Semantical Analysis of Contextual Types

Brigitte Pientka
McGill University, Canada
bpientka@cs.mcgill.ca

Ulrich Schöpp
LMU Munich, Germany
Ulrich.Schoepp@ifi.lmu.de

Abstract

Higher-order abstract syntax (HOAS) is an elegant and deceptively simple idea of encoding syntax that models binding and scope of variables by piggy-backing on binding in the meta-language. This requires a weak function space in the meta-language that does not admit recursion or pattern matching. An existing approach to nevertheless support recursion over HOAS syntax trees is by characterising them together with the context in which they are meaningful as contextual types and to embed contextual types into a computation language that has a strong function space.

In this paper, we give a semantic account of contextual types and of writing computations about them. We build on previous work on modelling higher-order abstract syntax using presheaf categories. We show that these presheaf models already have all the structure needed to model contextual types over a simply-typed lambda-calculus together with computations about them. This gives a simple semantic characterisation of the invariants of contextual types. It identifies contextual types as a type-theoretic presentation of these models with good algorithmic properties. To capture contextual types over dependently-typed lambda-calculi, such as the logical framework LF, we then generalise the approach by using presheaves over a Category with Attributes. We present the structure of the presheaf models in terms of their internal dependent type theory. This formulation makes working with type-dependencies manageable and provides an abstract core calculus for the interpretation of contextual types and computations about them. In particular, it provides a semantic reconstruction of the type-theoretic foundations underlying the proof environment Beluga.

2012 ACM Subject Classification Theory of computation → Program semantics

Keywords and phrases type theory, higher-order abstract syntax, presheaf semantics

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Introduction

A fundamental question when defining, implementing, and working with languages and logics is: How do we represent and analyse syntactic structures? — Defining such structures as abstract syntax trees (AST) allows us to define them via constructors needed to form phrases. However, traditionally ASTs do not capture one essential aspect of syntactic structures, namely variable binding and scope. As a consequence, the AST representations that use different bound variable names are distinct, while we want to treat them as α-equivalent.

Higher-order abstract syntax [15] (or lambda-tree syntax [13]) offers a more abstract view by choosing a more powerful meta-language for defining syntactic structures: instead of using a first-order meta-language, we choose a higher-order meta-language where we can map binders in our object language to binders (i.e. functions) in our meta-language.

In the logical framework LF [7], a dependently typed lambda-calculus, we can define a small functional programming language (object language) consisting of functions, function application, and let-expressions using a type `tm` as:

```
lam : (tm -> tm) -> tm.
app : tm -> tm -> tm.
letv: tm -> (tm -> tm) -> tm.
```


42nd Conference on Very Important Topics (CVIT 2016).
Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The object-language term (lam x.lam y.let $w = x\ y$ in $w\ y$) is then encoded as

```
lam \x.lam \y.letv (app x y) \w.app w y
```

using the LF abstractions to model binding. Object-level substitution is modelled through LF application; for instance, the fact that ((lam $x.M$) N) reduces to $M[N/x]$ in our object language is expressed as `(app (lam M) N)` reducing to `(M N)`.

This approach is elegant and can offer substantial benefits: we can treat two objects equivalent modulo renaming and two objects are substitution invariant. From a very practical point of view, we do not need to build up the basic mathematical infrastructure and can work at a higher-level of abstraction.

However, we not only want to construct HOAS trees, but also analyse them and select sub-trees. This is challenging, as sub-trees are context sensitive, i.e. they depend on a binder outside. For example, `letv (app x y) \w.app w y` only makes sense in a context `x:tm,y:tm`. Moreover, one cannot simply extend LF to allow syntax analysis. If one simply added a recursion combinator to LF, then it could be used to define many functions `M: tm -> tm` for which `lam M` would not represent a syntax term [9].

Contextual types [14, 16] offer a type-theoretic solution to these problems by reifying the typing judgement, i.e. that `letv (app x y) \w.app w y` has type `tm` in the context `x:tm,y:tm`, as a type (or proposition): we tie the object together with the context in which it is meaningful. This insight provides a handle on recursively analysing HOAS trees, separating cleanly the weak LF function space that is used to define HOAS trees from the strong function space needed for describing recursive functions about HOAS trees [16, 21, 17]. Contextual types hence allow us to mediate between HOAS trees and computations. The Beluga proof and programming language [22, 20] provides an implementation of these ideas.

In this paper, we aim to give a semantic analysis of contextual types and computations we do on them. There are a number of categorical models of abstract syntax with bindings [9, 5, 6] that are all closely related. Our work takes Hofmann's work [9] as a starting point, since it considers full Higher-Order Abstract Syntax (HOAS). Hofmann shows how a presheaf category soundly models full HOAS within a very expressive universe. While Hofmann has shown that the model justifies the axioms of the Theory of Contexts [10], these did not have computational content. In this paper, we first show how Hofmann's work also provides an explanation for computations over contextual HOAS trees, as for example found in Beluga.

We start by showing that Hofmann's model is a natural semantics for characterising contextual types over a simply-typed meta-language. In particular, computation over contextual types corresponds to natural transformations. Concentrating on a simply-typed meta-language will allow us to introduce the main idea without the additional complexity that choosing a dependently typed meta-language such as LF brings with it.

We then go a step further and extend Hofmann's approach to account also for contextual types over a dependently typed lambda-calculus, such as LF, together with computations about them. To this end, we use presheaves over a Category of Attributes [2].

Our work has several benefits: First, it highlights the relationship between Hofmann's categorical model and Beluga's type-theoretic foundation [21, 17]. In doing so, it provides a semantical reconstruction and justification of Beluga's proof language from a conceptually simple semantic idea. Second, the semantic model we describe provides a foundation for compiling contextual HOAS trees and hence lays the foundation for building generic libraries within proof assistants such as [1]. Last, this categorical view provides insights into building a general type theory with contextual types and we believe it also may serve as a foundation for the recent proposal Cocon [18, 19].

2 Presheaves for Higher-Order Abstract Syntax

Our work begins with the presheaf semantics for HOAS of [9, 5]. The key idea of those approaches is to integrate substitution-invariance in the computational universe in a controlled way. For the representation of abstract syntax, one wants to allow only substitution-invariant constructions. For example, `lam M` represents an object-level abstraction only if `M` is a function that uses its argument in a substitution-invariant way. For computation with abstract syntax, on the other hand, one wants to allow non-substitution-invariant constructions too. Presheaf categories allow one to choose the desired amount of substitution-invariance.

Let $\mathbb{D}$ be a small category. The presheaf category $\widehat{\mathbb{D}}$ is defined to be the category $\mathrm{Set}^{\mathbb{D}^{\mathrm{op}}}$. Its objects are functors $F\colon \mathbb{D}^{\mathrm{op}} \to \mathrm{Set}$, which are also called *presheaves*. Such a functor F is given by a set $F(\Gamma)$ for each object Γ of $\mathbb{D}$ together with a function $F(\sigma)\colon F(\Delta) \to F(\Gamma)$ for any $\sigma\colon \Gamma \to \Delta$ in $\mathbb{D}$, subject to the functor laws. The intuition is that F defines sets of elements in various $\mathbb{D}$-contexts, together with a $\mathbb{D}$-substitution action. A morphism $f\colon F \to G$ is a natural transformation, which is a family of functions $f_\Gamma\colon F(\Gamma) \to G(\Gamma)$ for any Γ. This family of functions must be natural, i.e. commute with substitution $f_\Gamma \circ F(\sigma) = F(\sigma) \circ f_\Delta$.

For the purposes of modelling higher-order abstract syntax, $\mathbb{D}$ will typically be the term model of some domain-level lambda-calculus. By domain-level, we mean the calculus that serves as the meta-level for object-language encodings. We use this term to avoid possible confusion between different meta-levels later. For simplicity, let us for now use a simply-typed lambda-calculus with functions and products as the domain language. It is sufficient to encode the example from the introduction and allows us to explain the main idea underlying our approach.

As is well known, the term model $\mathbb{D}$ of the simply-typed lambda-calculus is a cartesian closed category. The objects of $\mathbb{D}$ are simple types and a morphism $A \to B$ in $\mathbb{D}$ is a term $x : A \vdash t : B$. In the definition for morphisms, terms are identified up to $\beta\eta$-equality. The unit type 1 is a terminal object, the pair type $A \times B$ is a product, and the function type $A \,\texttt{->}\, B$ is an exponential. Using the cartesian product, contexts can also be considered as types, e.g. $x{:}\,\mathsf{tm}$, $y{:}\,\mathsf{tm}$ becomes $\mathsf{tm} \times \mathsf{tm}$. A morphism of type $\Gamma \to \Delta$ in $\mathbb{D}$ amounts to a substitution that provides a term in context Γ for each of the variables in Δ.

By choosing $\widehat{\mathbb{D}}$ as a computational universe, one can enforce some constructions to be invariant under substitutions, while allowing arbitrary constructions elsewhere, by choosing the objects appropriately. In one extreme, a normal set S can be represented by a constant presheaf $[S]$ such that $[S](\Gamma) = S$ and $[S](\sigma) = id$ for all Γ and σ. The Yoneda embedding represents the other extreme. For any object Δ of $\mathbb{D}$ the functor $\mathrm{y}(\Delta)\colon \mathbb{D}^{\mathrm{op}} \to \mathrm{Set}$ is defined by $\mathrm{y}(\Delta)(\Gamma) = \mathbb{D}(\Gamma, \Delta)$, which is the set of morphisms from Γ to Δ in $\mathbb{D}$. The functor action is pre-composition. The functor $\mathrm{y}(\Delta)$ should be understood as the type of all domain-level substitutions into Δ. An important example is $\mathsf{Tm} := y(\mathsf{tm})$, which is such that $\mathsf{Tm}(\Gamma)$ is the set of all terms in context Γ (when considered as substitutions of type $\Gamma \to \mathsf{tm}$).

The Yoneda embedding y is functorial in its first argument and defines a functor $\mathrm{y}\colon \mathbb{D} \to \widehat{\mathbb{D}}$. This functor embeds $\mathbb{D}$ into the category $\widehat{\mathbb{D}}$. It embeds $\mathbb{D}$ fully and faithfully, which means that y is a bijection from $\mathbb{D}(\Gamma, \Delta)$ to $\widehat{\mathbb{D}}(\mathrm{y}(\Gamma), \mathrm{y}(\Delta))$ for all Γ and Δ.

For terms $\mathsf{Tm} = \mathrm{y}(\mathsf{tm})$, we have that the morphisms $\mathsf{Tm} \to \mathsf{Tm}$ in $\widehat{\mathbb{D}}$ are in one-to-one correspondence with substitutions $\mathsf{tm} \to \mathsf{tm}$ in $\mathbb{D}$. These, in turn, correspond to α-equivalence classes of simply-typed lambda terms with one free variable. This shows that substitution invariance cuts down the morphisms from $\mathsf{Tm} \to \mathsf{Tm}$ just as much as we would like for adequate HOAS encodings. The argument to `lam` should be a function that represents a term with a free variable. This observation is the basis for Hofmann's presheaf model of HOAS [9].

3 From Presheaves to Contextual Types

The embedding of $\mathbb{D}$ into $\widehat{\mathbb{D}}$ naturally leads to a model of contextual types and computations about them. To explain this, we need to look closer at the structure of $\widehat{\mathbb{D}}$. Describing it directly in terms of functors and natural transformations is somewhat laborious and the technical details may obscure the basic idea of our approach. Instead, we use a dependent type theory for working with $\widehat{\mathbb{D}}$. This significantly reduces the amount of technical detail. It also makes it possible to use existing proof assistants to type-check constructions in $\widehat{\mathbb{D}}$. We have used Agda[1] for this purpose, which was quite helpful with type dependencies in Sec. 7.

In the following, we work in a standard dependent type theory with dependent products, sums and identity types. It is well-known that $\widehat{\mathbb{D}}$ has the structure to interpret a type theory with such types, see e.g. [8, §4]. The category $\widehat{\mathbb{D}}$ justifies proof-irrelevant extensional identity types. We shall use informal equational reasoning for working with them.

3.1 Yoneda Universe

For our purposes, the main feature of $\widehat{\mathbb{D}}$ is that it embeds $\mathbb{D}$ fully and faithfully via the Yoneda embedding. In type theoretic terms, one may think of $\mathbb{D}$ as a universe.

The set of objects of $\mathbb{D}$ can be represented in the type theory of $\widehat{\mathbb{D}}$ by a type Obj. We have seen above that any set can be represented as a presheaf with trivial substitution action, and Obj is one such example. Particular objects of $\mathbb{D}$ then appear as terms of type Obj. The cartesian closed structure gives us terms unit, times, arrow for the terminal object, finite products $\times$ and the exponential `->`. In our running example, where $\mathbb{D}$ is the term model of a simply-typed lambda calculus with a basic type tm, we also have a term for tm.

$$\vdash \mathsf{Obj}\ \text{type} \qquad \vdash \mathsf{tm}\colon \mathsf{Obj} \qquad \vdash \mathsf{times}\colon \mathsf{Obj} \to \mathsf{Obj} \to \mathsf{Obj}$$
$$\vdash \mathsf{unit}\colon \mathsf{Obj} \qquad \vdash \mathsf{arrow}\colon \mathsf{Obj} \to \mathsf{Obj} \to \mathsf{Obj}$$

We shall confuse objects of $\mathbb{D}$ with terms of type Obj.

The morphisms of $\mathbb{D}$ could similarly be encoded as a constant presheaf with constants, but it is easier to view Obj as a type-theoretic universe:

$$x\colon \mathsf{Obj} \vdash \mathsf{El}\, x\ \text{type}$$

The type $\mathsf{El}\,A$ corresponds to the presheaf $\mathrm{y}(A)$. This means that one can think of $\mathsf{El}\,A$ as the type of all morphisms of type $\Gamma \to A$ in $\mathbb{D}$ for arbitrary Γ, which are called generalised elements of A. If $\mathbb{D}$ is a term model as above, then a morphism of type $\Gamma \to A$ is an open term of type A that may refer to variables in Γ. In this case, the elements of $\mathsf{El}\,A$ are just domain-level terms of type A, both closed and open ones.

The universe El represents the morphisms of $\mathbb{D}$ fully and faithfully. This means that the type $\mathsf{El}\,A \to \mathsf{El}\,B$ represents the morphisms of type $A \to B$ in $\mathbb{D}$. Any closed term of type $\mathsf{El}\,A \to \mathsf{El}\,B$ corresponds to such a morphism and vice versa. Note that this says that the functions $\mathsf{El}\,A \to \mathsf{El}\,B$ consist not of arbitrary functions from generalised elements to generalised elements, but only of ones that arise from post-composition with a morphism of type $A \to B$. In essence, this is achieved by allowing only operations that are closed under substitution. If a function of type $\mathsf{El}\,A \to \mathsf{El}\,B$ maps a generalised element that represents a variable x to some term t, then it must map s to $t[s/x]$. This is enforced by naturality in

[1] Our Agda sources are available from: `http://github.com/uelis/contextual`

the construction of the presheaf category $\widehat{\mathbb{D}}$. In categorical terms, El represents the Yoneda embedding and the correspondence amounts to the Yoneda embedding being full and faithful.

We note that a term of type $\forall c, d\colon \mathsf{Obj}.\, \mathsf{El}\, c \to \mathsf{El}\, d$ corresponds to a family of terms $\mathsf{El}\, A \to \mathsf{El}\, B$ for all objects A and B in $\mathbb{D}$. This is because Obj is just a set, so that the naturality constraints of $\widehat{\mathbb{D}}$ are vacuous for functions out of Obj.

Since El represents the morphisms of $\mathbb{D}$, we can lift the structure of $\mathbb{D}$ to the meta-level of the internal type theory of $\widehat{\mathbb{D}}$. The following lemmas state that the Yoneda embedding preserves terminal object, binary products and the exponential.

▶ **Lemma 1.** *The internal type theory of $\widehat{\mathbb{D}}$ has a term $\vdash \mathit{terminal}\colon \mathit{El}\,\mathit{unit}$, such that $x = \mathit{terminal}$ is provable for any $x\colon \mathit{El}\,\mathit{unit}$.*

▶ **Lemma 2.** *The internal type theory of $\widehat{\mathbb{D}}$ has the following terms, which are such that $\mathit{fst}\ (\mathit{pair}\ x\ y) = x$ and $\mathit{snd}\ (\mathit{pair}\ x\ y) = y$ and $z = \mathit{pair}\ (\mathit{fst}\ z)\ (\mathit{snd}\ z)$ holds for all x, y and z.*

$$
\begin{aligned}
&c\colon \mathit{Obj}, d\colon \mathit{Obj} \vdash \mathit{fst}\colon \mathit{El}\,(\mathit{times}\ c\ d) \to \mathit{El}\, c\\
&c\colon \mathit{Obj}, d\colon \mathit{Obj} \vdash \mathit{snd}\colon \mathit{El}\,(\mathit{times}\ c\ d) \to \mathit{El}\, d\\
&c\colon \mathit{Obj}, d\colon \mathit{Obj} \vdash \mathit{pair}\colon \mathit{El}\, c \to \mathit{El}\, d \to \mathit{El}\,(\mathit{times}\ c\ d)
\end{aligned}
$$

▶ **Lemma 3.** *The internal type theory of $\widehat{\mathbb{D}}$ has terms*

$$
\begin{aligned}
&c\colon \mathit{Obj}, d\colon \mathit{Obj} \vdash \mathit{arrow\text{-}e}\colon \mathit{El}\,(\mathit{arrow}\ c\ d) \to \mathit{El}\, c \to \mathit{El}\, d\\
&c\colon \mathit{Obj}, d\colon \mathit{Obj} \vdash \mathit{arrow\text{-}i}\colon (\mathit{El}\, c \to \mathit{El}\, d) \to \mathit{El}\,(\mathit{arrow}\ c\ d)
\end{aligned}
$$

satisfying $\mathit{arrow\text{-}i}\ (\mathit{arrow\text{-}e}\ f) = f$ and $\mathit{arrow\text{-}e}\ (\mathit{arrow\text{-}i}\ g) = g$ for all f and g.

3.2 Higher-Order Abstract Syntax

The last lemma in the previous section states that $\mathsf{El}\, A \to \mathsf{El}\, B$ is isomorphic to $\mathsf{El}\,(\mathsf{arrow}\ A\ B)$. This is useful in particular to lift HOAS-encodings from $\mathbb{D}$ to $\widehat{\mathbb{D}}$. For instance, the object-level term constant `lam: (tm -> tm) -> tm` gives rise to an element of $\mathsf{El}\,(\mathsf{arrow}\ (\mathsf{arrow}\ \mathsf{tm}\ \mathsf{tm})\ \mathsf{tm})$. But this type is isomorphic to $(\mathsf{El}\,\mathsf{tm} \to \mathsf{El}\,\mathsf{tm}) \to \mathsf{El}\,\mathsf{tm}$, by the lemma.

This means that the higher-order abstract syntax constants lift to $\widehat{\mathbb{D}}$:

$$
\mathsf{app}\colon \mathsf{El}\,\mathsf{tm} \to \mathsf{El}\,\mathsf{tm} \to \mathsf{El}\,\mathsf{tm} \qquad\qquad \mathsf{lam}\colon (\mathsf{El}\,\mathsf{tm} \to \mathsf{El}\,\mathsf{tm}) \to \mathsf{El}\,\mathsf{tm}
$$

Once one recognises $\mathsf{El}\,\mathsf{tm}$ as $\mathrm{y}(\mathsf{tm})$, the adequacy of this higher-order abstract syntax encoding follows as in [9], as outlined in Sec. 2.

3.3 Closed Types

To model contextual types, we also need to be able to talk about closed terms. To this end, we can use the comonad $\flat\colon \widehat{\mathbb{D}} \to \widehat{\mathbb{D}}$ of closed sections. The presheaf $\flat F$ is the constant presheaf of the set of closed elements in F, which means $\flat F(\Gamma) = F(1)$ and $\flat F(\sigma) = \mathit{id}$. Informally, it restricts any presheaf to the set of its closed elements.

In our meta-level dependent type theory for $\widehat{\mathbb{D}}$, the comonad $\flat$ can be added as a modality $[-]$. For each type X, we add a boxed type $[X]$, which should be thought of as the type of the closed elements from X. The type $[X]$ can be formed if all its free variables also have a boxed type. Given any term $t\colon X$ whose free variables all have boxed type, we can form the term $[t]\colon [X]$. Finally, we have a let-term $\mathsf{let}\ [x] = t\ \mathsf{in}\ s$ that takes a term $t\colon [X]$ and binds it to a variable $x\colon X$. The let-term forgets that t denotes a closed element. In essence, the rules maintain the invariant that the free variables in a type $[X]$ or a term $[t]$ are all boxed.

For the purposes of this paper, this definition of the modality suffices. Its type theoretic and semantic foundation is being developed in crisp type theory [12] and it is closely related to modal type systems in [3, 14]. We have used a development version of Agda [24] that implements a $\flat$-modality.

One should think of $[X]$ as the type of 'closed' elements of X. In particular, $[\mathsf{El}\,A]$ represents *global* elements of A, i.e. the morphisms of type $1 \to A$ in $\mathbb{D}$. If $\mathbb{D}$ is the term model, then these would be closed domain-language terms. The type $\mathsf{El}\,A$, in contrast, contains both closed and open terms.

This applies also to the type $\mathsf{El}\,A \to \mathsf{El}\,B$. We have seen above that $\mathsf{El}\,A \to \mathsf{El}\,B$ is isomorphic to $\mathsf{El}\,(\mathsf{arrow}\ A\ B)$ and may therefore be thought of as containing the α-equivalence classes of terms of type B with a distinguished variable of type A. But, these α-equivalence classes may be open and may contain other free domain language variables. The type $[\mathsf{El}\,A \to \mathsf{El}\,B]$, on the other hand, contains only closed α-equivalence classes.

As a simple example for the usefulness of the modality, we note that the model of our type theory in $\widehat{\mathbb{D}}$ justifies a function like $\mathsf{is\text{-}lam}\colon [\mathsf{El}\,\mathsf{tm}] \to \mathsf{bool}$ that returns true if and only if the argument is a domain language lambda abstraction. Such a function cannot be defined with with type $\mathsf{El}\,\mathsf{tm} \to \mathsf{bool}$, since it would not be invariant under substitution. The argument ranges over terms that may be open; in particular it includes domain-level variables. The function would have to return false for them, since a domain-level variable is not a lambda-abstraction. But after substituting a lambda-abstraction for the variable, it would have to return true, so it could not be substitution-invariant.

We note that the type Obj consists only of closed elements, which makes it equal to $[\mathsf{Obj}]$.

3.4 Contextual Types

Using function types and the modality, it is now possible to work with contextual objects that represent domain level terms in a certain context, much like in [16, 17]. A contextual type is a boxed function type of the form $[\mathsf{El}\,\Gamma \to \mathsf{El}\,A]$. Let us use the notation $[\Gamma \vdash A]$ for it.

For example, object-level terms with up to two free variables now appear as terms of type $[\mathsf{times}\ \mathsf{tm}\ \mathsf{tm} \vdash \mathsf{tm}]$, as the following example of this type illustrates.

$$[\lambda\phi.\,\mathsf{app}\ (\mathsf{lam}\ (\lambda x.\,\mathsf{app}\ (\mathsf{fst}\ \phi)\ x)\ (\mathsf{snd}\ \phi))]\ :\ [\mathsf{times}\ \mathsf{tm}\ \mathsf{tm} \vdash \mathsf{tm}]$$

Of course, one can also introduce meta-level variables for the variables in the domain-level context ϕ by meta-level lets: $[\mathsf{let}\ x_1 = \mathsf{fst}\ \phi\ \mathsf{in}\ \mathsf{let}\ x_2 = \mathsf{snd}\ \phi\ \mathsf{in}\ \mathsf{app}\ (\mathsf{lam}\ (\lambda x.\,\mathsf{app}\ x_1\ x)\ x_2)]$.

The constants app and lam are easily lifted to contextual types

$$\phi\colon \mathsf{Obj} \vdash \mathsf{app}'\colon [\phi \vdash \mathsf{tm}] \to [\phi \vdash \mathsf{tm}] \to [\phi \vdash \mathsf{tm}]$$
$$\phi\colon \mathsf{Obj} \vdash \mathsf{lam}'\colon [\mathsf{times}\ \phi\ \mathsf{tm} \vdash \mathsf{tm}] \to [\phi \vdash \mathsf{tm}]$$

by $\mathsf{app}' := \lambda x, y.\,[\lambda\phi.\,\mathsf{app}\ (x\ \phi)\ (y\ \phi)]$ and $\mathsf{lam}' := \lambda f.\,[\lambda\phi.\,\mathsf{lam}\ (\lambda x.\ f\ (\mathsf{pair}\ \phi\ x))]$. This representation integrates substitution as usual. For example, for $s\colon [\mathsf{times}\ \phi\ \mathsf{tm} \vdash \mathsf{tm}]$ and $t\colon [\phi \vdash \mathsf{tm}]$, the term $[\lambda\phi.\,s\ (\mathsf{pair}\ \phi\ t)]$ represents substitution of t for the last variable in s.

A contextual type for domain-level variables (as opposed to arbitrary terms) can be defined by restricting $[\Gamma \vdash A]$ to the projections out of the context Γ. Projections are all functions that can be built using the terms fst and snd alone. The contextual type $[\Gamma \vdash_v A]$ is then defined as the subtype of $[\Gamma \vdash A]$ of all projections. We allow ourselves an implicit coercion from $[\Gamma \vdash_v A]$ to $[\Gamma \vdash A]$.

With these definitions, we can express a primitive recursion scheme for contextual types.

▶ **Lemma 4.** *Let $\psi\colon \mathit{Obj}$, $x\colon [\psi \vdash \mathit{tm}] \vdash A\ x$ type. Let X_{var}, X_{app} and X_{lam} be defined by:*

$$
\begin{aligned}
X_{\mathit{var}} &:= \forall\phi\colon \mathit{Obj}.\, \forall x\colon [\phi \vdash_v \mathit{tm}].\, A\ x\\
X_{\mathit{app}} &:= \forall\phi\colon \mathit{Obj}.\, \forall x, y\colon [\phi \vdash \mathit{tm}].\, A\ x \to A\ y \to A\ (\mathit{app}'\ x\ y)\\
X_{\mathit{lam}} &:= \forall\phi\colon \mathit{Obj}.\, \forall x\colon [\mathit{times}\ \phi\ \mathit{tm} \vdash \mathit{tm}].\, A\ x \to (\mathit{lam}'\ x)
\end{aligned}
$$

Then, $\widehat{\mathbb{D}}$ justifies a term $\vdash \mathit{rec}\colon X_{\mathit{var}} \to X_{\mathit{app}} \to X_{\mathit{lam}} \to \forall\phi\colon \mathit{Obj}.\, \forall x\colon [\phi \vdash \mathit{tm}].\, A\ x$ such that the following equations are valid.

$$
\begin{aligned}
\mathit{rec}\ t_{\mathit{var}}\ t_{\mathit{app}}\ t_{\mathit{lam}}\ \Phi\ x &= t_{\mathit{var}}\ \Phi\ x \text{ if } x\colon [\Phi \vdash_v \mathit{tm}]\\
\mathit{rec}\ t_{\mathit{var}}\ t_{\mathit{app}}\ t_{\mathit{lam}}\ \Phi\ [\mathit{app}'\ s\ t] &= t_{\mathit{app}}\ \Phi\ s\ t\\
\mathit{rec}\ t_{\mathit{var}}\ t_{\mathit{app}}\ t_{\mathit{lam}}\ \Phi\ [\mathit{lam}'\ s] &= t_{\mathit{lam}}\ \Phi\ s
\end{aligned}
$$

To outline the proof idea, note that $[\Phi \vdash \mathsf{tm}]$ represents the domain-level terms of type tm in context Φ up to $\beta\eta$-equality. To define a function $\forall\phi\colon \mathsf{Obj}.\, \forall x\colon [\phi \vdash \mathsf{tm}].\, A\ x$ in $\widehat{\mathbb{D}}$, it suffices to define a term $A\ t$ for each concrete object Φ and each domain-level term $t\colon [\phi \vdash \mathsf{tm}]$, since the naturality constraint for boxed types are vacuous.

As a simple example for the recursion combinator, we can define the function $\mathsf{is\text{-}lam}$ discussed above by $\mathsf{rec}\ (\lambda\phi, x.\, \mathsf{false})\ (\lambda\phi, M, r_M, N, r_N.\, \mathsf{false})\ (\lambda\phi, M, r_M.\, \mathsf{true})$.

4 Simple Contextual Types

We have outlined informally how the internal dependent type theory of $\widehat{\mathbb{D}}$ can model contextual types. In this section, we make this precise by giving the interpretation of a properly defined contextual type theory over a simply-typed domain language using this approach. Concentrating on a simply-typed domain will allow us to focus on the essential aspects of the semantic interpretation. The generalisation to LF in Sec. 7 will be conceptually straightforward, although more technical.

We first define a contextual type theory over a simply-typed domain concretely following [16] and [17]. It has the following types, where A is a domain-level type, U is a contextual type and τ is a computation type.

$$
\begin{array}{lcl@{\qquad\qquad}lcl}
A & ::= & \mathsf{tm} \mid A \to A & \Phi & ::= & \cdot \mid X \mid \Phi, x\colon A\\
U & ::= & (\Phi \vdash A) \mid \mathsf{ctx} & \Delta & ::= & \cdot \mid \Delta, X\colon U\\
\tau & ::= & [U] \mid \tau \to \tau \mid \forall X\colon U.\, \tau & \Gamma & ::= & \cdot \mid \Gamma, x\colon \tau
\end{array}
$$

For simplicity, we omit a contextual object $\Phi \vdash_v A$ for variables.

We define terms and typing rules in Fig. 1. We do not consider algorithmic aspects of the contextual type system (cf. [17]) and formulate equations simply as equations-in-context.

4.1 Interpretation

This simple contextual type theory is not hard to interpret with a cartesian closed Yoneda universe. Assume the dependent type theory from the previous section, including the basic type tm with the constants $\mathsf{app}\colon \mathsf{El}\,\mathsf{tm} \to \mathsf{El}\,\mathsf{tm} \to \mathsf{El}\,\mathsf{tm}$ and $\mathsf{lam}\colon (\mathsf{El}\,\mathsf{tm} \to \mathsf{El}\,\mathsf{tm}) \to \mathsf{El}\,\mathsf{tm}$.

The various types of the simple contextual type theory are translated to the dependent type theory for $\widehat{\mathbb{D}}$. A domain-level type A becomes a term $[\![A]\!]\colon \mathsf{Obj}$, a contextual type U becomes a type $[\![U]\!]$, and a computation type τ becomes a type $[\![\tau]\!]$,

$$
\begin{array}{rcl@{\qquad}rcl@{\qquad}rcl}
[\![\mathsf{tm}]\!] & = & \mathsf{tm} & [\![\mathsf{ctx}]\!] & = & \mathsf{Obj} & [\![[U]]\!] & = & [\![U]\!]\\
[\![A \to B]\!] & = & \mathsf{arrow}\ [\![A]\!]\ [\![B]\!] & [\![\Phi \vdash A]\!] & = & [\forall\gamma\colon \mathsf{El}\,[\![\Phi]\!].\, \mathsf{El}\,[\![A]\!]] & [\![\tau_1 \to \tau_2]\!] & = & [\![\tau_1]\!] \to [\![\tau_2]\!]\\
 & & & & & & [\![\forall X\colon U.\, \tau]\!] & = & \forall X\colon [\![U]\!].\, [\![\tau]\!]
\end{array}
$$

$\boxed{\Delta;\,\Phi \vdash t\colon A}$: simply-typed domain terms

$$\frac{\Phi(x) = A}{\Delta;\,\Phi \vdash x\colon A} \qquad \frac{\Delta;\,\Phi,\,x{:}\,A \vdash t\colon B}{\Delta;\,\Phi \vdash \lambda x{:}\,A.\,t\colon A \to B} \qquad \frac{\Delta;\,\Phi \vdash s\colon A \to B \quad \Delta;\,\Phi \vdash t\colon A}{\Delta;\,\Phi \vdash s\ t\colon B}$$

$$\frac{\Delta \vdash C\colon (\Phi \vdash A) \quad \Delta;\,\Psi \vdash \sigma\colon \Phi}{\Delta;\,\Psi \vdash \lfloor C \rfloor_\sigma\colon A} \qquad \frac{}{\Delta;\,\Phi \vdash \mathsf{app}\colon \mathsf{tm} \to \mathsf{tm} \to \mathsf{tm}} \qquad \frac{}{\Delta;\,\Phi \vdash \mathsf{lam}\colon (\mathsf{tm} \to \mathsf{tm}) \to \mathsf{tm}}$$

$\boxed{\Delta;\,\Phi \vdash \sigma\colon \Psi}$: domain-level substitutions

$$\frac{\Delta \vdash \Phi\colon \mathsf{ctx}}{\Delta;\,\Phi \vdash \cdot\colon \cdot} \qquad \frac{\Delta \vdash \Phi,\,\overline{x{:}\,A}\colon \mathsf{ctx}}{\Delta;\,\Phi,\,\overline{x{:}\,A} \vdash id_\Phi\colon \Phi} \qquad \frac{\Delta;\,\Phi \vdash \sigma\colon \Psi \quad \Delta;\,\Phi \vdash t\colon \mathsf{tm}}{\Delta;\,\Phi \vdash (\sigma, t)\colon (\Psi,\,x{:}\,\mathsf{tm})}$$

$\boxed{\Delta \vdash C\colon U}$: contextual objects

$$\frac{\Delta(X) = U}{\Delta \vdash X\colon U} \qquad \frac{\Delta;\,\Phi \vdash t\colon A}{\Delta \vdash (\Phi \vdash t)\colon (\Phi \vdash A)} \qquad \frac{}{\Delta \vdash \cdot\colon \mathsf{ctx}} \qquad \frac{\Delta \vdash \Phi\colon \mathsf{ctx}}{\Delta \vdash \Phi,\,x{:}\,\mathsf{tm}\colon \mathsf{ctx}}$$

$\boxed{\Delta \vdash \theta\colon \Delta'}$: contextual substitutions

$$\frac{}{\Delta \vdash \cdot\colon \cdot} \qquad \frac{\Delta \vdash \theta\colon \Delta' \quad \Delta \vdash C\colon U}{\Delta \vdash (\theta, C)\colon (\Delta',\,X{:}\,U)}$$

$\boxed{\Delta;\,\Gamma \vdash e\colon \tau}$: computations

$$\frac{\Delta \vdash C\colon U}{\Delta;\,\Gamma \vdash [C]\colon [U]} \qquad \frac{\Delta;\,\Gamma,\,y{:}\,\tau_1 \vdash e\colon \tau_2}{\Delta;\,\Gamma \vdash \mathsf{fn}\,y{:}\,\tau_1 \Rightarrow e\colon \tau_1 \to \tau_2} \qquad \frac{\Delta;\,\Gamma \vdash e_1\colon \tau_2 \to \tau_1 \quad \Delta;\,\Gamma \vdash e_2\colon \tau_2}{\Delta;\,\Gamma \vdash e_1\ e_2\colon \tau_1}$$

$$\frac{\Delta,\,X{:}\,U;\,\Gamma \vdash e\colon \tau}{\Delta;\,\Gamma \vdash \Lambda X{:}\,U \Rightarrow e\colon \forall X{:}\,U.\,\tau} \qquad \frac{\Delta;\,\Gamma \vdash e\colon \forall X{:}\,U.\,\tau \quad \Delta \vdash C\colon U}{\Delta;\,\Gamma \vdash e\ C\colon \tau[C/X]}$$

$$\frac{\Delta;\,\Gamma \vdash e_1\colon [U] \quad \Delta,\,,\,X{:}\,U;\,\Gamma \vdash e_2\colon \tau_2}{\Delta;\,\Gamma \vdash \mathsf{let}\ X = e_1\ \mathsf{in}\ e_2}$$

$$\frac{\begin{array}{r} \Delta,\,\phi{:}\,\mathsf{ctx},\,X{:}\,[\phi \vdash \mathsf{tm}];\,\cdot \vdash e_1 : \tau \\ \Delta,\,\phi{:}\,\mathsf{ctx},\,X{:}\,[\phi, x{:}\,\mathsf{tm} \vdash \mathsf{tm}];\,r{:}\,\tau \vdash e_2 : \tau \\ \Delta \vdash C\colon \Phi \vdash \mathsf{tm} \quad \Delta,\,\phi{:}\,\mathsf{ctx},\,X_1, X_2{:}\,[\phi \vdash \mathsf{tm}];\,r_1, r_2{:}\,\tau \vdash e_3 : \tau \end{array}}{\Delta;\,\Gamma \vdash \mathsf{rec\text{-}case}\ C\ \mathsf{of\ var} \Rightarrow e_1 \mid \mathsf{lam} \Rightarrow e_2 \mid \mathsf{app} \Rightarrow e_3 : \tau}$$

$\boxed{\Delta;\,\Gamma \vdash e_1 = e_2\colon \tau}$: equations

$$\frac{\Delta;\,\Phi,\,x{:}\,A \vdash t\colon B \quad \Delta;\,\Phi \vdash s\colon A}{\Delta;\,\Phi \vdash (\lambda x{:}\,A.\,t)\ s = t[s/x]\colon B} \qquad \frac{\Delta;\,\Phi \vdash t\colon A \quad \Delta;\,\Psi \vdash \sigma\colon \Phi}{\Delta;\,\Psi \vdash \lfloor \Phi.t \rfloor_\sigma = t[\sigma/\Phi]\colon A}$$

$$\frac{\Delta;\,\Gamma,\,X{:}\,U \vdash e\colon \tau \quad \Delta \vdash C\colon U}{\Delta;\,\Gamma \vdash (\Lambda X{:}\,U.\,e)\ C = e[C/X]\colon \tau} \qquad \frac{\Delta;\,\Gamma,\,y{:}\,\tau_1 \vdash e\colon \tau_2 \quad \Delta;\,\Gamma \vdash e_1\colon \tau_1}{\Delta;\,\Gamma \vdash (\mathsf{fn}\,y{:}\,\tau_1 \Rightarrow e)\ e_1 = e[e_1/y]\colon \tau_2}$$

$$\frac{\Delta;\,\Phi \vdash x\colon \mathsf{tm} \quad \ldots}{\Delta;\,\Gamma \vdash (\mathsf{rec\text{-}case}\ [\Phi \vdash x]\ \mathsf{of\ var} \Rightarrow e_1 \mid \mathsf{lam} \Rightarrow e_2 \mid \mathsf{app} \Rightarrow e_3) = e_1[[\Phi \vdash x]/X]\colon \tau}$$

$$\frac{\Delta;\,\Phi \vdash t\colon \mathsf{tm} \to \mathsf{tm} \quad \ldots}{\Delta;\,\Gamma \vdash (\mathsf{rec\text{-}case}\ [\Phi \vdash \mathsf{lam}\ t]\ \mathsf{of}\ \ldots) = e_2[[\Phi \vdash t]/X, \mathsf{rec\text{-}case}\ [(\Phi, x{:}\,\mathsf{tm}) \vdash t\ x] \ldots /r]\colon \tau}$$

$$\frac{\Delta;\,\Phi \vdash t_1 : \mathsf{tm} \quad \Delta;\,\Phi \vdash t_2 : \mathsf{tm} \quad \ldots}{\begin{array}{l} \Delta;\,\Gamma \vdash\ (\mathsf{rec\text{-}case}\ [\Phi \vdash \mathsf{app}\ t_1\ t_2]\ \mathsf{of}\ \ldots) \\ \qquad = e_3[[\Phi \vdash t_1]/X_1, [\Phi \vdash t_2]/X_2, \mathsf{rec\text{-}case}\ [\Phi \vdash t_1] \ldots /r_1, \mathsf{rec\text{-}case}\ [\Phi \vdash t_2] \ldots /r_2]\colon \tau \end{array}}$$

Figure 1 Typing Rules of Simple Contextual Types

STL terms:

$$
\begin{aligned}
[\![\Delta;\, \Phi \vdash x\colon A]\!] &= \pi_{\Phi,x}(\gamma)\\
[\![\lambda x\colon A.\, t]\!] &= \mathsf{arrow\text{-}i}\ (\lambda x\colon [\![A]\!].\, [\![t]\!][\mathsf{pair}\ \gamma\ x/\gamma])\\
[\![s\ t]\!] &= \mathsf{arrow\text{-}e}\ [\![s]\!]\ [\![t]\!]\\
[\![\lfloor C \rfloor_\sigma]\!] &= \mathsf{let}\ [X] = [\![C]\!]\ \mathsf{in}\ X\ [\![\sigma]\!]\\
[\![\mathsf{app}]\!] &= \mathsf{arrow\text{-}i}(\lambda x.\, \mathsf{arrow\text{-}i}\ (\lambda y.\, \mathsf{app}\ x\ y))\\
[\![\mathsf{lam}]\!] &= \mathsf{arrow\text{-}i}(\lambda f.\, \mathsf{lam}\ (\lambda x.\, \mathsf{arrow\text{-}e}\ f\ x))
\end{aligned}
$$

Contextual objects:

$$
\begin{aligned}
[\![\Phi \vdash t]\!] &= [\lambda\gamma\colon \mathsf{El}\, [\![\Phi]\!].\, [\![t]\!]]\\
[\![X]\!] &= X\\
[\![\cdot]\!] &= \mathsf{unit}\\
[\![\Phi,\, x\colon \mathsf{tm}]\!] &= \mathsf{pair}\ [\![\Phi]\!]\ \mathsf{tm}
\end{aligned}
$$

Contextual substitutions:

$$
\begin{aligned}
[\![\cdot]\!] &= \cdot\\
[\![(\theta, C)]\!] &= ([\![\theta]\!], [\![C]\!])
\end{aligned}
$$

STL substitutions:

$$
\begin{aligned}
[\![\cdot]\!] &= \mathsf{terminal}\\
[\![(\sigma, t)]\!] &= \mathsf{pair}\ [\![\sigma]\!]\ [\![t]\!]\\
[\![id_\Phi]\!] &= \pi_{\Psi,\Phi}(\gamma)\\
\pi_{\Phi,\Phi}(t) &= t\\
\pi_{(\Psi,\, x\colon A),\Phi}(t) &= \pi_{\Psi,\Phi}(\mathsf{fst}(t))\\
\pi_{(\Phi,\, x\colon A),x}(t) &= \mathsf{snd}(t)\\
\pi_{(\Phi,\, y\colon A),x}(t) &= \pi_{\Phi,x}(\mathsf{fst}(t))
\end{aligned}
$$

Expressions:

$$
\begin{aligned}
[\![[C]]\!] &= [\![C]\!]\\
[\![e_1\ e_2]\!] &= [\![e_1]\!]\ [\![e_2]\!]\\
[\![e\ C]\!] &= [\![e]\!]\ [\![C]\!]\\
[\![\mathsf{fn}\, y\colon \tau \Rightarrow e]\!] &= \lambda y\colon [\![\tau]\!].\, [\![e]\!]\\
[\![\Lambda X\colon U \Rightarrow e]\!] &= \lambda X\colon [\![U]\!].\, [\![e]\!]\\
[\![\mathsf{let}\ X = e_1\ \mathsf{in}\ e_2]\!] &= \mathsf{let}\ X = [\![e_1]\!]\ \mathsf{in}\ [\![e_2]\!]
\end{aligned}
$$

Figure 2 Interpretation of Simple Contextual Types

We extend these definitions to the contexts Δ and Γ in the canonical way via $[\![\cdot]\!] = \cdot$ and $[\![\Delta,\, X\colon U]\!] = [\![\Delta]\!],\, X\colon [\![U]\!]$ and $[\![\Gamma,\, x\colon \tau]\!] = [\![\Gamma]\!],\, x\colon [\![\tau]\!]$.

The domain-level contexts Φ are treated differently, since they can appear both as contexts and as terms. We define $[\![\Phi]\!]$ to capture the role of Φ as terms, and use $\gamma\colon \mathsf{El}\, [\![\Phi]\!]$ when it is used in the role of a context. We fix a freshly chosen variable γ for this purpose.

Terms and substitutions are translated by induction on the derivation. To simplify the notation, let us denote a derivation just by its conclusion, or even just the principal part of the conclusion, when this is clear from the context. For example, we write $[\![\Delta \vdash \Phi\colon \mathsf{ctx}]\!]$ or just $[\![\Phi]\!]$ for the interpretation of a derivation of $\Delta \vdash \Phi\colon \mathsf{ctx}$. Fig 2 gives the interpretation.

▶ **Lemma 5.** *The interpretation maintains the following invariants:*

- *If* $\Delta;\, \Phi \vdash t\colon A$ *then* $[\![\Delta]\!],\, \gamma\colon \mathit{El}\, [\![\Phi]\!] \vdash [\![t]\!]\colon \mathit{El}\, [\![A]\!]$.
- *If* $\Delta;\, \Phi \vdash \sigma\colon \Psi$ *then* $[\![\Delta]\!],\, \gamma\colon \mathit{El}\, [\![\Phi]\!] \vdash [\![\sigma]\!]\colon \mathit{El}\, [\![\Psi]\!]$.
- *If* $\Delta \vdash C\colon U$ *then* $[\![\Delta]\!] \vdash [\![C]\!]\colon [\![U]\!]$.
- *If* $\Delta \vdash \theta\colon \Delta'$ *then* $[\![\Delta]\!] \vdash [\![\theta]\!]\colon [\![\Delta']\!]$.
- *If* $\Delta;\, \Gamma \vdash e\colon \tau$ *then* $[\![\Delta]\!],\, [\![\Gamma]\!] \vdash [\![e]\!]\colon [\![\tau]\!]$.

The proof goes by induction on derivations. In the case for the third item, we can use a box introduction, since all types in Δ are equal to a boxed typed.

▶ **Proposition 6** (Soundness). *The following are true.*

- *If* $\Delta;\, \Phi \vdash s = t\colon A$ *then* $[\![\Delta]\!],\, \gamma\colon \mathit{El}\, [\![\Phi]\!] \vdash [\![s]\!] = [\![t]\!]\colon \mathit{El}\, [\![A]\!]$.
- *If* $\Delta;\, \Gamma \vdash e_1 = e_2\colon \tau$ *then* $[\![\Delta]\!],\, [\![\Gamma]\!] \vdash [\![e_1]\!] = [\![e_2]\!]\colon [\![\tau]\!]$.

5 Contextual LF

We have spelled out the interpretation of simple contextual types to explain the essence of our approach in a simple way. Realistic systems with contextual types, such as Beluga, use dependently-typed domain languages like LF.

Dependent domain languages are useful to represent object-level languages more precisely. In LF, our running example of the untyped lambda-calculus can be refined into an encoding

$\boxed{\Delta;\, \Phi \vdash A \text{ type}}$: well-formed LF type

$$\frac{\Delta \vdash \Phi \colon \mathsf{ctx}}{\Delta;\, \Phi \vdash \mathsf{ty} \text{ type}} \qquad \frac{\Delta;\, \Phi \vdash t \colon \mathsf{ty}}{\Delta;\, \Phi \vdash (\mathsf{tm}\ t) \text{ type}} \qquad \frac{\Delta;\, \Phi,\, x \colon A \vdash B \text{ type}}{\Delta;\, \Phi \vdash \Pi x \colon A.\, B \text{ type}}$$

$\boxed{\Delta;\, \Phi \vdash t \colon A}$: for LF terms

$$\frac{\Delta \vdash \Phi \colon \mathsf{ctx} \qquad \Delta;\, \Phi \vdash A \text{ type}}{\Delta;\, \Phi,\, x \colon A \vdash x \colon A} \qquad \frac{\Delta \vdash \Phi \colon t \colon B \qquad \Delta;\, \Phi \vdash A \text{ type}}{\Delta;\, \Phi,\, x \colon A \vdash t \colon B} \qquad \frac{\Delta;\, \Phi,\, x \colon A \vdash t \colon B}{\Delta;\, \Phi \vdash \lambda x \colon A.\, t \colon \Pi x \colon A.\, B}$$

$$\frac{\Delta;\, \Phi \vdash s \colon \Pi x \colon A.\, B \qquad \Delta;\, \Phi \vdash t \colon A}{\Delta;\, \Phi \vdash s\ t \colon B[t/x]} \qquad \frac{\Delta \vdash C \colon (\Phi \vdash A) \qquad \Delta;\, \Psi \vdash \sigma \colon \Phi}{\Delta;\, \Psi \vdash \lfloor C \rfloor_\sigma \colon A[\sigma/\Phi]} \qquad \frac{\Delta \vdash \Phi \colon \mathsf{ctx}}{\Delta;\, \Phi \vdash c \colon A}$$

where $c \colon A$ is one of the constants o, arr, app and lam, as described in the text.

$\boxed{\Delta;\, \Phi \vdash \sigma \colon \Psi}$: domain-level substitutions

$$\frac{\Delta \vdash \Phi \colon \mathsf{ctx}}{\Delta;\, \Phi \vdash \cdot \colon \cdot} \qquad \frac{\Delta \vdash \Phi,\, \overline{x \colon A} \colon \mathsf{ctx}}{\Delta;\, \Phi,\, \overline{x \colon A} \vdash id_\Phi \colon \Phi} \qquad \frac{\Delta;\, \Phi \vdash \sigma \colon \Psi \qquad \Delta;\, \Psi \vdash A \text{ type} \qquad \Delta;\, \Phi \vdash t \colon A[\sigma/\Psi]}{\Delta;\, \Phi \vdash (\sigma, t) \colon (\Psi,\, x \colon A)}$$

$\boxed{\Delta \vdash C \colon U}$: contextual objects

$$\frac{\vdash \Delta \text{ context} \qquad \Delta(X) = U}{\Delta \vdash X \colon U} \qquad \frac{\Delta;\, \Phi \vdash t \colon A}{\Delta \vdash (\Phi \vdash t) \colon (\Phi \vdash A)} \qquad \frac{\vdash \Delta \text{ context}}{\Delta \vdash \cdot \colon \mathsf{ctx}} \qquad \frac{\Delta \vdash \Phi \colon \mathsf{ctx} \qquad \Delta;\, \Phi \colon A \text{ type}}{\Delta \vdash \Phi,\, x \colon A \colon \mathsf{ctx}}$$

We omit the rules for $\Delta \vdash \theta \colon \Delta'$ and $\Delta;\, \Gamma \vdash e \colon \tau$, which are as in Fig. 1, as well as the straightforward well-formedness rules for $\vdash \Delta$ context, $\Delta \vdash \tau$ comp-type and $\Delta \vdash U$ ctx-type.

Figure 3 Typing Rules of Contextual LF

of the simply-typed lambda calculus that allows only well-typed terms. It is given by LF type constants ty and $a \colon \mathsf{ty} \vdash \mathsf{tm}\ a$ for object-level types and terms. Concrete object-level types are represented by a constant $\mathsf{o} \colon \mathsf{ty}$ for a base type and a constant $\mathsf{arr} \colon \mathsf{ty} \to \mathsf{ty} \to \mathsf{ty}$ for function types. Object-level terms are encoded using the constants $\mathsf{app} \colon \Pi a, b \colon \mathsf{ty}.\ \mathsf{tm}\ (\mathsf{arr}\ a\ b) \to \mathsf{tm}\ a \to \mathsf{tm}\ b$ and $\mathsf{lam} \colon \Pi a, b \colon \mathsf{ty}.\ (\mathsf{tm}\ a \to \mathsf{tm}\ b) \to \mathsf{tm}\ (\mathsf{arr}\ a\ b)$. The type dependencies are chosen so that one can only represent well-typed terms.

In the rest of this paper, we extend our semantical analysis to cover contextual types over LF. We consider the type system Contextual LF, which is obtained from the simple contextual type system by replacing the simply-typed domain language with LF. The typing rules of Contextual LF in Fig. 3 are a direct generalisation of the rules in Fig. 1.

6 Categories with Attributes

To define a semantics for Contextual LF, we need to model dependent types. There are a number of essentially-equivalent notions of models of dependent type theory, such as Categories with Families [4], Categories with Attributes [2], Comprehension Categories [11], etc. For our purposes, *Category with Attributes* (CwA) in the formulation of [8] are convenient.

▶ **Definition 7.** *A* category with attributes *given by the following data.*

- *A category* $\mathbb{C}$*, called* category of contexts*, with a terminal object.*
- *A functor* $\mathrm{Ty} \colon \mathbb{C}^{\mathrm{op}} \to \mathrm{Set}$.
- *For each type* $X \in \mathrm{Ty}(\Phi)$*, an object* $\Phi \ltimes X$ *and a morphism* $\pi_X \colon \Phi \ltimes X \to \Phi$ *in* $\mathbb{C}$*. We call* π_X *a* projection morphism*.*

- *For all* $\sigma\colon \Psi \to \Phi$ *in* $\mathbb{C}$ *and* $X \in \mathrm{Ty}(\Phi)$*, a morphism* $q(\sigma, X)$ *making the following diagram in* $\mathbb{C}$ *a pullback.*

$$\begin{tikzcd} \Psi \ltimes \mathrm{Ty}(\sigma)(X) \arrow[r, "q(\sigma,X)"] \arrow[d, "\pi_{\mathrm{Ty}(\sigma)(X)}"'] \arrow[dr, phantom, very near start, "\lrcorner"] & \Phi \ltimes X \arrow[d, "\pi_X"] \\ \Psi \arrow[r, "\sigma"'] & \Phi \end{tikzcd}$$

The category $\mathbb{C}$ represents contexts and substitutions. Its objects represent contexts. The terminal object is the empty context. A morphism $\sigma\colon \Phi \to \Psi$ in $\mathbb{C}$ represents a substitution that defines a term in context Φ for each variable in Ψ.

The functor Ty represents dependent types and type substitution. For an object Φ of $\mathbb{C}$, the set $\mathrm{Ty}(\Phi)$ is the set of types in context Φ. For any morphism $\sigma\colon \Psi \to \Phi$, the function $\mathrm{Ty}(\sigma)\colon \mathrm{Ty}(\Phi) \to \mathrm{Ty}(\Psi)$ explains how to apply the substitution σ to the types in context Φ.

The object $\Phi \ltimes X$ represents the context "$\Phi,\, x\colon X$". The projection π_X is the weakening substitution. For example, if we have a type $Y \in \mathrm{Ty}(\Phi)$, then $\mathrm{Ty}(\pi_X)(Y) \in \mathrm{Ty}(\Phi \ltimes X)$ should be understood as the same type Y after weakening with a variable of type X.

The morphism $q(\sigma, X)$ lifts the substitution σ to an extended context. It corresponds to a substitution of the form (σ, x) in the type theories of Secs. 4 and 5.

The definition of a CwA does not mention terms, since these are considered a derived concept. A term of type $X \in \mathrm{Ty}(\Phi)$ in context Φ can be identified with a *section of* π_X, which is a morphism $\sigma\colon \Phi \to \Phi \ltimes X$ with the property $\pi_X \circ \sigma = id$.

Having defined CwAs, we can say a few words about the type theory for $\widehat{\mathbb{D}}$ that we have described in Sec. 3. One can define a CwA with $\widehat{\mathbb{D}}$ as the category of contexts. Let us spell out the projections of a few types. The type $\mathsf{Obj} \in \mathrm{Ty}(1)$ has the projection $\pi_{\mathsf{Obj}}\colon O \to 1$, where $O(\Gamma)$ is the set of objects of $\mathbb{D}$. The type $\mathsf{El} \in \mathrm{Ty}(O)$ has the projection $\pi_{\mathsf{El}}\colon M \to O$, where $M(\Gamma)$ is the set of pairs (Δ, f) with $\Delta \in O(\Gamma)$ and $f \in \mathbb{D}(\Gamma, \Delta)$, and where $\pi_{\mathsf{El}}(\Delta, f) = \Delta$. Note that any object A of $\mathbb{D}$ defines a map $A\colon 1 \to O$, which corresponds to a term of type Obj. Substituting the type El with this map, gives us a type whose projection is (up to isomorphism) the pullback of π_{El} along A. This is easily seen to be just $\mathrm{y}A \to 1$, which justifies our view of El as a syntax for the Yoneda embedding.

7 Presheaves on a Small Category with Attributes

With the notion of CwA, we can now come to modelling Contextual LF. We still use a presheaf category $\widehat{\mathbb{D}}$ as before, but we now use a CwA $\mathbb{D}$ instead of a cartesian closed category. Thus, assume from now on that $\mathbb{D}$ is a CwA, e.g. the term model of LF. We now again consider the Yoneda embedding of the CwA $\mathbb{D}$ into $\widehat{\mathbb{D}}$ in type-theoretic terms, as in Sec. 3.

7.1 Yoneda CwA

We write Ctx for the type of all the objects of $\mathbb{D}$. In the canonical model, these would be LF contexts. The type $\mathsf{Ty}\ c$ is the set of types in context c, as defined as part of the CwA.

$$\vdash \mathsf{Ctx}\ \mathrm{type} \qquad\qquad c\colon \mathsf{Ctx} \vdash \mathsf{Ty}\ c\ \mathrm{type}$$

Both Ctx and $\mathsf{Ty}\ c$ have trivial presheaf structure, i.e. $[\mathsf{Ctx}] = \mathsf{Ctx}$ and $[\mathsf{Ty}\ c] = \mathsf{Ty}\ c$.

Because of the dependency structure of dependently typed contexts, we cannot simply model them anymore using products as we did in Sec 4. Instead, contexts are represented using the constants nil and cons:

$$\vdash \mathsf{nil}\colon \mathsf{Ctx} \qquad\qquad \vdash \mathsf{cons}\colon \forall c\colon \mathsf{Ctx}.\, \forall a\colon (\mathsf{Ty}\ c).\, \mathsf{Ctx}$$

The constant nil denotes the terminal object and $\mathsf{cons}\ c\ a$ stands for $c \ltimes a$ in the CwA $\mathbb{D}$.

As before, we consider the contexts as codes of a universe El.

$$c\colon \mathsf{Ctx} \vdash \mathsf{El}\, c \text{ type}$$

The type $\mathsf{El}\, c$ has the same definition as above and is essentially just the Yoneda embedding. It thus represents all global elements of the context c in $\mathbb{D}$, i.e. all substitutions with codomain c. One should therefore think of a term of type $\mathsf{El}\, c$ as a tuple of domain-level terms, one for each declaration in the context represented by c. A function $\mathsf{El}\, c \to \mathsf{El}\, d$ corresponds to a context morphism $c \to d$ in $\mathbb{D}$, as the Yoneda embedding is full and faithful.

The CwA-structure of $\mathbb{D}$ now induces the following terms in $\widehat{\mathbb{D}}$.

$$\begin{aligned}
&\vdash \mathsf{terminal}\colon \mathsf{El}\,\mathsf{nil}\\
c\colon \mathsf{Ctx},\ a\colon (\mathsf{Ty}\ c) &\vdash p\colon \mathsf{El}\ (\mathsf{cons}\ c\ a) \to \mathsf{El}\, c\\
c, d\colon \mathsf{Ctx} &\vdash \mathsf{sub}\colon \forall a\colon (\mathsf{Ty}\ d).\, \forall f\colon (\mathsf{El}\, c \to \mathsf{El}\, d).\, \mathsf{Ty}\ c\\
c, d\colon \mathsf{Ctx} &\vdash q\colon \forall a\colon (\mathsf{Ty}\ d).\, \forall f\colon (\mathsf{El}\, c \to \mathsf{El}\, d).\, \mathsf{El}\, (\mathsf{cons}\ c\ (\mathsf{sub}\ a\ f)) \to \mathsf{El}\, (\mathsf{cons}\ d\ a)
\end{aligned}$$

The following lemmas correspond to the CwA-axioms.

▶ **Lemma 8.** *Substitution is functorial: We have* $\mathsf{sub}\ a\ (\lambda x.x) = a$ *and* $\mathsf{sub}\ a\ (g \circ f) = \mathsf{sub}\ (\mathsf{sub}\ a\ g)\ f$ *for all* $c\colon \mathsf{Ctx}$*,* $a\colon (\mathsf{Ty}\ c)$*,* $f\colon \mathsf{El}\, e \to \mathsf{El}\, d$ *and* $g\colon \mathsf{El}\, d \to \mathsf{El}\, c$*.*

▶ **Lemma 9.** *The type* $\mathsf{El}\,\mathsf{nil}$ *is terminal, which means that any* $x\colon \mathsf{El}\,\mathsf{nil}$ *satisfies* $x = \mathsf{terminal}$*.*

The pullback property from Def. 7 is stated internally as follows:

▶ **Lemma 10.** *Let* $c, d\colon \mathsf{Ctx}$*,* $a\colon (\mathsf{Ty}\ c)$ *and* $f\colon \mathsf{El}\, d \to \mathsf{El}\, c$*. Then we have* $p\ (q\ a\ f\ \gamma) = f\ (p\ \gamma)$ *for all* $\gamma\colon \mathsf{El}\, c$*. Moreover, for all* $x\colon \mathsf{El}\, (\mathsf{cons}\ d\ a)$ *and* $\gamma\colon \mathsf{El}\, c$*, there exists a unique* $y\colon \mathsf{El}\, (\mathsf{cons}\ c\ (\mathsf{sub}\ a\ f))$ *with* $p\ y = \gamma$ *and* $q\ a\ f\ y = x$*.*

For working with the CwA structure, it is useful to define a dependent type

$$c\colon \mathsf{Ctx},\ a\colon (\mathsf{Ty}\ c),\ \gamma\colon (\mathsf{El}\, c) \vdash \mathsf{ElTm}\ a\ \gamma \text{ type}$$

by $\mathsf{ElTm}\ a\ \gamma\ :=\ \Sigma v\colon \mathsf{El}\, (\mathsf{cons}\ c\ a).\, (p\ v) = \gamma$. This Σ-type consists of all pairs $\langle v, w\rangle$ where v has type $\mathsf{El}\, (\mathsf{cons}\ c\ a)$ and where w is a proof of $(p\ v) = \gamma$.

The type $\mathsf{ElTm}\ a\ \gamma$ thus consists of all values in $\mathsf{El}\, (\mathsf{cons}\ c\ a)$ whose first projection is γ. If one considers $\gamma\colon \mathsf{El}\, c$ as a tuple of domain-level terms (one term for each variable in the context represented by c), then $\mathsf{ElTm}\ a\ \gamma$ represents all the terms that can be appended to this tuple to make it into one of type $\mathsf{El}\, (\mathsf{cons}\ c\ a)$. Accordingly, we can define a pairing operation and a second projection

$$\begin{aligned}
c\colon \mathsf{Ctx},\ a\colon (\mathsf{Ty}\ c) &\vdash \mathsf{pair}\colon \forall \gamma\colon (\mathsf{El}\, c).\, \mathsf{ElTm}\ a\ \gamma \to \mathsf{El}\, (\mathsf{cons}\ c\ a)\\
c\colon \mathsf{Ctx},\ a\colon (\mathsf{Ty}\ c) &\vdash p'\colon \forall \gamma\colon \mathsf{El}\, (\mathsf{cons}\ c\ a).\, \mathsf{ElTm}\ a\ (p\ \gamma)
\end{aligned}$$

by $\mathsf{pair} := \lambda\gamma.\, \lambda\langle v, p\rangle.\, v$ and $p' := \lambda\gamma.\, \langle\gamma, \mathrm{refl}\rangle$. The first projection p was already defined.

The next lemma relates ElTm to substitution. Its proof uses Lemma 11.

▶ **Lemma 11.** *For* $c, d\colon \mathsf{Ctx}$*,* $a\colon (\mathsf{Ty}\ c)$*,* $f\colon \mathsf{El}\, d \to \mathsf{El}\, c$ *and* $\gamma\colon (\mathsf{El}\, d)$*, there is an isomorphism* $\mathsf{subElTm}\colon \mathsf{ElTm}\ a\ (f\ \gamma) \to \mathsf{ElTm}\ (\mathsf{sub}\ a\ f)\ \gamma$*. We write* $\mathsf{subElTm}^{-1}$ *for its inverse.*

Finally, we can define a type of domain-level terms by $\mathsf{Tm}\ c\ a := \forall\gamma\colon (\mathsf{El}\, c).\, \mathsf{ElTm}\ a\ \gamma$. This type represents domain-level terms just as $\mathsf{Ty}\ c$ represents domain-level types. It is not hard to show that $\mathsf{Tm}\ c\ a$ is isomorphic to the type of sections of $p\colon \mathsf{El}\, (\mathsf{cons}\ c\ a) \to \mathsf{El}\, c$, cf. Sec. 6. We prefer to use ElTm over Tm, since it allows us to move domain-level abstractions into meta-level abstractions, e.g. in Lemma 12 below.

So far, we have only exposed the CwA structure of $\mathbb{D}$ in $\widehat{\mathbb{D}}$. Dependent products in the domain language are lifted to $\widehat{\mathbb{D}}$ by the following lemma, which generalises Lemma 3.

▶ **Lemma 12.** *If the CwA $\mathbb{D}$ has dependent products, then the internal type theory of $\widehat{\mathbb{D}}$ has the following terms, in which Γ abbreviates $c\colon \mathit{Ctx}$, $a\colon (\mathit{Ty}\ c)$, $b\colon (\mathit{Ty}\ (\mathit{cons}\ c\ a))$, $\gamma\colon \mathit{El}\,c$.*

$$
\begin{aligned}
c\colon \mathit{Ctx} &\vdash \Pi\colon \forall a\colon (\mathit{Ty}\ c).\ \mathit{Ty}\ (\mathit{cons}\ c\ a) \to \mathit{Ty}\ c\\
\Gamma &\vdash \Pi\text{-}e\colon \mathit{ElTm}\ (\Pi\ a\ b)\ \gamma \to \forall x\colon (\mathit{ElTm}\ a\ \gamma).\ \mathit{ElTm}\ b\ (\mathit{pair}\ \gamma\ x)\\
\Gamma &\vdash \Pi\text{-}i\colon (\forall x\colon (\mathit{ElTm}\ a\ \gamma).\ \mathit{ElTm}\ b\ (\mathit{pair}\ \gamma\ x)) \to \mathit{ElTm}\ (\Pi\ a\ b)\ \gamma
\end{aligned}
$$

Moreover, Π-i and Π-e are mutually inverse.

The term $(\Pi\ a\ b)$ in the type theory for $\widehat{\mathbb{D}}$ represents the dependent product type in $\mathbb{D}$. It is well-behaved with respect to substitution.

Object-level term constants in the type theory modelled by $\mathbb{D}$, such as ty, tm, app and lam from above can be lifted using ElTm. We use the same name for the lifted constants.

$$
\begin{aligned}
&c\colon \mathsf{Ctx} \vdash \mathsf{ty}\colon \mathsf{Ty}\ c && \Gamma \vdash \mathsf{o}\colon \mathsf{ElTm}\ \mathsf{ty}\ \gamma\\
&c\colon \mathsf{Ctx} \vdash \mathsf{tm}\colon \mathsf{Ty}\ (\mathsf{cons}\ c\ \mathsf{ty}) && \Gamma \vdash \mathsf{arr}\colon \mathsf{ElTm}\ \mathsf{ty}\ \gamma \to \mathsf{ElTm}\ \mathsf{ty}\ \gamma \to \mathsf{ElTm}\ \mathsf{ty}\ \gamma
\end{aligned}
$$

$$
\begin{aligned}
\Delta &\vdash \mathsf{app}\colon \mathsf{ElTm}\ \mathsf{tm}\ (\mathsf{pair}\ \gamma\ (\mathsf{arr}\ a\ b)) \to \mathsf{ElTm}\ \mathsf{tm}\ (\mathsf{pair}\ \gamma\ a) \to \mathsf{ElTm}\ \mathsf{tm}\ (\mathsf{pair}\ \gamma\ b)\\
&\vdash \mathsf{lam}\colon (\mathsf{ElTm}\ \mathsf{tm}\ (\mathsf{pair}\ \gamma\ a) \to \mathsf{ElTm}\ \mathsf{tm}\ (\mathsf{pair}\ \gamma\ b)) \to \mathsf{ElTm}\ \mathsf{tm}\ (\mathsf{pair}\ \gamma\ (\mathsf{arr}\ a\ b))
\end{aligned}
$$

where Γ abbreviates $c\colon \mathsf{Ctx}$, $\gamma\colon (\mathsf{El}\ c)$ and Δ abbreviates Γ, $a, b\colon (\mathsf{ElTm}\ \mathsf{ty}\ \gamma)$. Notice how lam uses higher-order abstract syntax at the meta level. For this, it seems to be essential to use ElTm rather than a formulation with Tm.

8 Interpreting Contextual LF

Having outlined the structure of presheaves over a CwA, we now use this structure to model Contextual LF in $\widehat{\mathbb{D}}$. We assume that $\mathbb{D}$ is a model of LF, i.e. a CwA with dependent products, and that it models the constants for ty and tm from the preceding section. The term model is a canonical example for $\mathbb{D}$.

With type dependencies, we cannot define the interpretation of types and terms separately, but must define the whole interpretation by induction on the derivation. As before, we denote derivations simply by their conclusion or the principal part thereof. With this understanding, we can define the interpretation of contextual types ($\Delta \vdash U$ ctx-type) and computation types ($\Delta \vdash \tau$ comp-type) almost exactly as before.

$$
\begin{aligned}
[\![\mathsf{ctx}]\!] &= \mathtt{Ctx} & [\![[U]]\!] &= [\![U]\!]\\
[\![\Phi \vdash A]\!] &= [\forall \gamma\colon \mathsf{El}\ [\![\Phi]\!].\ \mathsf{ElTm}\ [\![A]\!]\ \gamma] & [\![\tau_1 \to \tau_2]\!] &= [\![\tau_1]\!] \to [\![\tau_2]\!]\\
& & [\![\forall X\colon U.\ \tau]\!] &= \forall X\colon [\![U]\!].\ [\![\tau]\!]
\end{aligned}
$$

This definition makes reference to the interpretation of contextual objects (via $\Delta \vdash \Phi\colon \mathsf{ctx}$) and to LF types (via $\Delta;\ \Phi \vdash A$ type). Before we outline the interpretation of such judgements, it is useful to formulate the typing invariants of the interpretation.

▶ **Lemma 13.** *The interpretation maintains the following invariants:*

- *If $\Delta;\ \Phi \vdash A$ type, then $[\![\Delta]\!] \vdash [\![A]\!]\colon \mathit{Ty}\ [\![\Phi]\!]$.*
- *If $\Delta;\ \Phi \vdash t\colon A$ then $[\![\Delta]\!]$, $\gamma\colon \mathit{El}\,[\![\Phi]\!] \vdash [\![t]\!]\colon \mathit{ElTm}\ [\![A]\!]\ \gamma$.*
- *If $\Delta;\ \Phi \vdash \sigma\colon \Psi$ then $[\![\Delta]\!]$, $\gamma\colon \mathit{El}\,[\![\Phi]\!] \vdash [\![\sigma]\!]\colon \mathit{El}\,[\![\Psi]\!]$.*
- *If $\Delta \vdash U$ ctx-type then $[\![\Delta]\!] \vdash [\![U]\!]$ type.*
- *If $\Delta \vdash C\colon U$ then $[\![\Delta]\!] \vdash [\![C]\!]\colon [\![U]\!]$.*
- *If $\Delta \vdash \theta\colon \Delta'$ then $[\![\Delta]\!] \vdash [\![\theta]\!]\colon [\![\Delta']\!]$.*

- *If* $\Delta \vdash \tau$ *comp-type then* $[\![\Delta]\!] \vdash [\![\tau]\!]$ *type.*
- *If* $\Delta; \Gamma \vdash e\colon \tau$ *then* $[\![\Delta]\!], [\![\Gamma]\!] \vdash [\![e]\!] : [\![\tau]\!]$.

With these invariants in mind, the interpretation is essentially straigthforward. For example, LF types and contextual objects are interpreted by:

$$\begin{aligned}
[\![\Delta;\, \Phi \vdash \mathsf{ty}\ \mathrm{type}]\!] &= \mathsf{ty} \\
[\![\Delta;\, \Phi \vdash \mathsf{tm}\ t\ \mathrm{type}]\!] &= \mathsf{sub}\ \mathsf{tm}\ (\lambda\gamma.\, [\![\Delta;\, \Phi \vdash t\colon \mathsf{ty}]\!]) \\
[\![\Delta;\, \Phi \vdash \Pi x\colon A.\, B\ \mathrm{type}]\!] &= \Pi\ [\![\Phi \vdash A\ \mathrm{type}]\!]\ [\![\Delta;\, \Phi,\, x\colon A \vdash B\ \mathrm{type}]\!] \\
[\![\Delta;\, \Phi,\, x\colon A \vdash x\colon A]\!] &= \mathsf{subElTm}(p'\ \gamma)\ \text{(variable rule)} \\
[\![\Delta;\, \Phi,\, x\colon A \vdash t\colon A]\!] &= \mathsf{subElTm}\ [\![\Delta;\, \Phi \vdash t\colon A]\!][p\ \gamma/\gamma]\ \text{(weakening rule)} \\
[\![\lambda x\colon A.\, t]\!] &= \Pi\text{-}\mathsf{i}\ (\lambda x\colon [\![A]\!].\, [\![t]\!][\mathsf{pair}\ \gamma\ x/\gamma]) \\
[\![s\ t]\!] &= \mathsf{subElTm}\ (\Pi\text{-}\mathsf{e}\ [\![s]\!]\ [\![t]\!]) \\
[\![\lfloor C \rfloor_\sigma]\!] &= \mathsf{let}\ [X] = [\![C]\!]\ \mathsf{in}\ [\lambda\gamma\colon [\![\Psi]\!].\, \mathsf{subElTm}\ (X\ [\![\sigma]\!])]
\end{aligned}$$

The definition of π from Sec. 4.1 is now built into the explicit weakening rule. The term $\mathsf{subElTm}$ is used for type substitution. In the interpretation of application, for example, $(\Pi\text{-}\mathsf{e}\ [\![s]\!]\ [\![t]\!])$ has type $\mathsf{ElTm}\ [\![B]\!]\ (\mathsf{pair}\ \gamma\ [\![s]\!])$. By using $\mathsf{subElTm}$, we get a term of type $\mathsf{ElTm}\ (\mathsf{sub}\ [\![B]\!]\ (\lambda\gamma.\, \mathsf{pair}\ \gamma\ [\![s]\!]))\ \gamma$, which is equal to the required $\mathsf{ElTm}\ [\![B[s/x]]\!]\ \gamma$ (making use of extensional equality in the meta-theory).

Due to the type dependencies, the definition of the interpretation needs some care. In the example of application, we have used that $(\mathsf{sub}\ [\![B]\!]\ (\lambda\gamma.\, (\mathsf{pair}\ \gamma\ [\![s]\!])))$ and $[\![B[s/x]]\!]$ are equal, but this information is not available during the definition of $[\![-]\!]$. The standard approach, due to Streicher [23], is to consider the definition of $[\![-]\!]$ a priori as a partial function that is undefined if types do not match. Then one proves weakening and substitution lemmas using the partial definition of $[\![-]\!]$. With these lemmas, one can then show by induction on derivations that $[\![-]\!]$ is in fact total after all. We elide such details in this paper.

9 Conclusion

We have given a rational reconstruction of contextual types in presheaf models of higher-order abstract syntax. This provides a semantical way of understanding the invariants of contextual types independently of the algorithmic details of type checking. At the same time, we identify contextual type systems as a syntax for presheaf models of HOAS with good algorithmic properties. Considering the Yoneda embedding as a type-theoretic universe provides a manageable way of constructing contextual types in the model, especially in the dependent case. While various forms of universes are being studied in the context of functor categories, e.g. [1, 12], we are not aware of previous uses of presheaves over CwAs or similar.

In future work, one may consider using the model as a way of compiling contextual types, by implementing the semantics. In another direction, it may be interesting to apply the syntax of contextual types to other presheaf categories. The model may also help to provide a semantic foundations for the further development of contextual type system like Cocon [18]. In particular, the model can be extended to justify computations that are embedded into contextual LF and can be used to justify non-trival functions returning contexts, which are currently not available in type systems.

References

1 Guillaume Allais, Robert Atkey, James Chapman, Conor McBride, and James McKinna. A type and scope safe universe of syntaxes with binding: Their semantics and proofs. *Proc. ACM Program. Lang.*, 2(ICFP):90:1–90:30, July 2018.

2 John Cartmell. Generalised algebraic theories and contextual categories. *Annals of Pure and Applied Logic*, 32:209 – 243, 1986.

3 Rowan Davies and Frank Pfenning. A modal analysis of staged computation. *Journal of the ACM*, 48(3):555–604, 2001.

4 Peter Dybjer. Internal type theory. In *Types for Proofs and Programs (TYPES'95)*, pages 120–134, 1995.

5 M. Fiore, G. D. Plotkin, and D. Turi. Abstract syntax and variable binding. In *Logic in Computer Science (LICS'99)*, pages 193–202. IEEE Press, 1999.

6 Murdoch Gabbay and Andrew Pitts. A new approach to abstract syntax involving binders. In *Logic in Computer Science (LICS'99)*, pages 214–224. IEEE Press, 1999.

7 Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, January 1993.

8 Martin Hofmann. *Syntax and Semantics of Dependent Types*, page 79–130. Publications of the Newton Institute. Cambridge University Press, 1997.

9 Martin Hofmann. Semantical analysis of higher-order abstract syntax. In *Logic in Computer Science (LICS'99)*, pages 204–213. IEEE Press, 1999.

10 Furio Honsell, Marino Miculan, and Ivan Scagnetto. An axiomatic approach to metareasoning on nominal algebras in HOAS. In *International Colloquium on Automata, Languages and Programming (ICALP'01)*, LNCS 2076, pages 963–978. Springer, 2001.

11 Bart Jacobs. Comprehension categories and the semantics of type dependency. *Theor. Comput. Sci.*, 107(2):169–207, 1993.

12 Daniel R. Licata, Ian Orton, Andrew M. Pitts, and Bas Spitters. Internal universes in models of homotopy type theory. In *Formal Structures for Computation and Deduction (FSCD'18)*, pages 22:1–22:17, 2018.

13 Dale Miller and Catuscia Palamidessi. Foundational aspects of syntax. *ACM Comput. Surv.*, 31(3es), 1999.

14 Aleksandar Nanevski, Frank Pfenning, and Brigitte Pientka. Contextual modal type theory. *ACM Transactions on Computational Logic*, 9(3):1–49, 2008.

15 Frank Pfenning and Conal Elliott. Higher-order abstract syntax. In *Symposium on Language Design and Implementation (PLDI'88)*, pages 199–208, June 1988.

16 Brigitte Pientka. A type-theoretic foundation for programming with higher-order abstract syntax and first-class substitutions. In *Principles of Programming Languages (POPL'08)*, pages 371–382. ACM Press, 2008.

17 Brigitte Pientka and Andreas Abel. Well-founded recursion over contextual objects. In *Typed Lambda Calculi and Applications (TLCA'15)*, pages 273–287, 2015.

18 Brigitte Pientka, Andreas Abel, Francisco Ferreira, David Thibodeau, and Rébecca Zucchini. Cocon: Computation in contextual type theory. *CoRR*, abs/1901.03378, 2019.

19 Brigitte Pientka, Andreas Abel, Francisco Ferreira, David Thibodeau, and Rebecca Zucchini. A type theory for defining logics and proofs. Jan 2019.

20 Brigitte Pientka and Andrew Cave. Inductive Beluga: Programming Proofs (System Description). In *Conference on Automated Deduction (CADE-25)*, LNCS 9195, pages 272–281. Springer, 2015.

21 Brigitte Pientka and Joshua Dunfield. Programming with proofs and explicit contexts. In *Principles and Practice of Declarative Programming (PPDP'08)*, pages 163–173, 2008.

22 Brigitte Pientka and Joshua Dunfield. Beluga: a framework for programming and reasoning with deductive systems (System Description). In *International Joint Conference on Automated Reasoning (IJCAR'10)*, LNAI 6173, pages 15–21. Springer, 2010.

23 Thomas Streicher. *Semantics of Type Theory.* Birkhäuser, 1991.

24 Andrea Vezzosi. Agda with a flat modality. `https://github.com/agda/agda/tree/flat`, 2018.

A Structure of the Presheaf Category

In the main text, we have explained the structure of $\widehat{\mathbb{D}}$ in terms of its internal dependent type theory. Here we explain in more detail how the internal dependent type theory relates to the direct definition of $\widehat{\mathbb{D}}$ as a functor category.

A.1 Basic Structure

To fix notation, we recall the basic structure of $\widehat{\mathbb{D}}$.

A.1.1 Finite Limits and Colimits

Finite limits and colimits exist and are constructed pointwise. In particular, finite products and coproducts are given by:

$$
\begin{aligned}
&1(\Gamma) = \{*\} \qquad\qquad && (X \times Y)(\Gamma) = X(\Gamma) \times Y(\Gamma) \\
&0(\Gamma) = \emptyset && (X + Y)(\Gamma) = X(\Gamma) + Y(\Gamma)
\end{aligned}
$$

The Yoneda embedding preserves finite products.

A.1.2 Exponentials

The category $\widehat{\mathbb{D}}$ has exponentials. The exponential $(X \Rightarrow Y)$ can be calculated using the Yoneda lemma. We recall that the Yoneda lemma states that $Z(\Gamma)$ is naturally isomorphic to $\widehat{\mathbb{D}}(\mathrm{y}(\Gamma), Z)$. With this, we have:

$$(X \Rightarrow Y)(\Gamma) \;\cong\; \widehat{\mathbb{D}}(\mathrm{y}(\Gamma), X \Rightarrow Y) \;\cong\; \widehat{\mathbb{D}}(\mathrm{y}(\Gamma) \times X, Y)$$

Since y preserves finite products, we have in particular $(\mathrm{y}(A) \Rightarrow \mathrm{y}(B))(\Gamma) \cong \widehat{\mathbb{D}}(\mathrm{y}(\Gamma) \times \mathrm{y}(A), \mathrm{y}(B)) \cong \widehat{\mathbb{D}}(\mathrm{y}(\Gamma \times A), \mathrm{y}(B)) \cong \mathbb{D}(\Gamma \times A, B)$. In the case where $A = B = \mathsf{tm}$, this shows that the exponential $\mathsf{tm} \Rightarrow \mathsf{tm}$ represents terms with an additional bound variable.

A.1.3 Subobject Classifier

The category $\widehat{\mathbb{D}}$ has a subobject classifier. This is a map $\top \colon 1 \to \Omega$ such that, for any monomorphism m there is a unique map ξ making the diagram below a pullback.

$$
\begin{tikzcd}
Y \arrow[r] \arrow[d, tail, "m"'] \arrow[dr, phantom, "\lrcorner", very near start] & 1 \arrow[d, "\top"] \\
X \arrow[r, dashed, "\chi"'] & \Omega
\end{tikzcd}
$$

When given $\chi \colon X \to \Omega$, one can obtain the a morphism $m_\chi \colon \{\chi\} \rightarrowtail X$ by pullback of $\top$ along χ, as in the diagram. We can choose this morphism such that, at any stage Γ, it is just subset inclusion. This means that $\{\chi\}(\Gamma) \subseteq X(\Gamma)$ holds for all Γ and that $(m_\chi)_\Gamma$ is the inclusion function.

A.1.4 Partial Maps

As any elementary topos, $\widehat{\mathbb{D}}$ has a partial map classifier. It may be constructed as follows. Given Y in $\widehat{\mathbb{D}}$, let $Y_\perp$ be the presheaf obtained from Y by disjointly adding an element $\perp$ to each $Y(\Gamma)$. We assume that this is done such that $Y(\Gamma) \subseteq Y_\perp(\Gamma)$ holds. We then define $X \rightharpoonup Y$ as $X \Rightarrow Y_\perp$. This object represents partial maps from X to Y.

A.2 Dependent Types in a Presheaf Category

Presheaf categories have enough structure to model dependent types. One can think of the interpretation of dependent types as a slight generalisation of the set-theoretic interpretation of dependent types, where a type $\Phi \vdash X$ type is interpreted by a base set B and a predicate ϕ. With this data, the dependent type amounts to the set $\{(p, x) \mid p \in \Phi, x \in B, \phi(p, x)\}$. This standard construction works in the internal set-theory of any topos and so in particular in presheaf categories.

To interpret a dependent type theory in $\widehat{\mathbb{D}}$, it suffices to show that $\widehat{\mathbb{D}}$ has the structure of a CwA and to use an existing interpretation of the syntax, e.g. [8], in this structure.

A Category with Attributes for $\widehat{\mathbb{D}}$ is defined as follows.

- The category of contexts Ctx is $\widehat{\mathbb{D}}$ itself.
- The functor $\mathrm{Ty}\colon \mathrm{Ctx}^{\mathrm{op}} \to \mathrm{Set}$ is defined as follows.
  The set $\mathrm{Ty}(\Phi)$ is defined to be the set of all pairs $X = (B, \chi)$ where B is an object of $\widehat{\mathbb{D}}$ and $\phi\colon \Phi \times B \to \Omega$ is a predicate on $\Phi \times B$. The intuition is that this defines a dependent type $p\colon \Phi \vdash X(\phi)$ by $X(\phi) = \{x \in B \mid \chi(\phi, x) = \top\}$. Thus, the object B is a non-dependent type that can uniformly encode the values of the dependent type for arbitrary dependency, and ϕ formalises which dependencies are possible.
  Type substitution is defined by pre-composition. If $\sigma\colon \Psi \to \Phi$ is a morphism in Ctx and $X = (B, \chi)$ is a type, then $\mathrm{Ty}(\sigma)(X)$ is defined to be $(B, \chi \circ (\sigma \times B))$. We write $\sigma^* X$ for $\mathrm{Ty}(\sigma)(X)$.
- For each type $X \in \mathrm{Ty}(\Phi)$, there must be an object $\Phi \ltimes X$ in Ctx and a projection morphism $\pi_X\colon \Phi \ltimes X \to \Phi$.
  In the case of $\widehat{\mathbb{D}}$, the predicate $\chi\colon \Phi \times B \to \Omega$ in the type $X = (B, \chi)$ induces a monomorphism $m_\chi\colon \{\chi\} \rightarrowtail \Phi \times B$. We define $(\Phi \ltimes X)$ to be its domain $\{\chi\}$. The notation symbolises the intuition that $(\Phi \ltimes X)$ consists of all pairs (p, x) with $p\colon \Phi$ and $x\colon X(p)$. The projection morphism $\pi_X\colon (\Phi \ltimes X) \to \Phi$ is defined by $\pi_X := \pi_1 \circ m_\chi$.
- Finally, for each $\sigma\colon \Psi \to \Phi$ and each $X \in \mathrm{Ty}(\Phi)$, there must be a morphism $q(\sigma, X)\colon \Psi \ltimes \mathrm{Ty}(\sigma)(X) \to \Phi \ltimes X$ making the following diagram a pullback.

$$\begin{tikzcd}
\Psi \ltimes \mathrm{Ty}(\sigma)(X) \arrow[r, "q(\sigma,X)"] \arrow[d, "\pi"'] \arrow[dr, phantom, "\lrcorner", very near start] & \Phi \ltimes X \arrow[d, "\pi"] \\
\Psi \arrow[r, "\sigma"'] & \Phi
\end{tikzcd}$$

  In the case of $\widehat{\mathbb{D}}$, notice that $\Phi \ltimes X$ and $\Psi \ltimes \mathrm{Ty}(\sigma)(X)$ are subobjects of $\Phi \times B$ and $\Psi \times B$ for the same B. The map $q(\sigma, X)$ is the unique morphism over $\sigma \times B$.

This interpretation of dependent types is equivalent to approaches using display maps or the codomain fibration [11]. In these approaches, types are simply given by their projection maps π_X. The advantage of the above representation of π_X as a pair (B, χ) is that it comes with a suitable choice of the objects $\sigma^* X$, which are otherwise only determined up to isomorphism by pullbacks. This is important for the interpretation of syntax, as in [23]. The construction amounts to a splitting of the codomain fibration for $\widehat{\mathbb{D}}$.

A.3 Type Formers

To outline the interpretation of the type theoretic presentation of the structure of $\widehat{\mathbb{D}}$ from the main text, we need to spell out concretely the interpretation of the type formers for products and identity types.

To this end, notice that to define a type $X = (B, \chi)$ in the CwA for $\widehat{\mathbb{D}}$, it is sufficient to define B and a subset $\{\chi\}(\Gamma) \subseteq (\Phi \times B)(\Gamma)$ for any object Γ of $\mathbb{D}$. The subsets define a monomorphism $\{\chi\} \rightarrowtail X \times B$ that uniquely determines $\chi \colon \Phi \times B \to \Omega$ by the universal property of the subobject classifier.

The dependent product $\Pi_X Y \in \mathrm{Ty}(\Phi)$ is defined for $X \in \mathrm{Ty}(\Phi)$ and $Y \in \mathrm{Ty}(\Phi \ltimes X)$. Suppose $X = (B_X, \chi_X)$ and $Y = (B_Y, \chi_Y)$. Then, one can define $\Pi_X Y$ to be the pair $(B_X \rightharpoonup B_Y, \chi)$, where the predicate χ is defined such that $\chi(p, f)$ is equivalent to $\chi_X(p, x) \iff \chi_Y((p, x), f(x))$. The implication from left to right states that the function f must map any x from the argument to a result of the correct dependency. The implication from right to left states that f is not defined for arguments that are not actually in the right dependent type.

The identity type $\equiv_X$ in $\mathrm{Ty}(\Phi \ltimes X \ltimes \pi_X^* X)$ is defined by the pair $(1, \chi)$, where χ is specified by:

$$((p, x), y) \in \{\chi\}(\Gamma) \iff x = y$$

A.4 Yoneda Universe

It remains to define the type for the Yoneda universe and the rest of the structure identified in Secs. 3 and 7. We do this for representative cases.

We define $\mathsf{Obj} \in \mathrm{Ty}(1)$ as $(U, \top)$, where O is the constant presheaf of objects of $\mathbb{D}$ and $\top$ is the predicate that is always true.

For the Yoneda Universe, we define the type $\mathsf{El} \in \mathrm{Ty}(1 \ltimes \mathsf{Obj})$ to be the pair (M, χ), where M is the presheaf with $M(\Gamma) = \{f \in \mathbb{D}(\Gamma, \Delta) \mid \Delta \text{ an object of } \mathbb{D}\}$ and pre-composition as action and where $\chi \colon (1 \ltimes \mathsf{ctx}) \times M \to \Omega$ is defined by

$$((*, \Delta), f) \in \{\chi\}(\Gamma) \iff f \in \mathbb{D}(\Gamma, \Delta) \ .$$

Compare this to the definition of the Yoneda embedding:

$$f \in \mathrm{y}(\Delta)(\Gamma) \iff f \in \mathbb{D}(\Gamma, \Delta)$$

The type El is thus just the Yoneda embedding $\mathrm{y}(\Delta)$, considered as a type depending on a variable $\Delta \colon \mathsf{Obj}$.

We next consider the structure from Sec. 3 in this interpretation.

► **Lemma 14.** *When interpreted in the CwA for $\widehat{\mathbb{D}}$, closed terms of type Obj correspond to to objects of $\mathbb{D}$.*

The lemma allows us to identify terms of type Obj in the dependent type theory for $\widehat{\mathbb{D}}$ with objects in $\mathbb{D}$.

► **Lemma 15.** *In the interal type theory of $\widehat{\mathbb{D}}$, a term of type $\forall x \colon \mathsf{Obj}.\, X$ corresponds to a family of terms $\big(t_A \colon X[A/x]\big)_A$, where A ranges over all objects of $\mathbb{D}$.*

The point is that $\widehat{\mathbb{D}}$ does not impose a naturality condition on functions out of Obj. It suffices to look at the function value at all possible arguments.

These two lemmas justify the terms unit, times and arrow that represent the objects of $\mathbb{D}$ in $\widehat{\mathbb{D}}$.

We next look at the universe El itself. First we note that the interpretation of dependent types in a CwA is such that the interpretation of closed types amounts to the standard interpretation of types in a cartesian closed category. In particular, a closed type X corresponds to an object of $\mathbb{D}$ and a function type $X \to Y$ between closed types X and Y is the exponential.

▶ **Lemma 16.** *For closed A: Obj, the types El A and the object $\mathrm{y}(A)$ are isomorphic.*

This can be seen by substituting A in the type $\mathsf{El} \in \mathrm{Ty}(1 \ltimes \mathsf{Obj})$.

As a consequence, the denotation of closed terms of type $\mathsf{El}\,A \to \mathsf{El}\,B$ corresponds to morphisms $\mathrm{y}(A) \Rightarrow \mathrm{y}(B)$. The interpretation of a term of type $\forall a, b\colon \mathsf{Obj}.\, \mathsf{El}\,a \to \mathsf{El}\,a$ is just a family of such morphisms..

With this observation, Lemmas 1–3 correspond to the standard preservation properties of the Yoneda embedding. For example, that y preserves exponentials can be seen because we have the following natural isomorphism:

$$
\begin{aligned}
(\mathrm{y}A \Rightarrow \mathrm{y}B)(\Gamma) &\cong \widehat{\mathbb{D}}(\mathrm{y}\Gamma, \mathrm{y}A \Rightarrow \mathrm{y}B) \cong \widehat{\mathbb{D}}(\mathrm{y}\Gamma \times \mathrm{y}A, \mathrm{y}B) \\
&\cong \widehat{\mathbb{D}}(\mathrm{y}(\Gamma \times A), \mathrm{y}B) \cong \mathbb{D}(\Gamma \times A, B) \\
&\cong \mathbb{D}(\Gamma, A \Rightarrow B) \cong \widehat{\mathbb{D}}(\mathrm{y}\Gamma, \mathrm{y}(A \Rightarrow B)) \cong \mathrm{y}(A \Rightarrow B)(\Gamma)
\end{aligned}
$$

The box modality is interpreted using the $\flat$-comonad on $\widehat{\mathbb{D}}$ from the main text. Write $\varepsilon_X \colon \flat X \to X$ and $\nu_X \colon \flat X \to \flat\flat X$ for its counit and comultiplication (the latter is actually the identity in our case). We next describe the interpretation of the box modality $[-]$. It suffices to describe the action of $[-]$ on projection maps (for types) and their sections (for terms), since the CwA for $\widehat{\mathbb{D}}$ is a splitting of the codomain fibration on $\widehat{\mathbb{D}}$. The type $[X]$ is defined for any type X with a projection π_X of the form $\pi_X \colon \flat\Gamma \ltimes X \to \flat\Gamma$. It is defined as $\nu_X^*(\flat\pi_X)$. A term $t\colon X$ corresponds to a section $t\colon \flat\Gamma \to \{X\}$ of π_X. The term $[t]$ is interpreted by $\nu_X^*(\flat t)$. Finally, for a term $s\colon [X]$, which is a section s of $\nu_X^*(\flat\pi_X)$, we obtain a section of π_X by post-composing s with the counit. We use this to interpret the let-term for the box type.

The results in Sec. 7 are a direct internal formulation of the CwA-structure of $\mathbb{D}$. We consider Lemma 12 as a representative example. This lemma states an isomorphism between $\forall x\colon \mathsf{ElTm}\ A\ \gamma.\, \mathsf{ElTm}\ B\ (\mathsf{pair}\ \gamma\ x)$ and $\mathsf{ElTm}\ \Pi_A B\ \gamma$. To understand this, we first look at the interpretation of ElTm. Suppose $\Gamma \vdash A$ type and $\Gamma,\, x\colon A \vdash B$ type in the CwA $\mathbb{D}$. Write $\pi_A \colon \Gamma \ltimes A \to \Gamma$ and $\pi_B \colon \Gamma \ltimes A \ltimes B \to \Gamma \ltimes A$ for their projection morphmisn. The projection of the type $\gamma\colon \mathsf{El}\,\Gamma \vdash \mathsf{ElTm}\ A\ \gamma$ is then (up to isomorphism) given by $\mathrm{y}(\pi_A)$. The projection of the type $\gamma\colon \mathsf{El}\,(\mathsf{cons}\ \Gamma\ A) \vdash \mathsf{ElTm}\ B\ \gamma$ is given by $\mathrm{y}(\pi_B)$. The function pair from Sec. 7 amounts to a morphism $\mathsf{pair}\colon \mathrm{y}\Gamma \ltimes (\mathsf{ElTm}\ A) \to \mathrm{y}(\Gamma \ltimes A)$. With this, we have the following isomorphism in slice categories.

$$
\begin{aligned}
\mathbb{D}/\Gamma(id_\Gamma, \Pi_{y(\pi_A)}\mathsf{pair}^*\mathrm{y}(\pi_B)) &\cong \mathbb{D}/y(\Gamma \ltimes A)(id_{\Gamma\ltimes A}, \mathrm{y}(\pi_B)) \\
&\cong \mathbb{D}/(\Gamma \ltimes A)(id, \pi_B) \cong \mathbb{D}/\Gamma(id_\Gamma, \pi_{\Pi_A B}) \\
&\cong \mathbb{D}/\mathrm{y}\Gamma(id_\Gamma, \mathrm{y}(\pi_{\Pi_A B}))
\end{aligned}
$$

This expresses the isomorphism between $\forall x\colon \mathsf{ElTm}\ A\ \gamma.\, \mathsf{ElTm}\ B\ (\mathsf{pair}\ \gamma\ x)$ and $\mathsf{ElTm}\ \Pi_A B\ \gamma$.

B Interpretation of Simple Contextual Types

To show soundness of the interpretation lemmas of simple contextual types from Sec. 4, we need substitution lemmas, of which we spell out representative cases here.

Substitution variables in context Δ simply becomes substitution on the meta-level, since the computation part is translated with a shallow embedding.

▶ **Lemma 17.**

- $[\![\Delta,\ X{:}\,U,\ \Delta';\ \Phi \vdash t\colon A]\!][[\![\Delta \vdash C\colon U]\!]/X] = [\![\Delta,\ \Delta'[C/U];\ \Phi \vdash t[C/U]\colon A]\!]$.
- $[\![\Delta,\ X{:}\,U,\ \Delta';\ \Gamma \vdash e\colon \tau]\!][[\![\Delta \vdash C\colon U]\!]/X] = [\![\Delta,\ \Gamma[C/U];\ \Phi \vdash e[C/U]\colon \tau]\!]$.

Substitution on the Φ-context becomes substitution for the new variable γ, which is essentially composition, as usual. To define it, we need some notation. We define a lifting from $[\![\Delta]\!], \gamma\colon \mathsf{El}\,[\![\Phi]\!] \vdash t\colon \mathsf{El}\,[\![\Phi, x\colon A]\!]$ to $[\![\Delta]\!], \gamma\colon \mathsf{El}\,[\![\Phi, \Psi]\!] \vdash \mathit{lift}_\Psi(t)\colon \mathsf{El}\,[\![\Phi, x\colon A, \Psi]\!]$. It is defined by $\mathit{lift}_{(\cdot)}(t) = t$ and $\mathit{lift}_{(\Psi, x\colon C)}(t) = \lambda y.\, \mathsf{pair}\ (\mathit{lift}_\Psi(t)[\mathsf{fst}\ \gamma/\gamma])\ (\mathsf{snd}\ \gamma)$.

▶ **Lemma 18.** $[\![\Delta; \Phi, x\colon A, \Psi \vdash t\colon B]\!][\mathit{lift}_\Psi(\mathit{pair}\,\gamma\,[\![\Delta; \Phi \vdash s\colon A]\!])/\gamma] = [\![\Delta, \Phi, \Psi \vdash t[s/x]\colon B]\!]$.

Notice that in the interpretation of domain-level terms, a variable x from Φ is interpreted by projection from $\gamma\colon \mathsf{El}\,[\![\Phi]\!]$. In the main text, we have used projections, such as $\pi_{(\Phi, x\colon A), x}\colon \mathsf{El}\,(\mathsf{times}\ [\![\Phi]\!]\ [\![A]\!]) \to \mathsf{El}\,[\![A]\!]$ for this purpose. The lemma just states that pre-composing such a projection with a particular term amounts to substituting the term.

The soundness lemma then follows using meta-level equations and a substitution lemma.

C Interpretation of Contextual LF

We follow Streicher's approach [23] to the interpretation of the syntax of dependent types by first defining the interpretation as a partial function by induction on derivations. The non-trivial cases are given in the main text and in the Agda code[1]. To show that this interpretation is in fact total, we need weakening and substitution lemmas. We state these lemmas only for domain-level variables, as the other variables are again moved to the meta level as in the simply-typed case.

▶ **Lemma 19** (Weakening).

- $\mathit{sub}\ [\![\Delta; \Phi \vdash B\ \mathit{type}]\!]\ p = [\![\Delta; \Phi, x\colon A \vdash B\ \mathit{type}]\!]$
- $\mathit{subElTm}\ ([\![\Delta; \Phi \vdash t\colon B]\!][p\ \gamma/\gamma]) = [\![\Delta; \Phi, x\colon A \vdash t\colon B]\!]$

In the second part, recall that $[\![\Delta; \Phi \vdash t\colon B]\!]$ has type $\mathsf{ElTm}\ [\![\Delta; \Phi \vdash B\ \text{type}]\!]\ \gamma$, where γ is a variable of type $\mathsf{El}\,[\![\Phi]\!]$. Thus, $\mathsf{subElTm}\ ([\![\Delta; \Phi \vdash t\colon B]\!][p\ \gamma/\gamma])$ has type $\mathsf{ElTm}\ (\mathsf{sub}\ [\![\Delta; \Phi \vdash B\ \text{type}]\!]\ p)\ \gamma$, which by the first point has correct type for the right-hand side. The proof of the lemma goes by induction on the derivation of $[\![B]\!]$ and $[\![t]\!]$.

▶ **Lemma 20** (Substitution).

- $\mathit{sub}\ [\![\Delta; \Phi, x\colon A \vdash B\ \mathit{type}]\!]\ (\lambda\gamma.\, \mathit{pair}\ \gamma\ [\![\Delta; \Phi \vdash s\colon A]\!]) = [\![\Delta; \Phi \vdash B[s/x]\ \mathit{type}]\!]$
- $\mathit{subElTm}\ ([\![\Delta; \Phi, x\colon A \vdash t\colon B]\!][\mathit{pair}\ \gamma\ [\![\Delta; \Phi \vdash s\colon A]\!]/\gamma]) = [\![\Delta; \Phi \vdash t[s/x]\colon B[s/x]]\!]$

The proof goes by showing a slightly stronger property (with a context Ψ after the variable $x\colon A$) by induction on the derivation of B and t.

With the weakening and substitution lemmas, Lemma 13 follows by induction on the showing that the partial interpretation is in fact total by induction on derivations.