An Experiment with 2LTT Features in Agda

Elif Uskuplu

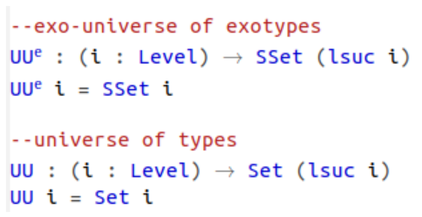
University of Southern California

In this talk, we present a formalization of basics of two level type theory (2LTT) [2] with the scope of the paper "The Univalence Principle" [1] using the proof assistant Agda. We make use of two flags of Agda, `--two-level` and `--cumulativity`, that are still in development.

Although there are some works written in Coq and Lean, there is no commonly accepted library of 2LTT in Agda. This motivates us to work on the project. The other motivation for the formalization is given in the paper "The Univalence Principle" [1]. This study provides a generalization of the univalence axiom from 1-categorical structures to higher categorical ones by using some notions obtained by 2LTT. To date, no attempt has been made to formalize this study in any of the proof assistants implementing 2LTT. We mainly work on the formalization of 2LTT to formalize this study in Agda.

New features of Agda that allows us work with 2LTT

The flag `--two-level` enables a new sort called `SSet`. This provides two distinct universes for us. Currently, Coq and Lean have no such a built-in feature. One stands for the universe of types (like in HoTT), the other stands for the universe of types with strict equalities. In [2], these are called the universe of *inner types* and the universe of *outer types*, respectively. In [1], these are called the universe of *of types* and the *exo*-universe of *exo-types*[1], respectively. We are using the second version. According to Voevodsky's approach in his *Homotopy Type System* [3], types are regarded as fibrant types, but exo-types are not necessarily fibrant. The following Agda code formalizes these two (polymophic) universes.



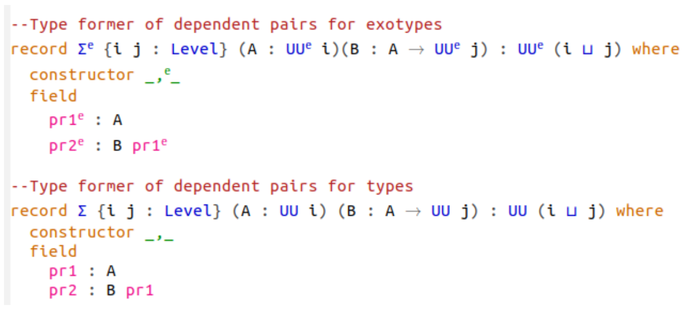
```
--exo-universe of exotypes
UUᵉ : (i : Level) → SSet (lsuc i)
UUᵉ i = SSet i

--universe of types
UU : (i : Level) → Set (lsuc i)
UU i = Set i
```

Following the idea in [2], we define each type former twice; one for types, one for exo-types. Whenever it is needed, we make distinction between them

[1]This term was originally suggested by Ulrik Buchholtz.

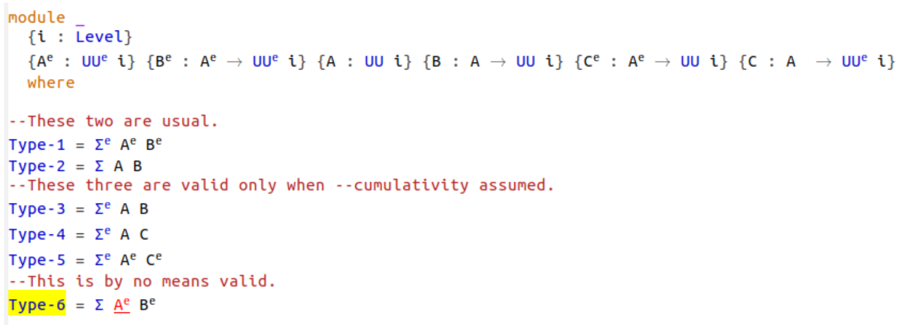
using the superscript $-^e$. The code below is an example of this approach for sigma types. The other basic types and type formers are defined similarly.



```
--Type former of dependent pairs for exotypes
record Σᵉ {i j : Level} (A : UUᵉ i)(B : A → UUᵉ j) : UUᵉ (i ⊔ j) where
  constructor _,ᵉ_
  field
    pr1ᵉ : A
    pr2ᵉ : B pr1ᵉ

--Type former of dependent pairs for types
record Σ {i j : Level} (A : UU i) (B : A → UU j) : UU (i ⊔ j) where
  constructor _,_
  field
    pr1 : A
    pr2 : B pr1
```

As for the relation between types and exo-types, while [2] assumes ”one can convert a type into an exo-type” via an additional conversion operation, the other work [1] assumes directly that every type is an exo-type, but not vice-versa. To formalize this principle, Agda has another new flag, which is `--cumulativity`. This enables the subtyping rule `Set i` $\leq$ `SSet i` for any level `i`. Thanks to the flag we can take types as arguments for the operations/formations which are originally defined for exo-types. The following Agda codes illustrate this idea. If we take an exo-type A^e, a type A, two families of exo-types $B^e : A^e \to \mathcal{U}^e$ and $C : A \to \mathcal{U}^e$, and two families of types $B : A \to \mathcal{U}$ and $C^e : A^e \to \mathcal{U}$, each one gives a suitable sigma type or sigma exo-type. Note that even with the cumulativity, we cannot evaluate a $\underline{\text{type former}}$ at exo-types.



```
module _
  {i : Level}
  {Aᵉ : UUᵉ i} {Bᵉ : Aᵉ → UUᵉ i} {A : UU i} {B : A → UU i} {Cᵉ : Aᵉ → UU i} {C : A  → UUᵉ i}
  where

--These two are usual.
Type-1 = Σᵉ Aᵉ Bᵉ
Type-2 = Σ A B
--These three are valid only when --cumulativity assumed.
Type-3 = Σᵉ A B
Type-4 = Σᵉ A C
Type-5 = Σᵉ Aᵉ Cᵉ
--This is by no means valid.
Type-6 = Σ Aᵉ Bᵉ
```

It’s then natural to ask whether there is a relation between $\sum_A^e B$ and $\sum_A B$. Lemma 2.11 in [2] answers this question. After defining the (exo-)isomorphism for exo-types, which is the analogue of equivalence for types, we show that they are exo-isomorphic, and the same can be shown for (exo-)dependent product and (exo-)unit, but not for (exo-)coproduct, (exo-)natural numbers, and (exo-) empty. (These can be assumed additionally, which is the axiom A1 in [2].) However, these experimental flags do not prevent undesired exo-isomorphisms for now. Using a lifting operation $L : \mathcal{U} \to \mathcal{U}^e$ as $L(A) = A$, one can show that $L(A + B)$ and $L(A) +^e L(B)$ are exo-isomorphic. The issues #5948 and #5761 in Agda Github page address this problem. As Andreas Abel pointed in these discussions, the problem with cumulativity is that the sort of a type is no longer a well-defined concept.

In this talk, we introduce the formalizations of all other type formers with the emphasis on two different equality types. Also, we discuss the pros and cons of using these flags in the formalization. Time permitting, we discuss the additional axioms in [2] and their formalizations. Also, we briefly mention the formalization of Univalence Principle paper in Agda. For the details, one can see the library itself. It's still in progress.

Acknowledgments. This project is conducted under the supervision of Michael Shulman. We also would like to thank Nicolai Kraus for many interesting discussions and insightful comments. Currently, the speaker is partially supported by NSF grant DMS-1902092, the Army Research Office W911NF-20-1-0075, and Simons Foundation.

References

[1] Benedikt Ahrens, Paige Randall North, Michael Shulman, and Dimitris Tsementzis. The Univalence Principle. arXiv preprint, 2021. arXiv:2102.06275

[2] Danil Annenkov, Paolo Capriotti, Nicolai Kraus, Christian Sattler. Two-Level Type Theory and Applications. arXiv preprint, 2019. arXiv:1705.03307

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