The Multiverse for a more math-friendly proof assistant

Gaëtan Gilbert, Kenji Maillard, Pierre-Marie Pédrot, Loïc Pujet, Matthieu Sozeau, Nicolas Tabareau, and Éric Tanter

Liber Abaci Kick-Off
Paris, September 20th 2022

What's not math-friendly in Coq?

Quotients are pervasively used in mathematics but hard to represent in (intentional) type theories.

Functional extensionality is an axiom!
Equivalent propositions are not equal?

One must resort to boring compatibility proofs or parametricity to derive basic results, e.g.
(forall x, f x = g x) -> forall l, map f l = map g l.
This requires a specific rewriting tactic to make it transparent to the user.

Equality of co-inductive types is just plain wrong! It should just be bisimilarity.

Quotients

Pervasive in mathematics, easy in set theory, hard in type theories:

- Extensional Type Theory: easy to add quotients, but lose decidable type-checking.

- Cubical Type Theory: radical change of foundation, introducing the concept of dimensions and another algebra of faces to an already complex system. Supports quotients naturally, but with a very different representation of propositions (hProp), not impredicative nor erasable.

Can we do something about it without changing the theory drastically?

Workarounds in Coq

- Mathematical Components
  Restrict to effective quotients, e.g. quotients definable by a total function computing a canonical element for each equivalence class.
  Not a problem for math-comp as it is developing mathematics on finite groups etc… but does not scale to more abstract math settings. Requires some ingenuity to make it work seamlessly for users as well.

- Lean
  Axiomatize quotients, losing metatheoretic properties (either subject reduction or normalization/ decidability). Kind of enforces a classical viewpoint. Could be adapted to Coq but some reticence for a less-than-perfect solution.

Observational Type Theory to the Rescue

Observational Equality Now! Altenkirch, McBride and Swiersta, PLPV 2007

Inspired from the setoid model of Hofmann & Streicher, just like HoTT is inspired by their groupoid model.

Regained a lot of attention recently:

- XTT - Sterling, Angiuli and Gratzer, FSCD 2019
- SetoidTT - Altenkirch, Boulier, Kaposi and Tabareau, MPC 2019
- TT^{obs} - Pujet and Tabareau, POPL 2022

A more “familiar-looking” foundation than cubical type theory, still conjectured to lift to higher dimensions:

Towards Higher Observational Type Theory -
Altenkirch, Kaposi and Schulmann, TYPES 2022

Observational Type Theory

Based on a definitionally proof-irrelevant sort (Prop) and a predicative hierarchy Type(i).

Propositional equality on terms is defined by induction on their type, e.g.:

```
f =_{∀ x : A, B} f' ≡ ∀ x : A, f x =_{B x} f' x
```

(= is propositional equality and ≡ is definitional equality in the following)

As well as their shape, for positive types:

```
0 = 0       ≡ ⊤
0 = S x     ≡ ⊥
S x = 0     ≡ ⊥
S x = S x'  ≡ x = x'
```

=> Definitional injectivity/discrimination and pattern-matching simplification.

Observational Type Theory

This allows to introduce new type constructors with a tailor-made equality. E.g. subsets and quotients:

```
s =_{{x : A | B}} s' ≡ s.1 =_{A} s'.1
```

Quotients `A // R` for the quotient of `A` by an equivalence `R`

(proof terms omitted here)

```
A term constructor:     inj : A -> A // R

An elimination principle:
  Q : A // R -> Type          q : ∀ x : A, Q (inj x)
  qp : ∀ x y : A, R x y -> q x = q y
  ---------
  Qelim Q q qp : ∀ x : A // R, Q x
  Qelim Q q qp (inj x) ≡ q x
And equality: inj x =_{A // R} inj y ≡ R x y
```

TTobs

Pujet and Tabareau prove consistency of the theory, decidability of type checking, and canonicity for computational objects (e.g. natural numbers).

There is ABSOLUTELY NO computation in proofs. TT^{obs} can hence be extended with any consistent propositional axiom and still compute. Even classical ones!

Observational Equality: Now for Good

LOÏC PUJET, Inria, France
NICOLAS TABAREAU, Inria, France

Building on the recent extension of dependent type theory with a universe of definitionally proof-irrelevant types, we introduce TT^{obs}, a new type theory based on the setoidal interpretation of dependent type theory. TT^{obs} equips every type with an identity relation that satisfies function extensionality, propositional extensionality, and definitional uniqueness of identity proofs (UIP). Compared to other existing proposals to enrich dependent type theory with these principles, our theory features a notion of reduction that is normalizing and provides an algorithmic canonicity result, which we formally prove in Agda using the logical relation framework of Abel et al. Our paper thoroughly develops the meta-theoretical properties of TT^{obs}, such as the decidability of the conversion and of the type checking, as well as consistency. We also explain how to extend our theory with quotient types, and we introduce a setoidal version of Swan's Id types that turn it into a proper extension of MLTT with inductive equality.

What's not math-friendly in TTobs?

Quotients are fine
Functional extensionality is by definition
Equivalent propositions are equal
Equality of co-inductive types is plain bisimilarity
Equality is itself proof-irrelevant (no higher-dimensional reasoning)

Caveat indexed inductive types are (rightfully) a bit delicate

So, let's have it?!

Very well, let's implement it!

Not so fast!

Changing a type theory and implementation from the ground up is not easy, let's see what we need:

- A definitionally proof-irrelevant sort
- A new universe hierarchy of setoidal types
- An extension of the system with new primitives and reduction rules

Universes

A universe classifies a collection of types and their closure properties.

Two main examples in Coq:

- Prop: impredicative, i.e. all quantifications allowed
- Type(i): predicative, i.e. raises levels

Ad-hoc rules to interact between them, e.g. singleton elimination from proofs in Prop to Type and the Prop <= Type cumulativity rule.

SProp

SProp is a variant of Prop with definitional proof-irrelevance.

Definitional Proof-Irrelevance without K*

GAËTAN GILBERT, Gallinette Project-Team, Inria, France
JESPER COCKX, Chalmers / Gothenburg University, Sweden
MATTHIEU SOZEAU, Pi.R2 Project-Team, Inria and IRIF, France
NICOLAS TABAREAU, Gallinette Project-Team, Inria, France

Definitional equality—or conversion—for a type theory with a decidable type checking is the simplest tool to prove that two objects are the same, letting the system decide just using computation. Therefore, the more things are equal by conversion, the simpler it is to use a language based on type theory. Proof-irrelevance, stating that any two proofs of the same proposition are equal, is a possible way to extend conversion to make a type theory more powerful. However, this new power comes at a price if we integrate it naively, either by making type checking undecidable or by realizing new axioms—such as uniqueness of identity proofs (UIP)—that are incompatible with other extensions, such as univalence. In this paper, taking inspiration from homotopy type theory, we propose a general way to extend a type theory with definitional proof irrelevance, in a way that keeps type checking decidable and is compatible with univalence. We provide a new criterion to decide whether a proposition can be eliminated over a type (correcting and improving the so-called singleton elimination of Coq) by using techniques coming from recent development on dependent pattern matching without UIP. We show the generality of our approach by providing implementations for both Coq and Agda, both of which are planned to be integrated in future versions of those proof assistants.

Adding that new sort to Coq was non-trivial (done by Gilbert)

- SProp: impredicative, erased, no singleton elimination, only empty elimination, restricted inductive types
- SProp(i) (in Agda): predicative variant

Again, ad-hoc rules of interaction between Prop and Type.

Ad-hoc implementations

Coq was not designed to deal with multiple different hierarchies from the beginning. High cost of integration payed for SProp.

This shows at the user level:

- Type inference prefers predicativity by default, e.g.:

```
forall P, P -> P \/ P
```

```
In environment
P : Type
The term "P" has type "Type" while it is expected to have type
"Prop" (universe inconsistency: Cannot enforce issue498.2 <= Prop).
```

At the theoretical level as well…

Ad-hoc theory, less ad-hoc

There is no general theory or framework for combining two or more universes with their specific rules, i.e. for modular consistency proofs.

Can we do better? We hope so!

The Multiverse: Logical Modularity for Proof Assistants

KENJI MAILLARD, Gallinette Project-Team, Inria, France
NICOLAS MARGULIES, ENS Paris-Saclay & Gallinette Project-Team, Inria, France
MATTHIEU SOZEAU, Gallinette Project-Team, Inria, France
NICOLAS TABAREAU, Gallinette Project-Team, Inria, France
ÉRIC TANTER, PLEIAD Lab, Computer Science Department (DCC), University of Chile, Chile

Mainly geared towards having the flexibility of adding new sorts to a system along with constants and reduction rules and getting modular canonicity and decidability of type checking proofs (using state of the art logical relation techniques).

The Multiverse

MuTT scales to adding Prop, SProp, and the exceptional type theory of Pédrot and Tabareau (which has 3 different universe hierarchies). Its implementation in Coq can reuse the groundwork of Gilbert.

A basis for new experiments, in particular:
- Implementing and verifying the meta theory of TT^{obs} together with other sorts.
- A special CProp with “classical” quotients à la Lean
- Using the Keller-Lasson sort hierarchy for parametricity and better erasability control (erasure of natural number indices for example).

MuTT allows to specify cleanly the possible communication between universe hierarchies, if available.

Sort polymorphism

Once we have a theory with lots of different sorts available, we will want polymorphic quantification over them. Justified by targeting MuTT.

It seems like a relatively straightforward generalisation of universe polymorphism, with new "sort" and elimination constraints rather than just "universe level" constraints. We have focused on the target MuTT for now.

When implemented right, will give the right answer to

```
forall P, P -> P \/ P
```

And whatever new sort we might want to add, have a single:

```
map : forall (U : Univ) (A B : U) (f : A -> B)
  (l : list A), list B.
```

Work plan

Multiverse Type Theory

(w/ MetaCoq and Agda proofs, Coq prototype)

Targets

Instance

Sort polymorphism

(w/ Coq prototype)

TT^{obs} in Coq

As an alternative, cooperating universe