*Math. Struct. in Comp. Science* (1997), *vol.* 7, *pp.* 1–47 Copyright © Cambridge University Press

# Categorical properties of logical frameworks

YUXI FU[†][‡]

*Department of Computer Science, University of Manchester, Manchester M13 9PL, England*

*Received 8 March 1993; revised 18 October 1995*

In this paper we define a logical framework, called $\lambda_{TT}$, that is well suited for semantic analysis. We introduce the notion of a fibration $\mathscr{L}_1 : \mathscr{T}_1 \longrightarrow \mathscr{C}_1$ being internally definable[§] in a fibration $\mathscr{L}_2 : \mathscr{T}_2 \longrightarrow \mathscr{C}_2$. This notion amounts to distinguishing an internal category $\mathsf{L}$ in $\mathscr{L}_2$ and relating $\mathscr{L}_1$ to the externalization of $\mathsf{L}$ through a pullback. When both $\mathscr{L}_1$ and $\mathscr{L}_2$ are term models of typed calculi $\mathscr{L}^1$ and $\mathscr{L}^2$, respectively, we say that $\mathscr{L}^1$ is an internal typed calculus definable in the frame language $\mathscr{L}^2$. We will show by examples that if an object language is adequately represented in $\lambda_{TT}$, then it is an internal typed calculus definable in the frame language $\lambda_{TT}$. These examples also show a general phenomenon: if the term model of an object language has categorical structure $S$, then an adequate encoding of the language in $\lambda_{TT}$ imposes an explicit internal categorical structure $S$ in the term model of $\lambda_{TT}$ and the two structures are related via internal definability. Our categorical investigation of logical frameworks indicates a sensible model theory of encodings.

## 1. Introduction

The notion of logical framework first appeared in Martin-Löf's work (Nordström *et al.* 1990). It is meant to be a simple language with which his type theory can be precisely defined. The idea is picked up and extended in Harper *et al.* (1987, 1993). 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 Harper *et al.* (1987, 1993), 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 that declare constant terms. In other words, signatures are constant contexts in which one can declare both constant objects and constant constructors. When coding up an object language, one first defines an appropriate signature so that logical operators become constant objects or constructors. An important feature of *ELF* is that rules in object languages are presented as objects in the framework. Logical derivations then amount to function applications.

[†] During the preparation of this paper, the author was supported by the CLICS-II project. In the later stage of the preparation, he was under the 863 Project.

[‡] Author's present address: Department of Computer Science, Shanghai Jiao Tong University, 1954 Hua Shan Road, Shanghai 200030, People's Republic of China.

[§] The definability as used in this paper should not be confused with Bénabou's 'definability' (Bénabou 1985).

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.

The core of Martin-Löf's logical framework and that of *ELF* are the same, but their emphases are different. Martin-Löf sees his logical framework as a foundational language upon which his constructive type 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 type theories are equational theories in the underlying calculus. On the other hand, the motivation in Harper *et al.* (1987, 1993) 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 internal representations of the definitional equalities 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.

A prominent question a designer of a logical framework must address is 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. One of the observations in this paper is that this variable identification has serious impact on model theory.

The proof theoretical and pragmatical aspects of logical frameworks have been extensively studied (Huet and Plotkin 1991; Huet and Plotkin 1993; Nordström *et al.* 1992). Their model theory, however, has not yet been paid as much attention as it deserves†. This paper sets out to remedy this. First, however, we need to make clear what we mean by a logical framework in this paper. A logical framework is a typed calculus upon (or in) which typed calculi are defined. Putting this differently, the theory of logical frameworks is about internal type theory. This unifies the views taken in Harper *et al.* (1987, 1993) and Nordström *et al.* (1990), and, arguably, also covers other more complex languages. The various well-known coding techniques (Böhm and Berarducci; Wraith 1989) belong to internal type theory. This viewpoint is semantically motivated. What internal categories are to an ambient category, object languages are to a logical framework.

In this paper we introduce the notions of internal definability, internal typed calculi and frame languages. Examples are given to show the potential usefulness of these definitions. 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 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.

† For those familiar with Simpson (1992), we mention that the logical framework considered in that paper can only encode consequence relations, not proof theories, of logics. The models described in *loc.cit.* are inappropriate for logical frameworks *à la* Martin-Löf.

Section 2 fixes some notations that will be employed later. Section 3 gives the definitions central to later development. Section 4 is an undemanding introduction to our logical framework $\lambda_{TT}$. Some sample encodings are given in Section 5. Section 6 details the fact that $\lambda_{TT}$ has a built-in mechanism to generate internal full subcategories, whose relationship to the variable convention is also explained. Sections 7 and 8 reveal some further categorical properties of $\lambda_{TT}$. Section 9 outlines the idea of model theory for encodings. Finally, in Section 10, we take a brief look at some questions in terms of internal definability.

## 2. Preliminaries

The technical definitions given in this section are needed to describe the categorical properties of $\lambda_{TT}$. The material is standard; see Bénabou (1985), Pavlović (1990), Barr and Wells (1990), and Jacobs (1991) for more on the theory of fibration and Hyland *et al.* (1990) and Pitts (1987) for relevant internal category theory.

Suppose $p : \mathscr{F} \longrightarrow \mathscr{B}$ is a functor. The morphism $Y \xrightarrow{f} X$ in $\mathscr{F}$ is said to be a *cartesian lifting* of $J \xrightarrow{u} I$ in $\mathscr{B}$ if

(i) $pf = u$ and

(ii) for any $Z \xrightarrow{g} X$ in $\mathscr{F}$ and $pZ \xrightarrow{m} J$ satisfying $pg = m;u$, there is a unique morphism $Z \xrightarrow{\phi} Y$ with $p\phi = m$ and $g = \phi;f$.

The functor $p$ is a *fibration* if for every morphism $J \xrightarrow{u} I$ in the *base* category $\mathscr{B}$ and every object $X$ in the *fibre* category $\mathscr{F}$ 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 $\{\overline{u}\}_{J \xrightarrow{u} I \in \mathscr{B}}$ 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 $\mathscr{B}$. When this is the case, $p$ can be presented as an *indexed category* $\mathscr{P} : \mathscr{B}^{op} \longrightarrow \mathbf{Cat}$, which is a 'functor' from $\mathscr{B}^{op}$ to the 'category' of categories. Suppose $I$ is an object in the base category of a split fibration $p : \mathscr{F} \longrightarrow \mathscr{B}$. The category $p^{-1}(I)$, the *fibre over* $I$, is the one whose objects and morphisms are those of $\mathscr{F}$ that are mapped by $p$ onto $I$ and $Id_I$, respectively. For each morphism $J \xrightarrow{u} I$ in $\mathscr{B}$, we have a *reindexing functor* $p^{-1}(I) \xrightarrow{u^\star} p^{-1}(J)$. The construction tells us how to transform a split fibration into an indexed category. The opposite construction is the so-called *Grothendieck construction* (Barr and Wells 1990). A split fibration has a *fibred cartesian closed structure* if each fibre has a fibred cartesian closed structure and the reindexing functors preserve the structure on the nose. A split fibration $p : \mathscr{F} \longrightarrow \mathscr{B}$ has *A-indexed products*, where $A$ is an object in $\mathscr{B}$, if for each $I$ in $\mathscr{B}$, the reindexing functor over $I \times A \rightarrow I$ has right adjoint $\Pi_I$ and these right adjoints satisfy the *Beck–Chevalley condition*: for each $J \xrightarrow{u} I$ in $\mathscr{B}$, we have $(u \times Id_A)^\star;\Pi_J = \Pi_I;u^\star$. A *cartesian functor* $H$ from $p : \mathscr{F} \longrightarrow \mathscr{B}$ to $q : \mathscr{E} \longrightarrow \mathscr{B}$ is a functor $H : \mathscr{F} \longrightarrow \mathscr{E}$ such that $p = H;q$, and $H$ sends cartesian liftings to cartesian liftings. A cartesian functor is full (faithful) if and only if its fibrewise functors are full (faithful). More generally, a *cartesian map* from $p : \mathscr{F} \longrightarrow \mathscr{B}$ to $q : \mathscr{G} \longrightarrow \mathscr{C}$ consists of two functors $H : \mathscr{F} \longrightarrow \mathscr{G}$ and $K : \mathscr{B} \longrightarrow \mathscr{C}$ such that $H;q = p;K$, and $H$ sends $p$-cartesian liftings to $q$-cartesian liftings. If both $p$

and $q$ are split, the cartesian map $(H, K)$ is *strict* if $H$ sends the chosen cartesian liftings to the chosen cartesian liftings. An object $U$ in a locally small category $\mathscr{B}$ determines a *representable functor* $\mathscr{B}[\_, U] : \mathscr{B}^{op} \longrightarrow \mathbf{Set}$ that sends $A \in \mathscr{B}$ to $\mathscr{B}[A, U]$ and $A \xrightarrow{f} B$ to $\mathscr{B}[f, U] : \mathscr{B}[B, U] \longrightarrow \mathscr{B}[A, U]$.

Let $\mathscr{F}$ and $\mathscr{B}$ be two categories. A *D-category* (Ehrhard 1988) is given by three functors $p, G : \mathscr{F} \longrightarrow \mathscr{B}$ and $I : \mathscr{B} \longrightarrow \mathscr{F}$ such that

(i) $p$ is a fibration;
(ii) $I$ is the fibred terminal object functor of $p$ (so $I; p = Id_{\mathscr{B}}$, $p \dashv I$ and $I$ is full embedding), and
(iii) $I \dashv G$.

We say that the $D$-category is split if $p$ is split.

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

$$\begin{tikzcd}
\mathsf{C}_2 \arrow[r, shift left=2, "\Pi_0"] \arrow[r, "\gamma" description] \arrow[r, shift right=2, "\Pi_1"'] & \mathsf{C}_1 \arrow[r, shift left=2, "d_0"] \arrow[r, shift right=2, "d_1"'] & \mathsf{C}_0 \arrow[l, "\mathsf{id}" description]
\end{tikzcd}$$

These morphisms in $\mathscr{B}$ must satisfy the well-known conditions:

(i) $\Pi_0$ ($\Pi_1$) is the pullback of $d_0$ ($d_1$) along $d_1$ ($d_0$);
(ii) the (co)domain of the composition of a pair of composable arrows is the (co)domain of the first (second) arrow: $\gamma; d_0 = \Pi_0; d_0$ and $\gamma; d_1 = \Pi_1; d_1$;
(iii) $\mathsf{id}; d_0 = \mathsf{id}; d_1 = Id_{\mathsf{C}_0}$;
(iv) $\mathsf{id}$ is a unit for composition: $(\mathsf{id} \times_0 Id_{\mathsf{C}_1}); \gamma = \Pi_1$ and $(Id_{\mathsf{C}_1} \times_0 \mathsf{id}); \gamma = \Pi_0$; and
(v) composition is associative: $(\gamma \times_0 Id_{\mathsf{C}_1}); \gamma = (Id_{\mathsf{C}_1} \times_0 \gamma); \gamma$.

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

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

The functor $dom_{\mathsf{C}}$ sends $(I \xrightarrow{f} J, I \xrightarrow{m} \mathsf{C}_1)$ onto $I \xrightarrow{f} J$. The canonical cartesian lifting over $I \xrightarrow{f} J$ with respect to $J \xrightarrow{\beta} \mathsf{C}_0$ is $(I \xrightarrow{f} J, I \xrightarrow{f} J \xrightarrow{\beta} \mathsf{C}_0 \xrightarrow{\mathsf{id}} \mathsf{C}_1)$. The corresponding indexed category is denoted by $[\_, \mathsf{C}] : \mathscr{B}^{op} \longrightarrow \mathbf{Cat}$. A morphism

$(I \xrightarrow{\alpha} \mathsf{C}_0) \longrightarrow (I \xrightarrow{\beta} \mathsf{C}_0)$ in $[I, \mathsf{C}]$ is a morphism $I \xrightarrow{m} \mathsf{C}_1$ in $\mathscr{B}$ such that the following diagram commutes.

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

Let $\mathbf{D}$ be a collection of morphisms, called *display maps* (Taylor 1986; Hyland and Pitts 1989), of $\mathscr{B}$ such that for any $A \xrightarrow{d} X$ in $\mathbf{D}$ and any morphism $Y \xrightarrow{f} X$, there is a (unique) morphism $A[f] \xrightarrow{d[f]} Y$ in $\mathbf{D}$ rendering the diagram

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

a pullback. Let $\mathscr{B}/\mathbf{D}$ be the full subcategory of the arrow category $\mathscr{B}^{\rightarrow}$ determined by $\mathbf{D}$. The definition of $\mathbf{D}$ simply says that $\mathscr{B}/\mathbf{D} \xrightarrow{cod} \mathscr{B}$ is a (cloven) fibration. The fibration $\mathscr{B}/\mathbf{D} \xrightarrow{cod} \mathscr{B}$ is split if it is cloven and for each display map $d$, we have $d[Id] = d$ and $d[f;g] = d[g][f]$.

Suppose $\mathscr{B}$ has finite products and $\mathbf{D}$ is a family of display maps. Let $V \xrightarrow{G} \mathsf{C}_0$ be in $\mathbf{D}$. We can transplant the well-known construction of an internal full subcategory in a locally cartesian closed category (Pitts 1987) to $\mathscr{B}$ if there are enough exponential structure in $\mathscr{B}/\mathbf{D} \xrightarrow{cod} \mathscr{B}$ (notice that $\mathscr{B}$ already has enough pullback structure guaranteed by $\mathbf{D}$). We will call the resulting $\mathsf{C}$ the internal category *induced* by $V \xrightarrow{G} \mathsf{C}_0$. For any $I \xrightarrow{a} \mathsf{C}_0$, we have a display map $G[a]$ that is the pulling-back of $G$ along $a$. We say that the $G$-induced internal category $\mathsf{C}$ is an *internal full subcategory* of $\mathscr{B}$ relative to $\mathbf{D}$ if

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

establishes a full embedding cartesian functor from $dom_{\mathsf{C}} : [\mathsf{C}] \longrightarrow \mathscr{B}$ to $\mathscr{B}/\mathbf{D} \xrightarrow{cod} \mathscr{B}$. We do not have room here to give a detailed explanation of this construction, but it will be given for the specific examples used in this paper.

We say that the internal category $\mathsf{C}$ induced by $V \xrightarrow{G} \mathsf{C}_0$ has *explicit cartesian closed (object) structure* if there are morphisms $\mathbf{1} \xrightarrow{\dot{\top}} \mathsf{C}_0$, $\mathsf{C}_0 \times \mathsf{C}_0 \xrightarrow{\dot{\times}} \mathsf{C}_0$ and $\mathsf{C}_0 \times \mathsf{C}_0 \xrightarrow{\dot{\to}} \mathsf{C}_0$ such that we have pullbacks as follows:

$$\begin{tikzcd}
\mathbf{1} \arrow[r] \arrow[d] & V \arrow[d, "G"] \\
\mathbf{1} \arrow[r, "\dot{\top}"'] & \mathsf{C}_0
\end{tikzcd}
\qquad
\begin{tikzcd}
\bullet \arrow[r] \arrow[d, "G_1 \times_{\mathsf{C}_0^2} G_2"] & V \arrow[d, "G"] \\
\mathsf{C}_0 \times \mathsf{C}_0 \arrow[r, "\dot{\times}"'] & \mathsf{C}_0
\end{tikzcd}
\qquad
\begin{tikzcd}
\bullet \arrow[r] \arrow[d, "G_1 \to_{\mathsf{C}_0^2} G_2"] & V \arrow[d, "G"] \\
\mathsf{C}_0 \times \mathsf{C}_0 \arrow[r, "\dot{\to}"'] & \mathsf{C}_0
\end{tikzcd}$$

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

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

where $G[ev]$ is the pulling-back of $G$ along $(A \to \mathsf{C}_0) \times A \xrightarrow{ev} \mathsf{C}_0$, and $\Pi_A$ is the right adjoint to the weakening functor over $(A \to \mathsf{C}_0) \times A \xrightarrow{\pi_1} (A \to \mathsf{C}_0)$.

If $\mathsf{C}$ has explicit cartesian closed structure ($A$-indexed products), the externalization of $\mathsf{C}$ has fibrewise cartesian closed structure and the reindexing functors preserve the structure on the nose (right adjoints to the weakening functors over morphisms $I \times A \xrightarrow{\pi_1} I$) and the *Beck–Chevalley condition* is satisfied.

## 3. Internal typed calculi and frame languages

The central theme of the theory of fibration is that an object in a base category should be regarded as a 'set' (and a morphism a 'function'). A fibration $p : \mathscr{E} \longrightarrow \mathscr{B}$ is a mathematical discipline in which one carries out mathematical activities with respect to the 'set theory' $\mathscr{B}$. Thinking 'set theoretically', it is important to distinguish between 'small' fibrations and 'non-small' ones. Roughly speaking, a 'small' fibration is obtained by externalizing some 'set-theoretical' gadget in $\mathscr{B}$. An externalization process transfers internal notions in $\mathscr{B}$ to external ones. The famous *Yoneda* lemma then acts a role similar to that of the extensionality axiom; it says that a 'small' fibration is essentially the same thing as a 'set' ('small' category *etc.*) in $\mathscr{B}$. So a 'small' fibration is a fibration that is 'definable' in the 'set theory' $\mathscr{B}$.

Given so many papers dealing with the relationship between typed calculi and fibrations, it is perhaps surprising that in type theory we have not yet seen the notion of 'internal definability' proposed, a notion that is fundamental in the theory of fibration. In this paper, we will show that this concept is potentially useful to type theory, especially to the study of logical frameworks.

Type theoretically, a strict cartesian map $(H, \mathscr{I})$ from a split fibration $\mathscr{L}_1 : \mathscr{T}_1 \longrightarrow \mathscr{C}_1$ to another $\mathscr{L}_2 : \mathscr{T}_2 \longrightarrow \mathscr{C}_2$ is a translation from the language $\mathscr{L}_1$ to the language $\mathscr{L}_2$. When both $H$ and $\mathscr{I}$ are embedding functors, the translation is faithful. In this case $\mathscr{L}_1$ can be seen as a sublanguage of $\mathscr{L}_2$. If, furthermore, both $H$ and $\mathscr{I}$ are full, $\mathscr{L}_2$ is a conservative extension of $\mathscr{L}_1$. A special case of this conservative extensionality relationship arises when the whole extension $\mathscr{L}_1 \hookrightarrow \mathscr{L}_2$ is determined by $\mathscr{C}_1 \hookrightarrow \mathscr{C}_2$, part of the extension. Categorically, this means that $\mathscr{L}_1$ is the pulling-back of $\mathscr{L}_2$ along $\mathscr{I}$.

Here it is most useful to combine the notion of 'internal definability' with that of conservative extensionality. So, typically, we have a strict cartesian map $(H, \mathscr{I})$ from a language $\mathscr{L} : \mathscr{T} \longrightarrow \mathscr{C}$, viewed as a fibration, to an externalization fibration $dom_{\mathsf{U}} : [\mathsf{U}] \longrightarrow \mathscr{D}$, where $\mathsf{U}$ is an internal category in $\mathscr{D}$; both $H$ and $\mathscr{I}$ are embedding functors. Because $dom_{\mathsf{U}}$ is completely determined by the internal category $\mathsf{U}$ in $\mathscr{D}$, it makes sense to require $\mathscr{L}$ be totally determined by $\mathscr{I} : \mathscr{C} \longrightarrow \mathscr{D}$ (that is, $\mathscr{L} : \mathscr{T} \longrightarrow \mathscr{C}$ is the pulling-back of $dom_{\mathsf{U}}$ along $\mathscr{I} : \mathscr{C} \longrightarrow \mathscr{D}$).

To introduce the central notion in this paper, it is helpful to give an auxiliary definition.

**Definition 3.1.** Suppose $p_1 : \mathscr{E}_1 \longrightarrow \mathscr{B}_1$ is a split fibration and $\mathscr{B}_2$ a category. Then $p_1$ is said to be *internally codable* in $\mathscr{B}_2$ by $\mathsf{L}$ via $\mathscr{I}$ if

(i) $\mathscr{I} : \mathscr{B}_1 \longrightarrow \mathscr{B}_2$ is an embedding functor;
(ii) $\mathsf{L}$ in $\mathscr{B}_2$ is an internal category, and
(iii) there is an embedding functor $H : \mathscr{E}_1 \hookrightarrow [\mathsf{L}]$ such that $(H, \mathscr{I})$ is a strict cartesian map from $p_1$ to $dom_{\mathsf{L}} : [\mathsf{L}] \longrightarrow \mathscr{B}_2$.

The fibration $p_1$ is said to be *internally definable* in $\mathscr{B}_2$ by $\mathsf{L}$ via $\mathscr{I}$ if in addition both $H$ and $\mathscr{I}$ are full and the following square is a pullback.

$$\begin{tikzcd}
\mathscr{E}_1 \arrow[r, "H"] \arrow[d, "p_1"'] & {[\mathsf{L}]} \arrow[d, "dom_{\mathsf{L}}"] \\
\mathscr{B}_1 \arrow[r, "\mathscr{I}"'] & \mathscr{B}_2
\end{tikzcd}$$

**Example 3.2.** Given a second-order $\lambda$-theory, we can construct its term model as a split fibration with fibred cartesian closed structures. In Pitts (1987), it is shown that this fibration is internally definable in a topos by an internal category equipped with explicit cartesian closed structure. □

**Example 3.3.** The higher-order polymorphic $\lambda$-calculus is a two level type system. At the first level we have kinds and constructors. The valid kinds are defined inductively as follows:

$$\begin{array}{rcl} & \Rightarrow & \top \text{ a kind} \\ & \Rightarrow & Type \text{ a kind} \\ K_1 \text{ kind}, \ K_2 \text{ kind} & \Rightarrow & K_1 \times K_2 \text{ kind} \\ K_1 \text{ kind}, \ K_2 \text{ kind} & \Rightarrow & K_1 \to K_2 \text{ kind.} \end{array}$$

A kinding context is a finite set of constructor variables with their specified kinds. Suppose $\Delta$ is $[X_1 : K_1, \ldots, X_n : K_n]$. Then $\Delta, X : K$ denotes the kinding context $[X_1 : K_1, \ldots, X_n : K_n, X : K]$. Given a kinding context, we can form valid constructors whose constructor variables are all declared in that context. These are the rules:

$$\begin{array}{rcl} \Delta \text{ a kinding context}, \ X : K \in \Delta & \Rightarrow & \Delta \vdash X : K \\ \Delta \text{ a kinding context} & \Rightarrow & \Delta \vdash \star_\top : \top \\ \Delta \vdash C_i : K_i \text{ for } i = 1, 2 & \Rightarrow & \Delta \vdash \langle C_1, C_2 \rangle : K_1 \times K_2 \\ \Delta \vdash C : K_1 \times K_2 & \Rightarrow & \Delta \vdash \pi_i C : K_i \text{ for } i = 1, 2 \\ \Delta, x : K_1 \vdash C : K_2 & \Rightarrow & \Delta \vdash \lambda x : K_1.C : K_1 \to K_2 \\ \Delta \vdash F : K_1 \to K_2, \ \Delta \vdash C : K & \Rightarrow & \Delta \vdash FC : K_2 \\ \Delta \text{ a kinding context} & \Rightarrow & \Delta \vdash \mathbf{1} : Type \\ \Delta \vdash A : Type, \ \Delta \vdash B : Type & \Rightarrow & \Delta \vdash A \times B : Type \\ \Delta \vdash A : Type, \ \Delta \vdash B : Type & \Rightarrow & \Delta \vdash A \to B : Type \\ \Delta, X : K \vdash A : Type & \Rightarrow & \Delta \vdash \forall X : K.A : Type. \end{array}$$

Constructors of kind $Type$ are called types.

A typing context $\Gamma$ is a set of statements of the form $x : A$ indicating that $A$ is the type of the free variable $x$. Let $\Gamma$ be $[x_1 : A_1, \ldots, x_n : A_n]$. Then $\Gamma, x : A$ denotes the typing context $[x_1 : A_1, \ldots, x_n : A_n, x : A]$. At the second level we have rules concerning objects. In these rules $\Gamma \vdash_\Delta a : A$ presupposes that $\Gamma$ is well defined under $\Delta$ and $\Delta \vdash A : Type$. Here a typing context $[x_1 : A_1, \ldots, x_n : A_n]$ is well defined under $\Delta$ if $\Delta \vdash A_1 : Type, \cdots, \Delta \vdash A_{n-1} : Type$ and $\Delta \vdash A_n : Type$. Here are the rules:

$$\begin{array}{rcl} \Gamma \text{ well defined under } \Delta, \ x : A \in \Gamma & \Rightarrow & \Gamma \vdash_\Delta x : A \\ \Gamma \text{ well defined under } \Delta & \Rightarrow & \Gamma \vdash_\Delta \star_{\mathbf{1}} : \mathbf{1} \\ \Gamma \vdash_\Delta a_i : A_i \text{ for } i = 1, 2 & \Rightarrow & \Gamma \vdash_\Delta \langle a_1, a_2 \rangle : A_1 \times A_2 \\ \Gamma \vdash_\Delta c : A_1 \times A_2 & \Rightarrow & \Gamma \vdash_\Delta \pi_i c : A_i \text{ for } i = 1, 2 \\ \Gamma, x : A \vdash_\Delta b : B & \Rightarrow & \Gamma \vdash_\Delta \lambda x : A.b : A \to B \end{array}$$

$$\begin{array}{rcl}
\Gamma \vdash_\Delta f : A{\to}B,\ \Gamma \vdash_\Delta a : A & \Rightarrow & \Gamma \vdash_\Delta fa : B \\
\Gamma \vdash_{\Delta,X:K} c : C,\ X \notin \Gamma & \Rightarrow & \Gamma \vdash_\Delta \Lambda X{:}K.c : \forall X{:}K.C \\
\Gamma \vdash_\Delta F : \forall X{:}K.C,\ \Delta \vdash D : K & \Rightarrow & \Gamma \vdash_\Delta FD : C[D/X],
\end{array}$$

where $X \notin \Gamma$ means that the constructor variable $X$ does not appear in any type in $\Gamma$. The set of equations is standard:

$$\begin{array}{rcl}
(\Lambda X{:}K.F)C & \overset{\beta^2}{=} & F[C/X] \\
\Lambda X{:}K.FX & \overset{\eta^2}{=} & F \\
(\lambda x{:}A.f)a & \overset{\beta}{=} & f[a/x] \\
\lambda x{:}A.fx & \overset{\eta}{=} & f \\
\pi_1\langle t_1, t_2\rangle & \overset{\pi_1}{=} & t_1 \\
\pi_2\langle t_1, t_2\rangle & \overset{\pi_2}{=} & t_2 \\
\langle \pi_1 t, \pi_2 t\rangle & \overset{\delta}{=} & t \\
u & \overset{!_\top}{=} & \star_\top \\
v & \overset{!_{\mathbf{1}}}{=} & \star_{\mathbf{1}},
\end{array}$$

where $u$ is of kind $\top$, $v$ is of type $\mathbf{1}$ and $t$, for example, is either of type $A_1{\times}A_2$ or of kind $K_1{\times}K_2$.

We can construct a split fibration $\mathscr{L}^\omega : \mathscr{T}^\omega \longrightarrow \mathscr{C}^\omega$ from the terms of the higher-order $\lambda$-calculus: the objects and morphisms of $\mathscr{C}^\omega$ are kinding contexts and realizations, respectively; an object over $\Delta$ is a typing judgement $\Delta \vdash A : Type$, and a morphism from $\Delta_1 \vdash A_1 : Type$ to $\Delta_2 \vdash A_2 : Type$ is a pair $(\Delta_1 \xrightarrow{r} \Delta_2, \vdash_{\Delta_1} f : A_1{\to}A_2[r])$, where $A_2[r]$ is obtained from $A_2$ by applying to $A_2$ the substitution prescribed by $r$. The functor $\mathscr{L}^\omega$ sends $(\Delta_1 \xrightarrow{r} \Delta_2, \vdash_{\Delta_1} f : A_1{\to}A_2[r])$ onto $\Delta_1 \xrightarrow{r} \Delta_2$. There is an internal category in $\mathscr{T}^\omega$ constructed as follows:

$$\begin{array}{rcl}
\mathsf{C}_0^\omega & \overset{\text{def}}{=} & X : Type \vdash \mathbf{1} : Type \\
\mathsf{C}_1^\omega & \overset{\text{def}}{=} & X, Y : Type \vdash X{\to}Y : Type \\
\mathsf{C}_2^\omega & \overset{\text{def}}{=} & X, Y, Z : Type \vdash (X{\to}Y){\times}(Y{\to}Z) : Type \\
d_0^\omega & \overset{\text{def}}{=} & ([X, Y : Type] \xrightarrow{(X)} [X : Type], \vdash_{X,Y:Type} \lambda f{:}X{\to}Y.\star_{\mathbf{1}} : (X{\to}Y){\to}\mathbf{1}) \\
d_1^\omega & \overset{\text{def}}{=} & ([X, Y : Type] \xrightarrow{(Y)} [Y : Type], \vdash_{X,Y:Type} \lambda f{:}X{\to}Y.\star_{\mathbf{1}} : (X{\to}Y){\to}\mathbf{1}) \\
\Pi_0^\omega & \overset{\text{def}}{=} & ([X, Y, Z : Type] \xrightarrow{(X,Y)} [X, Y : Type], \vdash_{X,Y,Z:Type} \lambda f{:}(X{\to}Y){\times}(Y{\to}Z).\pi_1 f) \\
\Pi_1^\omega & \overset{\text{def}}{=} & ([X, Y, Z : Type] \xrightarrow{(Y,Z)} [Y, Z : Type], \vdash_{X,Y,Z:Type} \lambda f{:}(X{\to}Y){\times}(Y{\to}Z).\pi_2 f) \\
\mathsf{id}^\omega & \overset{\text{def}}{=} & ([X : Type] \xrightarrow{(X,X)} [X, Y : Type], \vdash_{X:Type} \lambda z{:}\mathbf{1}.\lambda x{:}X.x) \\
\gamma^\omega & \overset{\text{def}}{=} & ([X, Y, Z : Type] \xrightarrow{(X,Z)} [X, Y : Type], \\
 & & \quad \vdash_{X,Y,Z:Type} \lambda f{:}(X{\to}Y){\times}(Y{\to}Z).\lambda x{:}X.(\pi_2 f)(\pi_1 f x)).
\end{array}$$

It can be readily shown that there is a pullback as shown by the diagram:

$$\begin{tikzcd}
\mathscr{T}^{\omega} \arrow[r, "H^{\omega}"] \arrow[d, "\mathscr{L}^{\omega}"'] & {[\mathsf{C}^{\omega}]} \arrow[d, "dom_{\mathsf{C}^{\omega}}"] \\
\mathscr{C}^{\omega} \arrow[r, "\top^{\omega}"'] & \mathscr{T}^{\omega}
\end{tikzcd}$$

where $\top^{\omega}$ sends $\Delta$ onto $\Delta \vdash \mathbf{1} : Type$. This means that $\mathscr{L}^{\omega}$ is internally definable in $\mathscr{T}^{\omega}$ by $\mathsf{C}^{\omega}$. □

Now we come to the central definition of this paper.

**Definition 3.4.** Suppose both fibrations $p_1 : \mathscr{E}_1 \longrightarrow \mathscr{B}_1$ and $p_2 : \mathscr{E}_2 \longrightarrow \mathscr{B}_2$ are split. Then $p_1$ is said to be *internally codable* (*definable*) in $p_2$ (notation $p_1 \lhd (\propto) p_2$) by $\mathsf{L}$ via $\mathscr{I}$ if

(i) $p_1$ is internally codable (definable) in $\mathscr{B}_2$ by $\mathsf{L}$ via $\mathscr{I}$, and
(ii) there is a full embedding functor $L : [\mathsf{L}] \longrightarrow \mathscr{E}_2$ that is cartesian from $dom_{\mathsf{L}}$ to $p_2$.

$$\begin{tikzcd}
\mathscr{E}_1 \arrow[r, "H"] \arrow[d, "p_1"'] & {[\mathsf{L}]} \arrow[r, "L"] \arrow[d, "dom_{\mathsf{L}}"'] & \mathscr{E}_2 \arrow[dl, "p_2"] \\
\mathscr{B}_1 \arrow[r, "\mathscr{I}"'] & \mathscr{B}_2 &
\end{tikzcd}$$

We call the cartesian map $(H, \mathscr{I})$ the *interpretation map.*

For typed $\lambda$-calculi $\mathscr{L}^1$ and $\mathscr{L}^2$, if their term models are split fibrations $\mathscr{L}_1$ and $\mathscr{L}_2$, respectively, then $\mathscr{L}^1$ is the *internal typed calculus* coded (defined) (notation $\mathscr{L}^1 \lhd (\propto) \mathscr{L}^2$) in $\mathscr{L}^2$ by $\mathsf{L}$ via $\mathscr{I}$ if $\mathscr{L}_1$ is internally coded (defined) in $\mathscr{L}_2$ by $\mathsf{L}$ via $\mathscr{I}$. When $\mathscr{L}^1 \propto \mathscr{L}^2$, we say that $\mathscr{L}^2$ is a *frame language* for $\mathscr{L}^1$.

The internal definability results to be established in Section 8 all possess some additional properties. Typically, the fibration $p_1$ has a fibred categorical structure $S$, say a fibred cartesian closed structure, and the internal category $\mathsf{L}$ is induced by a generic morphism and is equipped with explicit internal categorical structure $S$; besides the interpretation map $(H, \mathscr{I})$ preserves the fibred structure $S$. The structure $S$ is different in each application. To avoid the criticism of being imprecise, we refrain from giving a stronger definition.

**Example 3.5.** *(continued from Example 3.3)* There is a set $\mathbf{D}^{\omega}$ of display maps such that $cod^{\omega} \dashv I^{\omega} \dashv dom^{\omega} : \mathscr{T}^{\omega}/\mathbf{D}^{\omega} \longrightarrow \mathscr{T}^{\omega}$ form a $D$-category. The set $\mathbf{D}^{\omega}$ consists of all the maps in $\mathscr{T}^{\omega}$ of the form $(Id_{\Delta}, \vdash_{\Delta} \pi_1 : A \times B \rightarrow A) : (\Delta \vdash A \times B : Type) \longrightarrow (\Delta \vdash A : Type)$. The functor $I^{\omega}$ sends an object $\Delta \vdash A : Type$ in $\mathscr{T}^{\omega}$ onto the object $(Id_{\Delta}, \vdash_{\Delta} \pi_1 : A \times \top \rightarrow A)$ in $\mathscr{T}^{\omega}/\mathbf{D}^{\omega}$. It can be readily shown that $\mathsf{C}^{\omega}$ is an internal full subcategory of $cod^{\omega} : \mathscr{T}^{\omega}/\mathbf{D}^{\omega} \longrightarrow \mathscr{T}^{\omega}$. So $\mathscr{L}^{\omega}$ is internally definable by $\mathsf{C}^{\omega}$ in the fibration $cod^{\omega} : \mathscr{T}^{\omega}/\mathbf{D}^{\omega} \longrightarrow \mathscr{T}^{\omega}$. For a categorical account of this example and more, see Pitts (1987). $\square$

When the notion of internal definability is employed in this paper, $p_2$ is typically a $D$-category. Having a $D$-categorical structure allows us to switch between fibre category and base category. For example, if the base category of a $D$-category has an internal category, that internal category can be passed to the fibre category, and vice versa. We are going to prove a result about the relationship between the internal definability in the base category of a fibration and that in the fibre category of the same fibration. The next two lemmas will be used to prove the result.

**Lemma 3.6.** Suppose $I : \mathscr{B} \longrightarrow \mathscr{E}$ is a full embedding functor and $\mathsf{U}$ is an internal category in $\mathscr{B}$. If $I$ preserves finite limits, there is a pullback diagram as shown below and $(K, I)$ is a strict cartesian map from $dom_{\mathsf{U}}$ to $dom_{I\mathsf{U}}$.

$$\begin{tikzcd}
{[\mathsf{U}]} \arrow[r, "K"] \arrow[d, "dom_{\mathsf{U}}"'] & {[I\mathsf{U}]} \arrow[d, "dom_{I\mathsf{U}}"] \\
\mathscr{B} \arrow[r, "I"'] & \mathscr{E}
\end{tikzcd}$$

*Proof.* $K$ sends an object $X \xrightarrow{\alpha} \mathsf{U}_0$ onto $IX \xrightarrow{I\alpha} I\mathsf{U}_0$, a morphism $(X \xrightarrow{f} Y, X \xrightarrow{m} \mathsf{U}_1)$ onto $(IX \xrightarrow{If} IY, IX \xrightarrow{Im} I\mathsf{U}_1)$ and so on. $\square$

**Lemma 3.7.** Suppose the left exact functor $F : \mathscr{C} \longrightarrow \mathscr{D}$ is left adjoint to $G : \mathscr{D} \longrightarrow \mathscr{C}$. Let $\mathsf{U}$ be an internal category in $\mathscr{D}$. Then the externalization fibration of $dom_{G\mathsf{U}} : [G\mathsf{U}] \longrightarrow \mathscr{C}$ is the pullback of the fibration $dom_{\mathsf{U}} : [\mathsf{U}] \longrightarrow \mathscr{D}$ along $F : \mathscr{C} \longrightarrow \mathscr{D}$ and the pullback forms a strict cartesian map from $dom_{G\mathsf{U}}$ to $dom_{\mathsf{U}}$.

*Proof.* Because $G$ is a right adjoint and thus preserves limits, $G\mathsf{U}$ is an internal category in $\mathscr{C}$. First we need to define a functor $K : [G\mathsf{U}] \longrightarrow [\mathsf{U}]$ such that the following square commutes.

$$\begin{tikzcd}
{[G\mathsf{U}]} \arrow[r, "K"] \arrow[d, "dom_{G\mathsf{U}}"'] & {[\mathsf{U}]} \arrow[d, "dom_{\mathsf{U}}"] \\
\mathscr{C} \arrow[r, "F"'] & \mathscr{D}
\end{tikzcd}$$

For $X \xrightarrow{f} GY \in \mathscr{C}$ and $FX \xrightarrow{g} Y \in \mathscr{D}$, write $\hat{f}$ and $\bar{g}$, respectively, to denote their transposes across the adjunction. $K$ sends an object $\Gamma \xrightarrow{f} G\mathsf{U}_0$ to $F\Gamma \xrightarrow{\hat{f}} \mathsf{U}_0$ and a morphism $(\Gamma \xrightarrow{r} \Delta, \Gamma \xrightarrow{m} G\mathsf{U}_1)$ to $(F\Gamma \xrightarrow{Fr} F\Delta, F\Gamma \xrightarrow{\hat{m}} \mathsf{U}_1)$. It can be easily seen that $K$ preserves identities. Suppose $(\Gamma \xrightarrow{r} \Delta, \Gamma \xrightarrow{m} G\mathsf{U}_1) : \sigma \to \tau$ and $(\Delta \xrightarrow{s} \Theta, \Delta \xrightarrow{n} G\mathsf{U}_1) : \tau \to \delta$ are composable morphisms in $[G\mathsf{U}]$. Their composition is $(r;s, \langle m, r;n\rangle_0; G\gamma\rangle)$. This morphism is mapped onto $(F(r;s), \widehat{\langle m, r;n}\rangle_0;\gamma)$ by $K$. On the other hand, we can first apply $K$ and then compose. What we get is $(Fr;Fs, \langle \hat{m}, \widehat{r;n}\rangle_0;\gamma)$. So the problem is reduced to showing that if $\Gamma \xrightarrow{m_1} G\mathsf{U}_1$ and $\Gamma \xrightarrow{m_2} G\mathsf{U}_1$ are composable, $\langle \widehat{m_1, m_2}\rangle_0 = \langle \hat{m}_1, \hat{m}_2\rangle_0$. By definition, $\langle \widehat{m_1, m_2}\rangle_0$ is $F\langle m_1, m_2\rangle_0;\epsilon_{\mathsf{U}_2}$. The map $F\langle m_1, m_2\rangle_0$ is $\langle Fm_1, Fm_2\rangle_0$ because $F$ preserves limits. By the naturality of $\epsilon$ and a diagram-chasing, we have $\epsilon_{\mathsf{U}_2} = \epsilon_{\mathsf{U}_1} \times_0 \epsilon_{\mathsf{U}_1}$. Therefore

$$\begin{aligned}
\langle \widehat{m_1, m_2}\rangle_0 &= \langle Fm_1, Fm_2\rangle_0;\epsilon_{\mathsf{U}_2} \\
&= \langle Fm_1;\epsilon_{\mathsf{U}_1}, Fm_2;\epsilon_{\mathsf{U}_1}\rangle_0 \\
&= \langle \hat{m}_1, \hat{m}_2\rangle_0.
\end{aligned}$$

We conclude that $K$ is a functor. It is easily seen that $K$ renders the square commutative. Next we must show that it is a pullback. Suppose $M;F = N;dom_{\mathsf{U}}$.

$$\begin{tikzcd}
{[G\mathsf{U}]} \arrow[rr, "K"] \arrow[dd, "dom_{G\mathsf{U}}"'] & & {[\mathsf{U}]} \arrow[dd, "dom_{\mathsf{U}}"] \\
& \mathscr{B} \arrow[ul, "H"] \arrow[ur, "N"'] \arrow[dl, "M"] & \\
\mathscr{C} \arrow[rr, "F"'] & & \mathbf{D}
\end{tikzcd}$$

The functor $H : \mathscr{B} \longrightarrow [G\mathsf{U}]$ is defined as follows

$$\begin{aligned}
X &\longmapsto MX \xrightarrow{NX} G\mathsf{U}_0, \\
X \xrightarrow{f} Y &\longmapsto (Mf, \overline{Nf}),
\end{aligned}$$

where we have confused notationally $Nf$ with its second component. $H$ so defined clearly preserves identities. For morphisms $X \xrightarrow{f} Y$ and $Y \xrightarrow{g} Z$ in $\mathsf{B}$, the composition of $Nf$ and $Ng$ is

$$(FM(f;g), \langle Nf, FMf;Ng\rangle_0;\gamma).$$

So $H(f;g) = (M(f;g), \overline{\langle Nf, FMf;Ng\rangle_0};G\gamma)$. On the other hand,

$$\begin{aligned}
H(f);H(g) &= (M(f);M(g), \langle \overline{Nf}, Mf;\overline{Ng}\rangle_0;G\gamma) \\
&= (M(f;g), \langle \overline{Nf}, \overline{FMf;Ng}\rangle_0;G\gamma).
\end{aligned}$$

So we only have to show that if $FA \xrightarrow{\sigma} \mathsf{U}_1$ and $FA \xrightarrow{\tau} \mathsf{U}_1$ are composable, then $\langle \overline{\sigma}, \overline{\tau}\rangle_0 = \overline{\langle \sigma, \tau\rangle_0}$. But

$$\begin{aligned}
\langle \overline{\sigma}, \overline{\tau}\rangle_0 &= \langle \eta_A;G\sigma, \eta_A;G\tau\rangle_0 \\
&= \eta_A;\langle G\sigma, G\tau\rangle_0 \\
&= \eta_A;G\langle \sigma, \tau\rangle_0 \qquad G \text{ preserves limits} \\
&= \overline{\langle \sigma, \tau\rangle_0}.
\end{aligned}$$

Finally, we note that $(K, F)$ is a strict cartesian map from $dom_{G\mathsf{U}}$ to $dom_{\mathsf{U}}$ because it sends a canonical cartesian lifting $\Gamma \xrightarrow{\alpha} \Delta \xrightarrow{\sigma} G\mathsf{U}_0 \xrightarrow{G(\mathsf{id})} G\mathsf{U}_1$ onto the canonical cartesian lifting $F\Gamma \xrightarrow{F\alpha} F\Delta \xrightarrow{\hat{\sigma}} \mathsf{U}_0 \xrightarrow{\mathsf{id}} \mathsf{U}_1$. □

**Theorem 3.8.** Suppose $p_1 : \mathscr{E}_1 \longrightarrow \mathscr{B}_1$ is a split fibration and $p_2 \dashv I \dashv G : \mathscr{E}_2 \longrightarrow \mathscr{B}_2$ is a split $D$-category. Then $p_1$ is internally codable (definable) in $\mathscr{B}_2$ via $\mathscr{J} : \mathscr{B}_1 \longrightarrow \mathscr{B}_2$ iff it is internally codable (definable) in $\mathscr{E}_2$ via $\mathscr{J}; I : \mathscr{B}_1 \longrightarrow \mathscr{E}_2$.

*Proof.* One direction is established by Lemma 3.6.

$$\begin{tikzcd}
\mathscr{E}_1 \arrow[r, "J"] \arrow[d, "p_1"'] & {[G\mathsf{U}]} \arrow[r, "K"] \arrow[d, "dom_{G\mathsf{U}}"] & {[\mathsf{U}]} \arrow[d, "dom_{\mathsf{U}}"] \\
\mathscr{B}_1 \arrow[r, "\mathscr{J}"'] & \mathscr{B}_2 \arrow[r, "I"'] & \mathscr{E}_2
\end{tikzcd}$$

Suppose $(H, \mathscr{J}; I)$ is a strict cartesian map from $p_1$ to $dom_{\mathsf{U}}$. By Lemma 3.7, the right square in the above diagram is a pullback. So $H$ factors as $J; K$ for some $J : \mathscr{E}_1 \hookrightarrow [G\mathsf{U}]$. If the outer rectangle is a pullback, the left square is a pullback. If $H$ sends a canonical cartesian lifting to the canonical cartesian lifting $(IM \xrightarrow{I\alpha} IN, I\alpha; \sigma; \mathsf{id})$, then $J$ must send the former cartesian lifting to the canonical cartesian lifting $(M \xrightarrow{\alpha} N, \alpha; \overline{\sigma}; G(\mathsf{id}))$. That is $(J, \mathscr{J})$ is a strict cartesian map from $p_1$ to $dom_{G\mathsf{U}}$. □

## 4. Introducing $\lambda_{TT}$

The purpose of the first two subsections is to recap 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 both of the languages. It is our hope that by analysing these calculi at an elementary level, we can bring out the point and the advantage of $\lambda_{TT}$. Having reviewed the two languages, we will then introduce our logical framework $\lambda_{TT}$.

### 4.1. *Martin-Löf's logical framework*

Martin-Löf's logical framework (Nordström *et al.* 1990) has the following rules with the definitional equality being extensional. Since the language is only used for comparison, we omit the details of the judgemental equality rules. This comment also applies to the presentation of *ELF* given below.

**Context**

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

**Assumption**

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

**Type**

$$\frac{\Gamma \vdash A\ \textbf{type} \quad \Gamma, x : A \vdash B\ \textbf{type}}{\Gamma \vdash (x{:}A)B\ \textbf{type}}\ \textsf{Prod}$$

**Construction**

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

**Set**

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

**Conversion**

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash A = B}{\Gamma \vdash a : B}\ \textsf{Conv}$$

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

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

where we assume

$$\begin{array}{lcl} A & : & Set \\ B & : & (x{:}El(A))Set \\ f & : & (x{:}El(A))El(B(x)) \\ a & : & El(A). \end{array}$$

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 logics instead of just one group of them as in the case of Martin-Löf's framework. The system is structured into three levels: objects, constructors and kinds. The top level kinds provide the mechanism for introducing different universes. In any particular application, a fixed number of constants are introduced, which form a signature. These are the rules of *ELF*:

**Valid Signatures**

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

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

**Valid Contexts**

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

**Valid Kinds**

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

**Valid Constructors**

$$\frac{\vdash_\Sigma \Gamma \quad c : K \in \Sigma}{\Gamma \vdash_\Sigma c : K}\ \textsf{Cst Con} \qquad \frac{\Gamma, x : A \vdash_\Sigma B : Type}{\Gamma \vdash_\Sigma \Pi x{:}A.B : Type}\ \textsf{Pi Con}$$

$$\frac{\Gamma, x : A \vdash_\Sigma B : K}{\Gamma \vdash_\Sigma \lambda x{:}A.B : \Pi x{:}A.K}\ \textsf{Abs Con} \qquad \frac{\Gamma \vdash_\Sigma A : \Pi x{:}B.K \quad \Gamma \vdash_\Sigma M : B}{\Gamma \vdash_\Sigma AM : K[M/x]}\ \textsf{App Con}$$

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

**Valid Objects**

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

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

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

To give an idea of what encodings in *ELF* look like, we present a fragmentary encoding of simply typed $\lambda$-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, where $=_\tau$ stands for $=(\tau)$:

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

We should remark that in *ELF* equalities in object languages are ‘internalized’ by appropriate constructors. Following Martin-Löf’s logical framework, *ELF* employs the following principle:

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

We will call it the *variable convention.*

## 4.3. $\lambda_{TT}$*: a framework for defining type theories*

A typed calculus can only be a framework for a certain collection of typed calculi[†]. Some logics can be faithfully represented in such a framework because they have type theoretical formulations via the Curry–Howard isomorphism. Some other logics can be mimicked in the framework because some of their aspects can be captured type theoretically, and these aspects are considered to be important. A well-known example that cannot be coded up in *ELF* is 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. So, as long as one’s logical framework is a typed calculus, what one can code up in this framework are some other typed calculi; and that is all. It is for this reason that we think of $\lambda_{TT}$ as a framework for defining *t*ype *t*heories. It should be emphasized that $\lambda_{TT}$ is not meant to be a stronger logical framework. It is meant to be a manifestation of our view that a logical framework is a setting for defining frame languages.

The language $\lambda_{TT}$ is designed with the following goals in mind: it should incorporate ideas from both *ELF* and Martin-Löf’s logical framework; it should simplify model theoretical investigations; and it should have a built-in mechanism that formalizes processes to define internally definable typed calculi. There are several design decisions one has to make. Let us mention two of them:

(i) we decide that $\lambda_{TT}$ should have finite product types, and
(ii) we decide that $\lambda_{TT}$ should be an explicit language.

† We only consider languages where there is a notion of variable.

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.

There are usually two ways to present a typed calculus. In both approaches, one first defines the abstract syntax of the entities of the calculus and then gives rules defining the well-formed entities. A type theorist would go on to formulate reduction rules and define the definitional equality in terms of reduction. The notion of reduction is essential in proof-theoretical studies (Luo 1990a; Harper *et al.* 1993). A mathematician on the other hand prefers to define the definitional equality via judgemental equality given by judgemental equational rules (Pitts 1987; Hyland and Pitts 1989). This latter approach is favoured in semantical investigations as it renders no proof-theoretical problems when forming term models. Our logical framework $\lambda_{TT}$ as used in this paper will be given in the second style for the obvious reason.

The abstract syntax for types and objects in $\lambda_{TT}$ is described by the following grammar:

$$\begin{array}{rcl} \text{Types } A & ::= & C \mid U \mid \mathbf{1} \mid A \times A' \mid \Pi x{:}A.A' \mid \mathbf{t}_U(M) \\ \text{Objects } M & ::= & c \mid x \mid \star \mid \langle M, M' \rangle \mid \pi_1 M \mid \pi_2 M \mid \lambda x{:}A.M \mid M(M') \end{array}$$

where $C$ is a type constant declared in a signature, $U$ a universe declared in a universe declaration, $c$ an object constant declared in a signature and $x$ an object variable. Types of the form $\mathbf{t}_U(M)$ are nonstandard. The role of the type constructor $\mathbf{t}$ will be explained after the rules. $\lambda_{TT}$ has assertions of the following forms:

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

Substitutions into terms are defined as usual in metalanguage. The only case that is worth mentioning is

$$\mathbf{t}_U(M)[M'/x] \overset{\text{def}}{=} \mathbf{t}_U(M[M'/x]).$$

The rules concerning the well-formed entities of $\lambda_{TT}$ are as follows:

**Universe**

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

**Signature**

$$\frac{\vdash \Omega\ \mathbf{Uni}}{\vdash_\Omega \langle\rangle\ \mathbf{Sig}}\ \text{Empty Sig} \qquad \frac{\vdash_\Omega \Sigma\ \mathbf{Sig} \quad C \text{ fresh}}{\vdash_\Omega \Sigma, C\ \mathbf{Sig}}\ \text{Sig-Type}$$

$$\frac{[]\vdash_{\Omega;\Sigma} A\ \mathbf{Type} \quad c \text{ fresh}}{\vdash_\Omega \Sigma, c : A\ \mathbf{Sig}}\ \text{Sig-Obj}$$

**Context**

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

**Type**

$$\frac{\vdash_{\Omega;\Sigma} \Gamma\ \mathbf{Con} \quad U \text{ in } \Omega}{\Gamma \vdash_{\Omega;\Sigma} U\ \mathbf{Type}}\ \text{U-Type} \qquad \frac{\vdash_{\Omega;\Sigma} \Gamma\ \mathbf{Con} \quad C \text{ in } \Sigma}{\Gamma \vdash_{\Omega;\Sigma} C\ \mathbf{Type}}\ \text{C-Type}$$

$$\frac{\vdash_{\Omega;\Sigma} \Gamma\ \mathbf{Con}}{\Gamma \vdash_{\Omega;\Sigma} \mathbf{1}\ \mathbf{Type}}\ \text{Unit} \qquad \frac{\Gamma \vdash_{\Omega;\Sigma} A\ \mathbf{Type} \quad \Gamma \vdash_{\Omega;\Sigma} B\ \mathbf{Type}}{\Gamma \vdash_{\Omega;\Sigma} A \times B\ \mathbf{Type}}\ \text{Prod}$$

$$\frac{\Gamma, x : A \vdash_{\Omega;\Sigma} B\ \mathbf{Type}}{\Gamma \vdash_{\Omega;\Sigma} \Pi x : A.B\ \mathbf{Type}}\ \Pi\text{-Prod} \qquad \frac{\Gamma \vdash_{\Omega;\Sigma} M : U \quad U \text{ in } \Omega}{\Gamma \vdash_{\Omega;\Sigma} \mathbf{t}_U(M)\ \mathbf{Type}}\ \text{Reflection}$$

**Object**

$$\frac{\vdash_{\Omega;\Sigma} \Gamma\ \mathbf{Con} \quad c : C \text{ in } \Sigma}{\Gamma \vdash_{\Omega;\Sigma} c : C}\ \text{C-Obj} \qquad \frac{\vdash_{\Omega;\Sigma} \Gamma\ \mathbf{Con} \quad x : A \text{ in } \Gamma}{\Gamma \vdash_{\Omega;\Sigma} x : A}\ \text{V-Obj}$$

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

$$\frac{\vdash_{\Omega;\Sigma} \Gamma\ \mathbf{Con}}{\Gamma \vdash_{\Omega;\Sigma} \star : \mathbf{1}}\ \text{Singleton} \qquad \frac{\Gamma \vdash_{\Omega;\Sigma} M : A \quad \Gamma \vdash_{\Omega;\Sigma} N : B}{\Gamma \vdash_{\Omega;\Sigma} \langle M, N \rangle : A \times B}\ \text{Pair}$$

$$\frac{\Gamma \vdash_{\Omega;\Sigma} M : A \times B}{\Gamma \vdash_{\Omega;\Sigma} \pi_1 M : A}\ \text{L-Proj} \qquad \frac{\Gamma \vdash_{\Omega;\Sigma} M : A \times B}{\Gamma \vdash_{\Omega;\Sigma} \pi_2 M : B}\ \text{R-Proj}$$

**Conversion**

$$\frac{\Gamma \vdash_{\Omega;\Sigma} M : A \quad \Gamma \vdash_{\Omega;\Sigma} B\ \mathbf{Type} \quad \Gamma \vdash_{\Omega;\Sigma} A = B}{\Gamma \vdash_{\Omega;\Sigma} M : B}\ \text{Conv.}$$

The set of judgemental equational rules contains the following extensional rules

$$\frac{\Gamma \vdash_{\Omega;\Sigma} M : \mathbf{1}}{\Gamma \vdash_{\Omega;\Sigma} M = \star : \mathbf{1}}$$

$$\frac{\Gamma \vdash_{\Omega;\Sigma} M : \Pi x : A.B \quad \Gamma, x : A \vdash_{\Omega;\Sigma} B\ \mathbf{Type}}{\Gamma \vdash_{\Omega;\Sigma} \lambda x : A.Mx = M : \Pi x : A.B}$$

$$\frac{\Gamma \vdash_{\Omega;\Sigma} M : A \times B \quad \Gamma \vdash_{\Omega;\Sigma} A\ \mathbf{Type} \quad \Gamma \vdash_{\Omega;\Sigma} B\ \mathbf{Type}}{\Gamma \vdash_{\Omega;\Sigma} \langle \pi_1 M, \pi_2 M \rangle = M : A \times B}$$

and the rule that reflects the judgemental equality between objects of a universe to that between the corresponding types

$$\frac{\Gamma \vdash_{\Omega;\Sigma} M = N : U \quad U \text{ in } \Omega}{\Gamma \vdash_{\Omega;\Sigma} \mathbf{t}_U(M) = \mathbf{t}_U(N)}.$$

We omit the rest of the rules since they are well known.

In order to have an external view on equations in object languages we need equational contexts:

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

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

An equational context is just a finite list of definitional equations. If $\vdash_{\Omega;\Sigma} Th$ **Theory**, we write $\Gamma \vdash_{\Omega;\Sigma} M =_{Th} N : A$ if $\Gamma \vdash_{\Omega;\Sigma} M = N : A$ is derived from the equations in $Th$ as well as the definitional equations of $\lambda_{TT}$.

A careful reader must have noticed that by eliminating the higher-order structure in *ELF* we have collapsed rules about valid constructors and those about valid objects into one group.

A prominent feature of $\lambda_{TT}$, as compared to *ELF*, is that there is a built-in operator $\mathbf{t}_U$, for each universe $U$, that reflects an object $M$ of the universe to a type $\mathbf{t}_U(M)$. To appreciate this operator, one must first of all understand the role played by the kinds in *ELF*. The basic fact about the kinds in *ELF* is that they are all of the form $\Pi x_1 : A_1. \cdots \Pi x_n : A_n.Type$, where $A_1, \cdots, A_n$ are types. In *ELF* it is legitimate to declare a constant constructor of a closed kind in a signature; but it is forbidden to declare a variable constructor of any kind in a context. It follows that *ELF* is not a full-scale higher-order language. In *ELF* one never quantifies over a kind. The purpose of kinds is to introduce constant families of types indexed over types. This is the only place where genuine dependent types come into the language. It is this property that enables us to imitate *ELF* by a first-order language $\lambda_{TT}$ through the use of the lifting operator **t**. The role of $\Pi x_1 : A_1. \cdots \Pi x_n : A_n.Type$ is now played by $\Pi x_1 : A_1. \cdots \Pi x_n : A_n.U$ in $\lambda_{TT}$, where $U$ is a universe. This is possible because in *ELF* we never talk about variables of a kind. The type $\Pi x_1 : A_1. \cdots \Pi x_n : A_n.U$ classifies the families of names of types indexed over types $A_1, \ldots, A_n$.

In true type theoretical spirit, $\lambda_{TT}$ is an extension of Martin-Löf's logical framework. The additional features are product types and universe declaration[†]. 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 $\Sigma$-types, which are troublesome, and having only $\Pi$-types, which sometimes does not produce encodings in the way we want (the problem is even more serious when we

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

have universe declaration). For instance, in *ELF* the plus operator $+ : N{\to}N{\to}N$ has a drawback: $+(n)$ usually does not correspond to anything in the object language. In $\lambda_{TT}$ we can declare $+$ to be of type $N{\times}N{\to}N$. The problem now disappears.

There are at least three questions concerned with $\lambda_{TT}$:

1 If we forget about the equational contexts of $\lambda_{TT}$, we get a sublanguage, which we call $\lambda_{LF}$. Is $\lambda_{LF}$ confluent and strongly normalizing? More generally, is $\lambda_{LF}$ decidable?

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

3 What is the proper notion of semantics of $\lambda_{TT}$?

The answer to Question 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 the Church–Rosser property are heavily interwined. No trick has been invented to break the cycle. Our intuition tells us that $\lambda_{LF}$ is a kind-free formulation of *ELF* with finite product types. Unfortunately, we have not been able to give a reversable translation between them.

Question 2 is harder. It cannot be tackled before the proof theory of $\lambda_{TT}$ has been fully understood. Suppose we have defined a language $\mathscr{L}$ in $\lambda_{TT}$ and want to show that a property $\mathscr{P}$ holds of $\mathscr{L}$. Suppose further that $\mathscr{P}$ holds of $\lambda_{TT}$. From that we need to prove that $\mathscr{P}$ holds of the extended language $\lambda_{TT}+\mathscr{L}$. So the problem is closely related to that of compositional understanding of type theory – an issue that has not yet been properly addressed (as far as we know, the proof theory of the monomorphic Martin-Löf's type theory is unknown).

This paper attempts to give an answer to Question 3.

## 5. Coding object languages

In this section we give four examples of how to encode typed calculi and logics in $\lambda_{TT}$. The reader is encouraged to compare Examples 5.1 and 5.3 with those given in Sections 4.1 and 4.2. In the following, we will often use subscripts for application. For instance, $\mathsf{app}_{\sigma,\tau}$ stands for $\mathsf{app}(\sigma)(\tau)$.

**Example 5.1.** Throughout the rest of this paper, we fix a simply typed $\lambda$-calculus with a finite number of constant types. The following is a formulation of this calculus in $\lambda_{TT}$.

$$
\begin{array}{lllcl}
\Omega_\lambda & \text{is} & U & & \\
\Sigma_\lambda & \text{is} & \wedge & : & U{\times}U{\to}U \\
 & & \Rightarrow & : & U{\times}U{\to}U \\
 & & \diamond & : & U \\
 & & \heartsuit & : & \mathbf{t}_U(\diamond) \\
 & & \mathsf{abs} & : & \Pi\sigma:U.\Pi\tau:U.[\mathbf{t}_U(\sigma){\to}\mathbf{t}_U(\tau)]{\to}\mathbf{t}_U(\sigma{\Rightarrow}\tau) \\
 & & \mathsf{app} & : & \Pi\sigma:U.\Pi\tau:U.\mathbf{t}_U(\sigma{\Rightarrow}\tau){\times}\mathbf{t}_U(\sigma){\to}\mathbf{t}_U(\tau) \\
 & & \mathsf{pair} & : & \Pi\sigma:U.\Pi\tau:U.\mathbf{t}_U(\sigma){\times}\mathbf{t}_U(\tau){\to}\mathbf{t}_U(\sigma{\wedge}\tau) \\
 & & \mathsf{pr}_1 & : & \Pi\sigma:U.\Pi\tau:U.\mathbf{t}_U(\sigma{\wedge}\tau){\to}\mathbf{t}_U(\sigma) \\
 & & \mathsf{pr}_2 & : & \Pi\sigma:U.\Pi\tau:U.\mathbf{t}_U(\sigma{\wedge}\tau){\to}\mathbf{t}_U(\tau)
\end{array}
$$

$$
\begin{array}{lll}
Th_\lambda & \text{is} & \vdash_{\Omega;\Sigma} [\sigma : U, \tau : U, f : \mathbf{t}_U(\sigma)\to\mathbf{t}_U(\tau), x : \mathbf{t}_U(\sigma) \vdash \\
 & & \quad \mathsf{app}_{\sigma,\tau}(\mathsf{abs}_{\sigma,\tau}(f), x) = fx : \mathbf{t}_U(\tau)] \\
 & & \vdash_{\Omega;\Sigma} [\sigma : U, \tau : U, f : \mathbf{t}_U(\sigma\Rightarrow\tau) \vdash \\
 & & \quad \mathsf{abs}_{\sigma,\tau}(\lambda x : \mathbf{t}_U(\sigma).\mathsf{app}_{\sigma,\tau}(f, x)) = f : \mathbf{t}_U(\sigma \Rightarrow \tau)] \\
 & & \vdash_{\Omega;\Sigma} [\sigma : U, \tau : U, x : \mathbf{t}_U(\sigma), y : \mathbf{t}_U(\tau) \vdash (\mathsf{pr}_1)_{\sigma,\tau}(\mathsf{pair}_{\sigma,\tau}(x, y)) = x : \mathbf{t}_U(\sigma)] \\
 & & \vdash_{\Omega;\Sigma} [\sigma : U, \tau : U, x : \mathbf{t}_U(\sigma), y : \mathbf{t}_U(\tau) \vdash (\mathsf{pr}_2)_{\sigma,\tau}(\mathsf{pair}_{\sigma,\tau}(x, y)) = y : \mathbf{t}_U(\tau)] \\
 & & \vdash_{\Omega;\Sigma} [\sigma : U, \tau : U, z : \mathbf{t}_U(\sigma\wedge\tau) \vdash \mathsf{pair}_{\sigma,\tau}((\mathsf{pr}_1)_{\sigma,\tau}z, (\mathsf{pr}_2)_{\sigma,\tau}z) = z : \mathbf{t}_U(\sigma \wedge \tau)] \\
 & & \vdash_{\Omega;\Sigma} [x : \mathbf{t}_U(\diamond) \vdash \heartsuit = x : \mathbf{t}_U(\diamond)]
\end{array}
$$

Here $\diamond$ codes up the unit type in the simply typed $\lambda$-calculus; $\heartsuit$ provides an inhabitant of the type $\mathbf{t}_U(\diamond)$; and the accompanying definitional equation forces this inhabitant to be unique. $\mathscr{E}_\lambda$ will stand for this encoding. We do not give the encoding of the constants as it is obvious. The same remark also applies to the following examples. □

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

The encoding goes as follows:

$$
\begin{array}{llrcl}
\Omega_{PL} & \text{is} & U, K & & \\
\Sigma_{PL} & \text{is} & \dot{\wedge} & : & K\times K\to K \\
 & & \dot{\Rightarrow} & : & K\times K\to K \\
 & & \dot{\diamond} & : & K \\
 & & T & : & K \\
 & & \dot{\heartsuit} & : & \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) \\
 & & \mathsf{in} & : & \mathbf{t}_K(T)\to U \\
 & & \mathsf{out} & : & U\to\mathbf{t}_K(T) \\
 & & \wedge & : & \mathbf{t}_K(T)\times\mathbf{t}_K(T)\to\mathbf{t}_K(T) \\
 & & \Rightarrow & : & \mathbf{t}_K(T)\times\mathbf{t}_K(T)\to\mathbf{t}_K(T) \\
 & & \diamond & : & \mathbf{t}_K(T) \\
 & & \dot{\forall} & : & \Pi\kappa : K.(\mathbf{t}_K(\kappa)\to\mathbf{t}_K(T))\to\mathbf{t}_K(T) \\
 & & \heartsuit & : & \mathbf{t}_U(\mathsf{in}(\diamond)) \\
 & & \mathsf{abs} & : & \Pi\sigma : \mathbf{t}_K(T).\Pi\tau : \mathbf{t}_K(T).[\mathbf{t}_U(\mathsf{in}(\sigma))\to\mathbf{t}_U(\mathsf{in}(\tau))]\to\mathbf{t}_U(\mathsf{in}(\sigma\Rightarrow\tau)) \\
 & & \mathsf{app} & : & \Pi\sigma : \mathbf{t}_K(T).\Pi\tau : \mathbf{t}_K(T).\mathbf{t}_U(\mathsf{in}(\sigma\Rightarrow\tau))\times\mathbf{t}_U(\mathsf{in}(\sigma))\to\mathbf{t}_U(\mathsf{in}(\tau)) \\
 & & \mathsf{pair} & : & \Pi\sigma : \mathbf{t}_K(T).\Pi\tau : \mathbf{t}_K(T).\mathbf{t}_U(\mathsf{in}(\sigma))\times\mathbf{t}_U(\mathsf{in}(\tau))\to\mathbf{t}_U(\mathsf{in}(\sigma\wedge\tau)) \\
 & & \mathsf{pr}_1 & : & \Pi\sigma : \mathbf{t}_K(T).\Pi\tau : \mathbf{t}_K(T).\mathbf{t}_U(\mathsf{in}(\sigma\wedge\tau))\to\mathbf{t}_U(\mathsf{in}(\sigma))
\end{array}
$$

$$
\begin{array}{lcl}
\mathsf{pr}_2 & : & \Pi\sigma:\mathbf{t}_K(T).\Pi\tau:\mathbf{t}_K(T).\mathbf{t}_U(\mathsf{in}(\sigma\wedge\tau))\rightarrow\mathbf{t}_U(\mathsf{in}(\tau)) \\
\mathsf{Abs} & : & \Pi\kappa:K.\Pi F:\mathbf{t}_K(\kappa)\rightarrow\mathbf{t}_K(T).(\Pi\sigma:\mathbf{t}_K(\kappa).\mathbf{t}_U(\mathsf{in}(F\sigma)))\rightarrow \\
 & & \quad\mathbf{t}_U(\mathsf{in}(\dot\forall_\kappa(F))) \\
\mathsf{App} & : & \Pi\kappa:K.\Pi F:\mathbf{t}_K(\kappa)\rightarrow\mathbf{t}_K(T).\mathbf{t}_U(\mathsf{in}(\dot\forall_\kappa(F)))\rightarrow \\
 & & \quad\Pi\sigma:\mathbf{t}_K(\kappa).\mathbf{t}_U(\mathsf{in}(F\sigma))
\end{array}
$$

$$
\begin{array}{lll}
Th_{PL} & \text{is} & \vdash_{\Omega;\Sigma} [\kappa:K,\varrho:K,f:\mathbf{t}_K(\kappa)\rightarrow\mathbf{t}_K(\varrho),x:\mathbf{t}_K(\kappa) \\
 & & \qquad\vdash \dot{\mathsf{app}}_{\kappa,\varrho}(\dot{\mathsf{abs}}_{\kappa,\varrho}(f),x)=fx:\mathbf{t}_K(\varrho)] \\
 & & \vdash_{\Omega;\Sigma} [\kappa:K,\varrho:K,f:\mathbf{t}_K(\kappa\mathrel{\dot\Rightarrow}\varrho) \\
 & & \qquad\vdash \dot{\mathsf{abs}}_{\kappa,\varrho}(\lambda x:\mathbf{t}_K(\kappa).\dot{\mathsf{app}}_{\kappa,\varrho}(f,x))=f:\mathbf{t}_K(\kappa\mathrel{\dot\Rightarrow}\varrho)] \\
 & & \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:\mathbf{t}_K(\kappa)] \\
 & & \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:\mathbf{t}_K(\varrho)] \\
 & & \vdash_{\Omega;\Sigma} [\kappa:K,\varrho:K,z:\mathbf{t}_K(\kappa\mathbin{\dot\wedge}\varrho)\vdash\dot{\mathsf{pair}}_{\sigma,\tau}((\dot{\mathsf{pr}}_1)_{\sigma,\tau}z,(\dot{\mathsf{pr}}_2)_{\sigma,\tau}z)=z:\mathbf{t}_K(\kappa\mathbin{\dot\wedge}\varrho)] \\
 & & \vdash_{\Omega;\Sigma} [x:\mathbf{t}_K(\dot\diamond)\vdash\dot\heartsuit=x:\mathbf{t}_K(\dot\diamond)] \\
 & & \vdash_{\Omega;\Sigma} [x:\mathbf{t}_K(T)\vdash\mathsf{out}(\mathsf{in}(x))=x:\mathbf{t}_K(T)] \\
 & & \vdash_{\Omega;\Sigma} [x:U\vdash\mathsf{in}(\mathsf{out}(x))=x:U] \\
 & & \vdash_{\Omega;\Sigma} [\kappa:K,F:\mathbf{t}_K(\kappa)\rightarrow\mathbf{t}_K(T),x:\Pi\sigma:\mathbf{t}_K(\kappa).\mathbf{t}_U(\mathsf{in}(F\sigma)) \\
 & & \qquad\vdash\mathsf{App}_{\kappa,F}(\mathsf{Abs}_{\kappa,F}(x))=x:\Pi\sigma:\mathbf{t}_K(\kappa).\mathbf{t}_U(\mathsf{in}(F\sigma))] \\
 & & \vdash_{\Omega;\Sigma} [\kappa:K,F:\mathbf{t}_K(\kappa)\rightarrow\mathbf{t}_K(T),x:\mathbf{t}_U(\mathsf{in}(\dot\forall_\kappa(F))) \\
 & & \qquad\vdash\mathsf{Abs}_{\kappa,F}(\mathsf{App}_{\kappa,F}(x))=x:\mathbf{t}_U(\mathsf{in}(\dot\forall_\kappa(F)))] \\
 & & \vdash_{\Omega;\Sigma} [\sigma:\mathbf{t}_K(T),\tau:\mathbf{t}_K(T),f:\mathbf{t}_U(\mathsf{in}(\sigma))\rightarrow\mathbf{t}_U(\mathsf{in}(\tau)),x:\mathbf{t}_U(\mathsf{in}(\sigma)) \\
 & & \qquad\vdash\mathsf{app}_{\sigma,\tau}(\mathsf{abs}_{\sigma,\tau}(f),x)=fx:\mathbf{t}_U(\mathsf{in}(\tau))] \\
 & & \vdash_{\Omega;\Sigma} [\sigma:\mathbf{t}_K(T),\tau:\mathbf{t}_K(T),f:\mathbf{t}_U(\mathsf{in}(\sigma\Rightarrow\tau)) \\
 & & \qquad\vdash\mathsf{abs}_{\sigma,\tau}(\lambda x:\mathbf{t}_U(\mathsf{in}(\sigma)).\mathsf{app}_{\sigma,\tau}(f,x))=f:\mathbf{t}_U(\mathsf{in}(\sigma\Rightarrow\tau))] \\
 & & \vdash_{\Omega;\Sigma} [\sigma:\mathbf{t}_K(T),\tau:\mathbf{t}_K(T),x:\mathbf{t}_U(\mathsf{in}(\sigma)),y:\mathbf{t}_U(\mathsf{in}(\tau)) \\
 & & \qquad\vdash(\mathsf{pr}_1)_{\sigma,\tau}(\mathsf{pair}_{\sigma,\tau}(x,y))=x:\mathbf{t}_U(\mathsf{in}(\sigma))] \\
 & & \vdash_{\Omega;\Sigma} [\sigma:\mathbf{t}_K(T),\tau:\mathbf{t}_K(T),x:\mathbf{t}_U(\mathsf{in}(\sigma)),y:\mathbf{t}_U(\mathsf{in}(\tau)) \\
 & & \qquad\vdash(\mathsf{pr}_2)_{\sigma,\tau}(\mathsf{pair}_{\sigma,\tau}(x,y))=y:\mathbf{t}_U(\mathsf{in}(\tau))] \\
 & & \vdash_{\Omega;\Sigma} [\sigma:\mathbf{t}_K(T),\tau:\mathbf{t}_K(T),z:\mathbf{t}_U(\mathsf{in}(\sigma\wedge\tau)) \\
 & & \qquad\vdash\mathsf{pair}_{\sigma,\tau}((\mathsf{pr}_1)_{\sigma,\tau}z,(\mathsf{pr}_2)_{\sigma,\tau}z)=z:\mathbf{t}_U(\mathsf{in}(\sigma\wedge\tau))] \\
 & & \vdash_{\Omega;\Sigma} [x:\mathbf{t}_U(\mathsf{in}(\diamond))\vdash\heartsuit=x:\mathbf{t}_U(\mathsf{in}(\diamond))]
\end{array}
$$

In the above encoding, $K$ is the universe of kinds; the constant $\dot\diamond$ codes up the unit kind, $T$ the kind $Type$ of all types. The map in helps to code up the objects. Notice that the map out and the two accompanying equations, which forces in and out to be inverse to each other, are unnecessary from the proof theoretical point of view. They are included in the encoding to achieve a better categorical description. We will refer to the entire encoding of this higher-order polymorphism as $\mathscr{E}_{PL}$. □

**Example 5.3.** We now give an encoding of Martin-Löf's type theory built on a finite number of constants. The encoding should look familiar. In this example we leave out the subscript in $\mathbf{t}_{Set}$ and abbreviate the context $[A:Set,B:\mathbf{t}(A)\rightarrow Set]$ to $\Gamma$.

$$
\begin{array}{lll}
\Omega_{ML} & \text{is} & Set \\
\Sigma_{ML} & \text{is} & \begin{array}{rcl}
\pi & : & \Pi X:Set.(\mathbf{t}(X)\to Set)\to Set \\
\sigma & : & \Pi X:Set.(\mathbf{t}(X)\to Set)\to Set \\
\Lambda & : & \Pi X:Set.\Pi Y:\mathbf{t}(X)\to Set.\Pi f:(\Pi x:\mathbf{t}(X).\mathbf{t}(Y(x))).\mathbf{t}(\pi(X,Y)) \\
\bullet & : & \Pi X:Set.\Pi Y:\mathbf{t}(X)\to Set.\Pi f:\mathbf{t}(\pi(X,Y)).\Pi x:\mathbf{t}(X).\mathbf{t}(Y(x)) \\
\mathbf{pair} & : & \Pi X:Set.\Pi Y:\mathbf{t}(X)\to Set.\Pi x:\mathbf{t}(X).\Pi y:\mathbf{t}(Y(x)).\mathbf{t}(\sigma(X,Y)) \\
\mathbf{P}_1 & : & \Pi X:Set.\Pi Y:\mathbf{t}(X)\to Set.\Pi f:\mathbf{t}(\sigma(X,Y)).\mathbf{t}(X) \\
\mathbf{P}_2 & : & \Pi X:Set.\Pi Y:\mathbf{t}(X)\to Set.\Pi f:\mathbf{t}(\sigma(X,Y)).Y(\mathbf{P}_1(X,Y,f))
\end{array} \\
Th_{ML} & \text{is} & \begin{array}{l}
\vdash_{\Omega;\Sigma} [\Gamma, f : (\Pi x:\mathbf{t}(A).\mathbf{t}(B(x))), a : \mathbf{t}(A) \vdash \\
\quad \bullet(A,B,\Lambda(A,B,f),a) = f(a) : \mathbf{t}(B(a))] \\
\vdash_{\Omega;\Sigma} [\Gamma, f : \mathbf{t}(\pi(A,B)) \vdash \Lambda(A,B,\lambda x:\mathbf{t}(A).\bullet(A,B,f,x)) = f : \mathbf{t}(\sigma(A,B))] \\
\vdash_{\Omega;\Sigma} [\Gamma, a : \mathbf{t}(A), b : \mathbf{t}(B(a)) \vdash \mathbf{P}_1(A,B,\mathbf{pair}(A,B,a,b)) = a : \mathbf{t}(A)] \\
\vdash_{\Omega;\Sigma} [\Gamma, a : \mathbf{t}(A), b : \mathbf{t}(B(a)) \vdash \mathbf{P}_2(A,B,\mathbf{pair}(A,B,a,b)) = b : \mathbf{t}(B(a))] \\
\vdash_{\Omega;\Sigma} [\Gamma, f : \mathbf{t}(\sigma(A,B)) \vdash \\
\quad \mathbf{pair}(A,B,\mathbf{P}_1(A,B,f),\mathbf{P}_2(A,B,f)) = f : \mathbf{t}(\sigma(A,B))]
\end{array}
\end{array}
$$

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

The examples given above are all about encodings of typed calculi. That does not mean that $\lambda_{TT}$ cannot describe logics within it. In fact, $\lambda_{TT}$ can deal with logics just as well as *ELF*. In the following example, we only give fragmentary encodings. The reader is advised to compare them with those given in Harper *et al.* (1993).

**Example 5.4.** When formulating an object language in $\lambda_{TT}$, the first thing one has to decide is whether a syntactical class is coded by a universe in $\Omega$ or a constant type in $\Sigma$. The general principle is that if an entity of a syntactical class is itself inhabited by other entities, that syntactical class should be coded up by a universe. For instance, in first-order logic, there are a class of terms and a class of formulas. As formulas are inhabited by proofs, the latter syntactical class is encoded by a universe

$$\varrho \text{ in } \Omega.$$

The other syntactical class is coded by

$$\kappa \text{ in } \Sigma.$$

Some of the constants in $\Sigma$ are:

$$
\begin{array}{rcl}
0 & : & \kappa \\
\mathsf{s} & : & \kappa\to\kappa \\
+ & : & \kappa\times\kappa\to\kappa \\
= & : & \kappa\times\kappa\to\varrho \\
\neg & : & \varrho\to\varrho \\
\vee & : & \varrho\times\varrho\to\varrho \\
\forall & : & (\kappa\to\varrho)\to\varrho \\
\mathrm{RAA} & : & \Pi p:\varrho.\mathbf{t}_\varrho(\neg\neg p)\to\mathbf{t}_\varrho(p) \\
\mathrm{ALL}-\mathrm{E} & : & \Pi F:\kappa\to\varrho.\Pi x:\kappa.\mathbf{t}_\varrho(\forall(F))\to\mathbf{t}_\varrho(Fx).
\end{array}
$$

The last two code up the *reductio ad absurdum* rule and the elimination rule for the universal quantifier respectively. The situation in the case of higher-order logic is, however, slightly different. We still have the universe

$$\varrho \text{ in } \Omega,$$

but it no longer works to interpret the syntactical class of terms as a constant type in $\Sigma$. In order to code up higher-order quantifications uniformly, the class of terms and the 'name' of $\varrho$ must be encoded by the constants of the same type. So we have in $\Sigma$

$$\begin{array}{rcl} \kappa & : & H \\ \dot{\varrho} & : & H \\ \Rightarrow & : & H\times H\rightarrow H, \end{array}$$

where

$$H \text{ is in } \Omega.$$

Also in the $\Sigma$ are

$$\begin{array}{rcl} \mathsf{0} & : & \mathbf{t}_H(\kappa) \\ \mathsf{succ} & : & \mathbf{t}_H(\kappa)\rightarrow\mathbf{t}_H(\kappa) \\ + & : & \mathbf{t}_H(\kappa)\times\mathbf{t}_H(\kappa)\rightarrow\mathbf{t}_H(\kappa) \\ \leq & : & \mathbf{t}_H(\kappa)\times\mathbf{t}_H(\kappa)\rightarrow\mathbf{t}_H(\dot{\varrho}) \\ \supset & : & \mathbf{t}_H(\dot{\varrho})\times\mathbf{t}_H(\dot{\varrho})\rightarrow\mathbf{t}_H(\dot{\varrho}) \\ = & : & \Pi\sigma:H.\mathbf{t}_H(\sigma)\times\mathbf{t}_H(\sigma)\rightarrow\mathbf{t}_H(\dot{\varrho}) \\ \forall & : & \Pi\sigma:H.(\mathbf{t}_H(\sigma)\rightarrow\mathbf{t}_H(\dot{\varrho}))\rightarrow\mathbf{t}_H(\dot{\varrho}) \\ \mathsf{in} & : & \mathbf{t}_H(\dot{\varrho})\rightarrow\varrho \\ \mathsf{All-I} & : & \Pi\sigma:H.\Pi F:\mathbf{t}_H(\sigma)\rightarrow\mathbf{t}_H(\dot{\varrho}).(\Pi x:\mathbf{t}_H(\sigma).\mathbf{t}_\varrho(\mathsf{in}(Fx)))\rightarrow\mathbf{t}_\varrho(\mathsf{in}(\forall_\sigma(F))). \end{array}$$

In the full encoding, there should be application and abstraction constants and the accompanying definitional equations that ensure that $\mathbf{t}_H(\sigma_1\rightarrow\sigma_2)$ and $\mathbf{t}_H(\sigma_1)\rightarrow\mathbf{t}_H(\sigma_2)$ are 'isomorphic'. As in Example 5.2, the constant $\mathsf{in}$ helps to code up the logical rules. □

## 6. Internal categories in $\lambda_{TT}$

What does it mean to code up an object language in $\lambda_{TT}$? Since the term model of $\lambda_{TT}$ is a fibration, we can rephrase the question as 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 (internal categories). In $\lambda_{TT}$ there should be a routine way of constructing internal categories. It is the purpose of this section to show that $\lambda_{TT}$ has a built-in mechanism for defining internal categories.

Let $\mathscr{E}$ be an encoding in $\lambda_{TT}$. We can build a term model $(\lambda_{TT},\mathscr{E})$ from the constants in the universe declaration and the signature of $\mathscr{E}$ using the equational contexts in $\mathscr{E}$ (Seely 1984; Streicher 1989). It is common knowledge that the term model forms a fibration

$(\lambda_{TT}, \mathscr{E}) : \mathscr{T} \longrightarrow \mathscr{C}$. Here $\mathscr{C}$ is the category of contexts and context realizations. A context realization $r : \Gamma \longrightarrow [x_1 : A_1, \ldots, x_n : A_n]$ is a tuple $(a_1, a_2, \ldots, a_n)$ such that

$$\begin{array}{lll} \Gamma & \vdash & a_1 : A_1 \\ \Gamma & \vdash & a_2 : A_2[a_1/x_1] \\ & \vdots & \\ \Gamma & \vdash & a_n : A_n[a_1/x_1, \ldots, a_{n-1}/x_{n-1}]. \end{array}$$

The objects of $\mathscr{T}$ are types. For instance $\Gamma \vdash A$ is an object in $\mathscr{T}$ (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 $\Gamma \vdash f : A {\to} B[r]$. 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 $(\lambda_{TT}, \mathscr{E}) : \mathscr{T} \longrightarrow \mathscr{C}$ takes $\Gamma \vdash A$ to $\Gamma$ and $(r, f)$ to $r$.

There is a collection $\mathbf{D}$ of display maps in $\mathscr{C}$. A realization $\Gamma \xrightarrow{r} \Delta$ is in $\mathbf{D}$ 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 $\mathscr{T} \xrightarrow{(\lambda_{TT},\mathscr{E})} \mathscr{C}$ is essentially the same as the fibration $\mathscr{C}/\mathbf{D} \xrightarrow{cod} \mathscr{C}$.

In category theory there is a well-known technique for constructing an internal category from a generic morphism $T \xrightarrow{G} U$ (Pitts 1987; Hyland 1988). 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 those that have this property universes. Suppose a universe $U$ is declared in the encoding $\mathscr{E}$. A distinguished feature of $\lambda_{TT}$ is that $\mathscr{T}$ contains an internal category $\mathsf{U}$ constructed from $U$. This is made possible by the lifting operator $\mathbf{t}_U$. Using de Bruijn's notation, the components of this small category can be defined as follows:

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

In the above definition, $U^2$ is the context $[x_1 : U, x_2 : U]$; the pair $(\pi_1, \pi_2)$ is the context realization $[x_1 : U, x_2 : U, x_3 : U] \xrightarrow{(x_1,x_2)} [x : U, y : U]$; and $U^2 \vdash \mathbf{t}_U(1){\to}\mathbf{t}_U(2)$ for instance is $x_1 : U, x_2 : U \vdash \mathbf{t}_U(x_1){\to}\mathbf{t}_U(x_2)$.

**Lemma 6.1.** The above data form an internal category in the fibre category $\mathscr{T}$ of the fibration $(\lambda_{TT}, \mathscr{E}) : \mathscr{T} \longrightarrow \mathscr{C}$ associated with the encoding $\mathscr{E}$.

*Proof.* This is routine, but we will 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 $A$.

The objects $p_1$ and $p_2$ are $U^3 \vdash \lambda w : [\mathbf{t}_U(1) \to \mathbf{t}_U(2)] \times [\mathbf{t}_U(2) \to \mathbf{t}_U(3)].\pi_1 w : \mathbf{t}_U(1) \to \mathbf{t}_U(2)$ and $U^3 \vdash \lambda w : [\mathbf{t}_U(1) \to \mathbf{t}_U(2)] \times [\mathbf{t}_U(2) \to \mathbf{t}_U(3)].\pi_2 w : \mathbf{t}_U(2) \to \mathbf{t}_U(3)$, 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 $\mathsf{U}_3$. 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 $(\lambda_{TT}, \mathscr{E})$, there are another two obvious functors. The functor $\{\_ \vdash \mathbf{1}\} : \mathscr{C} \longrightarrow \mathscr{T}$ embeds $\mathscr{C}$ in $\mathscr{T}$ by sending a context $\Gamma$ to $\Gamma \vdash \mathbf{1}$. The functor $\{\_[\_]\} : \mathscr{T} \longrightarrow \mathscr{C}$ on the other hand is basically the context extension. Its actions on objects and morphisms are defined as follows:

$$\begin{array}{rcl}
\Gamma \vdash A & \longmapsto & \Gamma, x : A, \\
(r, f) : (\Gamma \vdash A) \to (\Delta \vdash B) & \longmapsto & (r, fx) : (\Gamma, x : A) \to (\Delta, y : B)
\end{array}$$

To our knowledge, the following fact was first observed by Ehrhard (Ehrhard 1988).

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

It is a direct consequence of Lemma 6.2 that if we apply the functor $\{\_[\_]\}$ to $\mathsf{U}$ we get an internal category $\dot{\mathsf{U}}$ in $\mathscr{C}$. This $\dot{\mathsf{U}}$ is isomorphic to the following internal category:

$$\begin{array}{rcl}
\mathsf{U}'_0 & \overset{\text{def}}{=} & [x : U] \\
\mathsf{U}'_1 & \overset{\text{def}}{=} & [x_1 : U, x_2 : U, f : \mathbf{t}_U(x_1) \to \mathbf{t}_U(x_2)] \\
\mathsf{U}'_2 & \overset{\text{def}}{=} & [x_1 : U, x_2 : U, x_3 : U, f : \mathbf{t}_U(x_1) \to \mathbf{t}_U(x_2), g : \mathbf{t}_U(x_2) \to \mathbf{t}_U(x_3)] \\
d'_0 & \overset{\text{def}}{=} & (x_1) : \mathsf{U}'_1 \to \mathsf{U}'_0 \\
d'_1 & \overset{\text{def}}{=} & (x_2) : \mathsf{U}'_1 \to \mathsf{U}'_0 \\
\mathsf{id}' & \overset{\text{def}}{=} & (x, x, \lambda y : \mathbf{t}_U(x).y) \\
\Pi'_0 & \overset{\text{def}}{=} & (x_1, x_2, f) : \mathsf{U}'_2 \to \mathsf{U}'_1 \\
\Pi'_1 & \overset{\text{def}}{=} & (x_2, x_3, g) : \mathsf{U}'_2 \to \mathsf{U}'_1 \\
\gamma' & \overset{\text{def}}{=} & (x_1, x_3, \lambda y : \mathbf{t}_U(x_1).g(fy)) : \mathsf{U}'_2 \to \mathsf{U}'_1.
\end{array}$$

From now on, we take $\dot{\mathsf{U}}$ to be this internal category.

**Proposition 6.3.** $\dot{\mathsf{U}}$ is an internal full subcategory of $\mathscr{C}$, relative to $\mathbf{D}$, induced by the display map $[x : U, y : \mathbf{t}_U(x)] \xrightarrow{(x)} [x : U]$.

$$\begin{tikzcd}
{[\dot{\mathsf{U}}]} \arrow[r, "H"] \arrow[dr, "dom_{\dot{\mathsf{U}}}"'] & \mathscr{C}/\mathbf{D} \arrow[r, "\cong"] \arrow[d, "cod"] & \mathscr{T} \arrow[dl, "(\lambda_{TT}, \mathscr{E})"] \\
 & \mathscr{C} &
\end{tikzcd}$$

*Proof.* In the above diagram, $H$ sends an object $(a) : \Gamma \longrightarrow [x : U]$ onto the display map $\Gamma, z : \mathbf{t}_U(a) \longrightarrow \Gamma$; it sends a morphism $(a, b, f) : \Gamma \longrightarrow [x : U, y : U, z : \mathbf{t}_U(x) \to \mathbf{t}_U(y)]$ in the fibre $dom_{\dot{\mathsf{U}}}^{-1}(\Gamma)$ onto the morphism $(Id_\Gamma, fz) : (\Gamma, z : \mathbf{t}_U(a) \to \Gamma) \longrightarrow (\Gamma, z : \mathbf{t}_U(b) \to \Gamma)$. It is full and faithful by the $\eta$-rule. The fact that $\dot{\mathsf{U}}$ is induced by $[x : U, y : \mathbf{t}_U(x)] \xrightarrow{(x)} [x : U]$ is also routine to check. □

Since $\mathscr{C}/\mathbf{D} \xrightarrow{cod} \mathscr{C}$ is isomorphic to $\mathscr{T} \xrightarrow{(\lambda_{TT}, \mathscr{E})} \mathscr{C}$, it makes sense to say that $\dot{\mathsf{U}}$ is an internal full subcategory of $\mathscr{T} \xrightarrow{(\lambda_{TT}, \mathscr{E})} \mathscr{C}$.

If there is more than one universe declared in an encoding, then for each universe one can construct an internal full subcategory.

More light can be cast on the above result if we look at it from a proof theoretical point of view. Suppose $\mathscr{E}$ encodes $\mathscr{L}$. As in *ELF*, variables of the encoded version of $\mathscr{L}$ in $\lambda_{TT}$ are identified with those in $\lambda_{TT}$. What is the model theoretical implication of this decision? Suppose $\Gamma \vdash_{\Omega,\Sigma} a : U$. Then a 'variable' of $a$ in the encoded langage is an indeterminate of the object $(a, \star)$ in the fibre $[\Gamma, \dot{\mathsf{U}}]$. On the other hand, a 'variable' of $a$ in $\lambda_{TT}$ is an indeterminate of the object $\Gamma \vdash \mathbf{t}_U(a)$ in $(\lambda_{TT}, \mathscr{E})^{-1}(\Gamma)$. If we are to identify a variable in the encoded language with the corresponding variable in $\lambda_{TT}$, then $[\Gamma, \dot{\mathsf{U}}]$ should be a *full* subcategory of $(\lambda_{TT}, \mathscr{E})^{-1}(\Gamma)$. We therefore conclude that the result stated in Proposition 6.3 is the categorical counterpart of the variable convention. This categorical explanation has a feedback to type theory: the variable convention forces a conservative extensionality relationship between an encoded calculus and the framework $\lambda_{TT}$.

## 7. Structures of universes

We have not told the full story about the internal categories discussed in Section 6. When defining a typed calculus in $\lambda_{TT}$, we force those internal categories to have specific structures. In this section we show that the internal categories induced by the universes declared in the first three examples in Section 5 have the necessary categorical structures that are usually associated with the categorical models of the three typed calculi, respectively. More specifically, we will show that in the encoding of the simply typed $\lambda$-calculus, the internal full subcategory $\dot{\mathsf{U}}$ induced by the universe $U$ is equipped with explicit cartesian closed structure. The internal categories $\dot{\mathsf{U}}$ and $\dot{\mathsf{K}}$ induced by the universes in the encoding of the polymorphic $\lambda$-calculus also possess the same structure. In addition, $\dot{\mathsf{U}}$ has explicit products over the encoded kinds. For the encoding of Martin-Löf type theory one can show that the internal full subcategory $\dot{\mathsf{Set}}$ induced by the universe

*Set* has an appropriate explicit internal structure, which, when externalized, gives rise to the left and right adjoints to weakening functors. We will omit an account of the Martin-Löf case, since the proofs are similar to those in the polymorphic case. In this section, we systematically omit the subscript in $\vdash_{\Omega;\Sigma}$.

**Lemma 7.1.** The equations in $Th_\lambda$ imply that the following diagrams are pullbacks in the base category of the fibration $(\lambda_{TT}, \mathscr{E}_\lambda)$, where $\otimes$ is $\Rightarrow$ (or $\wedge$) and $\dot{\otimes}$ is $\rightarrow$ (or $\times$); $m$ is $\mathsf{abs}_{x_1,x_2}(f)$ when $\otimes$ is $\Rightarrow$ and is $\mathsf{pair}_{x_1,x_2}(f)$ when $\otimes$ is $\wedge$.

$$\begin{tikzcd}
{[]} \arrow[r, "(\diamond, \heartsuit)"] \arrow[d, "()"'] & {[x : U, y : \mathbf{t}_U(x)]} \arrow[d, "(x)"] \\
{[]} \arrow[r, "(\diamond)"'] & {[x : U]}
\end{tikzcd}$$

$$\begin{tikzcd}
{[x_1 : U, x_2 : U, f : \mathbf{t}_U(x_1)\dot{\otimes}\mathbf{t}_U(x_2)]} \arrow[r, "(x_1 \otimes x_2, m)"] \arrow[d, "(x_1, x_2)"'] & {[x : U, y : \mathbf{t}_U(x)]} \arrow[d, "(x)"] \\
{[x_1 : U, x_2 : U]} \arrow[r, "(x_1 \otimes x_2)"'] & {[x : U]}
\end{tikzcd}$$

*Proof.* This is routine, as follows. Suppose we have a commuting diagram like this

$$\begin{tikzcd}
\Gamma \arrow[r, "(c, f)"] \arrow[d, "(a, b)"'] & {[x : U, y : \mathbf{t}_U(x)]} \arrow[d, "(x)"] \\
{[x_1 : U, x_2 : U]} \arrow[r, "(x_1 \Rightarrow x_2)"'] & {[x : U]}
\end{tikzcd}$$

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

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

Therefore

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

It follows that

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

The proofs for the other cases are similar. □

The categorical implication of Lemma 7.1 is obvious once we notice the facts in the next lemma.

**Lemma 7.2.**

(i) Both $[x : \mathbf{t}_U(\diamond)]$ and $[]$ are terminal objects.

(ii) $[x_1 : U, x_2 : U]$ is the product of $[x : U]$ and $[x : U]$.

(iii) $[x_1 : U, x_2 : U, y : \mathbf{t}_U(x_i)] \xrightarrow{(x_1,x_2)} [x_1 : U, x_2 : U]$ is the pulling-back of $[x : U, y : \mathbf{t}_U(x)] \xrightarrow{(x)} [x : U]$ along the projection $[x_1 : U, x_2 : U] \xrightarrow{(x_i)} [x : U]$.

(iv) $[x_1 : U, x_2 : U, f : \mathbf{t}_U(x_1)\times\mathbf{t}_U(x_2)]$ (or $[x_1 : U, x_2 : U, f : \mathbf{t}_U(x_1)\rightarrow\mathbf{t}_U(x_2)]$) is the product (or exponential) of $[x_1 : U, x_2 : U, y : \mathbf{t}_U(x_1)]$ and $[x_1 : U, x_2 : U, y : \mathbf{t}_U(x_2)]$ in the fibre over $[x_1 : U, x_2 : U]$.

We conclude that the internal full subcategory $\dot{\mathsf{U}}$ has explicit cartesian closed structure.

In the case of $\mathscr{E}_{PL}$, this property holds for both $\dot{\mathsf{U}}$ and $\dot{\mathsf{K}}$. But now the diagram associated with $\dot{\mathsf{U}}$ needs to be modified as follows: in the first diagram, $\diamond$ should be replaced by $\mathsf{in}(\diamond)$; in the second diagram, when $\dot{\otimes}$ is $\rightarrow$, $\otimes$ is $\mathsf{in}(\mathsf{out}(x_1) \Rightarrow \mathsf{out}(x_2))$ and $m$ is $\mathsf{abs}_{\mathsf{out}(x_1),\mathsf{out}(x_2)}(f)$; when $\dot{\otimes}$ is $\times$, $\otimes$ is $\mathsf{in}(\mathsf{out}(x_1)\wedge\mathsf{out}(x_2))$ and $m$ is $\mathsf{pair}_{\mathsf{out}(x_1),\mathsf{out}(x_2)}(f)$. In addition, we have the following lemma.

**Lemma 7.3.** In the encoding $\mathscr{E}_{PL}$ of the higher-order $\lambda$-calculus, the following is a pullback in the base category of the fibration $(\lambda_{TT}, \mathscr{E}_{PL})$ for any $\vdash_{\Omega;\Sigma} \kappa : K$.

$$\begin{tikzcd}
{[x : \mathbf{t}_K(\kappa)\rightarrow U, z : \Pi y{:}\mathbf{t}_K(\kappa).\mathbf{t}_U(xy)]} \arrow[r, "{(\mathsf{in}(\dot{\forall}_\kappa(\mathsf{out}\circ x)), \mathsf{Abs}_{\kappa,\mathsf{out}\circ x}(z))}"] \arrow[d, "(x)"'] & {[x : U, y : \mathbf{t}_U(x)]} \arrow[d, "(x)"] \\
{[x : \mathbf{t}_K(\kappa)\rightarrow U]} \arrow[r, "{(\mathsf{in}(\dot{\forall}_\kappa(\mathsf{out}\circ x)))}"'] & {[x : U]}
\end{tikzcd}$$

*Proof.* Suppose the following diagram commutes.

$$\begin{tikzcd}
\Gamma \arrow[r, "{(G,t)}"] \arrow[d, "(F)"'] & {[x : U, y : \mathbf{t}_U(x)]} \arrow[d, "(x)"] \\
{[x : \mathbf{t}_K(\kappa)\rightarrow U]} \arrow[r, "{(\mathsf{in}(\dot{\forall}_\kappa(\mathsf{out}\circ x)))}"'] & {[x : U]}
\end{tikzcd}$$

Clearly $G = \mathsf{in}(\dot{\forall}_\kappa(\mathsf{out}\circ F))$. Assume the mediating morphism is $(F, ?)$. Then we have

$$\mathsf{Abs}_{\kappa,\mathsf{out}\circ F}(?) = t : \mathbf{t}_U(\mathsf{in}(\dot{\forall}_\kappa(\mathsf{out}\circ F))).$$

So

$$? = \mathsf{App}_{\kappa,\mathsf{out}\circ F}(\mathsf{Abs}_{\kappa,\mathsf{out}\circ F}(?)) = \mathsf{App}_{\kappa,\mathsf{out}\circ F}(t).$$

This completes the proof. □

**Lemma 7.4.**

(i) $[x : \mathbf{t}_K(\kappa) \to U]$ is the exponential of $[x : \mathbf{t}_K(\kappa)]$ and $[x : U]$.

(ii) $[x : \mathbf{t}_K(\kappa) \to U, y : \mathbf{t}_K(\kappa)]$ is the product of $[x : \mathbf{t}_K(\kappa) \to U]$ and $[x : \mathbf{t}_K(\kappa)]$.

(iii) $[x : \mathbf{t}_K(\kappa) \to U, y : \mathbf{t}_K(\kappa)] \xrightarrow{(xy)} [x : U]$ is the evaluation map.

(iv) $[x : \mathbf{t}_K(\kappa) \to U, y : \mathbf{t}_K(\kappa), z : \mathbf{t}_U(xy)] \xrightarrow{(x,y)} [x : \mathbf{t}_K(\kappa) \to U, y : \mathbf{t}_K(\kappa)]$ is the pulling-back of $[x : U, y : \mathbf{t}_U(x)] \xrightarrow{(x)} [x : U]$ along the evaluation map in (iii).

(v) $[x : \mathbf{t}_K(\kappa) \to U, z : \Pi y{:}\mathbf{t}_K(\kappa).\mathbf{t}_U(xy)] \xrightarrow{(x)} [x : \mathbf{t}_K(\kappa) \to U]$ is obtained by applying to $[x : \mathbf{t}_K(\kappa) \to U, y : \mathbf{t}_K(\kappa), z : \mathbf{t}_U(xy)] \xrightarrow{(x,y)} [x : \mathbf{t}_K(\kappa) \to U, y : \mathbf{t}_K(\kappa)]$ the right adjoint to the relabelling functor over $[x : \mathbf{t}_K(\kappa) \to U, y : \mathbf{t}_K(\kappa)] \xrightarrow{(x)} [x : \mathbf{t}_K(\kappa) \to U]$.

In summary, Lemmas 7.3 and 7.4 together say that the internal category $\dot{\mathsf{U}}$ has explicit products over types lifted from 'kinds'.

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 $\lambda_{TT}$, 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.

## 8. 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 any specific proof theoretical properties of $\lambda_{TT}$.

In this section we are interested in two kinds of models: the closed term models and the open 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 $\lambda$-calculus is a cartesian closed category (Lambek and Scott 1986). On the other hand, the open term model of the same calculus is a fibration $\mathscr{L}_\lambda : \mathscr{T}_\lambda \longrightarrow \mathscr{C}_\lambda$. Here $\mathscr{C}_\lambda$ is the category of contexts and their realizations, and a morphism over $\Gamma \in \mathscr{C}_\lambda$ is an equivalence class of terms $\Gamma \vdash f : A \to B$. In the case of dependent typed calculi, however, we always talk about open term models.

Because of the presence of the equational contexts, the syntactic adequacy used in this paper is slightly different from that defined in Harper *et al.* (1987, 1993).

### 8.1. *Adequacy of $\mathscr{E}_\lambda$ and definability of simply typed $\lambda$-calculus*

Let $\mathscr{L}^\lambda$ and $\mathscr{L}_\lambda : \mathscr{T}_\lambda \longrightarrow \mathscr{C}_\lambda$ be the closed term model and the open term model of the simply typed $\lambda$-calculus, respectively. Here $\mathscr{L}^\lambda$ has types as objects. A morphism from $A$ to $B$ is a closed term of type $A \to B$. The category $\mathscr{C}_\lambda$ is the category of contexts and context realizations. An object in $\mathscr{T}_\lambda$ is a pair $(\Gamma, A)$ of a context and a type. A morphism in $\mathscr{T}_\lambda$ is a pair $(r, f) : (\Gamma, A) \to (\Delta, B)$ such that $\Gamma \xrightarrow{r} \Delta$ is a context realization and $\Gamma \vdash f : A \to B$. The functor $\mathscr{L}_\lambda$ is the first projection. The canonical cartesian lifting over $\Gamma \xrightarrow{r} \Delta$ with respect to $(\Delta, B)$ is $(r, \lambda x:B.x)$. The fibration $\mathscr{L}_\lambda$ has a fibred cartesian closed structure. For example, the product and exponential of the objects $(\Gamma, A)$ and $(\Gamma, B)$ in the fibre over $\Gamma$ are $(\Gamma, A\times B)$ and $(\Gamma, A\to B)$, respectively.

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

$$x_1 : A_1, \dots, x_n : A_n \vdash a : A \tag{1}$$

in the simply typed $\lambda$-calculus to a judgement (in this section, we are not going to distinguish notationally the universes (signatures) defined in the examples given in Section 5)

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

in $\lambda_{TT}$. We now define the translation inductively as follows:

$$\begin{array}{rrcl}
 & \hat{\mathbf{1}} & \stackrel{\text{def}}{=} & \diamond \\
 & \widehat{A\times B} & \stackrel{\text{def}}{=} & \hat{A}\wedge\hat{B} \\
 & \widehat{A\to B} & \stackrel{\text{def}}{=} & \hat{A}\Rightarrow\hat{B} \\
 & \hat{x} & \stackrel{\text{def}}{=} & x \\
 & \hat{\star} & \stackrel{\text{def}}{=} & \heartsuit \\
(f : A\to B,\ a : A) & \widehat{fa} & \stackrel{\text{def}}{=} & \mathsf{app}_{\hat{A},\hat{B}}(\hat{f}, \hat{a}) \\
(b : B) & \widehat{\lambda x{:}A.b} & \stackrel{\text{def}}{=} & \mathsf{abs}_{\hat{A},\hat{B}}(\lambda x{:}\mathbf{t}_U(\hat{A}).\hat{b}) \\
(a : A,\ b : B) & \widehat{\langle a, b\rangle} & \stackrel{\text{def}}{=} & \mathsf{pair}_{\hat{A},\hat{B}}(\hat{a}, \hat{b}) \\
(c : A\times B) & \widehat{\pi_1 c} & \stackrel{\text{def}}{=} & (\mathsf{pr}_1)_{\hat{A},\hat{B}}(\hat{c}) \\
(c : A\times B) & \widehat{\pi_2 c} & \stackrel{\text{def}}{=} & (\mathsf{pr}_2)_{\hat{A},\hat{B}}(\hat{c}).
\end{array}$$

The syntactic adequacy for the encoding $\mathscr{E}_\lambda$ consists of the following statements:

1 The map that sends a type $A$ to its translation $\hat{A}$ prescribes a bijective correspondence between types in the simply typed $\lambda$-calculus and judgements of the form $x_1 : \mathbf{t}_U(\hat{A}_1), \dots, x_n : \mathbf{t}_U(\hat{A}_n) \vdash_{\Omega;\Sigma} M : U$ in $\lambda_{TT}$ (with the encoding $\mathscr{E}_\lambda$) for any fixed types $A_1, \dots, A_n$ in the $\lambda$-calculus. Strictly speaking, $M$ should be an equivalence class, where the equivalence relation is induced by the definitional equality of $\lambda_{TT}$. Notice that intuitively $x_1 : \mathbf{t}_U(\hat{A}_1), \dots, x_n : \mathbf{t}_U(\hat{A}_n) \vdash_{\Omega;\Sigma} M : U$ if and only if $\vdash_{\Omega;\Sigma} M : U$. But that belongs to the proof theory of $\lambda_{TT}$.

2 The map that sends (1) onto (2) is a bijection between the judgements of the form (1) and the judgements of the form $x_1 : \mathbf{t}_U(\hat{A}_1), \dots, x_n : \mathbf{t}_U(\hat{A}_n) \vdash_{\Omega;\Sigma} M : \mathbf{t}_U(\hat{A})$. Strictly speaking, $M$ should be an equivalence class, where the equivalence relation is induced by the definitional equality of $\lambda_{TT}$.

3 $x_1 : A_1, \dots, x_n : A_n \vdash a = b : A$ if and only if $x_1 : \mathbf{t}_U(\hat{A}_1), \dots, x_n : \mathbf{t}_U(\hat{A}_n) \vdash_{\Omega;\Sigma} \hat{a} =_{Th_\lambda} \hat{b} : \mathbf{t}_U(\hat{A})$.

4 The translation is compositional, that is, $\widehat{a[b/x]}$ is syntactically the same as $\hat{a}[\hat{b}/x]$.

**Proposition 8.1.** If $\mathscr{E}_\lambda$ is syntactically adequate, then both $\mathscr{L}_\lambda$ and $\mathscr{L}^\lambda$ are internally definable by $\dot{\mathsf{U}}$ in $(\lambda_{TT}, \mathscr{E}_\lambda)$. In addition, the two interpretation maps send the fibred cartesian closed structures of $\mathscr{L}_\lambda$ and $\mathscr{L}^\lambda$, respectively, onto the fibred cartesian closed structure of $dom_{\dot{\mathsf{U}}}$ induced by the explicit cartesian closed structure of $\dot{\mathsf{U}}$.

*Proof.* Condition (ii) in Definition 3.4 is satisfied according to Proposition 6.3. This fact will not be mentioned in the proofs of Propositions 8.2 and 8.3 given below.
(i) Let us explain the functors $\mathscr{I}$ and $H$ in the following diagram.

$$\begin{tikzcd}
\mathscr{T}_\lambda \arrow[r, "H"] \arrow[d, "\mathscr{L}_\lambda"'] & {[\dot{\mathsf{U}}]} \arrow[d, "dom_{\dot{\mathsf{U}}}"] \\
\mathscr{C}_\lambda \arrow[r, "\mathscr{I}"'] & \mathscr{C}
\end{tikzcd}$$

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

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

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

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

Statement (4) means that $\mathscr{I}$ preserves composition. Statement (3) ensures that $\mathscr{I}$ is well defined as a functor and is faithful. Statement (3) implies that it is full; and Statement (1) says that it is injective on objects. So $\mathscr{I}$ is a full embedding functor.
Next, let us define the functor $H$. An object in the fibre $\mathscr{L}_\lambda^{-1}([x_1 : A_1, \dots, x_n : A_n])$ is a type $A$ in the simply typed $\lambda$-calculus. Define $H(A)$ to be the realization $(\hat{A}) : [x_1 : \mathbf{t}_U(\hat{A}_1), \dots, x_n : \mathbf{t}_U(\hat{A}_n)] \longrightarrow [x : U]$, which is essentially the sequent $[x_1 : \mathbf{t}_U(\hat{A}_1), \dots, x_n : \mathbf{t}_U(\hat{A}_n)] \vdash_{\Omega;\Sigma} \hat{A} : U$. But by the syntactic adequacy, there is a correspondence between types in the simply typed $\lambda$-calculus and sequents like $[x_1 : \mathbf{t}_U(\hat{A}_1), \dots, x_n : \mathbf{t}_U(\hat{A}_n)] \vdash_{\Omega;\Sigma} \hat{A} : U$. So $H$ establishes a bijection between the objects of $\mathscr{L}_\lambda^{-1}([x_1 : A_1, \dots, x_n : A_n])$ and those in $dom_{\dot{\mathsf{U}}}^{-1}([x_1 : \mathbf{t}_U(\hat{A}_1), \dots, x_n : \mathbf{t}_U(\hat{A}_n)])$. A morphism in $\mathscr{L}_\lambda^{-1}([x_1 : A_1, \dots, x_n : A_n])$ is a term $[x_1 : A_1, \dots, x_n : A_n] \vdash f : A \to B$. The syntactic adequacy implies that it is in bijective correspondence with $[x_1 : \mathbf{t}_U(\hat{A}_1), \dots, x_n : \mathbf{t}_U(\hat{A}_n)] \vdash_{\Omega;\Sigma} \hat{f} : \mathbf{t}_U(\hat{A} \Rightarrow \hat{B})$. But the latter is in bijective correpondence with $[x_1 : \mathbf{t}_U(\hat{A}_1), \dots, x_n : \mathbf{t}_U(\hat{A}_n)] \vdash_{\Omega;\Sigma} \lambda w : \mathbf{t}_U(\hat{A}).\mathsf{app}_{\hat{A},\hat{B}}(\hat{f}, w) : \mathbf{t}_U(\hat{A}) \to \mathbf{t}_U(\hat{B})$. Let $H(f)$ be

$$[x_1 : \mathbf{t}_U(\hat{A}_1), \dots, x_n : \mathbf{t}_U(\hat{A}_n)] \overset{(\hat{A}, \hat{B}, \lambda w : \mathbf{t}_U(\hat{A}).\mathsf{app}_{\hat{A},\hat{B}}(\hat{f}, w))}{\longrightarrow} [x : U, y : U, z : \mathbf{t}_U(x) \to \mathbf{t}_U(y)].$$

Clearly $H$ is full and faithful. It is routine to check the functoriality of $\mathscr{I}$ and $H$. Here we only show that $H$ preserves compositions. Suppose $[x_1 : A_1, \ldots, x_n : A_n] \vdash f : A \to B$ and $[x_1 : A_1, \ldots, x_n : A_n] \vdash g : B \to C$ are two sequents in the simply typed $\lambda$-calculus. Then by the definition of $H$, the composition $H(\hat{g}) \circ H(\hat{f})$ is

$$[x_1 : \mathbf{t}_U(\hat{A}_1), \ldots, x_n : \mathbf{t}_U(\hat{A}_n)] \xrightarrow{(\hat{A},\hat{C},\lambda w:\mathbf{t}_U(\hat{A}).\mathsf{app}_{\hat{B},\hat{C}}(\hat{g},\mathsf{app}_{\hat{A},\hat{B}}(\hat{f},w)))} [x : U, y : U, z : \mathbf{t}_U(x) \to \mathbf{t}_U(y)],$$

whereas $H(g \circ f)$ is

$$[x_1 : \mathbf{t}_U(\hat{A}_1), \ldots, x_n : \mathbf{t}_U(\hat{A}_n)] \xrightarrow{(\hat{A},\hat{C},\lambda w:\mathbf{t}_U(\hat{A}).\mathsf{app}_{\hat{A},\hat{C}}(\widehat{g \circ f},w))} [x : U, y : U, z : \mathbf{t}_U(x) \to \mathbf{t}_U(y)].$$

But

$$\begin{aligned}
\mathsf{app}_{\hat{A},\hat{C}}(\widehat{g \circ f}, w) &= \mathsf{app}_{\hat{A},\hat{C}}(\widehat{\lambda x : A.g(fx)}, w) \\
&= \mathsf{app}_{\hat{A},\hat{C}}(\mathsf{abs}_{\hat{A},\hat{C}}(\lambda x : \mathbf{t}_U(\hat{A}).\widehat{g(fx)}), w) \\
&= (\lambda x : \mathbf{t}_U(\hat{A}).\widehat{g(fx)})w \\
&= \widehat{g(fw)} \\
&= \mathsf{app}_{\hat{B},\hat{C}}(\hat{g}, \widehat{fw}) \\
&= \mathsf{app}_{\hat{B},\hat{C}}(\hat{g}, \mathsf{app}_{\hat{A},\hat{B}}(\hat{f}, w)).
\end{aligned}$$

We are done.
The diagram is also a pullback, so we conclude that $\mathscr{L}_\lambda \propto (\lambda_{TT}, \mathscr{E}_\lambda)$.
The verification that the interpretation map sends the fibred cartesian closed structure of $\mathscr{L}_\lambda$ onto the fibred cartesian closed structure of $dom_{\dot{\mathsf{U}}}$ induced by the explicit cartesian closed structure of $\dot{\mathsf{U}}$ is routine but long. The following is a snapshot of it. In the fibre $\mathscr{L}_\lambda(\Gamma)$, $(\Gamma, A \to B)$ is the exponential of $(\Gamma, A)$ and $(\Gamma, B)$. $H$ sends $(\Gamma, A \to B)$ onto $\hat{\Gamma} \xrightarrow{(\hat{A} \Rightarrow \hat{B})} [x : U]$. The latter is the exponential of $\hat{\Gamma} \xrightarrow{(\hat{A})} [x : U]$ and $\hat{\Gamma} \xrightarrow{(\hat{B})} [x : U]$ by Lemma 7.1.
(ii) For the closed term model $\mathscr{L}^\lambda$, a similar argument shows that the diagram below is a pullback,

$$\begin{tikzcd}
\mathscr{L}^\lambda \arrow[r, "H"] \arrow[d, "!"'] & {[\dot{\mathsf{U}}]} \arrow[d, "dom_{\dot{\mathsf{U}}}"] \\
\star \arrow[r, "{[]}"'] & \mathscr{C}
\end{tikzcd}$$

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

### 8.2. *Adequacy of $\mathscr{E}_{PL}$ and definability of polymorphic $\lambda$-calculus*

For higher-order polymorphic $\lambda$-calculus, there are two open term models: that of kinds and constructors $\mathscr{L}^K_{PL} : \mathscr{T}^K_{PL} \longrightarrow \mathscr{C}^K_{PL}$, and that of types and objects over the empty kinding context $\mathscr{L}^T_{PL} : \mathscr{T}^T_{PL} \longrightarrow \mathscr{C}^T_{PL}$. An object in $\mathscr{C}^T_{PL}$ is a context $x_1 : A_1, \ldots, x_n : A_n$, where $[] \vdash A_1, \ldots, [] \vdash A_n$. Let $\mathscr{L}^{PL} : \mathscr{T}^{PL} \longrightarrow \mathscr{C}^{PL}$ be the closed term model. Here the objects of $\mathscr{C}^{PL}$ are closed kinds; a morphism from object $K_1$ to object $K_2$ is a closed constructor $F : K_1{\to}K_2$. An object in $\mathscr{T}^{PL}$ over $K$ is of the form $x : K \vdash A : Type$. A morphism from $x_1 : K_1 \vdash A_1 : Type$ to $x_2 : K_2 \vdash A_2 : Type$ is a pair $(F : K_1{\to}K_2, \vdash_{[x_1:K_1]} f : A_1{\to}A_2[Fx_1/x_2])$. Both $\mathscr{L}^K_{PL}$ and $\mathscr{L}^T_{PL}$ have a fibred cartesian closed structure. It is well known that $\mathscr{L}^{PL}$ is equipped with a $PL$-categorical structure (Seely 1987). Roughly, this amounts to saying that the base category $\mathscr{C}^{PL}$ has a cartesian closed structure and $\mathscr{L}^{PL}$ has a fibred cartesian closed structure; in addition the fibration has right adjoints to weakening functors and these right adjoints satisfy the *Beck–Chevalley condition.*

We now define the translation inductively as follows:

— kind

$$\begin{aligned}
\hat{\top} &\overset{\mathrm{def}}{=} \dot{\diamond}\\
\widehat{Type} &\overset{\mathrm{def}}{=} T\\
\widehat{K_1{\times}K_2} &\overset{\mathrm{def}}{=} \hat{K}_1\dot{\wedge}\hat{K}_2\\
\widehat{K_1{\to}K_2} &\overset{\mathrm{def}}{=} \hat{K}_1\dot{\Rightarrow}\hat{K}_2
\end{aligned}$$

— constructor

$$\begin{array}{rrcl}
 & \hat{x} & \overset{\mathrm{def}}{=} & x\\
 & \widehat{\star_\top} & \overset{\mathrm{def}}{=} & \dot{\heartsuit}\\
(F : K_1{\to}K_2,\ C : K_1) & \widehat{FC} & \overset{\mathrm{def}}{=} & \dot{\mathsf{app}}_{\hat{K}_1,\hat{K}_2}(\hat{F},\hat{C})\\
(C : K_2) & \widehat{\lambda x{:}K_1.C} & \overset{\mathrm{def}}{=} & \dot{\mathsf{abs}}_{\hat{K}_1,\hat{K}_2}(\lambda x{:}\mathbf{t}_K(\hat{K}_1).\hat{C})\\
(C : K_1,\ D : K_2) & \widehat{\langle C,D\rangle} & \overset{\mathrm{def}}{=} & \dot{\mathsf{pair}}_{\hat{K}_1,\hat{K}_2}(\hat{C},\hat{D})\\
(C : K_1{\times}K_2) & \widehat{\pi_1 C} & \overset{\mathrm{def}}{=} & (\dot{\mathsf{pr}}_1)_{\hat{K}_1,\hat{K}_2}(\hat{C})\\
(C : K_1{\times}K_2) & \widehat{\pi_2 C} & \overset{\mathrm{def}}{=} & (\dot{\mathsf{pr}}_2)_{\hat{K}_1,\hat{K}_2}(\hat{C})
\end{array}$$

type (special constructor)

$$\begin{aligned}
\hat{\mathbf{1}} &\overset{\mathrm{def}}{=} \diamond\\
\widehat{A{\times}B} &\overset{\mathrm{def}}{=} \hat{A}\wedge\hat{B}\\
\widehat{A{\to}B} &\overset{\mathrm{def}}{=} \hat{A}\Rightarrow\hat{B}\\
\widehat{\forall x{:}K.A} &\overset{\mathrm{def}}{=} \dot{\forall}_{\hat{K}}(\lambda x{:}\mathbf{t}_K(\hat{K}).\hat{A})
\end{aligned}$$

— object

$$\begin{aligned}
\hat{x} &\overset{\mathrm{def}}{=} x\\
\widehat{\star_{\mathbf{1}}} &\overset{\mathrm{def}}{=} \heartsuit
\end{aligned}$$

$$
\begin{array}{rrcl}
(f : A{\to}B,\ a : A) & \widehat{fa} & \overset{\text{def}}{=} & \mathsf{app}_{\hat{A},\hat{B}}(\hat{f},\hat{a}) \\
(b : B) & \widehat{\lambda x{:}A.b} & \overset{\text{def}}{=} & \mathsf{abs}_{\hat{A},\hat{B}}(\lambda x{:}\mathbf{t}_U(\mathsf{in}(\hat{A})).\hat{b}) \\
(a : A,\ b : B) & \widehat{\langle a,b\rangle} & \overset{\text{def}}{=} & \mathsf{pair}_{\hat{A},\hat{B}}(\hat{a},\hat{b}) \\
(c : A{\times}B) & \widehat{\pi_1 c} & \overset{\text{def}}{=} & (\mathsf{pr}_1)_{\hat{A},\hat{B}}(\hat{c}) \\
(c : A{\times}B) & \widehat{\pi_2 c} & \overset{\text{def}}{=} & (\mathsf{pr}_2)_{\hat{A},\hat{B}}(\hat{c}) \\
(a : A) & \Lambda x{:}K.a & \overset{\text{def}}{=} & \mathsf{Abs}_{\hat{K},\lambda x:\mathbf{t}_K(\hat{K}).\hat{A}}(\lambda x{:}\mathbf{t}_K(\hat{K}).\hat{a}) \\
(F : \forall x{:}K.A) & \widehat{FC} & \overset{\text{def}}{=} & \mathsf{App}_{\hat{K},\lambda x:\mathbf{t}_K(\hat{K}).\hat{A}}(\hat{F},\hat{C}).
\end{array}
$$

The statements for the syntactical adequacy for $\mathscr{E}_{PL}$ are similar to those for $\mathscr{E}_\lambda$:

1 For any fixed kinds $K_1,\dots,K_n$, the map that sends a kind $K$ to the judgement $x_1 : \mathbf{t}_K(\hat{K}_1),\dots,x_n : \mathbf{t}_K(\hat{K}_n) \vdash_{\Omega;\Sigma} \hat{K} : K$ is a bijection between kinds and judgements of the form $x_1 : \mathbf{t}_K(\hat{K}_1),\dots,x_n : \mathbf{t}_K(\hat{K}_n) \vdash_{\Omega;\Sigma} M : K$ .

2 For any fixed kinds $K_1,\dots,K_n$ and $K$, the map that sends a constructor judgement $x_1 : K_1,\dots,x_n : K_n \vdash C : K$ to the judgement $x_1 : \mathbf{t}_K(\hat{K}_1),\dots,x_n : \mathbf{t}_K(\hat{K}_n) \vdash_{\Omega;\Sigma} \hat{C} : \mathbf{t}_K(\hat{K})$ is a bijection between the judgements of the form $x_1 : K_1,\dots,x_n : K_n \vdash C : K$ and the judgements of the form $x_1 : \mathbf{t}_K(\hat{K}_1),\dots,x_n : \mathbf{t}_K(\hat{K}_n) \vdash_{\Omega;\Sigma} M : \mathbf{t}_K(\hat{K})$.

3 For any fixed kinds $K_1,\dots,K_n$ and $K$, $x_1 : K_1,\dots,x_n : K_n \vdash C = D : K$ iff $x_1 : \mathbf{t}_K(\hat{K}_1),\dots,x_n : \mathbf{t}_K(\hat{K}_n) \vdash_{\Omega;\Sigma} \hat{C} =_{Th_{PL}} \hat{D} : \mathbf{t}_K(\hat{K})$.

4 For any fixed types $A_1,\dots,A_n$ valid under the kinding context $[y_1 : K_1,\dots,y_m : K_m]$, the map that sends a judgement $y_1 : K_1,\dots,y_m : K_m \vdash A : Type$ to the judgement $y_1 : \mathbf{t}_K(\hat{K}_1),\dots,y_m : \mathbf{t}_K(\hat{K}_m), x_1 : \mathbf{t}_U(\mathsf{in}(\hat{A}_1)),\dots,x_n : \mathbf{t}_U(\mathsf{in}(\hat{A}_n)) \vdash_{\Omega;\Sigma} \hat{A} : \mathbf{t}_K(T)$ is a bijection between judgements of the form $y_1 : K_1,\dots,y_m : K_m \vdash A : Type$ and the judgements of the form $y_1 : \mathbf{t}_K(\hat{K}_1),\dots,y_m : \mathbf{t}_K(\hat{K}_m), x_1 : \mathbf{t}_U(\mathsf{in}(\hat{A}_1)),\dots,x_n : \mathbf{t}_U(\mathsf{in}(\hat{A}_n)) \vdash_{\Omega;\Sigma} M : \mathbf{t}_K(T)$.

5 For any fixed types $A_1,\dots,A_n$ and $A$ valid under the kinding context $[y_1 : K_1,\dots,y_m : K_m]$, the map that sends an object judgement $x_1 : A_1,\dots,x_n : A_n \vdash_{[y_1:K_1,\dots,y_m:K_m]} a : A$ to the judgement $y_1 : \mathbf{t}_K(\hat{K}_1),\dots,y_m : \mathbf{t}_K(\hat{K}_m), x_1 : \mathbf{t}_U(\mathsf{in}(\hat{A}_1)),\dots,x_n : \mathbf{t}_U(\mathsf{in}(\hat{A}_n)) \vdash_{\Omega;\Sigma} \hat{a} : \mathbf{t}_U(\mathsf{in}(\hat{A}))$ is a bijection between the judgements of the form $x_1 : A_1,\dots,x_n : A_n \vdash_{[y_1:K_1,\dots,y_m:K_m]} a : A$ and the judgements of the form $y_1 : \mathbf{t}_K(\hat{K}_1),\dots,y_m : \mathbf{t}_K(\hat{K}_m), x_1 : \mathbf{t}_U(\mathsf{in}(\hat{A}_1)),\dots,x_n : \mathbf{t}_U(\mathsf{in}(\hat{A}_n)) \vdash_{\Omega;\Sigma} M : \mathbf{t}_U(\mathsf{in}(\hat{A}))$ .

6 For any fixed types $A_1,\dots,A_n$ and $A$ valid under the kinding context $y_1 : K_1,\dots,y_m : K_m$, we have $x_1 : A_1,\dots,x_n : A_n \vdash_{[y_1:K_1,\dots,y_m:K_m]} a = b : A$ iff $y_1 : \mathbf{t}_K(\hat{K}_1),\dots,y_m : \mathbf{t}_K(\hat{K}_m), x_1 : \mathbf{t}_U(\mathsf{in}(\hat{A}_1)),\dots,x_n : \mathbf{t}_U(\mathsf{in}(\hat{A}_n)) \vdash_{\Omega;\Sigma} \hat{a} =_{Th_{PL}} \hat{b} : \mathbf{t}_U(\mathsf{in}(\hat{A}))$.

7 The translation is compositional.

**Proposition 8.2.** If the encoding $\mathscr{E}_{PL}$ is syntactically adequate, then

(i) $\mathscr{L}^T_{PL}$ and $\mathscr{L}^K_{PL}$ are internally definable in $(\lambda_{TT},\mathscr{E}_{PL})$ by $\dot{\mathsf{U}}$ and $\dot{\mathsf{K}}$, respectively, and the interpretation maps for both $\mathscr{L}^T_{PL}$ and $\mathscr{L}^K_{PL}$ preserve fibred cartesian closed structure;

(ii) $\mathscr{L}^{PL}$ is internally definable in $(\lambda_{TT},\mathscr{E}_{PL})$ by $\dot{\mathsf{U}}$ and the interpretation map $(H,\mathscr{I})$ from $\mathscr{T}^{PL} \xrightarrow{\mathscr{L}^{PL}} \mathscr{C}^{PL}$ to $dom_{\dot{\mathsf{U}}}$ preserves the $PL$-categorical structure.

*Proof.* (i) By Proposition 8.1, both $\mathscr{L}^K_{PL}$ and $\mathscr{L}^T_{PL}$ are internally definable in $(\lambda_{TT}, \mathscr{E}_{PL})$ and the interpretation maps preserve the fibred cartesian closed structures.

$$\begin{tikzcd}
\mathscr{T}^K_{PL} \arrow[r, "H_K"] \arrow[d, "\mathscr{L}^K_{PL}"'] & {[\dot{\mathsf{K}}]} \arrow[d, "dom_{\dot{\mathsf{K}}}"] \\
\mathscr{C}^K_{PL} \arrow[r, "\mathscr{I}_K"'] & \mathscr{C}
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathscr{T}^T_{PL} \arrow[r, "H_T"] \arrow[d, "\mathscr{L}^T_{PL}"'] & {[\dot{\mathsf{U}}]} \arrow[d, "dom_{\dot{\mathsf{U}}}"] \\
\mathscr{C}^T_{PL} \arrow[r, "\mathscr{I}_T"'] & \mathscr{C}
\end{tikzcd}$$

Notice that the assumption that the types and objects appearred in $\mathscr{L}^T_{PL}$ do not contain constructor variables is important; otherwise the situation would not be similar to the one described in the proof of Proposition 8.1.
(ii) First we need to show that we have a pullback diagram.

$$\begin{tikzcd}
\mathscr{T}^{PL} \arrow[r, "H"] \arrow[d, "\mathscr{L}^{PL}"'] & {[\dot{\mathsf{U}}]} \arrow[d, "dom_{\dot{\mathsf{U}}}"] \\
\mathscr{C}^{PL} \arrow[r, "\mathscr{I}"'] & \mathscr{C}
\end{tikzcd}$$

In the diagram, $\mathscr{I}$ sends an object $K$ (a kind) to $[x : \mathbf{t}_K(\hat{K})]$ and a map $F : K_1 \to K_2$ onto $[x : \mathbf{t}_K(\hat{K}_1)] \overset{(\dot{\mathsf{app}}_{\hat{K_1},\hat{K_2}}(\hat{F},x))}{\longrightarrow} [x : \mathbf{t}_K(\hat{K}_2)]$. The functor $H$ sends an object $x : K \vdash A : Type$ onto $[x : \mathbf{t}_K(\hat{K})] \overset{(\mathsf{in}(\hat{A}))}{\longrightarrow} [x : U]$ and a map $(F : K_1 {\to} K_2, \vdash_{[x:K_1]} f : A_1 {\to} A_2[F]) : (x : K_1 \vdash A_1 : Type) \longrightarrow (x : K_2 \vdash A_2 : Type)$ onto a pair whose first component is

$$[x : \mathbf{t}_K(\hat{K}_1)] \overset{(\dot{\mathsf{app}}_{\hat{K_1},\hat{K_2}}(\hat{F},x))}{\longrightarrow} [x : \mathbf{t}_K(\hat{K}_2)]$$

and whose second component is

$$[x : \mathbf{t}_K(\hat{K}_1)] \overset{(\mathsf{in}(\hat{A}_1), \mathsf{in}(\widehat{A_2[F]}), \lambda x{:}\mathbf{t}_U(\mathsf{in}(\hat{A})).\mathsf{app}_{\hat{A}_1, \widehat{A_2[F]}}(\hat{f},x))}{\longrightarrow} [x : U, y : U, z : \mathbf{t}_U(x) {\to} \mathbf{t}_U(y)]).$$

The verification that $(H, \mathscr{I})$ preserves fibred cartesian closed structure and that $\mathscr{I}$ preserves the cartesian closed structure in the base category are routine. The proof that it preserves right adjoints to weakening functors is similar to that in the proof of Proposition 8.3 given below. □

## 8.3. *Adequacy of $\mathscr{E}_{ML}$ and definability of Martin-Löf type theory*

Let $\mathscr{T}_{ML} \overset{\mathscr{L}_{ML}}{\longrightarrow} \mathscr{C}_{ML}$ be the term model of the Martin-Löf type theory (see Seely (1984) for details). Here $\mathscr{C}_{ML}$ is the category of contexts. A morphism in $\mathscr{T}_{ML}$ is a pair $(r, f) : (\Gamma \vdash$

$A) \longrightarrow (\Delta \vdash B)$, where $\Gamma \xrightarrow{r} \Delta$ is a context realization and $\Gamma \vdash f : A \rightarrow B[r]$. A display map is a context realization of the form $[x_1 : A_1, \ldots, x_{n+1} : A_{n+1}] \xrightarrow{(x_1,\ldots,x_n)} [x_1 : A_1, \ldots, x_n : A_n]$. It is well known that the fibration $\mathscr{L}_{ML}$ has left and right adjoints to the reindexing functors over the display maps. These right adjoints satisfy the *Beck–Chevalley condition.* For example, the right adjoint to the reindexing functor over $\Gamma, x : A \longrightarrow \Gamma$ sends the object $\Gamma, x : A \vdash B$ onto $\Gamma \vdash \Pi x{:}A.B$.

The translation from Martin-Löf's type theory to $\mathscr{E}_{ML}$ is defined inductively as follows:

— context

$$\begin{array}{rcl} \hat{[]} & \longmapsto & [] \\ \widehat{\Gamma, x : A} & \longmapsto & \hat{\Gamma}, x : \mathbf{t}(\hat{A}), \end{array}$$

where $\hat{\Gamma} \vdash_{\Omega;\Sigma} \hat{A} : Set$ is the translation of $\Gamma \vdash A$;

— type

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

where $f : \Pi x{:}A.B$, $a : A$ and $c : \Sigma x{:}A.B$.

The syntactic adequacy consists of the following statements:

1 The map that sends $\Gamma \vdash A$ onto $\hat{\Gamma} \vdash_{\Omega;\Sigma} \hat{A} : Set$ is a bijection between judgements of the form $\Gamma \vdash A$ and judgements of the form $\hat{\Gamma} \vdash_{\Omega;\Sigma} M : Set$.

2 $\Gamma \vdash A = B$ if and only if $\hat{\Gamma} \vdash_{\Omega;\Sigma} \hat{A} =_{Th_{ML}} \hat{B} : Set$.

3 The map that sends $\Gamma \vdash a : A$ onto $\hat{\Gamma} \vdash_{\Omega;\Sigma} \hat{a} : \mathbf{t}(\hat{A})$ is a bijection between the judgements of the form $\Gamma \vdash a : A$ and the judgements of the form $\hat{\Gamma} \vdash_{\Omega;\Sigma} M : \mathbf{t}(\hat{A})$.

4 $\Gamma \vdash a = b : A$ if and only if $\hat{\Gamma} \vdash_{\Omega;\Sigma} \hat{a} =_{Th_{ML}} \hat{b} : \mathbf{t}(\hat{A})$.

5 The translation is compositional.

**Proposition 8.3.** If the encoding $\mathscr{E}_{ML}$ is syntactically adequate, the term model is internally definable in $(\lambda_{TT}, \mathscr{E}_{ML})$ by $\dot{S}et$. In addition, the interpretation map $(M, \mathscr{I})$ preserves the left and right adjoints along the display maps.

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

$$\begin{tikzcd}
\mathscr{T}_{ML} \arrow[r, "M"] \arrow[d, "\mathscr{L}_{ML}"'] & {[\dot{S}et]} \arrow[d, "dom_{\dot{S}et}"] \\
\mathscr{C}_{ML} \arrow[r, "\mathscr{I}"'] & \mathscr{C}
\end{tikzcd}$$

The functor $M : \mathscr{T}_{ML} \longrightarrow [\dot{S}et]$ is defined as follows, bearing in mind that a map from $\hat{\Gamma} \overset{(\hat{A})}{\rightarrow} \widehat{Set}_0$ to $\hat{\Delta} \overset{(\hat{B})}{\rightarrow} \widehat{Set}_0$ is a pair $(\hat{\Gamma} \xrightarrow{m} \hat{\Delta}, \hat{\Gamma} \xrightarrow{m'} \widehat{Set}_1)$:

$$\begin{array}{rcl}
\Gamma \vdash A & \longmapsto & \hat{\Gamma} \xrightarrow{(\hat{A})} \dot{S}et_0 \\
(\Gamma \vdash A) \xrightarrow{(r,f)} (\Delta \vdash B) & \longmapsto & (\hat{r}, (\hat{A}, \widehat{B[r]}, \lambda x : \mathbf{t}(\hat{A}). \bullet (\hat{A}, \lambda w : \mathbf{t}(\hat{A}).\widehat{B[r]}, \hat{f}, x))) \\
 & & : (\hat{\Gamma} \overset{(\hat{A})}{\rightarrow} \dot{S}et_0) \longrightarrow (\hat{\Delta} \overset{(\hat{B})}{\rightarrow} \dot{S}et_0),
\end{array}$$

where $\hat{\Gamma} \vdash_{\Omega;\Sigma} \hat{A} : Set$, $\hat{\Gamma} \vdash_{\Omega;\Sigma} \widehat{B[r]} : Set$, $\hat{\Gamma} \xrightarrow{\hat{r}} \hat{\Delta}$ and $\hat{\Gamma} \vdash_{\Omega;\Sigma} \hat{f} : \mathbf{t}(\pi(\hat{A}, \lambda w : \mathbf{t}(\hat{A}).\widehat{B[r]}))$. Syntactic adequacy implies that $M$ so defined is full, faithful and injective on objects (it also maps canonical cartesian liftings to canonical cartesian liftings). It can be readily verified that the diagram is also a pullback. We still need to show that $M$ preserves left and right adjoints along display maps in $\mathscr{C}_{ML}$. By the translation we have given, the actions of $M : \mathscr{T}_{ML} \longrightarrow [\dot{S}et]$ on $\Gamma, x : A \vdash B$ and $\Gamma \vdash \Pi x : A.B$, respectively, are as follows:

$$\begin{array}{rcl}
\Gamma, x : A \vdash B & \longmapsto & \hat{\Gamma}, x : \mathbf{t}(\hat{A}) \xrightarrow{(\hat{B})} \dot{S}et_0 \\
\Gamma \vdash \Pi x : A.B & \longmapsto & \hat{\Gamma} \xrightarrow{(\pi(\hat{A}, \lambda x : \mathbf{t}(\hat{A}).\hat{B}))} \dot{S}et_0.
\end{array}$$

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

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

and its inverse

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

We conclude that $M$ preserves right adjoints to reindexing functors over display maps. The fact that it also preserves left adjoints to reindexing functors over display maps can be similarly established. □

## 8.4. *Conclusion*

In all three examples above we have assumed that the number of constant types is finite. Let us look at an example where there is an infinite number of constant types. Forgetting the encoding of constant types, $\mathscr{E}_\lambda$ as it stands is an encoding of simply typed $\lambda$-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 $\lambda$-calculus is now coded up 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 occurring in $A$. It can be routinely checked that Proposition 8.1 still holds for $\mathscr{L}_\lambda$. But the functor $\mathscr{C}_\lambda \xrightarrow{\mathscr{I}} \mathscr{C}$ has to be redefined. It now 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 occurring in $A_1, \ldots, A_m$ and $A$.

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

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

So, to formulate an object language in $\lambda_{TT}$ is to identify a frame language that contains an internal version of the represented language. The richer the object language, the richer the frame language. A correct formulation of the encoded language and the frame language involves a build-up of an internal definability relationship between the two languages. The function of $\lambda_{TT}$ is to provide a framework on which this frame language is built. The good properties of $\lambda_{TT}$ ensure that to design the frame language is the same thing as to give an adequate representation of the object language in $\lambda_{TT}$.

## 9. The notion of models for logical frameworks

We now turn our attention to semantics. There are two levels of semantics. The model theory of $\lambda_{TT}$ has been well established, see, for instance, Hyland and Pitts (1989). A categorical model consists of a category $\mathscr{B}$ with finite products and a collection $\mathbf{D}$ of display maps. The fibration $\mathscr{B}/\mathbf{D} \xrightarrow{cod} \mathscr{B}$ must be a fibred cartesian closed category and be complete relative to $\mathbf{D}$. A generic judgement $x : U \vdash_{\Omega;\Sigma} \mathbf{t}_U(x)$ must be interpreted as a display map.

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 $\mathscr{L}$ is a typed calculus and $\mathscr{E} = (\Omega, \Sigma, Th)$ is an adequate encoding of $\mathscr{L}$ in $\lambda_{TT}$. There are two questions one can ask:

1 Given an interpretation of the frame language $(\lambda_{TT}, \mathscr{E})$, what does the interpretation of $\mathscr{E}$ constitute? Arguably, we are only interested in those interpretations whose restrictions to $\mathscr{E}$ constitute internal models of $\mathscr{L}$.
2 Given a model of $\mathscr{L}$, can we fully embed it into a model of $(\lambda_{TT}, \mathscr{E})$?

A categorical interpretation of $\mathscr{E} = (\Omega, \Sigma, Th)$ consists of a sound categorical interpretation of $\lambda_{TT}$ and an interpretation of constants in $\Omega$ and $\Sigma$. We say the interpretation of $\mathscr{E}$ is *sound* if it validates all the definitional equations in $Th$. We say a sound interpretation of $\mathscr{E}$ is *good* if the denotations of the constants in $\Omega$ and $\Sigma$ under this interpretation form an internal model of $\mathscr{L}$. We say the encoding $\mathscr{E}$ is *semantically adequate* if every sound interpretation of $\mathscr{E}$ is good.

**Proposition 9.1.** If both $\mathscr{E}_\lambda$ and $\mathscr{E}_{PL}$ are syntactically adequate, both are semantically adequate.

*Proof.* Lemma 7.1 (7.3) holds for any sound interpretation of $\mathscr{E}_\lambda$ ($\mathscr{E}_{PL}$), and the results follow. $\square$

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

$$\text{sound interpretations of } \mathscr{E} \quad \longrightarrow \quad \text{sound interpretations of } \mathscr{L}.$$

We say that the encoding $\mathscr{E}$ is *semantically complete* if the above map is surjective.

**Proposition 9.2.** If both $\mathscr{E}_\lambda$ and $\mathscr{E}_{PL}$ are syntactically adequate, both are semantically complete.

*Proof.* The semantic completeness for polymorphic $\lambda$-calculus is essentially the main result of Pitts (1987). The result for simply typed $\lambda$-calculus can be proved similarly. $\square$

## 10. Applications to other typed calculi

The central notion introduced in this paper is that of internal definability. We have seen the importance of this notion for the analysis of the logical framework $\lambda_{TT}$. As a matter of fact, internal definability (codability) can be found in a range of typed calculi. It is the purpose of this section to explain how some proof theoretical questions can be rephrased in terms of internal definability or internal codability.

## 10.1. *Polymorphic λ-calculus, continued*

We have already seen in Examples 3.3 and 3.5 that the higher-order polymorphic λ-calculus is, roughly speaking, internally definable in itself. For $i \in \{n \geq 2 \mid n \in \omega\}$, we can construct a term model $\mathscr{T}^i \xrightarrow{\mathscr{L}^i} \mathscr{C}^i$ for the $i$-th polymorphic λ-calculus. Similar to $\mathscr{L}^\omega$, we have an internal category $\mathsf{C}^i$ in $\mathscr{T}^i$ and a set $\mathbf{D}^i$ of display maps such that $cod^i \dashv I^i \dashv dom^i : \mathscr{T}^i/\mathbf{D} \longrightarrow \mathscr{T}^i$ is a $D$-category and $\mathsf{C}^i$ is an internal full subcategory of $cod^i$. Moreover, the following diagram is a pullback.

$$\begin{tikzcd}
\mathscr{T}^i \arrow[r, "H^i"] \arrow[d, "\mathscr{L}^i"'] & {[\mathsf{C}^i]} \arrow[d, "dom_{\mathsf{C}^i}"] \\
\mathscr{C}^i \arrow[r, "\top^i"'] & \mathscr{T}^i
\end{tikzcd}$$

Since the $i$-th polymorphic λ-calculus is a sublanguage of the $(i+1)$-th polymorphic λ-calculus, there is a natural cartesian map from $\mathscr{L}^i$ to $\mathscr{L}^{i+1}$ that sends everything to itself, so to speak. This cartesian map preserves all the relevant categorical structures. But is it full and faithful? In view of the above pullback diagram, the question can be couched in more informative terms:

Is the $i$-th polymorphic λ-calculus internally definable in the $(i+1)$-th polymorphic λ-calculus by $\mathsf{C}^{i+1}$ via the natural interpretation map?

As far as we know, this question is open. Notice that $\mathsf{C}^{i+1}$ has enough explicit categorical structures to be a model of the $i$-th polymorphic λ-calculus.

## 10.2. *Martin-Löf type theory*

There are two ways of presenting Martin-Löf's type theory. One approach is to define the type theory in a metalanguage. Typed calculi so defined are often *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*. As a monomorphic type theory is defined to be the encoding in Martin-Löf's logical framework, we have the obvious fact:

Monomorphic Martin-Löf type theories are internally definable in Martin-Löf's logical framework.

For a particular polymorphic Martin-Löf type theory, one can ask the question: is it the same as a monomorphic Martin-Löf type theory? Putting it differently, the question asks if the polymorphic Martin-Löf type theory can be adequately represented in Martin-Löf's logical framework. In our terminology, the question goes as follows:

Is the polymorphic Martin-Löf type theory internally definable in Martin-Löf's logical framework?

There are polymorphic Martin-Löf type theories that are not internally definable in Martin-Löf's logical framework. For example, type theories with 'extensional equality types' are not internally definable in Martin-Löf's logical framework (Nordström *et al.* 1990).

Suppose $ML^0$ is a Martin-Löf type theory without universes and $ML^1$ is a Martin-Löf type theory with a first universe $U$ that reflects the type structures of $ML^0$ on the object level ($ML^1$ must also contain a copy of the type structures of $ML^0$ on type level). The reader should refer to Nordström *et al.* (1990) for a detailed account. The point we would like to make here is that $U$ induces an internal category in the category of contexts of $ML^1$. There is a structure-preserving cartesian map from the term model of $ML^0$ to that of $ML^1$ that sends a type in $ML^0$ to its reflection. A question about how faithful $ML^0$ is reflected in $ML^1$ is whether $ML^1$ equates more objects of $ML^0$ than $ML^0$? This question can be rephrased as:

Is $ML^0$ internally codable in $ML^1$ via the said cartesian map?

If there are enough type structures in $ML^0$, then *Peano*'s fourth axiom is expressable in $ML^0$ as a type. This type is inhabited in $ML^1$ but not in $ML^0$ (Smith 1988). So $ML^0$ is in general not internally definable in $ML^1$. A natural question to ask is under what restrictions $ML^0$ is internally definable in $ML^1$.

## 10.3. *Calculus of constructions*

There are many variants of calculus of constructions. Here we take the view that a calculus of constructions is a Martin-Löf type theory with an encoded logic. The calculus of constructions we usually refer to is the one with an intended encoding of Church's higher-order logic. To emphasize that the internal logic in constructions can be seen as an encoded logic, we formulate the language in a slightly different way.

**The Ambient Calculus**

$$\frac{}{[]\ \textbf{valid}}\ \mathsf{C1} \qquad \frac{\Gamma \vdash A\ \textbf{type} \quad x \notin FV(\Gamma)}{\Gamma, x : A\ \textbf{valid}}\ \mathsf{C2} \qquad \frac{\Gamma, x : A, \Gamma'\ \textbf{valid}}{\Gamma, x : A, \Gamma' \vdash x : A}\ \mathsf{Var}$$

$$\frac{\Gamma \vdash A\ \textbf{type} \quad \Gamma, x : A \vdash B\ \textbf{type}}{\Gamma \vdash (x{:}A)B\ \textbf{type}}\ \mathsf{Prod} \qquad \frac{\Gamma \vdash M : (x{:}A)B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B[N/x]}\ \mathsf{App}$$

$$\frac{\Gamma \vdash A\ \textbf{type} \quad \Gamma, x : A \vdash B\ \textbf{type} \quad \Gamma, x : A \vdash M : B}{\Gamma \vdash (x{:}A)M : (x{:}A)B}\ \mathsf{Abs}$$

**Small Type**

$$\frac{\Gamma\ \textbf{valid}}{\Gamma \vdash \mathit{Type}\ \textbf{type}}\ \mathsf{Type} \qquad \frac{\Gamma \vdash A : \mathit{Type}}{\Gamma \vdash A\ \textbf{type}}\ \mathsf{Cum}$$

$$\frac{\Gamma\ \textbf{valid}}{\Gamma \vdash \mathit{Obj} : \mathit{Type}}\ \mathsf{Obj} \qquad \frac{\Gamma\ \textbf{valid}}{\Gamma \vdash \mathit{Prop} : \mathit{Type}}\ \mathsf{Prop}$$

$$\frac{\Gamma \vdash A : \mathit{Type} \quad \Gamma, x : A \vdash B : \mathit{Type}}{\Gamma \vdash \Pi x{:}A.B : \mathit{Type}}\ \Pi \qquad \frac{\Gamma \vdash f : \Pi x{:}A.B \quad \Gamma \vdash a : A}{\Gamma \vdash fa : B[a/x]}\ \mathsf{app}$$

$$\frac{\Gamma \vdash A : \mathit{Type} \quad \Gamma, x : A \vdash B : \mathit{Type} \quad \Gamma, x : A \vdash b : B}{\Gamma \vdash \lambda x{:}A.b : \Pi x{:}A.B}\ \mathsf{abs}$$

**Proposition**

$$\frac{\Gamma \vdash P : \mathit{Prop}}{\Gamma \vdash \mathbf{Prf}(P) : \mathit{Type}}\ \mathsf{Prf} \qquad \frac{\Gamma\ \textbf{valid}}{\Gamma \vdash \forall : (X{:}\mathit{Type})(Y{:}X\to\mathit{Prop})\mathit{Prop}}\ \forall$$

$$\frac{\Gamma\ \textbf{valid}}{\Gamma \vdash \Lambda : (X{:}\mathit{Type})(Y{:}X\to\mathit{Prop})(z{:}(x{:}X)\mathbf{Prf}(Y(x)))\mathbf{Prf}(\forall(X,Y))}\ \Lambda$$

$$\frac{\Gamma\ \textbf{valid}}{\Gamma \vdash \bullet : (X{:}\mathit{Type})(Y{:}X\to\mathit{Prop})(z{:}\mathbf{Prf}(\forall(X,Y)))(x{:}X)\mathbf{Prf}(Y(x))}\ \bullet$$

The definitional equality is defined in terms of the extensional reduction rules.

The following are the components of an internal category in the category of contexts.

$$\begin{aligned}
\mathsf{C}_0 &\overset{\text{def}}{=} [P : \mathit{Prop}]\\
\mathsf{C}_1 &\overset{\text{def}}{=} [P, Q : \mathit{Prop}, f : \mathbf{Prf}(P)\to\mathbf{Prf}(Q)]\\
\mathsf{C}_2 &\overset{\text{def}}{=} [P, Q, R : \mathit{Prop}, f : \mathbf{Prf}(P)\to\mathbf{Prf}(Q), g : \mathbf{Prf}(Q)\to\mathbf{Prf}(R)]\\
d_0 &\overset{\text{def}}{=} (P) : \mathsf{C}_1\to\mathsf{C}_0\\
d_1 &\overset{\text{def}}{=} (Q) : \mathsf{C}_1\to\mathsf{C}_0\\
\mathsf{id} &\overset{\text{def}}{=} (P, P, \lambda p{:}\mathbf{Prf}(P).p)\\
\Pi_0 &\overset{\text{def}}{=} (P, Q, f) : \mathsf{C}_2\to\mathsf{C}_1\\
\Pi_1 &\overset{\text{def}}{=} (Q, R, g) : \mathsf{C}_2\to\mathsf{C}_1\\
\gamma &\overset{\text{def}}{=} (P, R, \lambda x{:}\mathbf{Prf}(P).g(fx)) : \mathsf{C}_2\to\mathsf{C}_1
\end{aligned}$$

Luo (1990b) gives a type theoretical formulation of Church's higher-order logic and shows that a particular calculus of constructions is a conservative extension of the higher-order logic. Notice that conservativity as used in Luo (1990b) is at the level of provability. A stronger property is conservativity at the level of proofs[†]. Is the calculus of constructions conservative over the higher-order logic in this stronger sense? The aim of our particular presentation of the calculus of constructions is to make the question both precise and concise:

Is Church's higher-order logic[‡] internally definable by $\mathsf{C}$ in the calculus of constructions via the obvious interpretation map?

This question adds another dimension to that of conservativity at the level of provability.

### 10.4. *Edinburgh Logical Framework*

In *ELF*, a constant type $U : Type$ and a constant constructor $t : U \rightarrow Type$ in a signature of an encoding also induce an internal category in the category of typing contexts over that signature. We can use the notion of internal definability to give a categorical account of adequate encodings in *ELF*. But the situation is less satisfactory due to the absence of equational contexts. The encodings in *ELF* definitely do not have the categorical properties of the kind we have seen in Section 7. However, our intuition tells us that still there is something to be said. The reason that an encoding of simply typed $\lambda$-calculus does not have an explicit cartesian closed structure on the internal category induced by universe $U$ in the encoding is that the definitional equality of the object language is now represented by an internal map $=_U : U \times U \rightarrow B$. Consequently, the diagrams in Lemma 7.1 are not pullbacks (in fact they are not even weak pullbacks). But the map $=_U$ induces an equivalence relation $\sim$ on each homset of the category of typing contexts. If in the definition of pullbacks we replace the external equality $=$ by $\sim$, the above mentioned diagrams are 'pullbacks'. So what we need is a notion of relative categorical structures where mediating morphisms are required to be unique up to an internal equality (Leibniz equality) rather than up to the external equality $=$. This point needs further investigation.

The same can be said about $\lambda_{LF}$.

## 11. Conclusions and related work

This paper has attempted to give a model theoretical account of logical frameworks. We have argued 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 have proposed a logical framework that is well suited to model theoretical analysis. We have used examples to show how syntactic conditions force term models to possess certain categorical properties. One important point of view 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-hand column contains the semantic counterparts of the corresponding syntactic notions in the left-hand column.

[†] We propose to call the conservativity of provability the logical conservativity and the conservativity of proofs the type theoretical conservativity (or just conservativity).

[‡] In this question, Church's higher-order logic is assumed to have extensional definitional equality.

| **syntactical notion** | **semantical notion** |
|---|---|
| universes | generic objects |
| encodings of object calculi | internal categories |
| constants other than universes | explicit structures on internal categories |
| variable convention | internal full subcategories |
| syntactic adequacy | internal definability |

By characterizing the categorical properties of $\lambda_{TT}$ and some encodings within it, we hope that we have pointed out a sensible way of performing 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.

A selling point of logical frameworks 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. An assertion forms a range of significance. So by Russel's principle, it can be represented by a type and its inhabitants can be represented by its verifiers. In this paper we have concentrated our attention on constructive logics. For a non-constructive language $\mathscr{L}$, say Church's higher-order logic (Church 1940), what we code up in a logical framework is the 'image' of $\mathscr{L}$ in a 'constructive mirror'. It is this aspect of $\mathscr{L}$ that is captured in an encoding. So, in this case, the categorical properties are characterizations of this 'image'. In this way, one assigns denotations to classical proofs in suitable categories. These categories are not degenerated in any sense as we have not imposed any equality on proofs. An obvious question is whether we have any interesting model theory for classical proofs.

The development in Power (1994) 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 semantic studies of logical frameworks.

The language $ELF^+$ investigated in Gardner (1992) 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 $\lambda_{TT}$ along this line. For instance, a universe declaration can be broken into two parts: one contains universes that encode judgements; the other consists of universes that represent sorts. Sorts of course can also be represented in $\lambda_{TT}$ by constant types in a signature.

Gardner (1992) contains an observation similar to that in Section 8. Because of the presence of $Judge$ and $Sort$, the author of Gardner (1992) 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 aspect of this phenomenon.

In this paper, the definitional equality of $\lambda_{TT}$ is given by judgemental equality. If one is interested in the proof theory of $\lambda_{TT}$, one uses the version of $\lambda_{TT}$ where definitional equality is defined in terms of reduction. In this version we remove the unit type, since it complicates the notion of reduction. Now instead of equational contexts, we have in this version reduction contexts. Definitional equalities in object languages are all supposed to be defined in terms of reductions and they are represented in $\lambda_{TT}$ by reduction contexts. This version of $\lambda_{TT}$ is appropriate for the study of proof theory of encodings.

## 12. Acknowledgement

The material in this paper is taken from the author's Ph.D. thesis (Fu 1992). My supervisor David Rydeheard has been very helpful all the way over the last few years. He has read many previous versions of this paper and suggested a lot of improvement. I would like to take this opportunity to express my appreciation for what he has done for me as both a supervisor and a colleague.

I would also like to express my gratitude to two anonymous referees whose constructive suggestions and criticism have helped me considerably to improve the quality of this paper.

## References

Bénabou, J. (1985) Fibred categories and the foundation of naive category theory. *Journal of Symbolic Logic* **50** (1) 10–37.

Böhm, C. and Berarducci, A. (1985) Automatic synthesis of typed $\lambda$-program on term algebras. *Theoretical Computer Science* **39** 135–154.

Barr, M. and Wells, C. (1990) *Category Theory for Computing Science*, Prentice Hall.

Church, A. (1940) A formulation of the simple theory of types. *Journal of Symbolic Logic* **5** (1) 56–68.

Ehrhard, T. (1988) A categorical semantics of constructions. In: *Proceedings of the Third Symposium on Logic in Computer Science*, IEEE Computer Science Press 264–273.

Fu, Y. (1992) *Some Semantic Issues In Type Theory*. Ph.D. thesis, Department of Computer Science, University of Manchester.

Gardner, P. (1992) *Representing Logics in Type Theory*. Ph.D. thesis, LFCS, University of Edinburgh.

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

Harper, R., Honsell, F. and Plotkin, G. (1993) A framework for defining logics. *Journal of ACM* **40** (1) 143–184.

Huet, G. and Plotkin, G. (eds.) (1991) *Logical Frameworks*, Cambridge University Press.

Huet, G. and Plotkin, G. (eds.) (1993) *Logical Environments*, Cambridge University Press.

Hyland, J. and Pitts, A. (1989) The theory of constructions: Categorical semantics and topos-theoretic models. In: Gray, J. and Scedrov, A. (eds.) *Categories in Computer Science and Logic*, A.M.S 137–199.

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

Hyland, M. (1988) A small complete category. *Annals of Pure and Applied Logic* **40** 135–165.

Jacobs, B. (1991) *Categorical Type Theory*. Ph.D. thesis, Catholic University of Nijmegen.

Lambek, J. and Scott, P. (1986) *Introduction To Higher-Order Categorical Logic*, Cambridge Studies in Advanced Mathematics Volume **7**, Cambridge University Press.

Luo, Z. (1990a) *An Extended Calculus of Constructions*, Ph.D. thesis, LFCS, University of Edinburgh, Edinburgh.

Luo, Z. (1990b) *A Problem of Adequacy: conservativity of calculus of constructions over higher-order logic*, LFCS, University of Edinburgh.

Nordström, B., Petersson, K. and Plotkin, G. (eds.) (1992) *Proceedings of the 1992 Workshop on Types for Proofs and Programs.*

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

Pavlović, D. (1990) *Predicates and Fibrations*, Ph.D. thesis, University of Utrecht.

Pitts, A. (1987) Polymorphism is set-theoretic, constructively. In: Pitt, D., Poigné, A. and Rydeheard, D. (eds.) Category Theory and Computer Science. *Springer-Verlag Lecture Notes in Computer Science* **283** 12–39.

Power, J. (1994) Why tricategories? Technical Report ECS-LFCS-94-289, Department of Computer Science, University of Edinburgh.

Seely, R. (1984) Locally cartesian closed categories and type theory. *Mathematical Proceedings of Cambridge Philosophical Society* **95** 33–48.

Seely, R. (1987) Categorical semantics for higher-order polymorphic lambda calculus. *Journal of Symbolic Logic* **52** 969–989.

Simpson, A. (1992) Kripke semantics for a logical framework. In: Nordström, B., Petersson, K. and Plotkin, G. (eds.) *Proceedings of the 1992 Workshop on Types for Proofs and Programs* 333–362.

Smith, J. (1988) The independence of Peano's fourth axiom from Martin-Löf's type theory without universes. *Journal of Symbolic Logic* **53** (3).

Streicher, T. (1989) Independent results for calculi of dependent types. In: Pitt, D., Poigné, A. and Rydeheard, D. (eds.) Category Theory and Computer Science. *Springer-Verlag Lecture Notes in Computer Science* **389** 141–154.

Taylor, P. (1986) *Recursive Domains, Indexed Category Theory and Polymorphism*, Ph.D. thesis, University of Cambridge.

Wraith, G. (1989) A note on categorical datatypes. In: Pitt, D., Poigné, A. and Rydeheard, D. (eds.) Category Theory and Computer Science. *Springer-Verlag Lecture Notes in Computer Science* **389** 213–223.