A Restricted Version of Reflection Compatible with Univalent Homotopy Type Theory

Théo Winterhalter

A Restricted Version of Reflection Compatible with Univalent Homotopy Type Theory

Théo Winterhalter,
supervised by Andrej Bauer and Matthieu Sozeau

31 July, 2017

Abstract

We present our work conducted on the relation between the different notions of equality in type theory, particularly in the setting of homotopy type theory. We offer a novel notion of restricted reflection that is still consistent in a univalent setting while allowing us to derive useful applications in the field of interactive proving.

Acknowledgements

Although I do not mention them in the header, I would like to mention Philipp Haselwarter and Nicolas Tabareau that also worked with me on this topic, as well as Gaëtan Gilbert and Peter LeFanu Lumsdaine that I interacted with on these questions. I'm also grateful for all the feedback we got from people attending both EUtypes SSTT meeting and TYPES 2017 where we talked about our formalisation.

Introduction

When we want to make sure our mathematical developments hold formally, we can rely on the power of proof assistants such as Coq [10] and Agda [18]. These allow users to write their proofs in a formal language so that the machine can check derivability of such proofs. As useful as it might be, these tools can sometimes be annoying to use as they force the user to go into extreme details.

The job of the people working on these tools is—in part—to make them easier to use and ever stronger. One such way is to allow an intuitive use—meaning as easy as in informal blackboard mathematics—of the notion of equality that is often required in mathematical reasonings. This is however not a trivial problem,

and many notions of equality have been (and still are being) investigated in type theory (the logical foundation of the proof assistant).

Another part of this work is to make sure that the type theories we use actually make sense (e.g. are consistent). This can in turn be proven using the very tools we develop[1]. Part of my focus during this internship has been dedicated to the developpement of tools in the Coq proof assistant to prove meta-theorems of type theory. They are already available on github [4] and have been the subject of two talks given in Ljubljana and in Budapest.

Our main story will be about the reflection rule (introduced by Martin-Löf) that is used to bridge the gap between the two main notions of equality in type theory, and specifically about a restriction that is compatible with the homotopy type theory (HoTT) and the univalence axiom [21].

1 Context and motivations

1.1 Type theory

We will consider several type theories (variants of Martin-Löf type theory [16] and of the Calculus of Constructions [11], that are arguably at the core of respectively Agda and Coq). The definition of what it means to be "a type theory" is still a bit unclear formally, but informally this corresponds to giving a syntax of expressions and judgments relating them via a set of rules describing how to derive said judgments.

We will have the notions of terms $(u, v, \dots)$ and types $(A, B, \dots)$ together with the notion of a term being at some type $(u : A)$. This is in general coming with a list of assumptions, making up a context $(\Gamma, \Delta, \dots)$ and such that we can actually derive judgments of the form $\Gamma \vdash u : A$.

Without really introducing a general notion of type theory, we would like to point out that there is—as of yet—no unique notion of type theory, and we would rather talk about a particular collection of type theories. With that in mind, it still makes sense to consider the relation between several type theories, and translations from one to another (which we will explain in more detail in section 1.3).

[1] There is an inherent limitation in that, coming from Gödel's incompleteness theorems, but we are still able to formalise smaller type thoeries within Coq.

1.2 Equalities in type theory

1.2.1 Judgmental equality

Equality in type theory comes in different flavors. One of them is the judgmental equality that appear in the typing derivations. It is an extension of the familiar β rule from λ-calculus. This is the equality that is used in *type conversion* :

$$\frac{\Gamma \vdash u : A \qquad \Gamma \vdash A \equiv B}{\Gamma \vdash u : B}$$

This is the illustration of the principle that we can replace equals by equals . One rule for deriving judgemental equality would be the β rule, for instance:

$$\frac{\Gamma, x : A \vdash u : B \qquad \Gamma \vdash v : A}{\Gamma \vdash (\lambda x : A.u)\ v \equiv u[x \leftarrow v] : B[x \leftarrow v]}$$

where substitution $u[x \leftarrow v]$ stands for the term u where all occurrences of variable x were replaced by v (and same goes for $B[x \leftarrow v]$). This includes all the congurence rules (in particular, this allows us to use reduction under context).

In fact, in the sense that it is not a first-class object of the theory, the equality resides on the meta-level: during a coq proof, it is not possible to refer to it directly. We might still want to be able to talk about equalities of objects , and if possible replace equals for equals—as is possible with the judgmental equality—within the type theory.

1.2.2 Identity types

The widespread solution to that problem is to use what is usually called identity types $\mathsf{Id}_A(u, v)$. They are formed using—a variant of—the following rule.

$$\frac{\Gamma \vdash A\ \mathsf{type} \qquad \Gamma \vdash u : A \qquad \Gamma \vdash v : A}{\Gamma \vdash \mathsf{Id}_A(u, v)\ \mathsf{type}}$$

The underlying idea is that this type corresponds to the type of equalities between u and v of type A. The only constructor this relation has is *reflexivity*.

$$\frac{\Gamma \vdash u : A}{\Gamma \vdash \mathsf{refl}_A\ u : \mathsf{Id}_A(u, u)}$$

This could be interpreted as the identity type being the smallest reflexive relation. The trick is that, the typing of $\mathsf{refl}_A\ u$ will rely on the conversion of the type theory and then hold up to judgmental equality. For instance, we can witness the β-equality we mentioned earlier by deriving:

$$\Gamma \vdash \mathsf{refl}\ u[x \leftarrow v] : \mathsf{Id}_{B[x \leftarrow v]}((\lambda x : A.u)\ v, u[x \leftarrow v]).$$

Of course, proving equalities can be useful sometimes, but it is even better if we can make something out of them. We are faced with several options.

Reflection. Perhaps the thing we would like the most about equality is to be able to freely identify any two equal things. This can be achieved by using what is called "reflection" of equality:

$$\frac{\Gamma \vdash p : \mathsf{Id}_A(u, v)}{\Gamma \vdash u \equiv v : A}$$

The idea here is that every provable propositional equality (i.e. every equality that is provable from within the system) can be promoted to an equality at the level of judgments. This means that we identify the two notions of equality that we have at hand.

The addition of that rule yields extensional type theory, as opposed to intensional type theory and usually means that we lose the property of decidability of type checking (which is why it is no longer considered by some people). This doesn't prevent us from using this as a base for a proof assistant: Andromeda [2] and NuPrl [9] both rely on extensional type theories and semi-decidable type checking: sometimes the user has to provide hints to the proof assistant about the validity of some statements.

Inductive types elimination principle. The standard way to have propositional equality in Coq would be to consider the identity type as an inductive type. The elimination principle for identity types is called J. It informally corresponds to assuming one equality to be reflexivity in a given statement (think pattern-matching).

Streicher's axiom K. Alternatively, one could consider a different eliminator for the identity type. K—introduced by Streicher [20]—says that, for any type A and term $x : A$ and any predicate C over an equality $p : \mathsf{Id}_A(x, x)$, if we can prove $C\ (\mathsf{refl}_A\ x)$, then $C\ q$ holds for any $q : \mathsf{Id}_A(x, x)$.

This is complementary to J and adds the (otherwise not provable [14]) property that the only proof of $\mathsf{Id}_A(u,u)$ is itself the reflexivity $\mathsf{refl}_A\ u$, or, equivalently, that equalities are proof-irrelevant: called uniqueness of identity proofs (UIP for short), this property states that any two proofs p, q of $\mathsf{Id}_A(u,v)$ are themselves equal (in the sense of an inhabitant of $\mathsf{Id}(p,q)$).

Univalent equality. Axiom K is not compatible with the univalence axiom of homotopy type theory [21] as univalence allows two distinct equalities between $Bool$ and $Bool$.

Indeed, univalence states that the trivial map from $\mathsf{Id}(A,B)$ to the type of equivalences between A and B is itself an equalivalence, meaning that we can turn any equivalence into an equality. In the case of $\mathsf{Bool} = \mathsf{Bool}$, we have two equalities, one derived from the identity, and the other derived from the negation on booleans (which is an equivalence, and thus by univalence an equality). Homotopy type theory is about the study of the identity types and their structure (including equalities between equalities and so forth).

The incompatibility between K and univalence means that we might consider not having K in a development. For Agda, removing the reliance on K to prepare for HoTT took some work [7].

1.3 Translations between type theories

Proving consistency of equality reflection is a real concern and has already been achieved under restrictive axioms [19, 13]. There are two main ways of proving consistency: one is providing a model of the type theory we consider, and one is to provide a translation from the type theory we consider to another one that we "trust" (e.g. that we already know to be consistent). In the second case, we are also providing a model that is a syntactical model.

The idea of a translation is that we map judgments in the source type theory to judgments in the target type theory while preserving derivations and falsehood: the point is then that if falsehood is derivable in the source then it is as well in the target, making the source consistent relatively to the consistency of the target.

We can see several advantages to the translation way as it usually amounts to translating an extra feature that has been added to a trusted core (e.g. reflection of equality), meaning that we could use a constructive proof of the translation to extend the theory of a proof assistant, as well as usually being more robust, as scaling (extending the source and target with the same things) is *easier* (this is of course very informal). Educative examples of translations can be found in [5] where they are used to provide countermodels of various extensionality principles such as functional extensionality (two functions that are pointwise equal are

equal).

2 Restricting the notion of reflection

2.1 Reflection and univalence

As we have seen, many options exist when it comes to equality. There is one more thing we need to mention about them: univalence and reflection don't go well together. Indeed, if we consider a system with J in which reflection holds, and take $p : u = v$ (we will write $u = v$ for $\mathsf{Id}(u, v)$), then we have $u \equiv v$ by reflection, meaning we can now derive $\mathsf{refl}\ u : u = v$, and thus define the type $p = \mathsf{refl}\ u$. Now, if we take $C\ y\ e := e = \mathsf{refl}\ u$ (which is well-typed under the assumption $e : u = y$ thanks to the reflection rule), we can inhabit $C\ u\ (\mathsf{refl}\ u)$ which is convertible to $\mathsf{refl}\ u = \mathsf{refl}\ u$ by reflexivity, and then, with J, we get a proof of $C\ v\ p$ which is convertible to the $p = \mathsf{refl}\ u$ we wanted. This means that from reflection and J, we can derive K.

So even when we are not bothered by the undecidable aspect of extensional type theory, we might still not be able to use it for it is inconsistent with the univalence axiom.

2.2 Naive restriction

As we just showed, reflection on some type A and J imply UIP on A, something we don't want (assuming we want reflection with univalence, if not then Oury's translation [19] already tells us it is consistent to have reflection). However, the first idea we might have would be to restrict reflection to the types for which UIP already holds.

In homotopy type theory [21], types for which UIP holds are called hSets (for homotopy sets), the idea being that there is no specific structure on the equality: two elements of a set are equal or not and can't be equal in different ways. Of course, one might wonder what types do verify this condition. We know from Hedberg's theorem [12] that any type A with a decidable equality (in Coq, this would be a proof of the type `forall (x y : A), { x = y } + { x <> y }`, meaning there is a Coq function that tells us whether two elements of A are equal) satisfies UIP, i.e. is an hSet. This includes a great class of types one might consider such as the booleans or the natural numbers.

In HoTT, we also have the notion of hProp (homotopy proposition) that is types for which all elements are equal (bascially meaning it has only one element—equivalent to `True`—or none—equivalent to `False`) which is a subclass of hSets, as well as the notion of contractible types which are inhabited hProps (given by an

element and a proof that any other element is equal to it). Unfortunately, restricting reflection, even to contractible types is already incompatible with univalence.

Indeed, if we consider any type A with $a : A$, the type $\Sigma(x : A), a = x$, often called the singleton a (as it corresponds to the elements equal to a), is itself contractible: it is inhabited by $(a, \mathsf{refl}_A\ a)$ and if we take $u : \Sigma(x : A), a = x$, then we have $u.2 : a = u.1$, while the equality between $\mathsf{refl}_A\ a$ and $u.2$ (which aren't at the same type) hold up to the equality $u.2$ by reflexivity. If we assume we have $b : A$ and $p : a = b$, then $(b, p) : \Sigma(x : A), a = x$, and as we previously said, $(a, \mathsf{refl}\ a) = (b, p)$ which yields $(a, \mathsf{refl}\ a) \equiv (b, p)$ by reflection on contractible types. We can derive that the first projections are also convertible: $a \equiv b$. If we use reflection, then we must have provided two inhabitants of the type we use reflection on, so from reflection on contractible types we hence derived reflection on all types. This means we need to find a better restriction.

2.3 A syntactic criterion

We know reflection must be limited to only types that are hSets, but that this isn't enough of a restriction. We could not find any natural semantic restriction and instead seek a more syntactic approach. Our first idea was that we should be able to use reflection on at least the booleans and the natural numbers, otherwise we wouldn't imagine any type for which we could.

Our first line of attack was to produce a translation in the same fashion as Oury [19], where instead of using heterogenous equality (equality of terms over equality of types) we would use equalities over equalities of booleans, this way we could avoid using UIP (on `Type`) that is necessary to derive $(A, u) = (A, v) \to u = v$. However, the boolean equalities we consider may depend on variables in the context which makes them difficult to express after the application of an abstraction rule for instance: from $\Gamma, x : A \vdash u : B$ we derive $\Gamma \vdash \lambda(x : A).u : \Pi(x : A).B$; if the equality mentions x, there is no way to talk about it in Γ.

With our current approach, we can actually achieve reflection on a greater class of types, including `bool` and `nat`. We will develop this more once we sketch the proof in section 3.2 as this fact will arise naturally.

2.4 Reflection in practice

A legitimate question to ask is, *what for?* Indeed, is there still any appeal to such a restricted version of extensional type theory?

We claim there is a use, even for just reflection on `bool`! Let's take an example (in Coq syntax) and consider the dependent type `pnat : bool -> Type` of

natural numbers with their parity, such that `pnat true` represents *even* natural numbers, and `pnat false` the *odd* ones.

```
Inductive pnat (b : bool) : Type :=
| pz : pnat true
| ps : forall {b} (n : pnat b), pnat (not b).
```

Then we might want to have a generic notion of addition on them (where the function `eqb : bool -> bool -> bool` is the boolean equality).

```
Fixpoint add {b} (n : pnat b) {c} (m : pnat c) : pnat (eqb b c) :=
  match n with
  | pz   => m
  | ps n => ps (add n m)
  end.
```

Although this might seem to be the natural definition, this wouldn't type-check in Coq. First, `m : pnat c` where is expected `pnat (eqb true c)`. And then we have `ps (add n m) : pnat (not (eqb b c))` while the type it should have is `pnat (eqb (not b) c)`.

In a system with reflection on `bool` however, this is derivable as the equalities `(eqb true c) = c` and `not (eqb b c) = (eqb (not b) c)` hold propositionally (they are provable by simple case analysis—or pattern-matching).

The reflection on `nat` would yield more natural examples with for instance the vectors (list indexed by their length): we want `vec A (n+m)` to coincide with `vec A (m+n)` definitionally.

3 A model for the restricted reflection

3.1 Homotopy Type System

Because we failed to interpret our extended theory directly into Coq, we choose to give a model in a stronger theory. Introduced by Voevodsky, homotopy type system or HTS [22] is a type theory featuring two equality types, the usual intensional one of HoTT, and the extensional one that enjoys reflection. More formally, we have a global type theory of pretypes equipped with a strict equality $\overset{s}{=}$ that is an identity type with UIP and reflection, but we also have a subclass of (pre)types that are called fibrant types and that are there to interpret a univalent theory; the fibrant equality $=$ is the identity type of HoTT and has univalence. Both equalities feature the functional extensionality principle we already mentioned.

Basically fibrant types are structures that we consider only up to equivalence so that univalence can only be used on them. For instance, we can prove $x \overset{s}{=} y \to x = y$ as (pre)types eliminate to any type, but the converse $x = y \to x \overset{s}{=} y$ cannot generally be proven in HTS because fibrant types only eliminate to fibrant types. This last statement can be relaxed a little for some specific types, we say we have a strong theory when types such as `False`, **`bool`** and **`nat`** are fibrant while also having unrestricted elimination to any type.

3.2 The criterion in HTS

The idea behind the proof is to rely on the already built-in reflection of HTS to interpret our own reflection. Since we want to justify a theory compatible with HoTT, we want to translate all our types to fibrant types, in particular we want to send the identity type of the source to the fibrant equality of the target. As we mentioned before, the fibrant equality does not feature reflection (it wouldn't be consistent), however, in some cases, we have $x = y \to x \overset{s}{=} y$ which is sufficient to have reflection.

Consider a type A for which $\Pi(x\ y : A), x = y \to x \overset{s}{=} y$ holds; if we have $u =_A v$ then we have $u \overset{s}{=}_A v$, which in turn implies $u \equiv v$ by reflection. As we said, this property isn't true in general, however if we take Bool for instance, assuming it is strong and eliminates to any type, we can first make a case distinction on u and $v : \mathsf{Bool}$ and reduce the problem to proving the four statements: $\mathsf{true} = \mathsf{true} \to \mathsf{true} \overset{s}{=} \mathsf{true}$ (which holds by reflexivity), the same for false, and the two absurd cases where one assumes $\mathsf{true} = \mathsf{false}$ or $\mathsf{false} = \mathsf{true}$ which can be eliminated to the empty type by large elimination[2], and assuming it is strong as well, anything follows, including $\mathsf{true} \overset{s}{=} \mathsf{false}$ and $\mathsf{false} \overset{s}{=} \mathsf{true}$.

The same can be proved for the type of natural numbers by induction (case distinction isn't sufficient to deal with the successor case).

Generalising to arrows. Just accepting a list of examples would be a bother. We can actually derive this property for a class of types. First, since we have functional extensionality, if we consider a type R in the class, and any type A, then the type $A \to R$ is also in the class. Indeed, assume $f, g : A \to R$ and $f = g$, we want to show $f \overset{s}{=} g$. Now, by functional extensionality, we only need to prove for any $x : A$ that $f\ x \overset{s}{=} g\ x$. By using the fact that R enjoys the property we only need to prove $f\ x = g\ x$ which holds since $f = g$.

[2] This generally requires the notion of univeses, but we have them in Coq and HTS

The case of inductive types. We actually could also accept some inductive types depending on their shape. We only require their constructors to be of a given form in order to apply induction as we would for Nat. This is because, by discrimination, we only have to consider the cases where the two compared terms have the same constructor. For instance, if the constructor only has variables that appear linearly in the index of the inductive type, then it fits our requirements. This is in the same fashion as in Cockx's work [7, 6]; this works, because these variable are *forced* by dependent pattern-matching and will thus be syntactically equal on both sides of the equation. We can further extend this principle by requiring that any variable in the constructor that doesn't appear has a type that verifies the criterion (because then we use injectivity of constructors and the property on them to recover the wanted equality), and the same for variables that appear several times. This will be discussed in Gilbert's PhD thesis on definitional proof-irrelevance for propositions.

Completeness of the criterion. Unfortunately, this criterion cannot be complete if we want it to stay decidable[3]. Indeed, if we were to take A a type whose inhabitedness is undecidable and B a type that does not have the property (as a reminder $\Pi(x\ y : B), x = y \to x \overset{s}{=} y$), then we can construct the following Coq inductive type:

```
Inductive T : Type :=
| c : (A -> T) -> B -> T.
```

Now, if A is inhabited, T is actually empty, as inhabiting T can only be achieved by inhabiting A -> T and thus T. If not, then T is equivalent to B.

In the first case, T has the property, while in the second one, it doesn't. It is thus undecidable to know if a given type satisfies the property. In our case, we might still argue that we don't have decidability to begin with, but this property cannot be proven from within Coq either, while the decidability of type checking can be fixed by adding hints for the type checker when it needs to use the reflection rule.

3.3 Idea of the proof

In the following we shall assume we are only dealing with Nat and Bool for simplicity of the argument, but we keep in mind that we could devise a decidable criterion (and even refine it) to include many more cases.

[3] This remark is also due to Gaëtan Gilbert.

The first step of giving a model of our type theory with restricted reflection is to translate the terms and types into their fibrant counterpart in HTS. We will write $[\![e]\!]$ for the translation of an expression e. As we already said in section 1.3 what we want, to achieve the translation, is to prove that from $\Gamma \vdash u : A$ we can derive $[\![\Gamma]\!] \vdash [\![u]\!] : [\![A]\!]$ in HTS, as well as $\vdash \_ : [\![\bot_{\mathsf{S}}]\!] \to \bot_{\mathsf{HTS}}$ (where the subscript S and HTS repspectively denote expressions from the source theory and HTS, and where $\bot$ is the empty type). One thing to notice, is even though we don't feature univalence in the source, we give it a univalent model, meaning that we could add the axiom safely to the theory (this what we mean by *compatible* with univalence).

In order to prove $[\![\Gamma]\!] \vdash [\![u]\!] : [\![A]\!]$, one also needs to translate the other judgments. In particular, one has to be able to translate conversion, namely going from $\Gamma \vdash u \equiv v : A$ to $[\![\Gamma]\!] \vdash [\![u]\!] \equiv [\![v]\!] : [\![A]\!]$. Indeed $[\![\_]\!]$ on terms and types merely translates to the fibrant counterpart (and lifts pointwise to contexts), this means that we must preserve conversion. The only tricky part is the preservation of the reflection rule:

$$\frac{\Gamma \vdash p : u =_A v}{\Gamma \vdash u \equiv v : A}$$

So we have by induction hypothesis $[\![\Gamma]\!] \vdash [\![p]\!] : [\![u =_A v]\!]$ and we need to show $[\![\Gamma]\!] \vdash [\![u]\!] \equiv [\![v]\!] : [\![A]\!]$. There, we need to remember that the identity type is translated to fibrant equality (we avoid subscripts when there is no ambiguity): $[\![\Gamma]\!] \vdash [\![p]\!] : [\![u]\!] =_{[\![A]\!]} [\![v]\!]$. Now, in HTS, for $[\![A]\!]$ we have the property evidenced by some $\varphi_{[\![A]\!]} : \Pi(x\ y : [\![A]\!]),\ x =_{[\![A]\!]} y \to x \overset{s}{=}_{[\![A]\!]} y$. Then we derive

$$\frac{\dfrac{[\![\Gamma]\!] \vdash [\![p]\!] : [\![u]\!] =_{[\![A]\!]} [\![v]\!]}{[\![\Gamma]\!] \vdash \varphi_{[\![A]\!]}\ [\![p]\!] : [\![u]\!] \overset{s}{=}_{[\![A]\!]} [\![v]\!]}}{[\![\Gamma]\!] \vdash [\![u]\!] \equiv [\![v]\!] : [\![A]\!]}$$

using the reflection on the strict equality.

Theorem 1 (Translation into HTS)**.**

- *If* Γ ctx *then* $[\![\Gamma]\!]$ ctx*;*
- *If* $\Gamma \vdash A$ type *then* $[\![\Gamma]\!] \vdash [\![A]\!]$ type*;*
- *If* $\Gamma \vdash u : A$ *then* $[\![\Gamma]\!] \vdash [\![u]\!] : [\![A]\!]$*;*
- *If* $\Gamma \equiv \Delta$ *then* $[\![\Gamma]\!] \equiv [\![\Delta]\!]$*;*
- *If* $\Gamma \vdash A \equiv B$ *then* $[\![\Gamma]\!] \vdash [\![A]\!] \equiv [\![B]\!]$*;*

- *If* $\Gamma \vdash u \equiv v : A$ *then* $[\![\Gamma]\!] \vdash [\![u]\!] \equiv [\![v]\!] : [\![A]\!]$.

Proof. We prove the statements mutually by induction as shown above. □

Corollary 2 (Preservation of falsehood)**.** *If* $\bot_{\mathsf{S}}$ *is inhabited, then so is* $\bot_{\mathsf{HTS}}$.

Proof. Assume $\vdash u : \bot$, then, by theorem 1, $\vdash [\![u]\!] : [\![\bot]\!]$. By definition of our translation we have $[\![\bot_{\mathsf{S}}]\!] := \bot_{\mathsf{HTS}}$ and thus $\vdash [\![u]\!] : \bot$. □

Corollary 3 (Consistency)**.** *It is consistent to assume reflection for fibrant* Nat *and* Bool *as well as univalence.*

Now, the reason this proof works is because in HTS there is a seperation of concerns, we don't have to deal with higher equivalences because the strict equality is already there to be able to talk about the judgmental equality internally. Although we provide a model, this might seem a bit like cheating because we translate reflection to reflection. In order to complete the picture we will address that point in the next section.

4 Eliminating reflection from HTS

Even when it does not get in the way of univalence, reflection is not always desirable. Altenkirch et al [1] proposed their own variant of HTS without the reflection rule for equality but merely UIP.

In this section we adapt the translation of Oury [19] from extensional type theory to intensional type theory to the case of HTS to its two-level variant without reflection. In the original translation, several axioms are needed: UIP, functional extensionality and a third one that we will detail later. The two former axioms are already verified by the target's strict equality which doesn't make them a restriction that could harm univalence. For the latter we propose a way to avoid its need.

4.1 Principle of Oury's original translation

Translation of the reflection rule. This time there is no reflection available in the target, so we need to give a meaning to the reflection rule. The idea is that we do not translate conversion directly to conversion but instead we translate it to propositional equality. In the case of our two-level type theory, it would be $\overset{s}{=}$.

What we will use to translate conversion will then be what is called transport. Given $p : A \overset{s}{=} B$ we can build $p_* : A \to B$, the transport induced by p, using the

J_s eliminator. Conversion gets then interpreted to

$$\frac{\Gamma \vdash u : A \qquad \Gamma \vdash p : A \overset{s}{=} B}{\Gamma \vdash p_*\ u : B}.$$

This is actually not so simple as the terms and types involved are also translations and can already be labeled by transports in the conversion derivation. We switch from homegenous equality to heterogenous equality in order to deal with these cases.

Heterogenous equality. Even in laying out the grounds for the original translation, we take the liberty to adapt it a little to the language of homotopy type theory. Instead of relying on John Major equality [17] as heterogenous equality, we will use equalities between pointed types.

John Major equality is an inductive type defined in Coq as follows.

```
Inductive JMeq {A} (a : A) : forall {B}, B -> Type :=
| jm_refl : JMeq a a.
```

It is heterogenous in the sense that it allows to talk about equality of objects that don't even have the same type definitionally. Of course, to inhabit it (in the empty context) one needs to provide two objects of the same type that are actually convertible (like is the case for the identity type). The problem is that one cannot derive `forall {A} (x y : A), JMeq x y -> x = y` without UIP, and that fact isn't very clear to the naked eye.

We will switch to the language of HoTT instead to make things clearer, but this will be an equivalent notion. We rely on pointed types: $\Sigma(A : \mathcal{U}),\ A$ i.e. a type together with an inhabitant. Heterogenous equality is then simply interpreted as an equality between pointed types $(A, a) = (B, b)$. This definition gives us insight because $(A, a) = (B, b)$ is equivalent to having $p : A = B$ and $p_*\ a = b$. Then, if we are able to prove that $p = \mathsf{refl}\ \_$ we can deduce $a = b$. This is the case for types $((\mathcal{U}, A) = (\mathcal{U}, B) \to A = B)$ in the presence of UIP. We will take $\overset{s}{=}$ and not $=$ to be able to rely on that important fact.

Translation of derivations. Another important point to note is that, because the translation now depends on the derivation conversion, we are no longer translating expressions and then showing typing is preserved but rather directly translating typing derivations themselves. This raises the problem of coherence, how do two translations of the same expression but with different derivations relate?

In order to deal with that we define a syntactic relation between a term of the source and a term of the target, written $t \lhd t'$ this states that t (of the source) is a shape of t' (of the target), meaning t' is basically t only possibly labeled by transports.

We also introduce another relation, this time between expressions of the target and written $t_1 \bowtie t_2$ stating that two terms t_1 and t_2 have the same shape. This could be defined as the existence of a term t in the source such that $t_1 \rhd t \lhd t_2$, but we merely want this to imply $t_1 \bowtie t_2$ and rather define it inductively to reason about it more easily.

$$\frac{t_1 \bowtie t_2}{p_*\ t_1 \bowtie t_2} \qquad \frac{t_1 \bowtie t_2}{t_1 \bowtie p_*\ t_2} \qquad \frac{t_1 \bowtie t_2 \qquad u_1 \bowtie u_2}{t_1\ u_1 \bowtie t_2\ u_2} \qquad \dots$$

The idea is that every translated term admits its original term as a shape (which will also help preserve falsehood). Together with the following lemma, this property allows to make up for the coherence problem created by the fact we translate derivations.

Lemma 4 (Coherence of translation)**.** *If* $\Gamma \vdash u : A$ *and* $\Gamma \vdash v : A$ *and* $u \bowtie v$ *then there exists* p *such that* $\Gamma \vdash p : u \overset{s}{=} v$.

Proof. In order to prove this statement, we need to generalise it to heterogenous equality (so strict equality of pointed types). Then we reason by induction on the $\bowtie$ relation. Let's look at a few examples:

- $\Gamma \vdash p_*\ t_1 : T_1$ and $\Gamma \vdash t_2 : T_2$ and

  $$\frac{t_1 \bowtie t_2}{p_*\ t_1 \bowtie t_2}$$

  By inversion of typing we have $\Gamma \vdash p : T_1' \overset{s}{=} T_1$ and $\Gamma \vdash t_1 : T_1'$. Thus, by induction hypothesis, we have $\Gamma \vdash e : (T_1', t_1) \overset{s}{=} (T_2, t_2)$. Now, we need to remark that we have[4] $\Gamma \vdash (p, \mathsf{refl}\ \_) : (T_1', T_1) \overset{s}{=} (T_1, p_*\ t_1)$ in order to conclude.

- $\Gamma \vdash t_1\ u_1 : T_1$ and $\Gamma \vdash t_2\ u_2 : T_2$ and

  $$\frac{t_1 \bowtie t_2 \qquad u_1 \bowtie u_2}{t_1\ u_1 \bowtie t_2\ u_2}$$

[4] We don't strictly have that the equality of Σ is the Σ of the equalities, but this is equivalent and we use that implicitely, it does not matter as strict equalities are irrelevant.

This case is actually not provable without an axiom saying that when we apply equal functions to equal arguments, it yields equal results. This is the extra axiom that we mentioned earlier and that still doesn't hold in a two-level type theory. It is however satisfied in the model. We will detail our approach to avoid using an axiom in section 4.3.

- $\Gamma \vdash \lambda(x : A_1).\ t_1 : T_1$ and $\Gamma \vdash \lambda(x : A_2).\ t_2 : T_2$ and

$$\frac{A_1 \bowtie A_2 \qquad t_1 \bowtie t_2}{\lambda(x : A_1).\ t_1 \bowtie \lambda(x : A_2).\ t_2}$$

This time there is a little trick of comparing $\lambda(x : A_1).\ t_1$ with α-renamed $\lambda(y : A_2).\ t_2[x \leftarrow y]$ instead in order to use the extended context $\Gamma, x : A_1, y : A_2, e : (A_1, x) \overset{s}{=} (A_2, y)$ in the induction hypothesis. The rest is relatively straightforward.

□

Now that we have evidenced the connection between translations of the same expression, we can proceed with the translation itself.

Translation. The translation works by induction over the derivation. We remark that, thanks to lemma 4, we can always assume that types are given with the same head constructor as the original expression, meaning a function is translated to a function and so on. We will now state the theorem but not detail the proof here, most of the work has already been done.

As we are translating derivations, we can no longer talk about $[\![e]\!]$ as *the* translation of expression e, however, we will use notation $\overline{\mathcal{J}} \in [\![\mathcal{J}]\!]$ to state that judgment $\overline{\mathcal{J}}$ is *a* transltion of judgment $\mathcal{J}$. This means that $\mathcal{J} \triangleleft \overline{\mathcal{J}}$ and that $\overline{\mathcal{J}}$ is well-formed.

Theorem 5 (Elimination of reflection)**.**

- *If* Γ ctx *then there exists* $\overline{\Gamma}$ ctx $\in [\![\Gamma \text{ ctx}]\!]$.
- *If* $\Gamma \vdash A$ type*, then for any* $\overline{\Gamma} \in [\![\Gamma]\!]$ *we have some* $\overline{\Gamma} \vdash \overline{A}$ type $\in [\![\Gamma \vdash A \text{ type}]\!]$.
- *If* $\Gamma \vdash u : A$ *then for any* $\overline{\Gamma} \in [\![\Gamma]\!]$ *we have some* $\overline{\Gamma} \vdash \overline{u} : \overline{A} \in [\![\Gamma \vdash u : A]\!]$.
- *If* $\Gamma \vdash u \equiv v : A$ *then for any* $\overline{\Gamma} \in [\![\Gamma]\!]$ *we have some* $\overline{\Gamma} \vdash \overline{p} : (\overline{A}_1, \overline{u}) \overset{s}{=} (\overline{A}_2, \overline{v}) \in [\![\Gamma \vdash p : (A, u) \overset{s}{=} (A, v)]\!]$.

4.2 The new concerns

The proof we summarised in the previous subsection was introduced in a context where there was only one equality, and no distinction between fibrant types and pretypes. Thus, there was never any concern about whether or not elimination principles could be applied. In our case, we must limit the elimination principles of fibrant types to fibrant return types.

Hopefully, we only make and apply transports on the strict equality which can be eliminated to any type. The concern is thus, do these transport get in the way of fibrantness? We need to actually state in the theorem that, whenever we have $\Gamma \vdash A$ fib and any $\overline{\Gamma}$ ctx $\in [\![\Gamma \text{ ctx}]\!]$, we have some $\overline{A}$ such that $\overline{\Gamma} \vdash \overline{A}$ fib $\in [\![\Gamma \vdash A \text{ fib}]\!]$.

4.3 The third axiom

One of the problems that was already one in Oury's translation was the need of an axiom that states that equal functions applied to equal arguments yield equal results. Elementary, but also not provable in the case of heterogenous equality. We propose however a way to remedy this problem: annotating application. While we weren't really able to make a case for it, we believed that in presence of reflection, application should be annotated $u \,@^{A.B}\, v$ and not simply $u\ v$. The moral reason is that if we don't annotate, we might be able to reduce terms that don't make sense. One such instance would be to assume `nat -> nat = nat -> bool` (which is actually independent from the theory as they can be equated in the category of cardinals) and then to type the identity function

```
id : nat -> bool := fun (x : nat) => x
```

which then implies `id n : bool` and then `n : bool` for any `n : nat`. This seems to be threatening canonicity, but we still don't have any conclusive evidence.

Nevertheless, if we can't argue that not having them is bad, we can still claim that having annotations brings good. Indeed, with the annotated version, we for instance have

$$\frac{t_1 \bowtie t_2 \qquad A_1 \bowtie A_2 \qquad B_1 \bowtie B_2 \qquad u_1 \bowtie u_2}{t_1 \,@^{A_1.B_1}\, u_1 \bowtie t_2 \,@^{A_2.B_2}\, u_2}$$

meaning we can also use the induction hypothesis on $B_1 \bowtie B_2$ to get the missing piece necessary to recover an equality of both return types. We formalised that bit in Coq in order to be sure. Adapting the proof to a translation from HTS to two-level type theory wasn't a great ordeal but with our last trick we can gain a translation that doesn't require extra axioms to go through.

5 Formalisation

To be true to our objective of making formalisation of mathematics easier, we should as well put up an effort of formalising our work as well. This still takes a long time, and what we have been working on can be checked out on github [4] in the form a library for a modular formulation of type theory. Although we did not formalise (as of yet) the translations presented in the report, we still proved several results regarding type theory and even applied our library to one of the translations of [5].

Paranoia in type theory. When considering rules in type theory, there is a level of *paranoia* that can vary. For instance if we consider the introduction rule of the reflexivity, we could wonder whether it should be defined as $\dfrac{\Gamma\ \mathsf{ctx} \qquad \Gamma \vdash A\ \mathsf{type} \qquad \Gamma \vdash u : A}{\Gamma \vdash \mathsf{refl}_A\ u : \mathsf{Id}_A(u,u)}$ or as $\dfrac{\Gamma \vdash u : A}{\Gamma \vdash \mathsf{refl}_A\ u : \mathsf{Id}_A(u,u)}$. We call the left inference rule *paranoid* and the right one *economic*. There is a real design choice to be made between these two. We formalised type theory in Coq so that the choice between the two is as easy as setting a flag on or off (relying on the type class mechanism of Coq). The important result that we ship it with is that these two variants of type theory derive the same judgments.

Theorem 6 (Paranoid and economic rules are equivalent)**.** *$\Gamma \vdash \mathcal{J}$ is derivable in the paranoid variant if and only if it is derivable in the economic one.*

This means that we can freely chose whichever version, for instance, as a hypothesis it is usually much nicer to have a paranoid derivation, whereas the economic is preferable as a goal. With the theorem it is easy to switch between the two of them as it consists a pair of constructive translations.

Sanity of type theory. With our experience, we realised it was really easy to forget a rule or to formulate one the wrong way. We thought about different properties we would like our theory to verify in order to increase confidence in its formulation. The idea is that these theorems might also prove useful in developments, but actually checking what is often deemed trivial was worthwhile as it helped us "debug" our theory.

Theorem 7 (Sanity)**.**

- *If* $\Gamma \vdash A\ \mathsf{type}$ *then* $\Gamma\ \mathsf{ctx}$,
- *If* $\Gamma \vdash u : A$ *then* $\Gamma \vdash A\ \mathsf{type}$ *and* $\Gamma\ \mathsf{ctx}$,

- *If* $\Gamma \vdash u \equiv v : A$ *then* $\Gamma \vdash u : A$ *and* $\Gamma \vdash v : A$ *and* $\Gamma \vdash A$ type *and* Γ ctx*,*
- *and so on...*

This theorem actually requires a very long proof for which we developped a tactic that writes derivations on its own. Things become particularly lenghty when substitutions are involved.

Uniqueness of typing. Although this doesn't always need to hold in a type system (for instance in the presence of subtyping), uniqueness of typing is probably something that we want to hold if we didn't explicitely rule it out. The idea is quite simple.

Theorem 8 (Uniqueness of typing)**.** *If* $\Gamma \vdash u : A$ *and* $\Gamma \vdash u : B$*, then* $\Gamma \vdash A \equiv B$*.*

This is proven by simultaneously proving a stronger statement for terms and similar one for substitutions:

Lemma 9.

- *If* $\Gamma \vdash u : A$ *and* $\Delta \vdash u : B$ *and* $\Gamma \equiv \Delta$ *then* $\Gamma \vdash A \equiv B$*,*
- *If* $\sigma : \Gamma \to \Delta_1$ *and* $\sigma : \Gamma' \to \Delta_2$ *with* $\Gamma \equiv \Gamma'$ *then* $\Delta_1 \equiv \Delta_2$*.*

It is worth noting that in both cases, the *domain* context is fixed. Uniqueness wouldn't hold otherwise since typing of variables only depends on the context.

Modular formalisation. We mentioned we were doing a *modular* formalisation of type theory. This was shown a little with the choice of level of paranoia, but we can actually do much more.

First, we are modular in the rules that we have in our theory. The best example would be to have the reflection rule optional so that one can easily translate from a theory with reflection to a theory without, all by only having one definition of the rules (most formalisation of translations rely on copy-pasting for such purposes). This is actually even stronger than that, as it allows to state that, for instance, the presence of universes in the target and in the source is optional but set by the same flag (we can take universes in both, in only one of in none). This allows *modular translations* which is a nice way to show that some theorems *scale*. Also, all our sanity theorems are proven while being agnostic of the theory they are in, meaning that proving sanity of the Π-types rules didn't rely on the identity types rules, as well as being available directly for any instance of type theory we might want among our options.

We are also experimenting with a modular syntax that would allow our theorems to hold regardless of whether we have Tarski or Russel universes, explicit or implicit substitutions, with more or less explicit arguments to constructors... with the very interesting prospect of allowing to (maybe) prove these systems are equivalent. This is already functional, but needs a little more work to be practical.

All this is a foundation that we will use to formalise our work, and probably future work, but that is also available to everyone that would want to formalise their type theory.

6 Conclusion and future work

We addressed the question of restricting reflection in order for it to be compatible with the axiom of univalence. Unfortunately we didn't provide a translation that goes from Coq into Coq directly as we would have hoped. We instead provided a proof in several steps by first going into HTS and then into a two-level type system, our contribution adapts the proof of Oury in such a way that no extra axiom is required. The next step could be going from a two-level type system to some extension of Coq (like the HoTT Coq [3]) using currently researched techniques like forcing (see [15]) on cubical sets [8].

We also produced a Coq library [4] dedicated to formalisations of type theory, and will continue to improve it, as well as use it to formalise the work we described above.

References

[1] Thorsten Altenkirch, Paolo Capriotti, and Nicolai Kraus. Extending homotopy type theory with strict equality. *arXiv preprint arXiv:1604.03799*, 2016.

[2] Andrej Bauer, Gaëtan Gilbert, Philipp G. Haselwarter, Matija Pretnar, and Chris Stone. The 'andromeda' prover. Available at http://www.andromeda-prover.org/.

[3] Andrej Bauer, Jason Gross, Peter LeFanu Lumsdaine, Mike Shulman, Matthieu Sozeau, and Bas Spitters. The hott library: A formalization of homotopy type theory in coq. *CoRR*, abs/1610.04591, 2016.

[4] Andrej Bauer, Philipp G. Haselwarter, and Théo Winterhalter. The 'formal-type-theory' repository. Available at https://github.com/TheoWinterhalter/formal-type-theory.

[5] Simon Boulier, Pierre-Marie Pédrot, and Nicolas Tabareau. The next 700 syntactical models of type theory. In *Certified Programs and Proofs – CPP 2017*, pages 182–194, January 2017.

[6] Jesper Cockx. *Dependent Pattern Matching and Proof-Relevant Unification*. PhD thesis, FWO, June 2017.

[7] Jesper Cockx, Dominique Devriese, and Frank Piessens. Pattern matching without k. In *ACM SIGPLAN Notices*, volume 49, pages 257–268. ACM, 2014.

[8] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical type theory: a constructive interpretation of the univalence axiom. *arXiv preprint arXiv:1611.02108*, 2016.

[9] Robert L. Constable and Joseph L. Bates. The nuprl system, prl project. Available at http://www.nuprl.org/.

[10] The Coq development team. *The Coq proof assistant reference manual*. LogiCal Project, 2017. Version 8.6, available at http://coq.inria.fr.

[11] Thierry Coquand and Gérard Huet. The calculus of constructions. *Information and computation*, 76(2-3):95–120, 1988.

[12] Michael Hedberg. A coherence theorem for martin-löf's type theory. *Journal of Functional Programming*, 8(4):413–436, 1998.

[13] Martin Hofmann. Elimination of extensionality in martin-löf type theory. *Types for proofs and programs*, pages 166–190, 1994.

[14] Martin Hofmann and Thomas Streicher. The groupoid model refutes uniqueness of identity proofs. In *Logic in Computer Science, 1994. LICS'94. Proceedings., Symposium on*, pages 208–212. IEEE, 1994.

[15] Guilhem Jaber, Gabriel Lewertowski, Pierre-Marie Pédrot, Matthieu Sozeau, and Nicolas Tabareau. The Definitional Side of the Forcing. In *Logics in Computer Science*, New York, United States, May 2016.

[16] Per Martin-Löf. Constructive mathematics and computer programming. *Studies in Logic and the Foundations of Mathematics*, 104:153–175, 1982.

[17] Conor McBride. Elimination with a motive. In *International Workshop on Types for Proofs and Programs*, pages 197–216. Springer, 2000.

[18] Ulf Norell. *Towards a practical programming language based on dependent type theory*, volume 32. Citeseer, 2007.

[19] Nicolas Oury. Extensionality in the calculus of constructions. In *International Conference on Theorem Proving in Higher Order Logics*, pages 278–293. Springer, 2005.

[20] Thomas Streicher. *Investigations into intensional type theory*. 1993.

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

[22] Vladimir Voevodsky. A simple type system with two identity types. Available at https://ncatlab.org/homotopytypetheory/files/HTS.pdf.