Formalisations Using Two-Level Type Theory[*]

Danil Annenkov[1], Paolo Capriotti[2], and Nicolai Kraus[2]

[1] University of Copenhagen
[2] University of Nottingham

Abstract. We explain the basic idea of *two-level type theory* [1, 2], a version of Martin-Löf type theory with two equality types: the first acts as the usual equality of homotopy type theory, while the second allows us to reason about strict equality. In this system, we can formalise results of partially meta-theoretic nature. We demonstrate how we have implemented such results [2] in the proof assistant `Lean`.

Motivation. The motivation for two-level type theory is twofold.

Many results of homotopy type theory are completely internal to HoTT and can be formalised directly in a proof assistant, and a lot of work has been done using Agda, Coq, and Lean. Some other results are only *partially* internal to HoTT. One example is the construction of n-restricted semisimplicial types which we can do only after fixing a number n *externally* (i.e. we have to decide which n we use before we start writing it down in Agda). Another example is the work by Shulman on inverse diagrams [5], where we can do constructions in type theory once we fix a (finite) inverse category *in the meta-theory*. In many situations, one would like such constructions to be completely internal (using a variable $n : \mathbb{N}$ or an inverse category expressed internally) and formalisable in a proof assistant, but unfortunately, it is either unknown how this is doable or it is known to be impossible. Two-level type theory gives a way to completely formalise such results. This is the aspect that we explore in our paper [2].

A second motivation of two-level type theory is that it allows to extend homotopy type theory in a "controlled" way. It gives a framework which makes it easy to write down enhancements of the theory, and one can relatively easily check whether these assumptions hold in some models (models are explored in [3]).

Two-Level Type Theory. Two level type theory consists of two fragments: a *strict* fragment (a form of MLTTwith UIP) and a *fibrant* fragment (essentially HoTT). The fibrant fragment of our type theory has all the basic types and type formers found in HoTT [6, Appendix A.2]: $\mathbf{1}$, $\mathbf{0}$, $\mathbb{N}$, $=$ (the equality type), Π, Σ, $+$, along with a hierarchy $\mathcal{U}_0, \mathcal{U}_1, \ldots$ of universes, and possibly inductive and higher inductive types. The strict fragment has $\mathbf{0}^{\mathrm{s}}$, $\mathbb{N}^{\mathrm{s}}$, $+^{\mathrm{s}}$, $\overset{\mathrm{s}}{=}$ (the strict equality), a hierarchy $\mathcal{U}_0^{\mathrm{s}}, \mathcal{U}_1^{\mathrm{s}}, \ldots$ of strict universes. Type formers Π, Σ and the unit type $\mathbf{1}$ are shared by the two theories.

We refer to the elements of $\mathcal{U}_i$ as *fibrant types*, while the elements of $\mathcal{U}_i^{\mathrm{s}}$ are *pretypes*. The intuition is that fibrant types are the usual types that are found in HoTT, whereas pretypes are what one gets if one is allowed to talk about strict equality internally. In other words, strict equality serves as an internalised version of judgmental equality. These ideas of differentiating pretypes and fibrant types is inspired by Voevodsky's *Homotopy Type System* (HTS) [7], but there are some important differences. In particular, two-level type theory does not assume the reflection rule for the strict equality. Instead, we only require that it satisfies UIP. Another

[*]This work is supported by the HIPERFIT Research Centre, Danish Council for Strategic Research, contract number 10-092299, by USAF, Airforce office for scientific research, award FA9550-16-1-0029 and by the Engineering and Physical Sciences Research Council (EPSRC), grant reference EP/M016994/1

important difference is that HTS assumes that $\mathbf{0}$, $\mathbb{N}$ and $+$ from the fibrant fragment eliminate to arbitrary types. We are more minimalistic and do not hard-wire this principle in the theory as none of our results require it, although such assumptions can be added easily.

Applications and Results. In our paper [2], we develop some of the theory on inverse diagrams in two-level type theory to demonstrate how arguments that traditionally have to be done meta-theoretically can be expressed in a uniform type-theoretic way. A special case are semisimplicial types, which in our setting can be worked with internally. As an application, we show how to define the notion of *complete semi-Segal type*, which is a way to capture and study categorical structures built with types of arbitrary truncation levels (usually referred to as $(\infty, 1)$-categories), such as universes of fibrant types, and the ∞-groupoid associated to any fibrant type.

Formalisation. For the formalisation[1] of two-level type theory we have chosen to use the Lean proof assistant [4], although the overall idea of our implementation approach should be suitable for many existing proof assistants.[2] We work in "strict" Lean mode, which provides us with a close enough approximation for the strict fragment of two-level type theory. Fibrant types are implemented as a record type `Fib` with two fields: a pretype, and the property that it is fibrant. The `is_fibrant` property is defined using the type class mechanism provided by the language. Lean coercions are used to implement the rule that every fibrant type is also a pretype. The class instance resolution mechanism allows us to leave the property of being fibrant implicit in most cases. The fibrant equality type (i.e. the "HoTT equality") is a type family that we define ourselves, together with its eliminator J. The β-rule for J is postulated using Lean's propositional equality type, and reductions do therefore not always happen automatically. This issue can partially be resolved using proof automation techniques, but there are some cases when the need for the explicit application of the computation rule makes definitions awkward to work with.

Nevertheless, this setup allows us to experiment with two-level type theory in a currently available proof assistant, and our experience is that this works reasonably well. We demonstrate how one can use the fibrant fragment of two-level type theory to develop proofs in HoTT by formalising simple lemmas from the Lean HoTT library. Our experience shows that many proofs can be reused almost without change, provided that the same notation is used for basic definitions. As an example of the internalisation of meta-theoretic reasoning, we implemented the proof that, for I a finite inverse category and $D : I \to \mathcal{U}$ a Reedy fibrant diagram, the limit of D is a fibrant type again [5].

References

[1] Thorsten Altenkirch, Paolo Capriotti, and Nicolai Kraus. Extending Homotopy Type Theory with Strict Equality. In Jean-Marc Talbot and Laurent Regnier, editors, *25th EACSL Annual Conference on Computer Science Logic (CSL 2016)*, volume 62, pages 21:1–21:17, Dagstuhl, Germany, 2016.

[2] Danil Annenkov, Paolo Capriotti, and Nicolai Kraus. Two-Level Type Theory and Applications. *ArXiv e-prints*, May 2017. https://arxiv.org/abs/1705.03307.

[1]The code is available at https://github.com/annenkov/two-level.

[2]Boulier has a implemented a similar system in Coq, available at https://github.com/CoqHott/model-structures-Coq. In [2], we compare the approaches using Lean and Coq, and discuss the shortcomings that Agda seems to have here.

[3] Paolo Capriotti. *Models of Type Theory with Strict Equality.* PhD thesis, School of Computer Science, University of Nottingham, Nottingham, UK, 2016.

[4] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The Lean theorem prover. In *Automated Deduction - CADE-25, 25th International Conference on Automated Deduction*, 2015.

[5] Michael Shulman. Univalence for Inverse Diagrams and Homotopy Canonicity. *Mathematical Structures in Computer Science*, pages 1–75, Jan 2015.

[6] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics.* http://homotopytypetheory.org/book/, Institute for Advanced Study, 2013.

[7] Vladimir Voevodsky. A simple type system with two identity types, 2013. Unpublished note.