Shallowly Embedding Type Theories as Presheaf Models in Agda

Extended Abstract

Joris Ceulemans
Vrije Universiteit Brussel
joris.ceulemans@vub.be

Dominique Devriese
Vrije Universiteit Brussel
dominique.devriese@vub.be

1 Introduction

There are several type systems which extend Martin-Löf Type Theory (MLTT) by providing extra operations on types and terms or by including new axioms. Of course, the ability to use these new constructs when proving a theorem or when writing a program does not come for free, one has to check that the resulting type system remains consistent. In a lot of cases this can be accomplished by constructing a presheaf model of the type theory in which all the extra operations and axioms are given a semantic meaning [8]. Examples include presheaf models of type theories with support for guarded recursion [4], with support for parametricity [1, 11, 12], and of univalent/cubical type theory [5].

However, in order to use these new features in a proof assistant based on MLTT such as Agda, one basically has two options: either to postulate the new operations or axioms, in which case they will not have any computational content, or to implement an extension of the proof assistant, which has happened for instance with Agda’s cubical mode [15] but which requires a lot of work (and in general this effort needs to be repeated for every extension of MLTT one wants to consider). In both approaches, soundness of the extensions needs to be proven separately, meta-theoretically.

In this extended abstract, we present work in progress on a shallow embedding of extensions of MLTT in Agda as presheaf models. More concretely, the terms, types, ... of an object theory are represented in Agda using the presheaf construction and a user can manipulate them in the style of a category with families (CwF) [6, 8], with variables in de Bruijn form. Extra operations or axioms can be implemented by instantiating the framework with a suitable base category.

Most Agda definitions appearing in the text will only include the type, not the implementation. Details such as universe levels will be elided to enhance readability. The full code can be found at https://github.com/JorisCeulemans/shallow-presheaf-embedding/tree/tyde-2020.

2 Overview of the Framework

Our framework is parametrized by a small base category $C$ that will depend on the extension of MLTT under consideration. For such a category $C : \text{Category}$, we denote by $\text{Ob } C$ its type of objects and by $\text{Hom } C x y : \text{Set}$ the type of morphisms from an object $x$ to an object $y$.

The framework then consists of Agda (record) types for the different kinds of judgements present in MLTT. The overall structure is that of Dybjer’s internal CwFs [6]. This means that we first introduce the notion of contexts, then that of types in a context and finally that of terms of a type in a context.

First of all, a context is represented as a presheaf over the base category $C$ (i.e. a contravariant functor from $C$ to the category of Agda types and functions).

```agda
record Ctx (C : Category) : Set where
  field
    set : Ob C -> Set
    rel : {x y} -> Hom C x y -> set y -> set x

Here two fields expressing the functor laws were elided.

Similarly, we have for any context $\Gamma : \text{Ctx } C$ an Agda type of types in this context,

```agda
record Ty (Γ : Ctx C) : Set where

and for every type $T : \text{Ty } \Gamma$ there is an Agda type of terms of type $T$ in the context $\Gamma$.

```agda
record Tm (Γ : Ctx C) (T : Ty Γ) : Set where

The precise types of the fields in these records will not be important in the rest of the discussion.

Furthermore, we provide for any two contexts $\Delta$ and $\Gamma$ a type $\Delta \Rightarrow \Gamma$ of substitutions from $\Delta$ to $\Gamma$ and an action of a substitution on types and terms. More concretely, if $\sigma : \Delta \Rightarrow \Gamma$ and $T : \text{Ty } \Gamma$ and $t : \text{Tm } \Gamma T$, then $T [ \sigma ] : \text{Ty } \Delta$ and $t [ \sigma ] : \text{Tm } \Delta (T [ \sigma ])$. We can also extend a context $\Gamma$ with a type $T : \text{Ty } \Gamma$ to obtain a context $\Gamma , T$ (which would be written in MLTT as $\Gamma , x : T$) and then we get a substitution $\pi : \Gamma , T \Rightarrow \Gamma$ and a term $\xi : \text{Tm } (\Gamma , T) (T [ \pi ])$. This term $\xi$ corresponds to the variable rule in MLTT for the last variable in the context (so to the judgement $\Gamma , x : T \vdash x : T$). Finally, there are Agda types expressing equality of substitutions, of types and of terms. Each of these types will in the text be denoted by $\equiv$. Given an equality proof $e : T \equiv S$ for two types $T, S : \text{Ty } \Gamma$, a term $s : \text{Tm } \Gamma S$ can be converted into a term $[e] s : \text{Tm } \Gamma T$.

1Working with these custom-defined equality types turns out to be easier than with standard propositional equality.
Moreover, in any presheaf model (irrespective of the base category $C$) we can construct simple types, such as booleans and natural numbers, and some basic type operators and term constructors. For example, we have a type $\text{Nat}^\ast : Tm\ \Gamma$ of natural numbers and a term $\text{zero}^\ast : Tm\ \Gamma\ \text{Nat}^\ast$ for any context $\Gamma$ and we can implement simple product types

\[
\text{pair} : Tm\ \Gamma\ (\text{Tm}^\ast\ S) \rightarrow Tm\ \Gamma\ T
\]

and simple (non-dependent) function types.

\[
\text{app} : Tm\ \Gamma\ (T \Rightarrow S) \rightarrow Tm\ \Gamma\ T \rightarrow Tm\ \Gamma\ S
\]

and $\lambda$-abstractions.

\[
\text{str-cons} : Tm\ \Gamma\ (\text{Tm}^\ast\ (\text{Tm}^\ast\ \text{Stream})) \rightarrow Tm\ \Gamma\ \text{Stream}
\]

We can then use Löb induction to define a constant stream of zeros

\[
\text{zeros} : Tm\ \circ\ \text{Stream}
\]

\[
\text{zeros} = \text{ßLöb Stream}
\]

\[
(\text{ßLöb Stream}) (\text{[]}) \equiv (\text{str-cons} (\text{pair zero}^\ast (\text{[]})))
\]

where $\circ$ is the empty context and where the … represent two equality proofs for types. More concretely, the first proof has type $\text{Stream} [\pi]\equiv\text{Stream}$ and this fact is not surprising as $\text{Stream}$ is a non-dependent type (the proof itself is also very simple and provided as an operation available to the user). The second proof, on the other hand, is not straightforward although it also boils down to the fact that $\text{Stream}$ is a non-dependent type. We consider this an inconvenience in our framework that needs to be dealt with in order to obtain a system that is user-friendly.

### 3 A Concrete Example: Guarded Recursion

To demonstrate how we intend to use the embedding of presheaf models in Agda described above, we will consider in this section guarded recursion as a specific application. Guarded recursion was originally developed by Nakano [10], it is a technique for writing productive recursive definitions involving coinductive data types using a modality $\triangleright$ on types called "later". Presheaf models for guarded recursion were described in for instance [3, 4].

We can work with guarded recursion in our framework by instantiating it with the category $\omega$ as the base category (this is the category structure induced on the set $\mathbb{N}$ of natural numbers by its standard order relation). In this case we can define a later modality $\triangleright$ on types.$^2$

\[
\triangleright : [\Gamma : \text{Ctx}\ \omega] \rightarrow Tm\ \Gamma \rightarrow Tm\ \Gamma
\]

Using standard induction for natural numbers, we can also provide an operation corresponding to Löb induction (note that $\triangleright$ binds more tightly than $\Rightarrow$).

\[
\text{ßLöb} : (T : Tm\ \Gamma) \rightarrow Tm\ \Gamma\ (\triangleright\ T \Rightarrow T) \rightarrow Tm\ \Gamma\ T
\]

and show that it produces fixpoints.

\[
\text{ßLöb-fixpoint} : (f : Tm\ \Gamma\ (\triangleright\ T \Rightarrow T)) \rightarrow
\]

\[
\text{ßLöb} f \equiv \text{app} f (\text{next}^\prime (\text{ßLöb} T f))
\]

The prototypical example in the literature on guarded recursion is the type of guarded streams. We can define in our setting the type

\[
\text{Stream} : [\Gamma : \text{Ctx}\ \omega] \rightarrow Tm\ \Gamma
\]

of guarded streams of natural numbers. The intuition is that $\text{Stream}$ is isomorphic to the type $\text{Nat}^\ast \triangleright \text{Stream}$, and hence the constructor for streams has the following form.$^3$

\[
\text{str-cons} : Tm\ \Gamma\ (\text{Nat}^\ast\ (\triangleright\ \text{Stream})) \rightarrow Tm\ \Gamma\ \text{Stream}
\]

where $\circ$ is the empty context and where the … represent two equality proofs for types. More concretely, the first proof has type $\text{Stream} [\pi]\equiv\text{Stream}$ and this fact is not surprising as $\text{Stream}$ is a non-dependent type (the proof itself is also very simple and provided as an operation available to the user). The second proof, on the other hand, is not straightforward although it also boils down to the fact that $\text{Stream}$ is a non-dependent type. We consider this an inconvenience in our framework that needs to be dealt with in order to obtain a system that is user-friendly.

### 4 Related and Future Work

In [9], Jaber et al. describe a translation based on presheaves from extensions of the Calculus of Constructions (CoC) to CoC itself. However, their framework is quite different to work with (there is no CwF-like structure) and they make use of subtypes, which are not present in Agda. Velti van der Weide [14] provide presheaf semantics of guarded recursion in Agda in a way which is similar to ours. Both [9] and [14] consider presheaves over a preorder rather than a general category. Guarded recursion can also be presented in Agda using ordered families of equivalences, as in for example https://github.com/metaborg/mj.agda/tree/develop. Again, this approach does not generalize to arbitrary categories. In [2], Bickford formalizes presheaf models for general base categories using the structure of a CwF in Nuprl, which is a proof assistant implementing an extensional version of type theory. To the best of our knowledge, our framework is the first shallow embedding of type theories using presheaves over general categories in intensional type theory. Another approach to formalize presheaf models, is to use Agda’s type theory as the internal language of the presheaf topos, like in [13].

As discussed previously, we first want to make our framework more usable by reducing the number of type equality proofs that a user needs to provide. Next, we plan to study different extensions of MLTT using concrete presheaf models with specific base categories. In the first instance, we intend to consider applications involving non-dependent types, but we are planning to add dependent types and a universe type to the framework as well in a later phase. Another interesting path to explore in our framework is multimode type theory as described in [7].
Acknowledgments
Joris Ceulemans holds a PhD Fellowship from the Research Foundation – Flanders (FWO).

References