Shallow embedding of type theory is morally correct *

Ambrus Kaposi[1], András Kovács[1], and Nicolai Kraus[2]

[1] Eötvös Loránd University, Budapest, Hungary
akaposi|kovacsandras@inf.elte.hu
[2] University of Birmingham, United Kingdom
n.kraus@bham.ac.uk

Deep and shallow embeddings of monoids. In intensional type theory, when proving theorems that hold for every monoid, the usual method is assuming that there exists a pointed type with a binary operation and witnesses of some equalities (following Agda, we write $\equiv$ for the equality type and $=$ for definitional equality). We call this *deep embedding*, following the terminology for domain-specific languages [6].

$$\begin{array}{ll} \mathsf{M} & : \mathsf{Set} \\ \mathsf{u} & : \mathsf{M} \\ - \otimes - & : \mathsf{M} \to \mathsf{M} \to \mathsf{M} \end{array} \qquad \begin{array}{ll} \mathsf{ass} & : (a\,b\,c : \mathsf{M}) \to (a \otimes b) \otimes c \equiv a \otimes (b \otimes c) \\ \mathsf{idl} & : (a : \mathsf{M}) \to \mathsf{u} \otimes a \equiv a \\ \mathsf{idr} & : (a : \mathsf{M}) \to a \otimes \mathsf{u} \equiv a \end{array}$$

Combining the equalities idl, idr using congruence (ap) and transitivity, we prove the following example theorem.

$$\begin{aligned} &\mathsf{thm} : (a : \mathsf{M}) \to a \otimes (\mathsf{u} \otimes \mathsf{u}) \equiv a \\ &\mathsf{thm} := \lambda a.\mathsf{trans}\,(\mathsf{ap}\,(a \otimes -)\,(\mathsf{idl}\,\mathsf{u}))\,(\mathsf{idr}\,a) \end{aligned}$$

An alternative approach is *shallow embedding* of the monoid. Here we work with a *concrete* monoid such as the following one.

$$\begin{array}{ll} \mathsf{M} & := \mathsf{Bool} \to \mathsf{Bool} \\ \mathsf{u} & := \lambda x.x \\ a \otimes b & := \lambda x.a\,(b\,x) \end{array} \qquad \begin{array}{ll} \mathsf{ass}\,a\,b\,c & := \mathsf{refl}_{\lambda x.a\,(b\,(c\,x))} \\ \mathsf{idl}\,a & := \mathsf{refl}_a \\ \mathsf{idr}\,a & := \mathsf{refl}_a \end{array}$$

The advantage of using this monoid compared to a deeply embedded one is that the laws hold *definitionally*. For example, the proof of the above theorem now becomes trivial:

$$\begin{aligned} &\mathsf{thm} : (a : \mathsf{M}) \to a \otimes (\mathsf{u} \otimes \mathsf{u}) \equiv a \\ &\mathsf{thm} := \lambda a.\mathsf{refl}_a \end{aligned}$$

This monoid does not have more definitional equalities than a general monoid, e.g. we don't have the property that any two elements are equal (as would be the case if we used $\top \to \top$ as carrier). However, it has the property that there is an element propositionally unequal to u, e.g. $(\lambda a.\mathsf{true})$. Also, assuming function extensionality, we have $a \otimes a \otimes a \equiv a$ propositionally. These do not hold for every monoid, hence we can prove too many theorems.

*The first author was supported by by the National Research, Development and Innovation Fund of Hungary, financed under the Thematic Excellence Programme funding scheme, Project no. ED_18-1-2019-0030 (Application-specific highly reliable IT solutions), by the ÚNKP-19-4 New National Excellence Program of the Ministry for Innovation and Technology and by the Bolyai Fellowship of the Hungarian Academy of Sciences. The second author was supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002). The first and third authors were supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013 and EFOP-3.6.3-VEKOP-16-2017-00002). Finally, the third author acknowledges support by The Royal Society (URF\R1\191055).

We disallow such illegal constructions with an implementation hiding trick, using the record types of Agda with definitional η laws. The monoid is defined in the module Secret as a record M wrapping the function type Bool → Bool with constructor mkM and destructor unM. We only import Secret in the module Monoid and other modules are only allowed to import Monoid, but not Secret.

```
module Secret where
  record M := mkM {unM : Bool → Bool}
  u        := mkM (λx.x)
  a ⊗ b    := mkM (λx.unM a (unM b x))
```

```
module Monoid where
  import Secret privately
  M      := Secret.M
  u      := Secret.u
  _⊗_    := Secret._⊗_
```

A module importing Monoid only has access to M, u and $-\otimes-$, but not mkM and unM. However, the definitional behaviour of the operations is exported, so proofs are still as easy as for the naive shallow embedding.

In principle, we should be able to transfer any proof about the shallowly embedded monoid to a deeply embedded one.

Deep and shallow embeddings of type theory. Type theory can also be seen as an algebraic structure [5]. Compared to monoids, there are more sorts and many more operations and equations. Metatheoretic proofs about type theory can be seen as constructions on models of type theory. This is the case e.g. for normalisation [1], parametricity [2], or canonicity [4], the latter two being special cases of gluing [7], a construction on a weak homomorphism of models. When formalising such arguments for deeply embedded models, combining equalities and transporting over them becomes a huge bureaucratic burden, sometimes called "transport hell". However, as in the case of monoids, we are able to reuse properties of our metatheory (e.g. strict associativity of function composition) to define a shallow embedding of type theory, where (most) equalities are definitional. The shallow embedding is a concrete model (the *standard model* [2] – sometimes called *set model* or *metacircular interpretation*) in which all equations hold definitionally, and as before, we only export the interface. In this model, contexts are defined as Set, a type over Γ is a $\Gamma \to$ Set function, terms have dependent function type $(\gamma : \Gamma) \to A\,\gamma$.

Moral correctness. By proving all equations using refl, we can check that our shallow embedding has enough equalities. The implementation hiding makes sure that we cannot construct too many elements and proofs. However, we have to prove that we don't have too many definitional equalities. When showing this, we assume that Agda implements type theory correctly and we look at the standard model from outside of Agda. Externally, contexts are given by Tm · U (Agda-terms in the empty context of type U for the universe), types are in Tm · (El Γ ⇒ U), terms are in Tm · (Π (El Γ) (El (A $ var 0))), and so on. We prove that for any two syntactic terms, if their external standard interpretations are definitionally equal, then they are also definitionally equal. This shows that the standard model does not add more equalities than there are in the syntax.

Applications. Using this form of shallow embedding, for a type theory with an infinite hierarchy of universes, Π, Σ, Bool and Id types, we formalised [8] the syntactic logical predicate interpretation of type theory [3], canonicity [4], and gluing [7]. The line count of the parametricity proof is roughly 20% of the same proof for the deeply embedded syntax [2]. More details can be found in a paper presented at MPC 2019 [9].

References

[1] Thorsten Altenkirch and Ambrus Kaposi. Normalisation by evaluation for dependent types. In Delia Kesner and Brigitte Pientka, editors, *1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016)*, volume 52 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:16, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[2] Thorsten Altenkirch and Ambrus Kaposi. Type theory in type theory using quotient inductive types. In Rastislav Bodik and Rupak Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 18–29. ACM, 2016.

[3] Jean-Philippe Bernardy, Patrik Jansson, and Ross Paterson. Proofs for free — parametricity for dependent types. *Journal of Functional Programming*, 22(02):107–152, 2012.

[4] Thierry Coquand. Canonicity and normalization for dependent type theory. *Theor. Comput. Sci.*, 777:184–191, 2019.

[5] Peter Dybjer. Internal type theory. In *Lecture Notes in Computer Science*, pages 120–134. Springer, 1996.

[6] Jeremy Gibbons and Nicolas Wu. Folding domain-specific languages: Deep and shallow embeddings (functional pearl). *SIGPLAN Not.*, 49(9):339–347, August 2014.

[7] Ambrus Kaposi, Simon Huber, and Christian Sattler. Gluing for type theory. In Herman Geuvers, editor, *Proceedings of the 4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019)*, 2019.

[8] Ambrus Kaposi, András Kovács, and Nicolai Kraus. Formalisations in Agda using a morally correct shallow embedding, May 2019.

[9] Ambrus Kaposi, András Kovács, and Nicolai Kraus. Shallow embedding of type theory is morally correct. In Graham Hutton, editor, *Mathematics of Program Construction*, pages 329–365, Cham, 2019. Springer International Publishing.