Gamma \vdash A \arrow[r, "(G,t)"] \arrow[d, "(F,!)"'] & U \vdash \mathbf{t}_U(1) \arrow[d, "(Id,!)"] \\
\mathbf{t}_K(\kappa) \to U \vdash \mathbf{1} \arrow[r, "(\dot{\forall}_{\kappa,1},!)"'] & U \vdash \mathbf{1}
\end{tikzcd}$$

Clearly $G = \dot{\forall}_{\kappa,1}$. Assume the mediating morphism is $(F, ?)$. For a variable x of type A, we have

$$\mathsf{ABS}_{\kappa,F}(?(x)) = t(x) : \mathbf{t}_U(\dot{\forall}_{\kappa,F}).$$

So

$$?(x) = \mathsf{APP}_{\kappa,F}(\mathsf{ABS}_{\kappa,F}(?(x))) = \mathsf{APP}_{\kappa,F}(t(x)).$$

Therefore

$$? = \lambda x{:}A.\mathsf{APP}_{\kappa,F}(t(x)).$$

This completes the proof. ■

Again it is easy to see that (i) $\mathbf{t}_K(\kappa) \to U \vdash \mathbf{1}$ is the exponential of $\mathbf{t}_K(\kappa) \vdash \mathbf{1}$ and $U \vdash \mathbf{1}$; (ii) $(\mathbf{t}_K(\kappa) \to U, \mathbf{t}_K \vdash \mathbf{t}_U(1(2))) \xrightarrow{(\pi_1,!)} (\mathbf{t}_K(\kappa) \to U \vdash \mathbf{1})$ is the pulling-back of $(U \vdash \mathbf{t}_U(1)) \xrightarrow{(1_U,!)} (U \vdash \mathbf{1})$ and (iii) $(\mathbf{t}_K(\kappa) \to U \vdash \Pi x{:}\mathbf{t}_K(\kappa).\mathbf{t}_U(1(x))) \xrightarrow{(Id,!)} (\mathbf{t}_K(\kappa) \to U \vdash \mathbf{1})$ is obtained by applying the right adjoint to $(\mathbf{t}_K(\kappa) \to U, \mathbf{t}_K \vdash \mathbf{t}_U(1(2))) \xrightarrow{(\pi_1,!)} (\mathbf{t}_K(\kappa) \to U \vdash \mathbf{1})$. In summary, lemma 8.2 says that the internal category $\dot{\mathbb{U}}$ has explicit products over types lifted from a 'kind'.

We can generalize what is embodied in the above examples: If a collection of certain entities in an object language is represented by a universe in λ_{TT}, then to code up the operators associated with the collection is to equip the internal full subcategory, induced by the universe, with *explicit* categorical structures.

9 Syntactic Adequacy and Internal Definability

This section reveals the close tie between syntactic adequacy, a linguistic notion, and internal definability, a semantic notion. This is done by examining the encodings defined in section 5. We should point out that in order to demonstrate the relationship, we need not know if the encodings are syntactically adequate, nor do we have to know whether λ_{TT} has good proof theoretical properties or not.

9.1 Syntactic and Closed Term Models

In this section there are two kinds of models we are interested in: the *closed* term models and the *syntactic* term models. In the former we consider only closed terms whereas in the latter we consider terms with free variables. For instance the closed term model of a simply typed λ-calculus is a cartesian closed category; on the other hand the syntactic term model of the same calculus is a fibration $\mathcal{L}_\lambda : \mathcal{T}_\lambda \longrightarrow \mathcal{C}_\lambda$. Here $\mathcal{C}_\lambda$ is the category of contexts and their realizations; a morphism over $\Gamma \in \mathcal{C}_\lambda$ is an equivalence class of terms $\Gamma \vdash f : A \to B$.

In most semantic investigations of non-dependent typed calculi we focus on closed term models. In the case of logical frameworks however syntactic term models are more relevant. This is because encodings in a logical framework deals not only with closed terms but also with terms with free variables. This is manifested in the fact that syntactic adequacy theorems are statements about sequents instead of closed terms.

In the case of dependent typed calculi however we always talk about syntactic term models.

9.2 Syntactic Adequacy

The encoding $\mathcal{E}_\lambda$ provides us with a translation from a type A in the simply typed λ-calculus to a closed term $\hat{A}$ of type U in λ_{TT} and from a judgement

$$x_1 : A_1, \ldots, x_n : A_n \vdash t : A \tag{1}$$

in the simply typed λ-calculus to a judgement

$$x_1 : \mathbf{t}_U(\hat{A}_1), \ldots, x_n : \mathbf{t}_U(\hat{A}_n) \vdash_{\Omega;\Sigma} \hat{t} : \mathbf{t}_U(\hat{A}), \tag{2}$$

in λ_{TT}. Let's see some sample translations. Suppose A, B and $f : A \to B$ are constants. Their translations are declared in the signature of $\mathcal{E}_\lambda$. The following are some of the translations:

$$\begin{array}{rcl}
A \to B & \longmapsto & \vdash_{\Omega;\Sigma} \hat{A} \Rightarrow \hat{B} : U \\
\vdash f : A \to B & \longmapsto & \vdash_{\Omega;\Sigma} \hat{f} : \mathbf{t}_U(\hat{A} \Rightarrow \hat{B}) \\
x : A \vdash fx & \longmapsto & x : \mathbf{t}_U(\hat{A}) \vdash_{\Omega;\Sigma} \mathsf{app}_{\hat{A},\hat{B}}(\hat{f}, x) : \mathbf{t}_U(\hat{B}) \\
\vdash \lambda x{:}A.fx : A \to B & \longmapsto & \vdash_{\Omega;\Sigma} \mathsf{abs}_{\hat{A},\hat{B}}(\lambda x{:}\mathbf{t}_U(\hat{A}).\mathsf{app}_{\hat{A},\hat{B}}(\hat{f}, x)) : \mathbf{t}_U(\hat{A} \Rightarrow \hat{B}) \\
\vdash \lambda x{:}A.fx = f : A \to B & \longmapsto & \vdash_{\Omega;\Sigma} \mathsf{abs}_{\hat{A},\hat{B}}(\lambda x{:}\mathbf{t}_U(\hat{A}).\mathsf{app}_{\hat{A},\hat{B}}(\hat{f}, x)) = \hat{f} : \mathbf{t}_U(\hat{A} \Rightarrow \hat{B}).
\end{array}$$

The correctness of the encoding is given by the so-called syntactic adequacy theorem, which says that

1. there is a bijective correspondence between the types in the simply typed λ-calculus and judgements of the form $\vdash_{\Omega;\Sigma} \hat{A} : U$ in λ_{TT};
2. there is a bijection between the judgements of the form (1) and the judgements of the form (2); strictly speaking, $\hat{t}$ should be an equivalence class, where the equivalence relation is induced by the $\beta\eta$-equality of λ_{TT} excluding the definitional equations in Th_λ; the same remark applies to $\hat{A}$ in the above case.
3. $x_1 : A_1, \ldots, x_n : A_n \vdash s = t : A$ if and only if $x_1 : \mathbf{t}_U(\hat{A}_1), \ldots, x_n : \mathbf{t}_U(\hat{A}_n) \vdash_{\Omega;\Sigma} \hat{s} =_{Th_\lambda} \hat{t} : \mathbf{t}_U(\hat{A})$; here $=_{Th_\lambda}$ is induced by the equational contexts Th_λ as well as the $\beta\eta$-equality.
4. the translation is compositional, i.e. $\widehat{a[b/x]} = \hat{a}[\hat{b}/x]$.

We leave the translation of higher order polymorphic λ-calculus to $\mathcal{E}_{PL}$ and the statement of syntactic adequacy to the reader. The translation from Martin-Löf's type theory to $\mathcal{E}_{ML}$ is defined as follows, where $\widehat{[x_1 : A_1, \ldots, x_n : A_n]}$ stands for $x_1 : \mathbf{t}(\hat{A}_1), \ldots, x_n : \mathbf{t}(\hat{A}_n)$,

$$\begin{array}{rcl}
\Gamma \vdash \Pi x{:}A.B & \longmapsto & \hat{\Gamma} \vdash_{\Omega;\Sigma} \pi(\hat{A}, \lambda x{:}\mathbf{t}(\hat{A}).\hat{B}) \\
\Gamma \vdash \Sigma x{:}A.B & \longmapsto & \hat{\Gamma} \vdash_{\Omega;\Sigma} \sigma(\hat{A}, \lambda x{:}\mathbf{t}(\hat{A}).\hat{B}) \\
\Gamma \vdash f(a) : B[a/x] & \longmapsto & \hat{\Gamma} \vdash_{\Omega;\Sigma} \bullet(\hat{A}, \lambda x{:}\mathbf{t}(\hat{A}).\hat{B}, \hat{f}, \hat{a}) : \mathbf{t}(\widehat{B[a/x]}) \\
\Gamma \vdash \lambda x{:}A.b : \Pi x{:}A.B & \longmapsto & \hat{\Gamma} \vdash_{\Omega;\Sigma} \Lambda(\hat{A}, \lambda x{:}\mathbf{t}(\hat{A}).\hat{B}, \lambda x{:}\mathbf{t}(\hat{A}).\hat{b}) : \pi(\hat{A}, \lambda x{:}\mathbf{t}(\hat{A}).\hat{B}) \\
\Gamma \vdash \langle a, b \rangle : \Sigma x{:}A.B & \longmapsto & \hat{\Gamma} \vdash_{\Omega;\Sigma} \mathbf{pair}(\hat{A}, \lambda x{:}\mathbf{t}(\hat{A}).\hat{B}, \hat{a}, \hat{b}) : \sigma(\hat{A}, \lambda x{:}\mathbf{t}(\hat{A}).\hat{B}) \\
\Gamma \vdash \pi_1 c : A & \longmapsto & \hat{\Gamma} \vdash_{\Omega;\Sigma} \mathbf{P}_1(\hat{A}, \lambda x{:}\mathbf{t}(\hat{A}).\hat{B}, \hat{c}) : \mathbf{t}(\hat{A}) \\
\Gamma \vdash \pi_2 c : B[\pi_1 c/x] & \longmapsto & \hat{\Gamma} \vdash_{\Omega;\Sigma} \mathbf{P}_2(\hat{A}, \lambda x{:}\mathbf{t}(\hat{A}).\hat{B}, \hat{c}) : \mathbf{t}(\hat{B}[\mathbf{P}_1(\hat{A}, \lambda x{:}\mathbf{t}(\hat{A}).\hat{B}, \hat{c})/x])
\end{array}$$

The syntactic adequacy consists of the following statements:

1. there is a bijective correspondence between judgements $\Gamma \vdash A$ and judgements $\hat{\Gamma} \vdash_{\Omega;\Sigma} \hat{A}$;

2. there is a bijective correspondence between judgements $\Gamma \vdash a : A$ and judgements $\hat{\Gamma} \vdash_{\Omega;\Sigma} \hat{a} : \mathbf{t}(\hat{A})$;

3. $\Gamma \vdash a = b : A$ if and only if $\hat{\Gamma} \vdash_{\Omega;\Sigma} \hat{a} =_{Th_{ML}} \hat{b} : \mathbf{t}(\hat{A})$;

4. the translation is compositional.

For more discussions on syntactic adequacy, see [HHP87, HHP92].

9.3 Categorical Characterizations of Syntactic Adequacy

Let $\mathcal{L}_\lambda : \mathcal{T}_\lambda \longrightarrow \mathcal{C}_\lambda$ and $\mathcal{L}^\lambda$ be the syntactic term model and the closed term model of the simply typed λ-calculus respectively.

Proposition 9.1 *If $\mathcal{E}_\lambda$ is syntactically adequate, then $\mathcal{L}_\lambda$ and $\mathcal{L}^\lambda$ are internally definable in $(\lambda_{TT}, \mathcal{E}_\lambda)$ by $\mathbb{U}$.*

Proof: (i) First let's explain the functors $\mathcal{I}$ and H in the following diagram.

$$\begin{tikzcd}
\mathcal{T}_\lambda \arrow[r, hook, "H"] \arrow[d, "\mathcal{L}_\lambda"'] & {[\dot{\mathbb{U}}]} \arrow[r, hook] \arrow[d, "dom_{\dot{\mathbb{U}}}"'] & {[\mathbb{U}]} \arrow[d, "dom_{\mathbb{U}}"] \\
\mathcal{C}_\lambda \arrow[r, hook, "\mathcal{I}"'] & \mathcal{C} \arrow[r, hook, "_ \vdash \mathbf{1}"'] & \mathcal{T}
\end{tikzcd}$$

$\mathcal{I}$ sends a context $[x_1 : A_1, \ldots, x_n : A_n]$ in $\mathcal{C}_\lambda$ onto a context $[x_1 : \mathbf{t}_U(\hat{A}_1), \ldots, x_n : \mathbf{t}_U(\hat{A}_n)]$, the existence of which is guaranteed by 1. A morphism $r : [x_1 : A_1, \ldots, x_n : A_n] \longrightarrow [y_1 : B_1, \ldots, y_m : B_m]$ consists of

$$\begin{array}{rcl}
x_1 : A_1, \ldots, x_n : A_n & \vdash & b_1 : B_1, \\
 & \vdots & \\
x_1 : A_1, \ldots, x_n : A_n & \vdash & b_m : B_m.
\end{array}$$

2 says that we can define $\mathcal{I}(r)$ to be the realization consisting of

$$\begin{array}{rcl}
x_1 : \mathbf{t}_U(\hat{A}_1), \ldots, x_n : \mathbf{t}_U(\hat{A}_n) & \vdash_{\Omega;\Sigma} & \hat{b}_1 : \mathbf{t}_U(\hat{B}_1), \\
 & \vdots & \\
x_1 : \mathbf{t}_U(\hat{A}_1), \ldots, x_n : \mathbf{t}_U(\hat{A}_n) & \vdash_{\Omega;\Sigma} & \hat{b_m} : \mathbf{t}_U(\hat{B_m}).
\end{array}$$

3 means that $\mathcal{I}$ so defined is a full and faithful functor.

Next let's define the functor H. An object in the fibre $\mathcal{L}_\lambda^{-1}([x_1 : A_1, \ldots, x_n : A_n])$ is a type A in the simply typed λ-calculus. Define $H(A)$ to be the realization $(\hat{A}, \star) : [x_1 : \mathbf{t}_U(\hat{A}_1), \ldots, x_n : \mathbf{t}_U(\hat{A}_n)] \longrightarrow [x : U, y : \mathbf{1}]$, which is essentially the sequent $[x_1 : \mathbf{t}_U(\hat{A}_1), \ldots, x_n : \mathbf{t}_U(\hat{A}_n)] \vdash_{\Omega;\Sigma} \hat{A} : U$. Clearly $[x_1 : \mathbf{t}_U(\hat{A}_1), \ldots, x_n : \mathbf{t}_U(\hat{A}_n)] \vdash_{\Omega;\Sigma} \hat{A} : U$ iff $\vdash_{\Omega;\Sigma} \hat{A} : U$ since $\hat{A}$ is the translation of A and therefore a closed term of U. But by syntactic adequacy, there is a correspondence between types in the simply typed λ-calculus and sequents like $\vdash_{\Omega;\Sigma} \hat{A} : U$. So H establishes a bijection between the objects of $\mathcal{L}_\lambda^{-1}([x_1 : A_1, \ldots, x_n : A_n])$ and those in $dom_{\dot{\mathbb{U}}}^{-1}([x_1 : \mathbf{t}_U(\hat{A}_1), \ldots, x_n : \mathbf{t}_U(\hat{A}_n)])$. A morphism in $\mathcal{L}_\lambda^{-1}([x_1 : A_1, \ldots, x_n : A_n])$ is a term $[x_1 : A_1, \ldots, x_n : A_n] \vdash f : A \to B$. The syntactic adequacy implies that it is in bijective correspondence with $[x_1 : \mathbf{t}_U(\hat{A}_1), \ldots, x_n : \mathbf{t}_U(\hat{A}_n)] \vdash_{\Omega;\Sigma} \hat{f} : \mathbf{t}_U(\hat{A} \Rightarrow \hat{B})$. But the latter is in bijective correpondence with $[x_1 : \mathbf{t}_U(\hat{A}_1), \ldots, x_n : \mathbf{t}_U(\hat{A}_n)] \vdash_{\Omega;\Sigma} \lambda w{:}\mathbf{t}_U(\hat{A}).\mathsf{app}_{\hat{A},\hat{B}}(\hat{f}, w) : \mathbf{t}_U(\hat{A}) \to \mathbf{t}_U(\hat{B})$. Let $H(f)$ be $\lambda w{:}\mathbf{t}_U(\hat{A}).\mathsf{app}_{\hat{A},\hat{B}}(\hat{f}, w)$. Clearly H is full and faithful.

Finally observe that (a) the composition $\mathcal{I}; \{\_ \vdash \mathbf{1}\}$ is an embedding and (b) the right square in the above diagram is a pullback. For the proof of (b), one only has to notice that the proof of proposition 6.3 is not affected if new definitional equalities are added. Conclude that $\mathcal{L}_\lambda \multimapdotinv (\lambda_{TT}, \mathcal{E}_\lambda)$.

(ii) For the closed term model $\mathcal{L}^\lambda$, a similar argument shows that the left diagram below is a pullback,

$$
\begin{tikzcd}
\mathcal{L}^\lambda \arrow[r, hook, "G"] \arrow[d, "!"'] & {[\dot{\mathbb{U}}]} \arrow[r, hook] \arrow[d, "dom_{\dot{\mathbb{U}}}"'] & {[\mathbb{U}]} \arrow[d, "dom_{\mathbb{U}}"] \\
\star \arrow[r, hook, "{[]}"'] & \mathcal{C} \arrow[r, hook, "\_ \vdash \mathbf{1}"'] & \mathcal{T}
\end{tikzcd}
$$

where $\star$ is the terminal category and $[]$ sends the only object to the empty context. ■

For higher order polymorphic λ-calculus, there are two syntactic term models: that of kinds and constructors $\mathcal{L}_{PL}^K : \mathcal{T}_{PL}^K \longrightarrow \mathcal{C}_{PL}^K$ and that of types and terms $\mathcal{L}_{PL}^T : \mathcal{T}_{PL}^T \longrightarrow \mathcal{C}_{PL}^T$. Let $\mathcal{L}^{PL}$ be the closed term model.

Proposition 9.2 *If the encoding $\mathcal{E}_{PL}$ is syntactically adequate, $\mathcal{L}_{PL}^K$, $\mathcal{L}_{PL}^T$ and $\mathcal{L}^{PL}$ are all internally definable in $(\lambda_{TT}, \mathcal{E}_{PL})$ by $\mathcal{U}$ or $\mathcal{K}$.*

Proof: (i) By proposition 9.1, both $\mathcal{L}_{PL}^K$ and $\mathcal{L}_{PL}^T$ are internally definable in $(\lambda_{TT}, \mathcal{E}_{PL})$.

$$
\begin{tikzcd}
\mathcal{T}_{PL}^K \arrow[r, hook, "H_K"] \arrow[d, "\mathcal{L}_{PL}^K"'] & {[\dot{\mathbb{K}}]} \arrow[r, hook] \arrow[d, "dom_{\dot{\mathbb{K}}}"'] & {[\mathbb{K}]} \arrow[d, "dom_{\mathbb{U}}"] \\
\mathcal{C}_{PL}^K \arrow[r, hook, "\mathcal{I}_K"'] & \mathcal{C} \arrow[r, hook, "\_ \vdash \mathbf{1}"'] & \mathcal{T}
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathcal{T}_{PL}^T \arrow[r, hook, "H_T"] \arrow[d, "\mathcal{L}_{PL}^T"'] & {[\dot{\mathbb{U}}]} \arrow[r, hook] \arrow[d, "dom_{\dot{\mathbb{U}}}"'] & {[\mathbb{U}]} \arrow[d, "dom_{\mathbb{U}}"] \\
\mathcal{C}_{PL}^T \arrow[r, hook, "\mathcal{I}_T"'] & \mathcal{C} \arrow[r, hook, "\_ \vdash \mathbf{1}"'] & \mathcal{T}
\end{tikzcd}
$$

(ii) Proposition 9.1 essentially prescribes a full embedding functor $J : \mathcal{C}^{PL} \longrightarrow \mathcal{T}$, the reason being that $\mathcal{C}^{PL} \cong [\vdash \mathbf{1}, \mathbb{K}]$ and $[\vdash \mathbf{1}, \mathbb{K}]$ can be fully embedded in $\mathcal{T}$ by mapping $\vdash k : K$ to $\mathbf{t}_K(k) \vdash \mathbf{1}$ (remember that in $[\vdash \mathbf{1}, \mathbb{K}]$ a morphism from $\vdash \kappa : K$ to $\vdash \kappa' : K$ is precisely a term $\vdash f : \mathbf{t}_K(\kappa) \rightarrow \mathbf{t}_K(\kappa')$).

$$
\begin{tikzcd}
\mathcal{L}^{PL} \arrow[rr, "J"] \arrow[d, "\mathcal{L}^{PL}"'] & & {[\mathbb{U}]} \arrow[d, "dom_{\mathbb{U}}"] \\
\mathcal{C}^{PL} \arrow[r, "\cong"] & {[\vdash \mathbf{1}, \mathbb{K}]} \arrow[r] & \mathcal{T}
\end{tikzcd}
$$

By syntactic adequacy, $X : L \vdash A : Type$ corresponds to $X : \mathbf{t}_K(\hat{L}) \vdash \hat{A} : U$ and $X : L \vdash f : A \rightarrow B$ to $X : \mathbf{t}_K(\hat{L}) \vdash \hat{f} : \mathbf{t}_U(\hat{A} \Rightarrow \hat{B})$. Define the action of J to be as follows:

$$
\begin{array}{rcl}
\mathcal{T}^{PL} & \xrightarrow{J} & [\mathbb{U}] \\
X : L \vdash A : Type & \longmapsto & (\hat{A}, !) : (\mathbf{t}_K(\hat{L}) \vdash \mathbf{1}) \rightarrow \mathbb{U}_0 \\
X : L \vdash f : A \rightarrow B & \longmapsto & ((\hat{A}, \hat{B}), \lambda w{:}\mathbf{t}_U(\hat{A}).\mathsf{app}_{\hat{A},\hat{B}}(\hat{f}, w)) : (\mathbf{t}_K(\hat{L}) \vdash \mathbf{1}) \rightarrow \mathbb{U}_1.
\end{array}
$$

The verification of the rest is routine. ■

Proposition 9.3 *If the encoding $\mathcal{E}_{ML}$ is syntactically adequate, then the term model is internally definable in $(\lambda_{TT}, \mathcal{E}_{ML})$ by Set.*

Proof: In the diagram below, the functor $\mathcal{I}$ is defined as in the previous two cases. The only difference is that types are dependent in this case.

$$
\begin{tikzcd}
\mathcal{T}_{ML} \arrow[r, "M"] \arrow[d, "\mathcal{L}_{ML}"'] & {[\dot{Set}]} \arrow[r, hook] \arrow[d, "dom_{\dot{Set}}"'] & {[Set]} \arrow[d, "dom_{Set}"] \\
\mathcal{C}_{ML} \arrow[r, "\mathcal{I}"'] & \mathcal{C} \arrow[r, hook, "\_ \vdash \mathbf{1}"'] & \mathcal{T}
\end{tikzcd}
$$

The functor H is defined as follows:

$$
\begin{array}{rcl}
\mathcal{T}_{ML} & \xrightarrow{M} & [\dot{Set}] \\
\Gamma \vdash A & \longmapsto & (\hat{A}, !) : \hat{\Gamma} \longrightarrow \dot{Set}_0 \\
\Gamma \vdash f : A \rightarrow B & \longmapsto & (\hat{f}, !) : \hat{\Gamma} \longrightarrow \dot{Set}_1
\end{array}
$$

where $\hat{\Gamma} \vdash_{\Omega;\Sigma} \hat{A} : Set$ and $\hat{\Gamma} \vdash_{\Omega;\Sigma} \hat{f} : \mathbf{t}(\pi(\hat{A}, \lambda w{:}\mathbf{t}(\hat{A}).\hat{B}))$. Syntactic adequacy implies that H so defined is full and faithful. We need to show that M preserves left and right adjoints along display maps in $\mathcal{C}_{ML}$. According to the translation we have given, we have

$$
\begin{array}{rcl}
\mathcal{T}_{ML} & \xrightarrow{M} & [\dot{Set}] \\
\Gamma, x : A \vdash B & \longrightarrow & \hat{\Gamma}, x : \mathbf{t}(\hat{A}) \vdash_{\Omega;\Sigma} \hat{B} : Set \\
\Gamma \vdash \Pi x{:}A.B & \longrightarrow & \hat{\Gamma} \vdash_{\Omega;\Sigma} \pi(\hat{A}, \lambda x{:}\mathbf{t}(\hat{A}).\hat{B}) : Set.
\end{array}
$$

The following establishes a bijective correspondence between terms $\hat{\Gamma} \vdash_{\Omega;\Sigma} f : X \to \mathbf{t}(\pi(\hat{A}, \lambda x{:}\mathbf{t}(\hat{A}).\hat{B}))$ and terms $\hat{\Gamma}, x : \mathbf{t}(\hat{A}) \vdash_{\Omega;\Sigma} g : X \to \mathbf{t}(\hat{B})$,

$$\dfrac{\dfrac{\dfrac{\dfrac{\hat{\Gamma} \vdash_{\Omega;\Sigma} f : X \to \mathbf{t}(\pi(\hat{A}, \lambda x{:}\mathbf{t}(\hat{A}).\hat{B}))}{\hat{\Gamma}, y : X \vdash_{\Omega;\Sigma} fy : \mathbf{t}(\pi(\hat{A}, \lambda x{:}\mathbf{t}(\hat{A}).\hat{B}))}}{\hat{\Gamma}, y : X, x : \mathbf{t}(\hat{A}) \vdash_{\Omega;\Sigma} \bullet(\hat{A}, \lambda x{:}\mathbf{t}(\hat{A}).\hat{B}, fy, x) : \mathbf{t}(\hat{B})}}{\hat{\Gamma}, x : \mathbf{t}(\hat{A}), y : X \vdash_{\Omega;\Sigma} \bullet(\hat{A}, \lambda x{:}\mathbf{t}(\hat{A}).\hat{B}, fy, x) : \mathbf{t}(\hat{B})}}{\hat{\Gamma}, x : \mathbf{t}(\hat{A}) \vdash_{\Omega;\Sigma} \lambda y{:}X. \bullet (\hat{A}, \lambda x{:}\mathbf{t}(\hat{A}).\hat{B}, fy, x) : X \to \mathbf{t}(\hat{B})}$$

and its inverse

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\hat{\Gamma}, x : \mathbf{t}(\hat{A}) \vdash_{\Omega;\Sigma} f : X \to \mathbf{t}(\hat{B})}{\hat{\Gamma}, x : \mathbf{t}(\hat{A}), y : X \vdash_{\Omega;\Sigma} fy : \mathbf{t}(\hat{B})}}{\hat{\Gamma}, y : X, x : \mathbf{t}(\hat{A}) \vdash_{\Omega;\Sigma} fy : \mathbf{t}(\hat{B})}}{\hat{\Gamma}, y : X \vdash_{\Omega;\Sigma} \lambda x{:}\mathbf{t}(\hat{A}).fy : \Pi x{:}\mathbf{t}(\hat{A}).\mathbf{t}(\hat{B})}}{\hat{\Gamma}, y : X \vdash_{\Omega;\Sigma} \Lambda(\hat{A}, \lambda x{:}\mathbf{t}(\hat{A}).\hat{B}, \lambda x{:}\mathbf{t}(\hat{A}).fy) : \mathbf{t}(\pi(\hat{A}, \lambda x{:}\mathbf{t}(\hat{A}).\mathbf{t}(\hat{B})))}}{\hat{\Gamma} \vdash_{\Omega;\Sigma} \lambda y{:}X.\Lambda(\hat{A}, \lambda x{:}\mathbf{t}(\hat{A}).\hat{B}, \lambda x{:}\mathbf{t}(\hat{A}).fy) : X \to \mathbf{t}(\pi(\hat{A}, \lambda x{:}\mathbf{t}(\hat{A}).\mathbf{t}(\hat{B})))}.$$

The rest is trivial. ■

In all the three examples we have assumed that the number of constant types is finite. Let's look at an example where there is an infinite number of constant types. Forgetting the encoding of constant types, $\mathcal{E}_\lambda$ as it stands is an encoding of simply typed λ-calculus with an infinite number of constant types. The idea is to assume a bijective correspondence between the constant types and variables of type U. A type A in the simply typed λ-calculus is now coded by $\sigma_1 : U, \ldots, \sigma_n : U \vdash_{\Omega;\Sigma} \hat{A} : U$ where $\sigma_1, \ldots, \sigma_n$ correspond to the constant types occurred in A. It can be routinely checked that proposition 9.1 still holds for $\mathcal{L}_\lambda$. For instance the functor $\mathcal{C}_\lambda \xrightarrow{\mathcal{I}} \mathcal{C}$ sends $[x_1 : A_1, \ldots, x_n : A_n]$ onto $[\sigma_1 : U, \ldots, \sigma_m : U, x_1 : \mathbf{t}(\hat{A}_1), \ldots, x_n : \mathbf{t}(\hat{A}_n)]$ where $\sigma_1, \ldots, \sigma_n$ correspond to the constant types occurred in $A_1, \ldots, A_n$ and A.

The technique used above can be applied to the other two examples.

The general phenomenon seem to be this: If the encoding $\mathcal{E}_\mathcal{L}$ of a typed calculus $\mathcal{L}$ in λ_{TT} is syntactically adequate then, in a canonical way (determined by the translation of $\mathcal{L}$ into $\mathcal{E}_\mathcal{L}$) $\mathcal{L}$ is internally definable in $(\lambda_{TT}, \mathcal{E}_\mathcal{L})$ (or, waving hand, $\mathcal{L} \multimapdotinv \lambda_{TT}$). In other words, a faithful encoding of $\mathcal{L}$ defines a frame language such that $\mathcal{L}$ is an internal typed calculus definable in the frame language via the encoding of $\mathcal{L}$.

So to formulate an object language in λ_{TT} is to identify a frame language which contains somehow an internal version of the represented language. The richer the object language, the richer the frame calculus. A correct identification of the frame language involves the build-up of an internal definability relationship between the object language and the frame language.

10 The Notion of Models

We now turn our attention to semantics. There are two levels of semantics. The model theory of λ_{TT} has been well-established, see for instance [HP89]. A categorical model consists of a category $\mathcal{B}$ with finite products and a collection $\mathbf{D}$ of display maps. The fibration $\mathcal{B}/\mathbf{D} \xrightarrow{cod} \mathcal{B}$ must be a fibred cartesian closed category and is complete relative to $\mathbf{D}$. A generic judgement $x : U \vdash_{\Omega;\Sigma} \mathbf{t}_U(x)$ must be interpreted as a display map. There is not much to say.

The proper model theory of logical frameworks is about the meanings of encodings and how to relate the model theory of an object language to that of a framework. It is this aspect of semantics we have been trying to understand. Suppose $\mathcal{L}$ is a typed calculus and $\mathcal{E} = (\Omega, \Sigma, Th)$ is an encoding of $\mathcal{L}$ in λ_{TT}. There are two questions one can ask:

1. Given an interpretation of λ_{TT}, what does the interpretation of $\mathcal{E}$ make up? Arguably, we are only interested in those interpretations whose restriction to $\mathcal{E}$ constitutes an internal model of $\mathcal{L}$.
2. Given a model of $\mathcal{L}$, can we fully embed it into a model of λ_{TT}?

Definition 10.1 *A (categorical) interpretation of $\mathcal{E} = (\Omega, \Sigma, Th)$ consists of a sound interpretation of λ_{TT} and an interpretation of constants in Ω and Σ. We say the interpretation is* sound *if it validates all the definitional equations in Th. In this case we also say that it is an* interpretation of the encoding $\mathcal{E}$. *We say an interpretation of $\mathcal{E}$ is* good *if it gives rise to an internal model of $\mathcal{L}$. We say the encoding $\mathcal{E}$ is* semantically adequate *if every sound interpretation of $\mathcal{E}$ is good.*

Proposition 10.2 *Both $\mathcal{E}_\lambda$ and $\mathcal{E}_{PL}$ are semantically adequate.*

Proof: Lemma 8.1 (8.2) holds for any interpretation of $\mathcal{E}_\lambda$ ($\mathcal{E}_{PL}$). The results follow immediately. ∎

Suppose $[\![\_]\!]$ is an interpretation of λ_{TT}. Usually there are many ways of extending $[\![\_]\!]$ to $[\![\_]\!]_\Omega^\Sigma$ so that it also interprets the constants in both Ω and Σ. If the encoding $\mathcal{E}$ is syntactically adequate then the translation $\hat{\_} : \mathcal{L} \longrightarrow \mathcal{E}$ gives rise to a map

$$\text{models of } \mathcal{E} \quad \longmapsto \quad \text{models of } \mathcal{L}.$$

We say that the translation $\hat{\_} : \mathcal{L} \longrightarrow \mathcal{E}$ is *semantically complete* if the above map is surjective.

Proposition 10.3 *The encodings in examples 5.1 and 5.2 are semantically complete.*

Proof: The semantic adequacy for polymorphism is essentially the main result of [Pit87]. The result for the simply typed λ-calculus can be proved similarly. ∎

11 Conclusion and Related Work

This paper attempts to give a model theoretical account of logical frameworks. We argue that the real issues in the semantics of logical frameworks are about the meanings of encodings and how they are related to the model theories of the object languages. We propose a logical framework that is well-suited to model theoretical analysis. Examples are given to show how syntactic conditions force term models to possess certain categorical properties. One important point taken in this paper is that a logical framework is a setting for defining frame languages. According to this view, to code up an object language is to search for a frame language within which the object language is internally definable. In the following table, the right column contains the semantic counterparts of the corresponding syntactic notions in the left column.

proof theoretical notion	**model theoretical notion**
universes	generic objects
encodings of object calculi	internal categories
variable convention	internal codability
constants other than universes	explicit structures on internal categories
syntactic adequacy	internal definability

By characterizing the categorical properties of λ_{TT} and some encodings within it, we hope we have pointed out a sensible way of semantic investigations. It is our personal opinion that the general ideas expressed in this paper are helpful when one designs a logical framework or tries to code up an object language.

It seems that the categorical explanations make perfect sense when the object calculi are constructive. For a non-constructive language $\mathcal{L}$, say Church's simple type theory, what we code up in a logical framework is the 'image' of $\mathcal{L}$ in a 'constructive mirror'. The point is that a demonstration of "something is a proof" is always constructive. It is this aspect of $\mathcal{L}$ that is captured in an encoding. So in this case, the categorical properties are characterizations of this 'image'.

The development in [Pow92] is at a more abstract level. There the author seeks a categorical setting within which semantical constructions of typed calculi can be carried out. There are two issues: one is to search for a semantic framework; the other is to study type theory semantically within the framework. The dichotomy corresponds to the one we advocate in this paper. The difference is that we concentrate on a particular typed calculus and its semantic description. Our approach emphasizes the importance of internal categories in semantics of logical frameworks.

The language ELF^+ investigated in [Gar92] is a refined version of ELF. In addition to the kind $Type$, ELF^+ introduces two new kinds: $Judge$ and $Sort$. The basic idea is that in an encoding of an object language, the basic judgements should be coded by inhabitants of $Judge$ whereas sorts correspond to objects of $Sort$. The purpose is to achieve a close correspondence between a language and its representation in ELF^+. We can modify λ_{TT} along this line. For instance the universe declarations can be broken into two parts: one contains universes that encode judgements; the other consists of universes which represent sorts.

[Gar92] contains an observation similar to that in section 9.3. Because of the presence of *Judge* and *Sort*, the author is able to prove that adequate encodings of a class of logics are essentially the same as the logics themselves when both are viewed as indexed categories. This 'isomorphism' phenomenon is implicit in our definition of internal definability. The nice thing about our approach is that it brings out the internal categorical aspects of this phenomenon.

12 Acknowledgement

The material in this paper is taken from the author's Ph.D thesis ([Fu92]). My supervisor David Rydeheard has been very helpful all the way over the last two years. The examiners Peter Aczel and Martin Hyland have raised some interesting points and made quite a few comments.

During the preparation of this paper, the author was supported by the following person, organizations and project: Yu-Gang Bau, the British Council, the State Education Commission of China and the CLICS-II project.

References

[B̀85] J. Bènabou. Fibred categories and the foundation of naive category theory. *Journal of Symbolic Logic*, 50(1):10–37, March 1985.

[BB85] C. Böhm and A. Berarducci. Automatic synthesis of typed λ-program on term algebras. *Theoretical Computer Science*, 39:135–154, 1985.

[CH88] T. Coquand and G. Huet. The calculus of constructions. *Information and Computation*, 76(2/3):95–120, 1988.

[dB80] N. de Bruijn. A survey of the project AUTOMATH. In J. Seldin and J. Hindley, editors, *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus and Formalism*, pages 589–606. Academic Press, 1980.

[ea86] R. Constable et al. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice-Hall, 1986.

[Ehr88] T. Ehrhard. A categorical semantics of constructions. In *Proceedings of the Third Symposium on Logic in Computer Science*, pages 264–273. IEEE Computer Science Press, July 1988.

[Fu92] Yuxi Fu. *Some Semantic Issues In Type Theory.* PhD thesis, Department of Computer Science, University of Manchester, May 1992.

[Gar92] P. Gardner. *Representing Logics in Type Theory.* PhD thesis, LFCS, University of Edinburgh, July 1992.

[HHP87] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. In *Proceedings of the Second Symposium on Logic in Computer Science*, pages 194–204. IEEE Computer Science Press, June 1987.

[HHP92] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 1992.

[HMM90] R. Harper, J. Mitchell, and E. Moggi. Higher-order modules and the phase distinction. In *17th ACM Symposium on Principles of Programming Languages*, pages 341–354, San Francisco, January 1990. ACM.

[HP91] G. Huet and G. Plotkin, editors. *Logical Frameworks.* Cambridge University Press, 1991.

[HP89] J. Hyland and A. Pitts. The theory of constructions: Categorical semantics and topos-theoretic models. In J. Gray and A. Scedrov, editors, *Categories in Computer Science and Logic*, pages 137–199, Boulder, June 89. A.M.S.

[HPJ91] G. Huet, G. Plotkin, and C. Jones, editors. *Logical Frameworks*, 1991.

[HRR90] M. Hyland, E. Robinson, and G. Rosolini. The discrete objects in the effective topos. *Proc. London Mathematical Society*, 60(3):1–36, 1990.

[Jac91] B. Jacobs. *Categorical Type Theory.* PhD thesis, Catholic University of Nijmegen, 1991.

[Luo90] Z. Luo. *An Extended Calculus of Constructions.* PhD thesis, LFCS, University of Edinburgh, Edinburgh, July 1990.

[Luo91] Z. Luo. A higher-order calculus and theory abstraction. *Information and Computation*, 90(1):107–137, January 1991.

[Luo92] Z. Luo. A unifying theory of dependent types: the schematic approach. In A. Nerode and M. Taitslin, editors, *Logical Foundations of Computer Science–Tver '92, LNCS 620.* Springer, 1992.

[MA88] N. Mendler and P. Aczel. The notion of a framework and a framework for LTC. In *Proceedings of the Third Symposium on Logic in Computer Science*, pages 392–399. IEEE Computer Science Press, July 1988.

[Mog89] E. Moggi. A category-theoretic account of program modules. In D. Pitt, D. Rydeheard, P. Dybier, A. Pitts, and A. Poigné, editors, *Category Theory and Computer Science, LNCS 389*, pages 101–117, Manchester, September 1989. Springer Verlag.

[Mog91] E. Moggi. A category theoretical account of program modules. *Mathematical Structures in Computer Science*, 1:103–139, 1991.

[NPS90] B. Nordström, K. Petersson, and J. Smith. *Programming in Martin-Löf's Type Theory –an introduction*, volume 7 of *International series of monographs on computer science*. Oxford University Press, Oxford, 1990.

[Pav90] D. Pavlović. *Predicates and Fibrations*. PhD thesis, University of Utrecht, 1990.

[Pit87] A. Pitts. Polymorphism is set-theoretic, constructively. In D. Pitt, A.Poigné, and D. Rydeheard, editors, *Category Theory and Computer Science, LNCS 283*, pages 12–39, Edinburgh, U.K., September 1987. Springer Verlag.

[Pow92] J. Power. *Why tricategories?*, June 1992.

[Tay86] P. Taylor. *Recursive Domains, Indexed Category Theory and Polymorphism*. PhD thesis, University of Cambridge, 1986.

[Wra89] G. Wraith. A note on categorical datatypes. In D. Pitt, Rydeheard, P. Dybier, A. Pitts, and A. Poigné, editors, *Category Theory and Computer Science, LNCS 389*, pages 213–223, Manchester, September 1989. Springer Verlag.