Reducing Inductive-Inductive Types to Indexed Inductive Types

Thorsten Altenkirch[1], Ambrus Kaposi[2], András Kovács[2], and Jakob von Raumer[1]

[1] University of Nottingham, United Kingdom
thorsten.altenkirch@nott.ac.uk, jakob@von-raumer.de
[2] Eötvös Loránd University, Budapest, Hungary
{akaposi, kovacsandras}@inf.elte.hu

1 Motivation

Many dependently typed languages which are built on foundations like the calculus of constructions (CoC) provide support for indexed inductive types. These are type families which are inductively defined using constructors that create an instance over arbitrary elements of the base type (the type of $\mathbb{N}$-indexed vectors being a prominent example). Most of these languages, e. g. Coq [2] and Lean [3], don't allow for so called inductive-inductive types in which, for example, the user mutually defines a type A and a family B in which A may appear in the index, and where constructors of A and B may refer to the constructors of each other.

Use cases for inductive-inductive types encompass important applications like the internalization of the syntax of dependent type theory itself ("type theory in type theory" [1]) and the definition of the Cauchy construction of the real numbers [7]. We thus ask the question whether inductive-inductive types can be emulated in a language that only provides indexed inductive types, that is, most importantly, how to construct an appropriate eliminator for these types. The quesion whether this is possible has been brought up in previous studies on inductive-inductive types [6, 4].

2 Approach

We are given a list of *sorts* which we want to define and a list of *constructors*, each with potential references to others. First we define the category of *typed algebras*, of which we aim to construct the initial element. Next, we also define *untyped algebras* which we obtain by erasing all the indices from sorts and constructors. The initial object can be constructed using indexed induction only. To make up for the missing indexing in the untyped algebras, we introduce an inductively defined *well-typedness predicate* which contains the information that a given element of an untyped sort is really indexed by a given index element. The desired initial object of the category of typed algebras is then given by using this predicate to filter for well-typed elements. To prove initiality, we construct an inductive relation between elements of the initial untyped algebra and elements of an arbitrary typed algebra. We then show that this relation is functional.

3 Results & Future Work

The approach has been formalized for the example of a fragment of an inductive-inductive syntax of type theory in Agda, and ported to Lean. We aim to use Lean's meta-language [5] to

automate the construction and provide a user-defined command to create inductive-inductive types.

4 Example

As a first example, we looked at the type Con of *contexts* and the type Ty of *types* in a formalized syntax of a type theory, with a constructor $\mathsf{nil} : \mathsf{Con}$ for an empty context, $\mathsf{ext} : \prod_{\Gamma:\mathsf{Con}} \mathsf{Ty}(\Gamma) \to \mathsf{Con}$ for context extension, $\mathsf{unit} : \prod_{\Gamma:\mathsf{Con}} \mathsf{Ty}(\Gamma)$ for an atomic unit type and $\mathsf{pi} : \prod_{\Gamma:\mathsf{Con},A:\mathsf{Ty}(\Gamma)} \mathsf{Ty}(\mathsf{ext}(\Gamma, A)) \to \mathsf{Ty}(\Gamma)$ for a Π-type. For this example the typed and untyped algebras are represented by the following Lean code:

```
structure CT :=
  (C : Type u)
  (T : C → Type u)
  (nil : C)
  (ext : Π Γ, T Γ → C)
  (unit : Π (Γ : C), T Γ)
  (pi : Π Γ A,
    T (ext Γ A) → T Γ)

structure CT′ :=
  (C : Type u)
  (T : Type u)
  (nil : C)
  (ext : C → T → C)
  (unit : C → T)
  (pi : C → T → T → T)
```

```
inductive S′0 : bool → Type u
| nil : S′0 ff
| ext : S′0 ff → S′0 tt → S′0 ff
| unit : S′0 ff → S′0 tt
| pi : S′0 ff → S′0 tt → S′0 tt → S′0 tt

parameters (M : CT)
def rel_arg : bool → Type u
| ff := M.C
| tt := Σ γ, M.T γ
inductive rel : Π b, S′0 b → rel_arg b → Prop
| nil : rel ff S′0.nil M.nil
| ext : Π Γ A γ a, rel ff Γ γ → rel tt A ⟨γ, a⟩
    → rel ff (S′0.ext Γ A) (M.ext γ a)
| unit : Π Γ γ, rel ff Γ γ → rel tt (S′0.unit Γ) ⟨γ, M.unit γ⟩
| pi : Π Γ A B γ a b, rel ff Γ γ → rel tt A ⟨γ, a⟩ →
    rel tt B ⟨M.ext γ a, b⟩ → rel tt (S′0.pi Γ A B) ⟨γ, M.pi γ a b⟩
```

References

[1] 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.

[2] Bruno Barras, Samuel Boutin, Cristina Cornes, Judicaël Courant, Jean-Christophe Filliatre, Eduardo Gimenez, Hugo Herbelin, Gerard Huet, Cesar Munoz, Chetan Murthy, et al. The Coq proof assistant reference manual: Version 6.1. 1997.

[3] Leonardo Mendonça de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The Lean theorem prover (system description). In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, volume 9195 of *Lecture Notes in Computer Science*, pages 378–388. Springer, 2015.

[4] Gabe Dijkstra. *Quotient inductive-inductive definitions*. PhD thesis, University of Nottingham, 2017.

[5] Gabriel Ebner, Sebastian Ullrich, Jared Roesch, Jeremy Avigad, and Leonardo de Moura. A metaprogramming framework for formal verification. *Proceedings of the ACM on Programming Languages*, 1(ICFP):34, 2017.

[6] Fredrik Nordvall Forsberg. *Inductive-inductive definitions*. PhD thesis, Swansea University, 2013.

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