Categorical Semantics for Contextual Types

JASON HU, McGill University, Canada

This course project is to extend a recent work on categorical semantics for contextual types in a dependently typed system.

Additional Key Words and Phrases: type theory, category theory, contextual types, dependent types

ACM Reference Format:
Jason Hu. 2018. Categorical Semantics for Contextual Types. *Proc. ACM Program. Lang.* 1, CONF, Article 1 (January 2018), 18 pages.

1 INTRODUCTION

It has been a long lasting problem how to apply comonadic modality to metaprogramming in Martin-Löf style dependent type theory. There are two general approaches: the syntactical point of view and the semantic point of view. The former starts from and reasons about the syntactical rules of the systems while the latter derives the syntax from intended mathematical models. Recently, many researchers have made progress towards the solution. Among them, Gratzer et al. [2019]; Nanevski et al. [2008]; Pfenning and Davies [2001]; Pientka et al. [2019] take the syntactical approach while Birkedal et al. [2019]; Licata et al. [2018]; Shulman [2018] take the semantic approach. These directions have led to insights from different angles. Yet, these results are made by only taking one approach. We might be able to obtain deeper insight if we apply semantic methods to analyze a syntactically designed system.

In this report, we interpret a simplified system from Pientka et al. [2019] to a categorical model, in the hope that the theoretical framework of category theory can bring us to a richer and more powerful foundation.

2 BACKGROUND

2.1 Higher Order Abstract Syntax

When representing a programming language on paper, one is granted α-equivalence "for free" a lá Barendregt [1985], i.e. types and terms only differ only by names are considered equal. However, when we actually formalize the language in a proof assistant, name representation becomes an immediate problem. There are many solutions to this problem: locally nameless representation [Charguéraud 2012], de Bruijn indices [de Bruijn 1972], or more naively the string representation. One disadvantage of all these methods is that the users need to manually manage substitution and prove many properties of it. Consider a definition of untyped lambda calculus using any aforementioned method:

```
data Trm = Var Name | App Trm Trm | Lam Trm
```

Notice that we have a case `Var` for variables and `Name` is altered depending on the concrete method. We only need this case because we need to refer to variables in the `Lam` case. The following is a rule in operational semantics describing a step from a function application to another term by means of substitution:

$$(\lambda x.t_1)\ t_2 \longrightarrow t_1[t_2/x]$$

Author's address: Jason Hu, School of Computer Science, McGill University, 3480 University St. Montréal, Quebec, H3A 0E9, Canada, zhong.s.hu@mail.mcgill.ca.

2018. 2475-1421/2018/1-ART1 $15.00
https://doi.org/

where $t_1[t_2/x]$ denotes substituting t_2 for x in t_1. In all aforementioned methods, this substitution needs to be explicitly written and much equational reasoning is required to show properties of the calculus. This adds extra burden and thus difficulties to the formalization. Contrarily, higher order abstract syntax (HOAS) [Pfenning and Elliott 1988] completely shifts this burden to the meta-language. In HOAS, the untyped lambda calculus is formalized as follows:

```
data Trm' = App' Trm' Trm' | Lam' (Trm' -> Trm')
```

Notice that the case for variables is gone; instead, `Lam'` takes a function, and variables are directly represented by variables on the meta-level. As a result, the rule of the function application on the object level is directly represented as a function application on the meta-level and uses the substitution operation on the meta-level. This allows the subtle substitution operation to be defined correctly "once and for all". HOAS is employed in Edinburgh's Logical Framework (LF) [Harper et al. 1993] as a typical name representation.

However, HOAS is not a silver bullet; it seemingly contradicts common dependent type theories. In a typical dependent type theory, e.g. Martin-Löf type theory (MLTT) [Martin-Löf 1984] or Calculus of Inductive Constructions (CIC) [Paulin-Mohring 1989], (co)inductive definitions must satisfy the *strict positivity condition*, which requires a recursive reference not to appear in the argument position. In the `Lam'` case above, the type `Trm'` being defined occurs as an parameter type in `Trm' -> Trm'`, and thus violates the condition.

Despeyroux et al. [1997] observed the possibility of embedding LF in MLTT in a comonadic modality. Pientka et al. [2019] took a big step and proposed Cocon, a MLTT-like dependent type theory which allows not only coexistence of LF and MLTT, but also induction principles of LF terms. In this type theory, HOAS can be used in LF, which is in turn encapsulated in contextual modal types. This encapsulation is the essential reason for these two systems to coexist.

2.2 Modal Types and Contextual Types

Modal types [Pfenning and Davies 2001] is found to have applications in metaprogramming [Davies and Pfenning 2001]. Box modality can be used to contain code and participate computation. Box modality is captured by the following rules:

$$\frac{\Gamma; \cdot \vdash M : A}{\Gamma; \Psi \vdash \mathtt{box}\ M : \Box A} \qquad \frac{\Gamma; \Psi \vdash M : \Box A \qquad \Gamma, u : A; \Psi \vdash N : B}{\Gamma; \Psi \vdash \mathtt{let\ box}\ u = M\ \mathtt{in}\ N : B}$$

where Γ denotes the global context and Ψ denotes the local context. One limitation of box modality is seen from the introduction rule: since the local context is empty in the premise, t must represent closed code. To lift this limitation, Nanevski et al. [2008] introduces *contextual types*, which internalize contexts in box modality. Contextual types are captured by the following rules:

$$\frac{\Gamma; \Phi \vdash M : A}{\Gamma; \Psi \vdash \mathtt{box}\ \Phi.M : [\Phi]A} \qquad \frac{\Gamma; \Psi \vdash M : [\Phi]A \qquad \Gamma, u : A[\Phi]; \Psi \vdash N : B}{\Gamma; \Psi \vdash \mathtt{let\ box}\ u = M\ \mathtt{in}\ N : B}$$

In the introduction rule, $\Box A$ is generalized to $[\Phi]A$, allowing the capturing of a local context. The local context in the premise is no longer necessarily empty. In the elimination rule, A conditioned on Φ is added to the global context in the second premise, denoted by $A[\Phi]$.

In addition to the encapsulation of code, box modality in Cocon is used to provide a universe which LF can live in. Universes in Cocon are separated in two levels: the LF level and the computation level. Users can go from one to the other by using box and unbox:

$$\frac{\Gamma; \Psi \vdash M : A}{\Gamma \vdash \lceil \hat{\Psi} \vdash M \rceil : \lceil \Psi \vdash A \rceil} \qquad \frac{\Gamma \vdash t : \lceil \Phi \vdash A \rceil \qquad \Gamma; \Psi \vdash \sigma : \Phi}{\Gamma; \Psi \vdash \lfloor M \rfloor_\sigma : A}$$

The computation level only needs to keep track of one context for computation while the LF level needs to keep track of both the global and the local contexts. The introduction rule is quite similar to the previous rule in contextual types. The difference is that the conclusion lives on the computation level. Dually, the elimination rule converts a boxed term on the computation level on the LF level. Since the local contexts of the current environment and of the term t might not be the same, the elimination requires a substitution of Φ for Ψ, as indicated by $\Gamma; \Psi \vdash \sigma : \Phi$.

2.3 Categorical Semantics

Category theory [Awodey 2010; MacLane 1971] is an enormous theoretical framework which allows us to discover deep connections between two seemingly unrelated theories. Category theory has found many applications in computer science, especially in the field of programming languages. The community often discovers essential correspondences between frequently studied categories and type systems. For instance, Lambek [1985] reveals the connection between simply typed λ calculus (STLC) and cartesian closed categories (CCC), forming Curry-Howard-Lambek correspondence. Clairambault and Dybjer [2011]; Seely [1984] show that extensional type theory corresponds to locally cartecian closed categories (LCCC). These tight connection in turn guides us to design new type systems. Applying category theory to analysis of semantics for programming languages has become a common practice in the field.

Typically, we first axiomatize the structure of a particular kind of categories of interest. This kind of categories is intended to model the language. The signature (or the data) together with the axioms derives an *internal logic* of the category, which allows us to convert between logical and categorical statements. If we can interpret Cocon in an internal logical language, the corresponding categorical structure can then serve as a semantic model of Cocon. Via this model, we can investigate further how we should extend Cocon and even discover potential relations between Cocon and other theories. In this report, we present an interpretation of Cocon into category theory.

3 A SIMPLIFIED COCON

In this section, we introduce a simplified version of Cocon which differs slightly from the version presented by Pientka et al. [2019]. The original Cocon supports predicative levels of universes on the computation level, as well as dependent kinds on the LF level. These features are removed from the version presented here in favor of a simpler semantic model. Instead, the goal of this report is to extend a simply typed version [Pientka and Schöpp 2020] with dependent types. A model for full Cocon can be extended based on this report and is left as a future work.

The syntax of Cocon is presented in Figure 1. In this version, we support dependent types on both the computation and the LF levels. In addition, we define a simply typed language as an object language on the LF level:

```
ty  : type
o   : ty
arr : ty -> ty -> ty

trm : ty -> type
lam : (trm a -> trm b) -> trm (arr a b)
app : trm (arr a b) -> trm a -> trm b
```

In the object language, `ty` represents types and `trm` represents terms. Notice that `trm` is indexed by `ty` and thus dependently typed and the lambda abstraction is encoded using HOAS.

Judgments on the LF level are shown in Figure 2. Since we do not have kinding on the LF level, we use $\Gamma; \Psi \vdash A$ `type` to assert that A is a well-formed LF type. For this reason,we cannot directly

LF Types	$A, B ::= ty \mid tm\ M \mid \Pi x : A.B$
LF Terms	$M, N ::= \lambda x.M \mid x \mid c \mid M\ N \mid \lfloor t \rfloor_\sigma$
LF Constants	$c ::= o \mid arr \mid lam \mid app$
LF Contexts	$\Phi, \Psi ::= \phi \mid \cdot \mid \Phi, x : A$
LF Contexts (Erased)	$\hat{\Phi}, \hat{\Psi} ::= \phi \mid \cdot \mid \hat{\Phi}, x$
LF Substitutions	$\sigma ::= \cdot \mid wk_{\hat{\Phi}} \mid \sigma, M$
Contextual Types	$T ::= \Phi \vdash A \mid \Phi \vdash_v A$
Contextual Objects	$C ::= \hat{\Phi} \vdash M$
Domain of Discourse	$\check{\tau} ::= \tau \mid ctx$
Types	$\tau ::= \lceil T \rceil \mid (y : \check{\tau}_1) \Rightarrow \tau_2$
Terms	$t, s ::= y \mid \lceil C \rceil \mid \mathtt{fn}\ y \Rightarrow t \mid t_1\ t_2 \mid \mathtt{rec}^{\mathcal{I}}\ \mathcal{B}\ \Psi\ t$
Branches	$\mathcal{B} ::= \Gamma \mapsto t$
Contexts	$\Gamma ::= \cdot \mid \Gamma, y : \check{\tau}$

Fig. 1. Syntax of the simplified Cocon [Pientka et al. 2019]

formulate the well-formedness of trm as an axiom of $trm : ty \to type$. As usual, we use $A \to B$ to indicate $\Pi x : A.B$ when x is not free in B.

Judgments on the computation level is shown in Figure 3. Both ty and trm have induction principles on the computation level. It is worth noting that the induction principle of trm seems quite weak; it is necessary that the context of $z : \lceil \vdash ty \rceil$ in the signature is empty. Consider a generalized case where z has any context, then the signature of the recursor must become

$$(\phi, \psi : ctx) \Rightarrow (z : \lceil \phi \vdash ty \rceil) \Rightarrow (y : \lceil \psi \vdash trm \lfloor z \rfloor_{???} \rceil) \Rightarrow \tau$$

It is uncertain how to substiute from ψ to ϕ. It would be helpful if we can also internalize substitution, so that a more general induction principle can be obtained:

$$(\phi, \psi : ctx) \Rightarrow (\sigma : \psi \to \phi) \Rightarrow (z : \lceil \phi \vdash ty \rceil) \Rightarrow (y : \lceil \psi \vdash trm \lfloor z \rfloor_\sigma \rceil) \Rightarrow \tau$$

4 INTERPRETATION IN CATEGORIES WITH ATTRIBUTES

In this section, we show how Cocon presented in Section 3 can be interpreted in categories with attributes, a classic categorical structure modeling dependent type theories.

4.1 Categories with Attributes

In a dependent type theory, types can depend on terms. One typical example is the function application rule in Cocon:

$$\frac{\Gamma \vdash t : (y : \hat{\tau_1}) \Rightarrow \tau_2 \qquad \Gamma \vdash s : \hat{\tau_1}}{\Gamma \vdash t\ s : \tau_2[s/y]}$$

$\Gamma \vdash \Psi\ \ \mathtt{ctx}$ denotes the well-formedness of the LF context Ψ:

$$\frac{}{\Gamma \vdash \cdot\ \mathtt{ctx}} \qquad \frac{\Gamma(\psi) = ctx}{\Gamma \vdash \psi\ \mathtt{ctx}} \qquad \frac{\Gamma \vdash \Phi\ \mathtt{ctx} \qquad \Gamma; \Psi \vdash A\ \mathtt{type}}{\Gamma \vdash \Phi, x : A\ \mathtt{ctx}}$$

From now on, whenever Ψ presents, $\Gamma \vdash \Psi\ \ \mathtt{ctx}$ is assumed.

$\Gamma; \Psi \vdash A\ \ \mathtt{type}$ denotes the well-formedness of the LF type A:

$$\frac{}{\Gamma; \Psi \vdash ty\ \ \mathtt{type}} \qquad \frac{\Gamma; \Psi \vdash M : ty}{\Gamma; \Psi \vdash trm\ M\ \ \mathtt{type}} \qquad \frac{\Gamma; \Psi \vdash A\ \ \mathtt{type} \qquad \Gamma; \Psi, x : A \vdash B\ \ \mathtt{type}}{\Gamma; \Psi \vdash \Pi x : A.B\ \ \mathtt{type}}$$

$\Gamma; \Psi \vdash t : A$ denotes the LF typing judgments:

$$\frac{x : A \in \Psi}{\Gamma; \Psi \vdash x : A} \qquad \frac{\Gamma; \Psi, x : A \vdash M : B}{\Gamma; \Psi \vdash \lambda x.M : \Pi x : A.B} \qquad \frac{\Gamma; \Psi \vdash M : \Pi x : A.B \qquad \Gamma; \Psi \vdash N : A}{\Gamma; \Psi \vdash M\ N : B[N/x]}$$

$$\frac{\Gamma \vdash t : \lceil \Phi \vdash A \rceil \text{ or } \Gamma \vdash t : \lceil \Phi \vdash_v A \rceil \qquad \Gamma; \Psi \vdash \sigma : \Phi}{\Gamma; \Psi \vdash \lfloor t \rfloor_\sigma : A[\sigma/\hat{\Phi}]} \qquad \frac{}{\Gamma; \Psi \vdash o : ty} \qquad \frac{\Gamma; \Psi \vdash a : ty \quad \Gamma; \Psi \vdash b : ty}{\Gamma; \Psi \vdash arr\ a\ b : ty}$$

$$\frac{\Gamma; \Psi \vdash a : ty \qquad \Gamma; \Psi \vdash b : ty \qquad \Gamma; \Psi \vdash f : trm\ a \to trm\ b}{\Gamma; \Psi \vdash lam\ a\ b\ f : trm\ (arr\ a\ b)}$$

$$\frac{\Gamma; \Psi \vdash a : ty \qquad \Gamma; \Psi \vdash b : ty \qquad \Gamma; \Psi \vdash m : trm\ (arr\ a\ b) \qquad \Gamma; \Psi \vdash n : trm\ a}{\Gamma; \Psi \vdash app\ a\ b\ m\ n : trm\ b}$$

$\Gamma; \Psi \vdash \sigma : \Phi$ denotes substitution σ mapping Ψ to Φ:

$$\frac{\Gamma \vdash \Psi\ \ \mathtt{ctx}}{\Gamma; \Psi \vdash \cdot : \cdot} \qquad \frac{\Gamma; \Psi \vdash \sigma : \Phi \qquad \Gamma; \Psi \vdash M : A[\sigma/\hat{\Phi}]}{\Gamma; \Psi \vdash \sigma, M : \Phi, x : A} \qquad \frac{\Gamma \vdash \Psi, \overrightarrow{x : A}\ \mathtt{ctx}}{\Gamma; \Psi, \overrightarrow{x : A} \vdash wk_{\hat{\Psi}} : \Psi}$$

Fig. 2. LF related judgments [Pientka et al. 2019]

When t is applied, occurrences of y in τ_2 are substituted for s. Substitution is a fundamental operation in a dependent type theory, so it is very essential and convenient for the categorical model to characterize substitution properly.

A category with Attributes (CwA) [Cartmell 1986; Hofmann 1997] is a category with a particular structure, which has substitution as part of the invariant.

Definition 4.1. A category with attributes C consists of the following data:

(1) a terminal object $\top$,
(2) a functor $Ty : C^{op} \to \mathcal{S}et$, and $-\{\sigma\} = Ty(\sigma)$ for $\sigma : \Psi \to \Phi$,
(3) a context comprehension . so that given $\Phi \in C$ and $A \in Ty(\Phi)$, $\Phi.A \in C$,
(4) a projection morphism of context comprehension $p(A) : \Phi.A \to \Phi$, and
(5) a morphism $q(\sigma, A) : \Psi.A\{\sigma\} \to \Phi.A$ for each $\sigma : \Psi \to \Phi$.
(6) the following pullback diagram for $\sigma : \Psi \to \Phi$:

$\Gamma \vdash C : T$ denotes typing judgments of contextual objects:

$$\frac{}{\Gamma \vdash (\hat{\Psi} \vdash M) : (\Psi \vdash A)} \qquad \frac{x : A \in \Psi}{\Gamma \vdash (\hat{\Psi} \vdash x) : (\Psi \vdash_v A)} \qquad \frac{x : \lceil \Phi \vdash_v A \rceil \in \Gamma \qquad \Gamma; \Psi \vdash wk_{\hat{\Psi}} : \Phi}{\Gamma \vdash (\hat{\Psi} \vdash \lfloor x \rfloor_{wk_{\hat{\Psi}}}) : (\Psi \vdash_v A)}$$

$\Gamma \vdash t : \tau$ denotes the typing judgments in the computation level:

$$\frac{y : \hat{\tau} \in \Gamma}{\Gamma \vdash y : \hat{\tau}} \qquad \frac{\Gamma \vdash C : T}{\Gamma \vdash \lceil C \rceil : \lceil T \rceil} \qquad \frac{\Gamma, y : \hat{\tau_1} \vdash t : \tau_2}{\Gamma \vdash \mathtt{fn}\ y \Rightarrow t : (y : \hat{\tau_1}) \Rightarrow \tau_2}$$

$$\frac{\Gamma \vdash t : (y : \hat{\tau_1}) \Rightarrow \tau_2 \qquad \Gamma \vdash s : \hat{\tau_1}}{\Gamma \vdash t\ s : \tau_2[s/y]}$$

Recursor over ty: $\mathcal{I} = (\psi : \mathtt{ctx}) \Rightarrow (y : \lceil \psi \vdash ty \rceil) \Rightarrow \tau$

$$\frac{\Gamma \vdash t : \lceil \Psi \vdash ty \rceil \qquad \Gamma \vdash b_o : \mathcal{I} \qquad \Gamma \vdash b_{arr} : \mathcal{I}}{\Gamma \vdash \mathtt{rec}^{\mathcal{I}}\ (b_o \mid b_{arr})\ \Psi\ t : \tau[\Psi, t/\psi, y]}$$

Branch for the o case $\dfrac{\Gamma, \psi : \mathtt{ctx} \vdash t_o : \tau[\lceil \psi \vdash o \rceil / y]}{\Gamma \vdash (\psi \mapsto t_o) : \mathcal{I}}$

Branch for the arr case

$$\frac{\Gamma, \psi : \mathtt{ctx}, m, n : \lceil \psi \vdash ty \rceil, f_m : \tau[m/y], f_n : \tau[n/y] \vdash t_{arr} : \tau[\lceil \psi \vdash arr \lfloor m \rfloor \lfloor n \rfloor \rceil / y]}{\Gamma \vdash (\psi, m, n, f_m, f_n \mapsto t_{arr}) : \mathcal{I}}$$

Recursor over $trm : \mathcal{I} = (\psi : \mathtt{ctx}) \Rightarrow (z : \lceil \vdash ty \rceil) \Rightarrow (y : \lceil \psi \vdash trm\ \lfloor z \rfloor . \rceil) \Rightarrow \tau$

$$\frac{\Gamma \vdash t : \lceil \vdash ty \rceil \qquad \Gamma \vdash t' : \lceil \Psi \vdash trm\ \lfloor t \rfloor . \rceil \qquad \Gamma \vdash b_v : \mathcal{I} \qquad \Gamma \vdash b_{lam} : \mathcal{I} \qquad \Gamma \vdash b_{app} : \mathcal{I}}{\Gamma \vdash \mathtt{rec}^{\mathcal{I}}\ (b_v \mid b_{lam} \mid b_{app})\ \Psi\ t\ t' : \tau[\Psi, t, t'/\psi, z, y]}$$

Branch for the variable case $\dfrac{\Gamma, \psi : \mathtt{ctx}, a : \lceil \vdash ty \rceil, t : \lceil \psi \vdash_v trm\ \lfloor a \rfloor . \rceil \vdash t_v : \tau[a, t/z, y]}{\Gamma \vdash (\psi, a, t \mapsto t_v) : \mathcal{I}}$

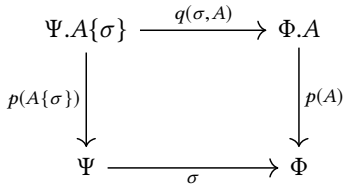
Branch for the lam case

$$\frac{\begin{array}{c}\Gamma, \psi : ctx, a, b : \lceil \vdash ty \rceil, m : \lceil \psi, x : trm\ \lfloor a \rfloor . \vdash trm\ \lfloor b \rfloor . \rceil, \\ f_m : \tau[(\psi, trm\ \lfloor a \rfloor .), b, m/\psi, x, y] \vdash t_{lam} : \tau[\lceil \vdash arr \lfloor a \rfloor \lfloor b \rfloor \rceil, \lceil \psi \vdash lam \lfloor a \rfloor . \lfloor b \rfloor . (\lambda x . \lfloor m \rfloor) \rceil / z, y]\end{array}}{\Gamma \vdash (\psi, a, b, m, f_m \mapsto t_{lam}) : \mathcal{I}}$$

Branch for the app case

$$\frac{\begin{array}{c}\Gamma, \psi : ctx, a, b : \lceil \vdash ty \rceil, m : \lceil \psi \vdash trm\ (arr \lfloor a \rfloor . \lfloor b \rfloor .) \rceil, n : \lceil \psi \vdash trm\ \lfloor a \rfloor . \rceil, \\ f_m : \tau[\lceil \vdash arr \lfloor a \rfloor \lfloor b \rfloor \rceil, m/z, y], f_n : \tau[a, n/z, y] \vdash t_{app} : \tau[b, \lceil \psi \vdash app \lfloor a \rfloor . \lfloor b \rfloor . \lfloor m \rfloor \lfloor n \rfloor \rceil / z, y]\end{array}}{\Gamma \vdash (\psi, a, b, m, n, f_m, f_n \mapsto t_{app}) : \mathcal{I}}$$

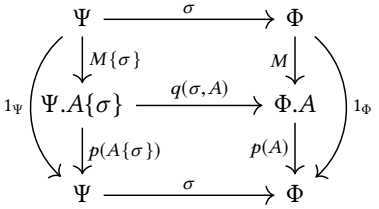
Fig. 3. Judgments in the comptuation level [Pientka et al. 2019]



The axioms are the following:

(1) $q(1_\Phi, A) = 1_{\Phi.A}$,
(2) $q(\sigma \circ \delta, A) = q(\sigma, A) \circ q(\delta, A\{\sigma\})$, and

We regard the objects in $\mathcal{C}$ as contexts and the morphisms as substitution of one context for another. Based on this understanding, $Ty(\Phi)$ denotes the set of semantic types under context Φ. Given a semantic type $A \in Ty(\Phi)$, $A\{\sigma\}$ is thus substituting the context for Ψ given $\sigma : \Psi \to \Phi$ and the set of semantic terms of this type $Tm(\Phi, A)$ is defined as the set of sections of $p(A)$, $Sect(p(A)) = \{M \mid p(A) \circ M = 1_\Phi\}$. Substitution for semantic terms $M\{\sigma\}$ is naturally defined due to the top square in the following diagram:



All squares in this diagram commute. $M\{\sigma\}$ is uniquely defined due to the pullback square at the bottom.

Provided $\sigma : \Psi \to \Phi$ and $M \in Tm(\Psi, A\{\sigma\})$, the substitution extension $\langle \sigma, M \rangle : \Psi \to \Phi.A$ is defined as $q(\sigma, A) \circ M$. The second projection of context comprehension thus becomes $v_A \in Tm(\Phi.A, A\{p(A)\}) = \{M \mid p(A\{p(A)\}) \circ M = 1_{\Phi.A}\}$.

Up until this point, we have obtained a generic categorical structure for dependent type theory with substitution as an invariant. In order to model dependent function space, we need an additional type former.

Definition 4.2. Semantic Π types in a CwA $\mathcal{C}$ have the following data:

(1) A semantic type $\Pi(A, B) \in Ty(\Phi)$ for each $A \in Ty(\Phi)$ and $B \in Ty(\Phi.A)$,
(2) a semantic term $\Lambda_{A,B}(M) \in Tm(\Phi, \Pi(A, B))$ for each $M \in Tm(\Phi.A, B)$, and
(3) a semantic term $App_{A,B}(M, N) \in Tm(\Phi, B\{\langle 1_\Phi, N \rangle\})$ for each $M \in Tm(\Phi, \Pi(A, B))$ and $N \in Tm(\Phi, A)$.

so that the following axioms are satisfied:

(1) $\Pi(A, B)\{\sigma\} = \Pi(A\{\sigma\}, B\{q(\sigma, A)\}) \in Ty(\Phi)$ for $\sigma : \Phi \to \Psi$,
(2) $\Lambda_{A,B}(M)\{\sigma\} = \Lambda_{A\{\sigma\}, B\{q(\sigma,A)\}}(M\{q(\sigma, A)\}) \in Tm(\Phi, \Pi(A, B)\{\sigma\})$ for $M \in Tm(\Psi.A, B)$ and $\sigma : \Phi \to \Psi$,
(3) $App_{A,B}(M, N)\{\sigma\} = App_{A\{\sigma\}, B\{q(\sigma,A)\}}(M\{\sigma\}, N\{\sigma\}) \in Tm(\Phi, B\{\langle 1_\Psi, N \rangle\}\{\sigma\}) = Tm(\Phi, B\{\langle \sigma, N\{\sigma\} \rangle\})$ for $M \in Tm(\Psi, \Pi(A, B))$, $N \in Tm(\Psi, A)$ and $\sigma : \Phi \to \Psi$,
(4) $App_{A,B}(\Lambda_{A,B}(M), N) = M\{\langle 1_\Phi, N \rangle\} \in Tm(\Phi, B\{\langle 1_\Phi, N \rangle\})$ for each $M \in Tm(\Phi.A, B)$ and $N \in Tm(\Phi, A)$, and
(5) $\Lambda_{A,B}(App_{A,B}(M\{p(A)\}, v_A)) = M \in Tm(\Phi, \Pi(A, B))$.

We often omit the subscripts of Λ and App in favor of conciseness when they can be unambiguously inferred. The types of the third equation are reasoned as follows:

$$\begin{aligned}
App(M,N)\{\sigma\} &\in Tm(\Psi, B\{\langle 1_\Psi, N\rangle\})\{\sigma\} \\
&= Tm(\Phi, B\{\langle 1_\Psi, N\rangle\}\{\sigma\}) \\
&= Tm(\Phi, B\{\langle 1_\Psi, N\rangle \circ \sigma\}) \\
&= Tm(\Phi, B\{q(1_\Psi, A) \circ N \circ \sigma\}) \\
&= Tm(\Phi, B\{q(1_\Psi, A) \circ q(\sigma, A) \circ N\{\sigma\}\}) && \text{from the property of } q \text{ above} \\
&= Tm(\Phi, B\{q(\sigma, A) \circ N\{\sigma\}\}) \\
&= Tm(\Phi, B\{\langle \sigma, N\{\sigma\}\rangle\})
\end{aligned}$$

$$\begin{aligned}
App(M\{\sigma\}, N\{\sigma\}) &\in Tm(\Phi, B\{q(\sigma, A)\}\{\langle 1_\Phi, N\{\sigma\}\rangle\}) \\
&= Tm(\Phi, B\{q(\sigma, A) \circ \langle 1_\Phi, N\{\sigma\}\rangle\}) \\
&= Tm(\Phi, B\{q(\sigma, A) \circ q(1_\Phi, A\{\sigma\}) \circ N\{\sigma\}\}) \\
&= Tm(\Phi, B\{q(\sigma, A) \circ N\{\sigma\}\}) \\
&= Tm(\Phi, B\{\langle \sigma, N\{\sigma\}\rangle\})
\end{aligned}$$

Thus their types agree.

The fourth and the fifth equations correspond to β and η rule in the type system.

With the axiomization, we are able to capture the dependently typed nature and the existence of dependent function space of both the LF level and the computation level. We can also axiomatize other types like ty and trm and their constructors. However, we need more to interpret Cocon:

(1) CwA does not provide a direct connection between the LF and the computation level, and
(2) the LF terms exhibit different substitution behavior from the computation terms.

Therefore we need to fit the LF level on the computation level in the semantic model as well as alter the substitution behavior of the LF terms. These two are obtained simultaneously by adding a layer of presheaf category over a CwA $\mathcal{C}$.

4.2 Presheaf Model

A presheaf from a category $\mathcal{C}$ is a functor $F : \mathcal{C}^{op} \to \mathcal{S}et$. The presheaf category is a category where the objects are presheaves from a fixed category $\mathcal{C}$ and the morphisms are natural transformations between the presheaves. We call $\mathcal{C}$ the *index category* or the *domain category* and we use $\hat{\mathcal{C}}$ to denote the presheaf category. Presheaf categories enjoy many structures and useful properties.

Theorem 4.3. *For any small category $\mathcal{C}$, its presheaf category is a topos.*

That is, a presheaf category has the cartesian closed structure and has all finite limits. Hofmann [1997] showed that presheaf categories are a useful instance to model dependent type theories.

Theorem 4.4. *[Hofmann 1997, Section 4.1] Presheaf categories have the CwA structure.*

Theorem 4.5. *[Hofmann 1997, Section 4.2] Presheaf categories support dependent function types.*

Considering Cocon is a type theory with two levels of dependent type theories, if the index category is a CwA, then we have obtained a similar two-level structure from the presheaf model. We can use the index category to model the LF level and the presheaf category to model the computation level. From the presheaf category, we have access to the index category via Yoneda embedding $El : \mathcal{C} \to \mathcal{S}et^{\mathcal{C}^{op}}$. We also need a comonadic modality to model box modality. This is obtained by the endofunctor $\flat : \mathcal{S}et^{\mathcal{C}^{op}} \to \mathcal{S}et^{\mathcal{C}^{op}}$, which maps presheaves to constants, i.e. $\flat F(\Psi) = F(\top)$.

4.3 Semantic Model

The semantic model is based on the presheaf model described above, where the index category is a CwA. The axioms are presented using the internal logical language of the dependent type theory of the presheaf model, so the axioms can be conveniently stated as judgments. Following Section 4.1, we define the morphisms in the semantic model in the same order. The presentation resembles the axiomatization above as much as possible.

$$\Gamma \vdash \mathtt{Ctx}\ \mathtt{type} \qquad \Gamma, \Phi : \mathtt{Ctx} \vdash El(\Phi)\ \mathtt{type} \qquad \Gamma, \Phi : \mathtt{Ctx} \vdash Ty(\Phi)\ \mathtt{type} \qquad \Gamma \vdash \top : \mathtt{Ctx}$$

$$\Gamma \vdash \top : El(\top) \qquad \Gamma \vdash . : (\Phi : \mathtt{Ctx}) \to (A : Ty(\Phi)) \to \mathtt{Ctx} \qquad \Gamma, \Phi : \mathtt{Ctx} \vdash ! : El(\Phi) \to El(\top)$$

$$\Gamma, \Phi : \mathtt{Ctx}, \Psi : \mathtt{Ctx}, \sigma : El(\Phi) \to El(\Psi) \vdash -\{\sigma\} : Ty(\Psi) \to Ty(\Phi)$$

$$\Gamma, \Phi : \mathtt{Ctx}, A : Ty(\Phi) \vdash p : El(\Phi.A) \to El(\Phi)$$

$$\Gamma, \Phi : \mathtt{Ctx}, \Psi : \mathtt{Ctx}, A : Ty(\Psi), \sigma : El(\Phi) \to El(\Psi) \vdash q(\sigma, A) : El(\Phi.A\{\sigma\}) \to El(\Psi.A)$$

The LF contexts are lifted by the Yoneda embedding. The morphisms are justified by the signature of CwA, only lifted by Yoneda embedding to the presheaf category. ! is the unique morphism going from any context morphism to the terminal context. Tm is the set of sections of p:

$$\Gamma, \Phi : \mathtt{Ctx}, A : Ty(\Phi) \vdash Tm(\Phi, A) \equiv \Sigma(El(\Phi) \to El(\Phi.A))(\lambda M.p \circ M = 1_\Phi)\ \mathtt{type}$$

That is, the LF terms of an LF type A in a given LF context Γ are modeled by those semantic substitutions which left composed with p produce the identity substitution. Here we further use the fact that the Σ and equality exists in the presheaf model.

Since the LF contexts are lifted by Yoneda embedding, it is convenient to actually represent semantic terms in the presheaf category, so that we can take advantage of the function space in the internal logic. Pientka and Schöpp [2020] proposed to use a slightly different definition:

$$\Gamma, \Phi : \mathtt{Ctx}, u : El(\Phi), A : Ty(\Phi) \vdash TmEl(u, A) \equiv \Sigma(El(\Phi.A))(\lambda w.p(w) = u)\ \mathtt{type}$$

It is easy to show these two definitions are isomorphic.

Theorem 4.6. *Tm and $TmEl$ are isomorphic.*

Thus we use $TmEl$ in place of Tm. Following Section 4.1, we continue state the morphisms in the semantic model. First, we lift the pullback diagram from $\mathcal{C}$ to $\hat{\mathcal{C}}$ using Yoneda embedding. This is possible because the Yoneda embedding preserves all limits. Then we obtain the semantic substitution for semantic terms due to essentially the same diagram:

$$\Gamma, \Phi, \Psi : \mathtt{Ctx}, u : El(\Phi), A : Ty(\Psi), \sigma : El(\Phi) \to El(\Psi) \vdash -\{\sigma\} : TmEl(\sigma(u), A) \to TmEl(u, A\{\sigma\})$$

The same as in Section 4.1, this morphism is unique up to isomorphism.

Other morphisms can be defined in a similar manner:

$$\Gamma, \Phi : \mathtt{Ctx}, u : El(\Phi), A : Ty(\Phi), M : TmEl(u, A) \vdash \mathtt{pair}(u, M) : El(\Phi.A)$$

$$\Gamma, \Phi : \mathtt{Ctx}, A : Ty(\Phi) \vdash v : (u : El(\Phi.A)) \to TmEl(u, A\{p\})$$

`pair` is defined by applying the underlying substitution of M to u.

Next we state the morphisms for Π type on the LF level. Luckily we can reuse the symbols because the symbols we used in Section 3 are consistent with the axiomization.

$$\frac{\Gamma, \Phi : \mathtt{Ctx} \vdash A : Ty(\Phi) \qquad \Gamma, \Phi : \mathtt{Ctx} \vdash B : Ty(\Phi.A)}{\Gamma, \Phi : \mathtt{Ctx} \vdash \Pi(A, B) : Ty(\Phi)}$$

$$\frac{\Gamma, \Phi : \mathtt{Ctx} \vdash M : (u : El(\Phi)) \to (x : TmEl(u, A)) \to TmEl(\mathtt{pair}(u, x), B)}{\Gamma, \Phi : \mathtt{Ctx} \vdash \Lambda(M) : (u : El(\Phi)) \to TmEl(u, \Pi(A, B))}$$

$$\frac{\Gamma, \Phi : \mathtt{Ctx} \vdash M : (u : El(\Phi)) \to TmEl(u, \Pi(A, B)) \qquad \Gamma, \Phi : \mathtt{Ctx} \vdash N : (u : El(\Phi)) \to TmEl(u, A)}{\Gamma, \Phi : \mathtt{Ctx} \vdash App(M, N) : (u : El(\Phi)) \to TmEl(u, B\{\lambda u'.\mathtt{pair}(u', N(u'))\})}$$

The same as Section 4.1, we omit subscripts of Λ and *App* as many as possible because they can usually be inferred. In the return type of the *App* case, we want to substitute M in B, which requires us to provide the (Yoneda embedded) substitution morphism $El(\Phi) \to El(\Phi.A)$, which is why we are defining this morphism using a lambda. For conciseness, we define the following notation for this kind of weakening morphism:

$$\Gamma, \Phi : \mathtt{Ctx}, N : (u : El(\Phi)) \to TmEl(u, A) \vdash \overline{N} \equiv (\lambda u'.\mathtt{pair}(u', N(u'))) : El(\Phi) \to El(\Phi.A)$$

With dependent types, we need to prepare for substitution in the result types, so we cannot assume a fixed Yoneda embedded context:

$$\Gamma, \Phi : \mathtt{Ctx}, u : El(\Phi) \vdash App(M, N) : TmEl(u, B\{???\})$$

If we used this judgmental form, we would not be able to write down a proper substitution in B. For this purpose, in general, we interpret a LF term M in the model to the following form:

$$\Gamma, \Phi : \mathtt{Ctx} \vdash M : (u : El(\Phi)) \to TmEl(u, A)$$

for a proper LF type A.

The object language is stated as follows:

$$\Gamma, \Phi : \mathtt{Ctx} \vdash ty : Ty(\Phi) \qquad\qquad \Gamma, \Phi : \mathtt{Ctx} \vdash trm : Ty(\Phi.ty)$$

$$\Gamma, \Phi : \mathtt{Ctx} \vdash o : (u : El(\Phi)) \to TmEl(u, ty)$$

$$\Gamma, \Phi : \mathtt{Ctx} \vdash arr :$$

$$((u : El(\Phi)) \to TmEl(u, ty)) \to ((u : El(\Phi)) \to TmEl(u, ty)) \to ((u : El(\Phi)) \to TmEl(u, ty))$$

$$\Gamma, \Phi : \mathtt{Ctx} \vdash lam : (a, b : (u : El(\Phi)) \to TmEl(u, ty)) \to$$

$$(((u : El(\Phi)) \to TmEl(u, trm\{\overline{a}\})) \to (u : El(\Phi)) \to TmEl(u, trm\{\overline{b}\})) \to$$

$$((u : El(\Phi)) \to TmEl(u, trm\{\overline{arr(a, b)}\}))$$

$$\Gamma, \Phi : \mathtt{Ctx} \vdash app : (a, b : (u : El(\Phi)) \to TmEl(u, ty)) \to$$

$$((u : El(\Phi)) \to TmEl(u, trm\{\overline{arr(a, b)}\})) \to$$

$$((u : El(\Phi)) \to TmEl(u, trm\{\overline{a}\})) \to ((u : El(\Phi)) \to TmEl(u, trm\{\overline{b}\}))$$

Interpretation of LF types:

$$\begin{aligned}
[\![\Gamma; \Psi \vdash \Pi x : A.B \ \ \texttt{type}]\!] &= \Pi([\![\Gamma; \Psi \vdash A \ \ \texttt{type}]\!], [\![\Gamma; \Psi, x : A \vdash B \ \ \texttt{type}]\!]) \\
[\![\Gamma; \Psi \vdash ty \ \ \texttt{type}]\!] &= ty \\
[\![\Gamma; \Psi \vdash trm\ M \ \ \texttt{type}]\!] &= trm\{\overline{[\![\Gamma; \Psi \vdash M : ty]\!]}\}
\end{aligned}$$

Interpretation of LF contexts to objects in C:

$$\begin{aligned}
[\![\Gamma \vdash \cdot \ \ \texttt{ctx}]\!] &= \top \\
[\![\Gamma \vdash \Psi, x : A \ \ \texttt{ctx}]\!] &= [\![\Gamma \vdash \Psi \ \ \texttt{ctx}]\!].[\![\Gamma; \Psi \vdash A \ \ \texttt{type}]\!] \\
[\![\Gamma \vdash \psi \ \ \texttt{ctx}]\!] &= \psi
\end{aligned}$$

Interpretation of LF substitutions where $\Psi' = [\![\Gamma \vdash \Psi \ \ \texttt{ctx}]\!]$:

$$\begin{aligned}
[\![\Gamma; \Psi \vdash \cdot : \cdot]\!] &= ! : El(\Psi') \to El(\top) \\
[\![\Gamma; \Psi \vdash \sigma, M : \Phi, x : A]\!]_u &= q(e_1, A') \circ \overline{e_2} \\
&\quad : El(\Psi') \to El(\Phi'.A') \\
&\quad \text{where } A' = [\![\Gamma; \Phi \vdash A \ \ \texttt{type}]\!] \text{ and } \Phi' = [\![\Gamma \vdash \Phi \ \ \texttt{ctx}]\!] \\
&\quad e_1 = [\![\Gamma; \Psi \vdash \sigma : \Phi]\!] : El(\Psi') \to El(\Phi') \\
&\quad e_2 = [\![\Gamma; \Psi \vdash M : A[\sigma/\hat{\Phi}]]\!] : (u : El(\Psi')) \to TmEl(u, A'\{e_1\}) \\
[\![\Gamma; \Psi, \overrightarrow{x : A} \vdash wk_{\hat{\Psi}} : \Psi]\!]_u &= \lambda u.p^k(u) : El([\![\Gamma \vdash \Psi, \overrightarrow{x : A} \ \ \texttt{ctx}]\!]) \to El(\Psi')
\end{aligned}$$

Fig. 4. Interpretation of LF objects

For convenience, we want to assert that ty is unique in all LF contexts. This is established by requiring substitution always bring ty in context to another ty in another context:

$$\Gamma, \Phi, \Psi : \texttt{Ctx}, \sigma : El(\Phi) \to El(\Psi) \vdash ty\{\sigma\} \equiv ty : Ty(\Phi)$$

Here the first ty is $Ty(\Psi)$ and the second one is $Ty(\Phi)$.

On the other hand, trm does not have this property. In fact, substitution of trm is meaningful and is used in our interpretation.

Thus the goal is to find an interpretation function $[\![\,]\!]$ for types and contexts in both levels, so that the following properties hold.

Conjecture 4.7. *The following are the intended theorems:*

(1) *If* $\Gamma \vdash \Phi \ \ \texttt{ctx}$, *then* $[\![\Gamma]\!] \vdash [\![\Gamma \vdash \Phi \ \ \texttt{ctx}]\!] : \texttt{Ctx}$.
(2) *If* $\Gamma; \Psi \vdash A \ \ \texttt{type}$, *then* $[\![\Gamma]\!] \vdash [\![\Gamma; \Psi \vdash A \ \ \texttt{type}]\!] : Ty([\![\Gamma \vdash \Psi \ \ \texttt{ctx}]\!])$.
(3) *If* $\Gamma; \Psi \vdash M : A$, *then* $[\![\Gamma]\!] \vdash [\![\Gamma; \Psi \vdash M : A]\!] : (u : El([\![\Gamma \vdash \Psi \ \ \texttt{ctx}]\!])) \to TmEl(u, [\![\Gamma; \Psi \vdash A \ \ \texttt{type}]\!]_u)$.
(4) *If* $\Gamma; \Psi \vdash \sigma : \Phi$, *then* $[\![\Gamma]\!] \vdash [\![\Gamma; \Psi \vdash \sigma : \Phi]\!] : El([\![\Gamma \vdash \Psi \ \ \texttt{ctx}]\!]) \to El([\![\Gamma \vdash \Phi \ \ \texttt{ctx}]\!])$.
(5) *If* $\Gamma \vdash C : T$, *then* $[\![\Gamma]\!] \vdash [\![\Gamma \vdash C : T]\!] : [\![T]\!]$.
(6) *If* $\Gamma \vdash t : \check{\tau}$, *then* $[\![\Gamma]\!] \vdash [\![\Gamma \vdash t : \check{\tau}]\!] : [\![\check{\tau}]\!]$.

4.4 Interpretation of the LF Level

In this section, we show the interpretation of the LF level in Cocon in Section 3 to the semantic model presented above. The interpretation functions are shown in Figure 4 and Figure 5.

In the interpretation of LF types in Figure 4, since trm has a semantic object has type $Ty(\Phi.ty)$, we apply substitution for the ty on top of the context.

$$
\begin{aligned}
&\text{Let } \Psi' = [\![\Gamma \vdash \Psi \ \texttt{ctx}]\!]\text{:}\\
[\![\Gamma; \Psi \vdash x : A]\!] &= \lambda u.v(p^k(u))\{p^k\}\\
&\quad : (u : El(\Psi')) \to TmEl(u, A\{p^{k+1}\})\\
&\qquad \text{where } \Psi = \Psi_0, x : A, \overrightarrow{y_i : B_i} \text{ and } |\overrightarrow{y_i : B_i}| = k\\
[\![\Gamma; \Psi \vdash \lambda x.M : \Pi x : A.B]\!] &= \Lambda(\lambda u' : El(\Psi').\lambda x : TmEl(u', A').e_1(\texttt{pair}(u', x)))\\
&\quad : (u : El(\Psi')) \to TmEl(u, \Pi(A', B'))\\
&\qquad \text{where } A' = [\![\Gamma; \Psi \vdash A \ \texttt{type}]\!] \text{ and } B' = [\![\Gamma; \Psi, x : A \vdash B \ \texttt{type}]\!],\\
&\qquad\quad e_1 = [\![\Gamma; \Psi, x : A \vdash M : B]\!] : (u : El(\Psi'.A')) \to TmEl(u, B')\\
[\![\Gamma; \Psi \vdash M\ N : B[N/x]]\!] &= App(e_1, e_2)\\
&\quad : (u : El(\Psi')) \to TmEl(u, B\{\overline{e_2}\})\\
&\qquad \text{where } A' = [\![\Gamma; \Psi \vdash A \ \texttt{type}]\!] \text{ and } B' = [\![\Gamma; \Psi, x : A \vdash B \ \texttt{type}]\!],\\
&\qquad\quad e_1 = [\![\Gamma; \Psi \vdash M : \Pi x : A.B]\!] : (u : El(\Psi')) \to TmEl(u, \Pi(A', B')),\\
&\qquad\quad e_2 = [\![\Gamma; \Psi \vdash N : A]\!] : (u : El(\Psi')) \to TmEl(u, A)\\
[\![\Gamma; \Psi \vdash o : ty]\!] &= o : (u : El(\Psi')) \to TmEl(u, ty)\\
[\![\Gamma; \Psi \vdash arr\ a\ b : ty]\!] &= arr([\![\Gamma; \Psi \vdash a : ty]\!], [\![\Gamma; \Psi \vdash b : ty]\!]) : (u : El(\Psi')) \to TmEl(u, ty)\\
[\![\Gamma; \Psi \vdash lam\ a\ b\ f : trm\ (arr\ a\ b)]\!] &= lam(a', b', \lambda x.App(e', x))\\
&\quad : (u : El(\Psi')) \to TmEl(u, trm\{\overline{arr(a', b')}\})\\
&\qquad \text{where } a' = [\![\Gamma; \Psi \vdash a : ty]\!], b' = [\![\Gamma; \Psi \vdash b : ty]\!],\\
&\qquad\quad b'' = [\![\Gamma; \Psi, x : trm\ a \vdash b : ty]\!],\\
&\qquad \text{and } e' = [\![\Gamma; \Psi \vdash f : trm\ a \to trm\ b]\!] : (u : El(\Psi')) \to TmEl(u, \Pi(trm\{\overline{a'}\}, trm\{\overline{b''}\}))\\
[\![\Gamma; \Psi \vdash app\ a\ b\ m\ n : trm\ b]\!] &= app(a', b', e_1, e_2) : (u : El(\Psi')) \to TmEl(u, trm\{\overline{b'}\})\\
&\qquad \text{where } a' = [\![\Gamma; \Psi \vdash a : ty]\!], b' = [\![\Gamma; \Psi \vdash b : ty]\!],\\
&\qquad\quad e_1 = [\![\Gamma; \Psi \vdash m : trm\ (arr\ a\ b)]\!] \text{ and } e_2 = [\![\Gamma; \Psi \vdash n : trm\ a]\!]\\
[\![\Gamma; \Psi \vdash \lfloor t \rfloor_\sigma : A[\sigma/\hat{\Phi}]]\!] &= \lambda u.\texttt{let box } f = e_1 \texttt{ in } (f(e_2(u)))\{e_2\}\\
&\quad : (u : El(\Psi')) \to TmEl(u, A'\{e_2\})\\
&\qquad \text{where } \Phi' = [\![\Gamma \vdash \Phi \ \texttt{ctx}]\!], A' = [\![\Gamma; \Phi \vdash A \ \texttt{type}]\!],\\
&\qquad\quad e_1 = [\![\Gamma \vdash t : \lceil \Phi \vdash A \rceil]\!] : \flat((w : El(\Phi')) \to TmEl(w, A')),\\
&\qquad\quad \text{and } e_2 = [\![\Gamma; \Psi \vdash \sigma : \Phi]\!] : El(\Psi') \to El(\Phi')
\end{aligned}
$$

Fig. 5. Interpretation of LF terms

The interpretation of LF terms in Figure 5 is relatively complex. Let us examine the cases carefully. Consider the variable case:

$$
\begin{aligned}
u : El([\![\Gamma \vdash \Psi \ \texttt{ctx}]\!]) &= El([\![\Gamma \vdash \Psi_0, x : A, \overrightarrow{y : B} \ \texttt{ctx}]\!])\\
p^k(u) &: El([\![\Gamma \vdash \Psi_0, x : A \ \texttt{ctx}]\!])\\
v(p^k(u)) &: Tm(p^k(u), A\{p\})\\
v(p^k(u))\{p^k\} &: Tm(u, A\{p^{k+1}\})
\end{aligned}
$$

The λ abstraction case has:

$$e_1(\mathtt{pair}(u', x)) : TmEl(\mathtt{pair}(u', x), B')$$
$$\lambda u'.\lambda x.e_1(\mathtt{pair}(u', x)) : (u' : El(\Psi')) \to (x : TmEl(u', A')) \to TmEl(\mathtt{pair}(u', x), B')$$
$$\Lambda(\lambda u'.\lambda x.e_1(\mathtt{pair}(u', x))) : (u : El(\Psi')) \to TmEl(u, \Pi(A', B'))$$

The *App*, *o* and *arr* cases are immediate.

The *lam* case is more subtle. Since $trm\ a \to trm\ b$ is $\Pi x : trm\ a.trm\ b$ on the LF level, this LF type is interpreted to $\Pi(trm\{\overline{a'}\}, trm\{\overline{b''}\})$, where b'' is b reinterpreted in an extended context with $x : trm\ a$ on which b does not depend. This extension of context is necessary to make the Π type well-formed.

We should prove the following conjecture of context strengthening:

Conjecture 4.8. *If* $\Gamma; \Psi, x : A \vdash M : B$ *and* $x \notin FV(M) \bigcup FV(B)$, *then* $[\![\Gamma; \Psi, x : A \vdash M : B]\!] = \lambda u.[\![\Gamma; \Psi \vdash M : B]\!](p(u))\{p\}$.

We then have

$$\begin{aligned}
App(e', x) : (u : El(\Psi')) \to TmEl(u, trm\{\overline{b''}\}\{\overline{x}\}) \qquad & \text{where } x : (u : El(\Psi')) \to TmEl(u, trm\{\overline{a'}\}) \\
&= (u : El(\Psi')) \to TmEl(u, trm\{\overline{b''} \circ \overline{x}\}) \\
&= (u : El(\Psi')) \to TmEl(u, trm\{\lambda u'.\mathtt{pair}(\mathtt{pair}(u', x(u')), b''(\mathtt{pair}(u', x(u'))))\}) \\
&= (u : El(\Psi')) \to TmEl(u, trm\{\lambda u'.\mathtt{pair}(\mathtt{pair}(u', x(u')), b'(p(\mathtt{pair}(u', x(u'))))\{p\})\}) \\
&= (u : El(\Psi')) \to TmEl(u, trm\{\lambda u'.\mathtt{pair}(\mathtt{pair}(u', x(u')), b'(u')\{p\})\}) \\
&\qquad \text{we need an axiom here to characterize the substitution property of } trm \\
&= (u : El(\Psi')) \to TmEl(u, trm\{\lambda u'.\mathtt{pair}(u', b'(u'))\}) \\
&= (u : El(\Psi')) \to TmEl(u, trm\{\overline{b'}\})
\end{aligned}$$

The *app* case is immediate.

The unbox case type checks as follows:

$$f : (w : El(\Phi')) \to Tm(w, A')$$
$$f(e_2(u)) : Tm(e_2(u), A')$$
$$(f(e_2(u)))\{e_2\} : Tm(u, A'\{e_2\})$$

4.5 Interpretation of the Computation Level

In this section, we present the interpretation of the computation level of Cocon. Interpretation of the computation level is simpler because the internal logic we operate in is readily dependently typed and thus we simply need to define a straightforward embedding.

The interpretation functions without the recursors are shown in Figure 6. They resemble the simply typed case greatly. We only need to restrict the variable morphism $(u : El(\Phi)) \to_v Tm(u, A)$ as the following form:

$$\lambda u.v(p^k(u))\{p^k\}$$

for $k \in \mathbb{N}$.

Next we discuss the recursors for the LF types. The recursor for *ty* is immediate. Since there is no non-positive occurrence of *ty* in its definition, it can be modeled in a conventional manner using initial algebras, which corresponds tightly with the syntax in Figure 3. Hence we omit the concrete formulation here.

Intepretation of computation types

$$[\![\lceil T \rceil]\!] = \flat [\![T]\!]$$
$$[\![(x : \check{\tau}_1) \Rightarrow \tau_2]\!] = (x : [\![\check{\tau}_1]\!]) \to [\![\tau_2]\!]$$
$$[\![\texttt{ctx}]\!] = \texttt{Ctx}$$

Interpretation of computation contexts

$$[\![\cdot]\!] = \cdot$$
$$[\![\Gamma, x : \check{\tau}]\!] = [\![\Gamma]\!], x : [\![\check{\tau}]\!]$$

Interpretation of contextual objects

$$[\![\Gamma \vdash (\hat{\Psi} \vdash M) : (\Psi \vdash A)]\!] = [\![\Gamma; \Psi \vdash M : A]\!]$$
$$[\![\Gamma \vdash (\hat{\Psi} \vdash M) : (\Psi \vdash_v A)]\!] = [\![\Gamma; \Psi \vdash M : A]\!]$$

Interpretation of contextual types

$$[\![\Gamma \vdash (\Psi \vdash A)]\!] = (u : El([\![\Gamma \vdash \Psi \ \ \texttt{ctx}]\!])) \to TmEl(u, [\![\Gamma; \Psi \vdash A \ \ \texttt{type}]\!])$$
$$[\![\Gamma \vdash (\Psi \vdash_v A)]\!] = (u : El([\![\Gamma \vdash \Psi \ \ \texttt{ctx}]\!])) \to_v TmEl(u, [\![\Gamma; \Psi \vdash A \ \ \texttt{type}]\!])$$

Interpretation of computation terms

$$[\![\Gamma \vdash \lceil C \rceil : \lceil T \rceil]\!] = \texttt{box}\ [\![\Gamma \vdash C : T]\!]$$
$$[\![\Gamma \vdash t_1\ t_2 : \tau]\!] = [\![\Gamma \vdash t_1 : (x : \check{\tau}_2) \Rightarrow \tau]\!]\ [\![\Gamma \vdash t_2 : \check{\tau}_2]\!]$$
$$[\![\Gamma \vdash \texttt{fn}\ x \Rightarrow t : (x : \check{\tau}_1) \Rightarrow \tau_2]\!] = \lambda x : [\![\check{\tau}_1]\!].[\![\Gamma, x : \check{\tau}_1 \vdash t : \tau_2]\!]$$
$$[\![\Gamma \vdash x : \tau]\!] = x$$

Fig. 6. Interpretation of computation objects (without recursors)

$$\frac{\Gamma \vdash A : Ty(\top) \qquad \Gamma \vdash \Psi : \texttt{Ctx} \qquad \Gamma \vdash x : \flat(TmEl(\top, A))}{\Gamma \vdash \texttt{lift}(\Psi, x) \equiv \texttt{let box}\ x' = x\ \texttt{in}\ \lambda u.x'\{!\} : (u : El(\Psi)) \to TmEl(u, A\{!\})}$$

$$\frac{\begin{array}{c}\Gamma \vdash \Psi : \texttt{Ctx} \qquad \Gamma \vdash a : (u : El(\Psi)) \to TmEl(u, ty) \\ \Gamma \vdash b : (u : El(\Psi)) \to TmEl(u, ty) \qquad \Gamma \vdash f : \flat((u : El(\Psi.trm\{\overline{a}\})) \to TmEl(u, trm\{\overline{b'}\}))\end{array}}{\Gamma \vdash lam'(\Psi, f) : \flat((u : El(\Psi)) \to TmEl(u, trm\{\overline{arr(a, b)}\}))}$$

$$b' \equiv \lambda u.b(p(u))\{p\} : (u : El(\Psi.trm\{\overline{a}\})) \to TmEl(u, ty)$$

$$lam'(\Psi, f) \equiv \texttt{let box}\ f' = f\ \texttt{in box}\ lam(a, b, \lambda x.\lambda u, f'(\texttt{pair}(u, x(u)))\{\overline{x}\})$$

$$\frac{\begin{array}{c}\Gamma \vdash \Psi : \texttt{Ctx} \qquad \Gamma \vdash a : (u : El(\Psi)) \to TmEl(u, ty) \qquad \Gamma \vdash b : (u : El(\Psi)) \to TmEl(u, ty) \\ \Gamma \vdash m : \flat((u : El(\Psi)) \to TmEl(u, trm\{\overline{arr(a, b)}\})) \qquad \Gamma \vdash n : \flat((u : El(\Psi)) \to TmEl(u, trm\{\overline{a}\}))\end{array}}{\Gamma \vdash app'(\Psi, m, n) : \flat((u : El(\Psi)) \to TmEl(u, trm\{\overline{b}\}))}$$

$$app'(\Psi, m, n) \equiv \texttt{let box}\ m' = m\ \texttt{in let box}\ n' = n\ \texttt{in box}\ app(a, b, m', n')$$

Fig. 7. Auxiliary functions for defining semantic recursor of *trm*

$$\frac{\Gamma \vdash \Psi : \mathtt{Ctx} \qquad \Gamma \vdash z : \flat(TmEl(u, ty)) \qquad \Gamma \vdash y : \flat((u : El(\Psi)) \to TmEl(u, trm\{\overline{\mathtt{lift}(\Psi, z)}\}))}{\Gamma \vdash R(\Psi, z, y)\ \ \mathsf{type}}$$

$$\frac{\Gamma \vdash \Psi : \mathtt{Ctx} \qquad \Gamma \vdash a : \flat(TmEl(\top, ty)) \qquad \Gamma \vdash t : \flat((u : El(\Psi)) \to_v TmEl(u, trm\{\overline{\mathtt{lift}(\Psi, a)}\}))}{\Gamma \vdash B_v(\Psi, a, t) : R(\Psi, a, t)}$$

$$\frac{\begin{array}{c}\Gamma \vdash \Psi : \mathtt{Ctx} \qquad \Gamma \vdash a : \flat(TmEl(\top, ty)) \\ \Gamma \vdash b : \flat(TmEl(\top, ty)) \qquad \Gamma \vdash m : \flat((u : El(\Psi.trm\{\overline{\mathtt{lift}(\Psi, a)}\})) \to TmEl(u, trm\{\overline{\mathtt{lift}(\Psi, b)}\})) \\ \Gamma \vdash f_m : R(\Psi.trm\{\overline{\mathtt{lift}(\Psi, a)}\}, b, m)\end{array}}{\Gamma \vdash B_{lam}(\Psi, m, f_m) : R(\Phi, \mathsf{box}\ (arr(\mathtt{lift}(\cdot, a), \mathtt{lift}(\cdot, b))), lam'(\Psi, m))}$$

$$\frac{\begin{array}{c}\Gamma \vdash \Psi : \mathtt{Ctx} \qquad \Gamma \vdash a : \flat(TmEl(\top, ty)) \\ \Gamma \vdash b : \flat(TmEl(\top, ty)) \qquad \Gamma \vdash m : \flat((u : El(\Psi)) \to TmEl(u, trm\{\overline{arr(\mathtt{lift}(\Psi, a), \mathtt{lift}(\Psi, b))}\})) \\ \Gamma \vdash n : \flat((u : El(\Psi)) \to TmEl(u, trm\{\overline{\mathtt{lift}(\Psi, a)}\})) \\ \Gamma \vdash f_m : R(\Psi, \mathsf{box}\ (arr(\mathtt{lift}(\cdot, a), \mathtt{lift}(\cdot, b))), m) \qquad \Gamma \vdash f_n : R(\Psi, a, n)\end{array}}{\Gamma \vdash B_{app}(\Psi, m, n, f_m, f_n) : R(\Psi, b, app'(\Psi, m, n))}$$

$$\frac{}{\begin{array}{c}\Gamma \vdash rec_{trm}(B_v, B_{lam}, B_{app}) : \\ (\Psi : \mathtt{Ctx}) \to (z : \flat(TmEl(\top, ty))) \to (y : \flat((u : El(\Psi)) \to TmEl(u, trm\{\overline{\mathtt{lift}(\Psi, z)}\}))) \to R(\Psi, z, y)\end{array}}$$

Axiomatic equations

$$\begin{array}{l} rec_{trm}(B_v, B_{lam}, B_{app}, \Psi, z, x) = B_v(\Psi, z, x) \qquad x : \flat((u : El(\Psi)) \to_v TmEl(u, trm\{\overline{\mathtt{lift}(\Psi, z)}\})) \\ rec_{trm}(B_v, B_{lam}, B_{app}, \Psi, \mathsf{box}\ (arr(\mathtt{lift}(\cdot, a), \mathtt{lift}(\cdot, b))), lam'(\Psi, m)) \\ = B_{lam}(\Psi, f, rec_{trm}(B_v, B_{lam}, B_{app}, \Psi.trm\{\overline{\mathtt{lift}(\Psi, a)}\}, b, m)) \\ rec_{trm}(B_v, B_{lam}, B_{app}, \Psi, b, app'(\Psi, m, n)) \\ = B_{lam}(\Psi, f, m, n, rec_{trm}(B_v, B_{lam}, B_{app}, \Psi, \mathsf{box}\ (arr(\mathtt{lift}(\cdot, a), \mathtt{lift}(\cdot, b))), m), rec_{trm}(B_v, B_{lam}, B_{app}, \Psi, a, n)) \end{array}$$

Fig. 8. Induction principle for trm

Contrarily, the semantic recursor for tm is interesting, because the lam case uses HOAS. We first define a number of auxiliary functions in Figure 7. These function provides a wrapper for the $\flat$ modality so that the presentation is more concise. lam' is flagged red because it requires assumption of some substitution properties of the semantic LF type trm. For conciseness, certain parameters are omitted and they can be inferred. The semantic recursor is presented in Figure 8. The axiomization is a relatively straightforward extension of the one in the simply typed case and corresponds to the syntactical recursor. In many places, we have the pattern $trm\{\overline{\mathtt{lift}(\Psi, z)}\}$ where $z : \flat(TmEl(\top, ty))$. This pattern correspond to $\lfloor z \rfloor$. in the syntax. The reasoning is the following:

$$\begin{array}{c} z : \flat(TmEl(\top, ty)) \\ \mathtt{lift}(\Psi, z) : (u : El(\Psi)) \to TmEl(u, ty) \\ trm\{\overline{\mathtt{lift}(\Psi, z)}\} : Ty(\Psi) \end{array}$$

Thus this pattern is necessary to have the terms well-formed.

$$\mathcal{I} = (\psi : \texttt{ctx}) \Rightarrow (z : \lceil \vdash ty \rceil) \Rightarrow (y : \lceil \psi \vdash trm\ \lfloor z \rfloor . \rceil) \Rightarrow \tau$$

$$\begin{aligned}
[\![\Gamma \vdash \texttt{rec}^{\mathcal{I}}\ (b_v \mid b_{lam} \mid b_{app})\ \Psi\ t\ t' : \tau[\Psi, t, t'/\psi, z, y]]\!] &= rec_{trm}(e_v, e_{lam}, e_{app}, e_\Psi, e_t, e_{t'}) \\
\text{where } e_\Psi &= [\![\Gamma \vdash \Psi\ \texttt{ctx}]\!] \\
e_t &= \texttt{let box } x = [\![\Gamma \vdash t : \lceil \vdash ty \rceil]\!] \texttt{ in box } (x(\top)) \\
e_{t'} &= [\![\Gamma \vdash t' : \lceil \Psi \vdash trm\ \lfloor t \rfloor . \rceil]\!] \\
e_v &= [\![\Gamma \vdash b_v : \mathcal{I}]\!] \\
e_{lam} &= [\![\Gamma \vdash b_{lam} : \mathcal{I}]\!] \\
e_{app} &= [\![\Gamma \vdash b_{app} : \mathcal{I}]\!]
\end{aligned}$$

Interpretation of branches

$$\begin{aligned}
[\![\Gamma \vdash b_v : \mathcal{I}]\!] &= \lambda \Psi\ a\ t \rightarrow e \\
\text{where } e &= [\![\Gamma, \Psi : \texttt{ctx}, a : \lceil \vdash ty \rceil, t : \lceil \Psi \vdash_v trm\ \lfloor a \rfloor . \rceil \vdash t_v : \tau[a, t/z, y]]\!] \\
[\![\Gamma \vdash b_{lam} : \mathcal{I}]\!] &= \lambda \Psi\ a\ b\ m\ f_m \rightarrow e \\
\text{where } e &= [\![\text{the premise judgment}]\!] \\
[\![\Gamma \vdash b_{app} : \mathcal{I}]\!] &= \lambda \Psi\ a\ b\ m\ n\ f_m\ f_n \rightarrow e \\
\text{where } e &= [\![\text{the premise judgment}]\!]
\end{aligned}$$

Fig. 9. Interpretation of the recursor of trm

We can then interpret the syntactical recursor to the semantic recursor. The interpretation function is shown in Figure 9. The interpretation is very straightforward due to their similarity.

5 CONCLUSION

In this report, we show the interpretation of a version of Cocon with dependent types to a presheaf model with two level categories with attributes. Unfortunately, the interpretation is not sound; there are various mistakes in the interpretation of the LF objects. The primary difficulty is the occurrences of computation level terms indexing tm. As a future work, it would be beneficial to reconsider a semantic model for contextual types with dependent types. An ideal semantic model for contextual types should reflect arbitrary computation on the LF level and also be suggestive of multi-level computation beyond two levels.

REFERENCES

Steve Awodey. 2010. *Category Theory* (2nd ed.). Oxford University Press, Inc., USA.

Hendrik Pieter Barendregt. 1985. *The lambda calculus - its syntax and semantics*. Studies in logic and the foundations of mathematics, Vol. 103. North-Holland.

Lars Birkedal, Ranald Clouston, Bassel Mannaa, Rasmus Ejlers Møgelberg, Andrew M. Pitts, and Bas Spitters. 2019. Modal dependent type theory and dependent right adjoints. *Mathematical Structures in Computer Science* (2019), 1–21. https://doi.org/10.1017/S0960129519000197

John Cartmell. 1986. Generalised algebraic theories and contextual categories. *Ann. Pure Appl. Log.* 32 (1986), 209–243. https://doi.org/10.1016/0168-0072(86)90053-9

Arthur Charguéraud. 2012. The Locally Nameless Representation. *J. Autom. Reasoning* 49, 3 (2012), 363–408. https://doi.org/10.1007/s10817-011-9225-2

Pierre Clairambault and Peter Dybjer. 2011. The Biequivalence of Locally Cartesian Closed Categories and Martin-Löf Type Theories. In *Typed Lambda Calculi and Applications - 10th International Conference, TLCA 2011, Novi Sad, Serbia,*

June 1-3, 2011. Proceedings (Lecture Notes in Computer Science), C.-H. Luke Ong (Ed.), Vol. 6690. Springer, 91–106. https://doi.org/10.1007/978-3-642-21691-6_10

Rowan Davies and Frank Pfenning. 2001. A modal analysis of staged computation. *J. ACM* 48, 3 (2001), 555–604. https://doi.org/10.1145/382780.382785

Nicolaas Govert de Bruijn. 1972. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae (Proceedings)* 75, 5 (1972), 381 – 392. https://doi.org/10.1016/1385-7258(72)90034-0

Joëlle Despeyroux, Frank Pfenning, and Carsten Schürmann. 1997. Primitive Recursion for Higher-Order Abstract Syntax. In *Typed Lambda Calculi and Applications, Third International Conference on Typed Lambda Calculi and Applications, TLCA '97, Nancy, France, April 2-4, 1997, Proceedings (Lecture Notes in Computer Science)*, Philippe de Groote (Ed.), Vol. 1210. Springer, 147–163. https://doi.org/10.1007/3-540-62688-3_34

Daniel Gratzer, Jonathan Sterling, and Lars Birkedal. 2019. Implementing a modal dependent type theory. *PACMPL* 3, ICFP (2019), 107:1–107:29. https://doi.org/10.1145/3341711

Robert Harper, Furio Honsell, and Gordon D. Plotkin. 1993. A Framework for Defining Logics. *J. ACM* 40, 1 (1993), 143–184. https://doi.org/10.1145/138027.138060

Martin Hofmann. 1997. *Syntax and Semantics of Dependent Types.* Cambridge University Press, 79–130. https://doi.org/10.1017/CBO9780511526619.004

Joachim Lambek. 1985. Cartesian Closed Categories and Typed Lambda- calculi. In *Combinators and Functional Programming Languages, Thirteenth Spring School of the LITP, Val d'Ajol, France, May 6-10, 1985, Proceedings (Lecture Notes in Computer Science)*, Guy Cousineau, Pierre-Louis Curien, and Bernard Robinet (Eds.), Vol. 242. Springer, 136–175. https://doi.org/10.1007/3-540-17184-3_44

Daniel R. Licata, Ian Orton, Andrew M. Pitts, and Bas Spitters. 2018. Internal Universes in Models of Homotopy Type Theory. In *3rd International Conference on Formal Structures for Computation and Deduction, FSCD 2018, July 9-12, 2018, Oxford, UK (LIPIcs)*, Hélène Kirchner (Ed.), Vol. 108. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 22:1–22:17. https://doi.org/10.4230/LIPIcs.FSCD.2018.22

Saunders MacLane. 1971. *Categories for the Working Mathematician.* Springer-Verlag, New York. ix+262 pages. Graduate Texts in Mathematics, Vol. 5.

Per Martin-Löf. 1984. *Intuitionistic type theory.* Studies in proof theory, Vol. 1. Bibliopolis.

Aleksandar Nanevski, Frank Pfenning, and Brigitte Pientka. 2008. Contextual modal type theory. *ACM Trans. Comput. Log.* 9, 3 (2008), 23:1–23:49. https://doi.org/10.1145/1352582.1352591

Christine Paulin-Mohring. 1989. *Program Extraction in the Calculus of Constructions.* Ph.D. Dissertation. Université Paris-Diderot - Paris VII. https://tel.archives-ouvertes.fr/tel-00431825

Frank Pfenning and Rowan Davies. 2001. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science* 11, 4 (2001), 511–540. https://doi.org/10.1017/S0960129501003322

Frank Pfenning and Conal Elliott. 1988. Higher-Order Abstract Syntax. In *Proceedings of the ACM SIGPLAN'88 Conference on Programming Language Design and Implementation (PLDI), Atlanta, Georgia, USA, June 22-24, 1988*, Richard L. Wexelblat (Ed.). ACM, 199–208. https://doi.org/10.1145/53990.54010

Brigitte Pientka and Ulrich Schöpp. 2020. Semantical Analysis of Contextual Types. In submission to *Foundations of Software Science and Computation Structures - 23nd International Conference, FOSSACS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020 (Lecture Notes in Computer Science).* Springer.

Brigitte Pientka, David Thibodeau, Andreas Abel, Francisco Ferreira, and Rébecca Zucchini. 2019. A Type Theory for Defining Logics and Proofs. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019.* IEEE, 1–13. https://doi.org/10.1109/LICS.2019.8785683

Robert A. G. Seely. 1984. Locally cartesian closed categories and type theory. *Mathematical Proceedings of the Cambridge Philosophical Society* 95, 1 (1984), 33–48. https://doi.org/10.1017/S0305004100061284

Michael Shulman. 2018. Brouwer's fixed-point theorem in real-cohesive homotopy type theory. *Mathematical Structures in Computer Science* 28, 6 (2018), 856–941. https://doi.org/10.1017/S0960129517000147

A APPENDIX

A.1 Comparison with Pientka and Schöpp [2020, Section 6]

The semantic model represented in this report has some significant differences with the one scatched in Pientka and Schöpp [2020, Section 6]. In particular, the semantic model in this report is heavily based on semantic substitution. This section briefly discusses the rationale behind.

The object language is stated as follows:

$$\Gamma, \Phi : \texttt{Ctx} \vdash ty : Ty(\Phi) \qquad \Gamma, \Phi : \texttt{Ctx} \vdash tm : Ty(\Phi.ty) \qquad \Gamma, \Phi : \texttt{Ctx}, u : El(\Phi) \vdash o : Tm(u, ty)$$

$$\Gamma, \Phi : \texttt{Ctx}, u : El(\Phi) \vdash arr : Tm(u, ty) \to Tm(u, ty) \to Tm(u, ty)$$

$$\Gamma, \Phi : \texttt{Ctx}, u : El(\Phi), a, b : Tm(u, ty) \vdash$$

$$lam : (Tm(\texttt{pair}(u, a), tm) \to Tm(\texttt{pair}(u, b), tm)) \to Tm(\texttt{pair}(u, (arr\ a\ b)), tm)$$

$$\Gamma, \Phi : \texttt{Ctx}, u : El(\Phi), a, b : Tm(u, ty) \vdash$$

$$app : Tm(\texttt{pair}(u, (arr\ a\ b)), tm) \to Tm(\texttt{pair}(u, a), tm) \to Tm(\texttt{pair}(u, b), tm)$$

Consider the following LF judgment and the semantic type of $[\![y]\!]$:

$$\frac{}{\cdot; x : ty, y : tm\ x \vdash y : tm\ x}$$

If we avoid using substitution, we fall into the problem of having to use the top most term to represent the indexer of tm:

$$Tm(\texttt{pair}(\texttt{pair}(\texttt{pair}(\top, [\![x]\!]), [\![y]\!]), [\![x]\!]), tm)$$

Notice that x needs to be interpreted twice in order to keep $[\![x]\!]$ on top of the stack. The Yoneda embedded context has length three while the syntactical context has length two.

Consider another variation:

$$\frac{}{\cdot; y : tm\ o \vdash y : tm\ o}$$

$[\![y]\!]$ must have the following semantic type:

$$Tm(\texttt{pair}(\texttt{pair}(\texttt{pair}(\top, o), [\![y]\!]), o), tm)$$

In this case, the semantic context still has length three but the length of the syntactical context has been reduced to one.

In general, with the semantic model presented in Pientka and Schöpp [2020, Section 6], a semantic context is at least as long as, often longer than, the corresponding syntactical context, but the difference in length depends how the type dependencies are formulated and therefore the interpretation of the LF contexts needs to be intertwined with the interpretation of the LF types, which at the very least is not the case in this report. If the lengths of the interpreted contexts cannot be determined, then the target soundness theorems as stated in Conjecture 4.7 cannot be properly formulated.