Vrije Universiteit Brussel

Shallowly Embedding Type Theories as Presheaf Models in Agda

Ceulemans, Joris; Devriese, Dominique

Publication date:
2020

Document Version:
Final published version

Link to publication

Citation for published version (APA):
Ceulemans, J., & Devriese, D. (2020). *Shallowly Embedding Type Theories as Presheaf Models in Agda: Extended Abstract*. Abstract from Workshop on Type-Driven Development, Jersey City, New Jersey, United States.

Copyright
No part of this publication may be reproduced or transmitted in any form, without the prior written permission of the author(s) or other rights holders to whom publication rights have been transferred, unless permitted by a license attached to the publication (a Creative Commons license or other), or unless exceptions to copyright law apply.

Take down policy
If you believe that this document infringes your copyright or other rights, please contact openaccess@vub.be, with details of the nature of the infringement. We will investigate the claim and if justified, we will take the appropriate steps.

Download date: 25. May. 2025

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 : Category, we denote by Ob C : Set its type of objects and by Hom $C\ x\ y$: 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).

```
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 Γ : Ctx C an Agda type of types in this context,

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

and for every type T : Ty Γ there is an Agda type of terms of type T in the context Γ.

```
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 Δ and Γ a type Δ ⇒ Γ of substitutions from Δ to Γ and an action of a substitution on types and terms. More concretely, if σ : Δ ⇒ Γ and T : Ty Γ and t : Tm Γ T, then T [σ] : Ty Δ and t [σ]' : Tm Δ (T [σ]). We can also extend a context Γ with a type T : Ty Γ to obtain a context Γ „ T (which would be written in MLTT as $\Gamma, x : T$) and then we get a substitution π : Γ „ T ⇒ Γ and a term ξ : Tm (Γ „ T) (T [π]). This term ξ 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.[1] Each of these types will in the text be denoted by _≅_. Given an equality proof e : $T \cong S$ for two types T, S : Ty Γ, a term s : Tm Γ S can be converted into a term ι[e] s : Tm Γ T.

[1] Working with these custom-defined equality types turns out to be easier than with standard propositional equality.

TyDe 2020, August 23, 2020, online
2020.

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 Nat' : Ty Γ of natural numbers and a term zero' : Tm Γ Nat' for any context Γ and we can implement simple product types

```
_⊠_ : Ty Γ → Ty Γ → Ty Γ
pair : Tm Γ T → Tm Γ S → Tm Γ (T ⊠ S)
fst : Tm Γ (T ⊠ S) → Tm Γ T
snd : Tm Γ (T ⊠ S) → Tm Γ S
```

and simple (non-dependent) function types.

```
_⇛_ : Ty Γ → Ty Γ → Ty Γ
lam : (T : Ty (Γ „ T)) (S [ π ]) → Tm Γ (T ⇛ S)
app : Tm Γ (T ⇛ S) → Tm Γ T → Tm Γ S
```

Note that the body of a lambda abstraction has type $S\,[\,\pi\,]$ rather than S because the latter is not a type in context $\Gamma \mathbin{„} T$.

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 ▹ 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 ω 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 ▹' on types.[2]

```
▹' : {Γ : Ctx ω} → Ty Γ → Ty Γ
next' : Tm Γ T → Tm Γ (▹' T)
```

Using standard induction for natural numbers, we can also provide an operation corresponding to Löb induction (note that ▹' binds more tightly than ⇛)

```
löb : (T : Ty Γ) → Tm Γ (▹' T ⇛ T) → Tm Γ T
```

and show that it produces fixpoints.

```
löb-is-fixpoint : (f : Tm Γ (▹' T ⇛ T)) →
                  löb T f ≅ app f (next' (löb 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

```
Stream : {Γ : Ctx ω} → Ty Γ
```

of guarded streams of natural numbers. The intuition is that Stream is isomorphic to the type Nat' ⊠ ▹' Stream, and hence the constructor for streams has the following form.[3]

```
str-cons : Tm Γ (Nat' ⊠ (▹' Stream)) → Tm Γ Stream
```

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

```
zeros : Tm ⋄ Stream
zeros = löb Stream
          (lam (▹' Stream) (ι[ … ] (str-cons
            (pair zero' (ι[ … ] ξ)))))
```

where ⋄ is the empty context and where the … represent two equality proofs for types. More concretely, the first proof has type Stream [π] ≅ Stream and this fact is not surprising as 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 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. Veltri and 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].

[2] Note that in the Agda code the modality ▹' is defined in terms of a more basic modality ▹ that we will not discuss in this abstract. This explains the prime symbol in its name.

[3] This is not completely accurate regarding the difference between ▹ and ▹'. See the full code for details.

Acknowledgments

Joris Ceulemans holds a PhD Fellowship from the Research Foundation – Flanders (FWO).

References

[1] Robert Atkey, Neil Ghani, and Patricia Johann. 2014. A Relationally Parametric Model of Dependent Type Theory. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Diego, California, USA) *(POPL '14)*. Association for Computing Machinery, New York, NY, USA, 503–515. https://doi.org/10.1145/2535838.2535852

[2] Mark Bickford. 2018. Formalizing Category Theory and Presheaf Models of Type Theory in Nuprl. *CoRR* abs/1806.06114 (2018). arXiv:1806.06114 http://arxiv.org/abs/1806.06114

[3] Lars Birkedal, Rasmus Ejlers Møgelberg, Jan Schwinghammer, and Kristian Støvring. 2012. First steps in synthetic guarded domain theory: step-indexing in the topos of trees. *Logical Methods in Computer Science* Volume 8, Issue 4 (Oct. 2012). https://doi.org/10.2168/LMCS-8(4:1)2012

[4] Aleš Bizjak and Rasmus Ejlers Møgelberg. 2018. Denotational semantics for guarded dependent type theory. *CoRR* abs/1802.03744 (2018), 40 pages. arXiv:1802.03744 http://arxiv.org/abs/1802.03744

[5] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. 2016. Cubical Type Theory: a constructive interpretation of the univalence axiom. *CoRR* abs/1611.02108 (2016), 34. arXiv:1611.02108 http://arxiv.org/abs/1611.02108

[6] Peter Dybjer. 1996. Internal type theory. In *Types for Proofs and Programs*, Stefano Berardi and Mario Coppo (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 120–134.

[7] Daniel Gratzer, G.A. Kavvos, Andreas Nuyts, and Lars Birkedal. 2020. Multimodal Dependent Type Theory. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science* (Saarbrücken, Germany) *(LICS '20)*. ACM, New York, NY, USA, 15. https://doi.org/10.1145/3373718.3394736

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

[9] Guilhem Jaber, Nicolas Tabareau, and Matthieu Sozeau. 2012. Extending type theory with forcing. In *27th Annual IEEE Symposium on Logic in Computer Science*. IEEE, 395–404. https://doi.org/10.1109/LICS.2012.49

[10] H. Nakano. 2000. A modality for recursion. In *Proceedings Fifteenth Annual IEEE Symposium on Logic in Computer Science*. 255–266.

[11] Andreas Nuyts and Dominique Devriese. 2018. Degrees of Relatedness: A Unified Framework for Parametricity, Irrelevance, Ad Hoc Polymorphism, Intersections, Unions and Algebra in Dependent Type Theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science* (Oxford, United Kingdom) *(LICS '18)*. Association for Computing Machinery, New York, NY, USA, 779–788. https://doi.org/10.1145/3209108.3209119

[12] Andreas Nuyts, Andrea Vezzosi, and Dominique Devriese. 2017. Parametric Quantifiers for Dependent Type Theory. *Proc. ACM Program. Lang.* 1, ICFP, Article 32 (Aug. 2017), 29 pages. https://doi.org/10.1145/3110276

[13] Ian Orton and Andrew M. Pitts. 2018. Axioms for Modelling Cubical Type Theory in a Topos. *Logical Methods in Computer Science* Volume 14, Issue 4 (Dec. 2018). https://doi.org/10.23638/LMCS-14(4:23)2018

[14] Niccolò Veltri and Niels van der Weide. 2019. Guarded recursion in Agda via sized types. In *4th International Conference on Formal Structures for Computation and Deduction* (Dortmund, Germany). Dagstuhl: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 32:1–32:18. https://hdl.handle.net/2066/204586

[15] Andrea Vezzosi, Anders Mörtberg, and Andreas Abel. 2019. Cubical Agda: A Dependently Typed Programming Language with Univalence and Higher Inductive Types. *Proc. ACM Program. Lang.* 3, ICFP, Article 87 (July 2019), 29 pages. https://doi.org/10.1145/3341691