Eliminating Reflection from Type Theory

To the Legacy of Martin Hofmann

ANONYMOUS AUTHOR(S)

Type theories with equality reflection, such as extensional type theory (ETT), are convenient theories in which to formalise mathematics, as they allow to consider provably equal terms as convertible. Although type-checking is undecidable in this context, variants of ETT have been implemented, for example in NuPRL and more recently in Andromeda. The actual objects that can be checked are not proof-terms, but derivations of proof-terms. This suggests that any derivation of ETT can be translated as a typecheckable proof term of intensional type theory (ITT). However, this result, investigated categorically by Hofmann in 1995, and 10 years later more syntactically by Oury, has never given rise to an effective translation. In this paper, we provide the first syntactical translation from ETT to ITT with uniqueness of identity proofs and function extensionality. This translation has been defined and proven correct in Coq and gives rise to an executable plugin that translates a derivation in ETT into an actual Coq typing judgment.

CCS Concepts: • **Theory of computation** → **Logic**; **Proof theory**; **Constructive mathematics**; **Type theory**; *Logic and verification*; *Automated reasoning*;

Additional Key Words and Phrases: dependent types, translation, formalisation

ACM Reference Format:
Anonymous Author(s). 2018. Eliminating Reflection from Type Theory: To the Legacy of Martin Hofmann. *Proc. ACM Program. Lang.* 1, ICFP, Article 1 (September 2018), 25 pages.

1 INTRODUCTION

Type theories with equality reflection, such as extensional type theory (ETT), are convenient theories in which to formalise mathematics, as they allow to consider provably equal terms as convertible, as expressed in the following typing rule:

$$\frac{\Gamma \vdash_{\mathsf{x}} e : u =_A v}{\Gamma \vdash_{\mathsf{x}} u \equiv v : A}$$

Here, the type $u =_A v$ is Martin-Löf's identity type with only one constructor $\mathsf{refl}\ u : u =_A u$ which allows to internalise proofs of equality inside type theory, whereas $u \equiv v : A$ means that the u and v are convertible in the theory—and can thus be silently replaced one by another in any term. Several variants of ETT have been considered and implemented, for example in NuPRL [Constable and Bates 2014] and more recently in Andromeda [Bauer et al. 2016]. The prototypical example of the use of equality reflection is the definition of a coercion function between two types A and B that are equal (but not convertible) by taking a term of type A and simply returning it as a term of type B:

$$\lambda\ (A\ B : \square_0)\ (e : A = B)\ (x : A) \Rightarrow x : \Pi(A\ B : \square_0).\ A = B \to A \to B.$$

In intensional type theory (ITT), this term does not type-check because x of type A can not be given the type B by conversion. In ETT, however, equality reflection can be used to turn the witness of equality into a proof of conversion and thus the type system validates the fact that x can be given

2475-1421/2018/9-ART1
https://doi.org/

the type B. But, this means that one needs to guess equality proofs during type-checking, because the witness of equality has been lost at the application of the reflection rule, which was not so hard in this example but is in general undecidable, as one can for instance encode the halting problem of any Turing machine as an equality in ETT. That is, the actual objects that can be checked in ETT are not terms, but instead derivations of terms. It thus seems natural to wonder whether any derivation of ETT can be translated as a typecheckable term of ITT. And indeed, it is well know that one can find a corresponding term of the same type in ITT by *explicitely* transporting the term x of type A using the elimination of internal equality on the witness of equality e, noted e_*:

$$\lambda\ (A\ B : \square_0)\ (e : A = B)\ (x : A) \Rightarrow e_*\ x : \Pi(A\ B : \square_0).\ A = B \to A \to B.$$

This can be seen as a way to make explicit the *silent* use of reflection. Furthermore, by making the use of transport as economic as possible, the corresponding ITT term can be seen as a compact witness of the derivation tree of the original ETT term.

This result has first been investigated categorically in the pioneering work of Hofmann [Hofmann 1995, 1997], by showing that the term model of ITT can be turned into a model of ETT by quotienting this model with propositional equality. However, it is not clear how to extend this categorical construction to an explicit and constructive translation for a derivation in ETT to a term of ITT. In 2005, this result has been investigated more syntactically by Oury [Oury 2005]. However, his presentation does not give rise to an effective translation and requires some additional axioms in ITT that can hardly be motivated.

Contributions. In this paper, we present the first syntactical translation from ETT to ITT (assuming uniqueness of identity proof (UIP) and function extensionality in ITT). By syntactical translation, we mean an explicit translation from a derivation $\Gamma \vdash_{\mathsf{x}} t : T$ of ETT (the x index testifies that it is a derivation in ETT) to a context Γ', term t' and type T' of ITT such that $\Gamma' \vdash t' : T'$ in ITT. This translation enjoys the additional property that if T can be typed in ITT, *i.e.,* $\Gamma \vdash T$, then $T' \equiv T$. This means in particular that a theorem proven in ETT but whose statement is also valid in ITT can be automatically transfered to a theorem of ITT. This allows for instance to use *locally* an extension of the Coq proof assistant with a reflection rule, without being forced to rely on the reflection in the entire development.

This translation can be seen as a way to build a syntactical model of ETT from a model of ITT as described more generally in [Boulier et al. 2017] and has been entirely programmed and formalised in Coq [Coq development team 2017]. For this, we rely on TemplateCoq[1], which provides a reifier for Coq terms as represented in Coq kernel as well as a formalisation of the type system of Coq. Thus, our formalisation of ETT is just given by adding the reflection rule to the original type system of Coq. This allows us to extract concrete Coq terms and types from a closed derivation of ETT and opens the way to a direct generalisation of the translation to inductive types (we left this generalisation for future work).

Outline of the paper. Before going into the technical development of the translation, we explain its main ingredients and differences with previous works. Then, in Section 2, we define the extensional and intensional type theories we consider. In Section 3, we define the main ingredient of the translation, which is a relation between terms of ETT and term in ITT. Then, the translation is given in Section4. Section 5 describes the Coq formalisation and Section 6 discusses related work. The main proof are given in Appendix A and B.

The Coq formalisation is provided as an anonymous supplementary material.

[1] https://template-coq.github.io/template-coq/

On the need of UIP and functional extensionality.

Our translation targets ITT plus UIP and functional extensionality, which correspond to the two following axioms

$$\begin{array}{rcl} \mathtt{UIP} & : & \Pi(A : \square_i).\, \Pi(x\ y : A).\, \Pi(e\ e' : x = y).\, e = e' \\ \mathtt{FunExt} & : & \Pi(A\ B : \square_i).\, \Pi(f\ g : \Pi(x : A).\, B).(\Pi(x : A).\, fx = gx) \to f = g \end{array}$$

The first axiom says that any two proofs of the same equality are equal, and the other one says that two (dependent) functions are equal when they are pointwise equal[2]. These two axioms are perfectly valid statement of ITT and they can be proven in ETT. Indeed, UIP is can be shown to be equivalent to the Streicher K axiom

$$\mathtt{K} \quad : \quad \Pi(A : \square_i).\, \Pi(x : A).\, \Pi(e : x = x).\, e = \mathsf{refl}_x$$

using the elimination on the identity type. But K is provable in ETT by considering the type

$$\Pi(A : \square_i).\, \Pi(x\ y : A).\, \Pi(e : x = y).\, e = \mathsf{refl}_x$$

which is well typed (using the reflection rule to show that e has type $x = x$) and which can be inhabited by elimination of the identity type. In the same way, functional extensionality is provable in ETT because

$$\begin{array}{lcll} \Pi(x : A).\, f\ x = g\ x & \to & x : A \vdash f\ x \equiv g\ x & \text{by reflection} \\ & \to & (\lambda(x : A).f\ x) \equiv (\lambda(x : A).g\ x) & \text{by congruence of } \equiv \\ & \to & f \equiv g & \text{by } \eta\text{-law} \\ & \to & f = g & \end{array}$$

Therefore, applying our translation to the proofs of those theorems in ETT gives corresponding proofs of the same theorems in ITT. However, UIP is independent from ITT, as first shown by Hofmann and Streicher using the groupoid model [Hofmann and Streicher 1998], which has recently been extended in the setting of univalent type theory using the simplicial or cubical models [Bezem et al. 2013; Kapulkin and Lumsdaine 2012]. Similarly, Boulier *et al.* have shown that functional extensionality is independent from ITT using a simple syntactical translation [Boulier et al. 2017].

Therefore, our translation provides proofs of axioms independent from ITT, which means that the target of the transation already needs to have both UIP and functional extensionatity. Part of our work is to show formally that they are the only axioms required.

Heterogeneous equality and parametricity translation.

The basic idea behind the translation from ETT to ITT is to interpret conversion using the internal notion of equality, *i.e.,* the identity type. But this means that two terms of two convertible types that were comparable in ETT become comparable in ITT only up-to the equality between the two types. One possible solution of this problem is to consider a native heterogeneous equality, such as *John Major's equality* introduced by McBride [McBride 2000]. However, to avoid adding additional axioms to ITT, we prefer to encode this heterogeneous equality using the following dependent sums:

$$t\ {}_T{\cong}_U\ u := \Sigma(p : T = U).\, p_*\ t = u.$$

During the translation, the same term occurring twice can be translated in two different manners, if the corresponding typing derivations are different. Even the types of the two different translations may be different. However, we have the strong property that any two translations of the same term only differ on places where transports of proof of equality have been injected. To keep track of this

[2]In Homotopy Type Theory (HoTT) [Univalent Foundations Program 2013], the functional extensionality axiom is stated in a more complete way, using the notion of adjoint equivalences, but this more complete way collapses to our simpler statement in presence of UIP.

s	$::=$	$\square_i \ (i \in \mathbb{N})$	sorts (universes)
T, A, B, t, u, v	$::=$	$x \mid \lambda(x : A).B.t \mid t \ @_{x:A.B} \ u$	λ-terms
		$\mid \langle u; v \rangle_{x:A.B} \mid \pi_1^{x:A.B} \ p \mid \pi_2^{x:A.B} \ p$	pair terms
		$\mid \mathsf{refl}_A \ u \mid \mathsf{J}(A, u, x.e.P, w, v, p)$	equality terms
		$\mid \mathsf{funext}(x : A, B, f, g, e) \mid \mathsf{uip}(A, u, v, p, q)$	equality axioms
		$\mid s \mid \Pi(x : A).\, B \mid \Sigma(x : A).\, B \mid u =_A v$	types
Γ, Δ	$::=$	$\bullet \mid \Gamma, x : A$	contexts

Fig. 1. Syntax of ETT/ITT

property, we introduce the relation $t \sim t'$ between two terms of ITT, of possibly different types. The crux of the proof of the translation is to guarantee that for every two terms t_1 and t_2 such that $\Gamma \vdash t_1 : T_1$, $\Gamma \vdash t_2 : T_2$ and $t_1 \sim t_2$, there exists p such that $\Gamma \vdash p : t_1 \ {}_{T_1}{\cong}_{T_2} \ t_2$. However, during the proof, variables of different but equal types are introduced and the context cannot be maintained to be the same for both t_1 and t_2. Therefore, the translation needs to keep track of this duplication of variables, plus a proof that they are heterogeneously equal. This mechanism is similar to what happens in the (relational) internal parametricity translation in ITT introduced by Bernardy *et al.* [Bernardy et al. 2012] and recently rephrased in the setting of TemplateCoq [Anand et al. 2018]. Namely, a context is not translated as a telescope of variables, but as a telescope of triples consisting of two variables plus a witness that they are in the parametric relation. In our setting, this amounts to consider telescope of triples consisting of two variables plus a witness that they are heterogeneously equal. We can express this by considering the following dependent sums:

$$\mathsf{Pack} \ A_1 \ A_2 := \Sigma(x : A_1).\, \Sigma(y : A_2).\, x \ {}_{A_1}{\cong}_{A_2} \ y.$$

This presentation inspired by the parametricity translation is crucial in order to get an effective translation, because it is necessary to keep track of the evolution of contexts when doing the translation on open terms. This ingredient is missing in Oury's work [Oury 2005], which prevents him to deduce an effective translation from his theorem.

2 DEFINITIONS OF EXTENSIONAL AND INTENSIONAL TYPE THEORIES

This section presents the common syntax, typing and main properties of ETT and ITT. Our type theories are full-blended, featuring a universe hierarchy, dependent products and sums as well as Martin Löf's identity types.

2.1 Syntax of ETT and ITT

The common syntax of ETT and ITT is given in Figure 1. It features: dependent products $\Pi(x : A).\, B$, with λ-terms and (annotated) applications, negative dependent sums $\Sigma(x : A).\, B$ with (annotated) projections, sorts $\square_i$, identity types $u =_A v$ with reflection and elimination as well as terms realising UIP and functional extensionality. Annotating terms with otherwise computationally irrelevant typing information is a common practice when studying precisely the syntax of type theory (see [Streicher 1993] for a similar example). We will write $A \to B$ for $\Pi(\_ : A).\, B$ the non-dependent product.

We consider a fixed universe hierarchy without cumulativity, which ensures in particular uniqueness of typing (2.2) which is important for the translation.

2.2 The Typing Systems

As usual in dependent type theory, we consider contexts which are telescopes whose declarations may depend on any variable already introduced. We note $\Gamma \vdash t : A$ to say that t as type A in context Γ. $\Gamma \vdash A$ shall stand for $\Gamma \vdash A : s$ for some sort s.

We use two relations $(s, s') \in \mathsf{Ax}$ (simply noted (s, s')) and $(s, s', s'') \in \mathsf{R}$ (simply noted (s, s', s'')) to constrain the sorts in the typing rules for universes, dependent products and dependent sums, as is done in any Pure Type System (PTS). In our case, because we do not have cumulativity, the rules are as follows:

$$(\Box_i, \Box_{i+1}) \in \mathsf{Ax} \qquad\qquad (\Box_i, \Box_j, \Box_{\max(i,j)}) \in \mathsf{R}$$

We give the typing rules of ITT in Figure 2. The rules are standard and we do not explain them. Let us just point out the conversion rule, which says that $u : A$ can be given the type $u : B$ when $A \equiv B$, *i.e.,* when A and B are convertible. As the notion of conversion is central in our work—the conversion of ETT being translated to an equality in ITT—we provide an exhaustive definition of it, with computational conversion rules (including β-conversion or reduction of the elimination principle of equality over reflexivity, see Figure 3) and congruence conversion rules (Figure 4). Although pretty straightforward, being precise here is very important, as for instance the congruence rule for λ-terms is the reason why functional extensionality is derivable in ETT.

ETT is thus simply an extension of ITT (we write $\vdash_{\mathsf{x}}$ for the associated typing judgment) with the reflection rule on equality, allowing to consider propositionally equal terms as convertible:

$$\frac{\Gamma \vdash_{\mathsf{x}} e : u =_A v}{\Gamma \vdash_{\mathsf{x}} u \equiv v : A}$$

Note that, as already mentioned, in the presence of reflection and J, UIP is derivable so we could remove it from ETT, but keeping it allows us to share a common syntax which makes the statements of theorems simpler and does not affect the development.

2.3 General Properties of ITT and ETT

We now state the main properties of both ITT and ETT. We do not detail their proof as they are standard and can be found the Coq formalisation.

First, although not explicit in the typing system, weakening is admissible in ETT and ITT.

Lemma 2.1 (Weakening). *If* $\Gamma \vdash \mathcal{J}$ *and* Δ *extends* Γ *(possibly interleaving variables) then* $\Delta \vdash \mathcal{J}$.

Then, as mentioned above, the use of a non-cumulative hierarchy allows use to prove that a term t can be given at most one type in a context Γ, up-to conversion.

Lemma 2.2 (Uniqueness of typing). *If* $\Gamma \vdash u : T_1$ *and* $\Gamma \vdash u : T_2$ *then* $\Gamma \vdash T_1 \equiv T_2$.

Finally, an important property of the typing system (seen as a mutual inductive definition) is the possibility to deduce hypotheses from their conclusion, thanks to inversion of typing. Note that it is important here that our syntax is annotated for applications and projections as it provides a richer inversion principle.

Lemma 2.3 (Inversion of typing).

(1) *If* $\Gamma \vdash x : T$ *then* $(x : A) \in \Gamma$ *and* $\Gamma \vdash A \equiv T$.
(2) *If* $\Gamma \vdash \Box_i : T$ *then* $\Gamma \vdash \Box_{i+1} \equiv T$.
(3) *If* $\Gamma \vdash \Pi(x : A).\, B : T$ *then* $\Gamma \vdash A : s$ *and* $\Gamma, x : A \vdash B : s'$ *and* $\Gamma \vdash s'' \equiv T$ *for some* (s, s', s'').
(4) *If* $\Gamma \vdash \lambda(x : A).B.t : T$ *then* $\Gamma \vdash A : s$ *and* $\Gamma, x : A \vdash B : s'$ *and* $\Gamma, x : A \vdash t : B$ *and* $\Gamma \vdash \Pi(x : A).\, B \equiv T$.

Well-formedness of contexts.

$$\frac{}{\vdash \bullet} \qquad \frac{\vdash \Gamma \qquad \Gamma \vdash A}{\vdash \Gamma, x : A}(x \notin \Gamma)$$

Types.

$$\frac{\vdash \Gamma}{\Gamma \vdash s : s'}(s, s') \qquad \frac{\Gamma \vdash A : s \qquad \Gamma, x : A \vdash B : s'}{\Gamma \vdash \Pi(x : A).\, B : s''}(s, s', s'')$$

$$\frac{\Gamma \vdash A : s \qquad \Gamma, x : A \vdash B : s'}{\Gamma \vdash \Sigma(x : A).\, B : s''}(s, s', s'') \qquad \frac{\Gamma \vdash A : s \qquad \Gamma \vdash u : A \qquad \Gamma \vdash v : A}{\Gamma \vdash u =_A v : s}$$

Structural rules.

$$\frac{\vdash \Gamma \qquad (x : A) \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\Gamma \vdash u : A \qquad \Gamma \vdash A \equiv B}{\Gamma \vdash u : B}$$

λ-calculus terms.

$$\frac{\Gamma \vdash A : s \qquad \Gamma, x : A \vdash B : s' \qquad \Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda(x : A).B.t : \Pi(x : A).\, B}$$

$$\frac{\Gamma \vdash A : s \qquad \Gamma, x : A \vdash B : s' \qquad \Gamma \vdash t : \Pi(x : A).\, B \qquad \Gamma \vdash u : A}{\Gamma \vdash t \mathbin{@_{x:A.B}} u : B[x \leftarrow u]}$$

$$\frac{\Gamma \vdash u : A \qquad \Gamma \vdash A : s \qquad \Gamma, x : A \vdash B : s' \qquad \Gamma \vdash v : B[x \leftarrow u]}{\Gamma \vdash \langle u; v \rangle_{x:A.B} : \Sigma(x : A).\, B} \qquad \frac{\Gamma \vdash p : \Sigma(x : A).\, B}{\Gamma \vdash \pi_1^{x:A.B}\, p : A}$$

$$\frac{\Gamma \vdash p : \Sigma(x : A).\, B}{\Gamma \vdash \pi_2^{x:A.B}\, p : B[x \leftarrow \pi_1^{x:A.B}\, p]}$$

Equality terms.

$$\frac{\Gamma \vdash A : s \qquad \Gamma \vdash u : A}{\Gamma \vdash \mathsf{refl}_A\, u : u =_A u} \qquad \frac{\Gamma \vdash e_1, e_2 : u =_A v}{\Gamma \vdash \mathsf{uip}(A, u, v, e_1, e_2) : e_1 = e_2}$$

$$\frac{\begin{array}{c}\Gamma \vdash A : s \\ \Gamma \vdash u, v : A \qquad \Gamma, x : A, e : u =_A x \vdash P : s' \qquad \Gamma \vdash p : u =_A v \qquad \Gamma \vdash w : P[x \leftarrow u, e \leftarrow \mathsf{refl}_A\, u]\end{array}}{\Gamma \vdash \mathsf{J}(A, u, x.e.P, w, v, p) : P[x \leftarrow v, e \leftarrow p]}$$

$$\frac{\Gamma \vdash f, g : \Pi(x : A).\, B \qquad \Gamma \vdash e : \Pi(x : A).\, f \mathbin{@_{x:A.B}} x =_B g \mathbin{@_{x:A.B}} x}{\Gamma \vdash \mathsf{funext}(x : A, B, f, g, e) : f = g}$$

Fig. 2. Typing rules

Computation.

$$\frac{\Gamma \vdash A : s \qquad \Gamma, x : A \vdash B : s' \qquad \Gamma, x : A \vdash t : B \qquad \Gamma \vdash u : A}{\Gamma \vdash (\lambda(x : A).B.t)\ @_{x:A.B}\ u \equiv t[x \leftarrow u] : B[x \leftarrow u]}$$

$$\frac{\Gamma \vdash A : s \qquad \Gamma \vdash u : A \qquad \Gamma, x : A, e : u =_A x \vdash P : s' \qquad \Gamma \vdash w : P[x \leftarrow u, e \leftarrow \mathsf{refl}_A\ u]}{\Gamma \vdash \mathsf{J}(A, u, x.e.P, w, u, \mathsf{refl}_A\ u) \equiv w : P[x \leftarrow u, e \leftarrow \mathsf{refl}_A\ u]}$$

$$\frac{\Gamma \vdash A : s \qquad \Gamma \vdash u : A \qquad \Gamma, x : A \vdash B : s' \qquad \Gamma \vdash v : B[x \leftarrow u]}{\Gamma \vdash \pi_1^{x:A.B}\ \langle u; v\rangle_{x:A.B} \equiv u : A}$$

$$\frac{\Gamma \vdash A : s \qquad \Gamma \vdash u : A \qquad \Gamma, x : A \vdash B : s' \qquad \Gamma \vdash v : B[x \leftarrow u]}{\Gamma \vdash \pi_2^{x:A.B}\ \langle u; v\rangle_{x:A.B} \equiv v : B[x \leftarrow u]}$$

Equivalence relation.

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

Conversion.

$$\frac{\Gamma \vdash t_1 \equiv t_2 : T_1 \qquad \Gamma \vdash T_1 \equiv T_2}{\Gamma \vdash t_1 \equiv t_2 : T_2}$$

Fig. 3. Main conversion rules

(5) *If* $\Gamma \vdash u\ @_{x:A.B}\ v : T$ *then* $\Gamma \vdash A : s$ *and* $\Gamma, x : A \vdash B : s'$ *and* $\Gamma \vdash u : \Pi(x : A).\ B$ *and* $\Gamma \vdash v : A$ *and* $\Gamma \vdash B[x \leftarrow u] \equiv T$.
(6) *… Analogous for the remaining term and type constructors.*

Proof. Each case is proven by induction on the derivation (which corresponds to any number of applications of the conversion rule following one introduction rule). □

3 RELATING TRANSLATED EXPRESSIONS

We want to define a relation on terms that equates two terms that are the same up to transport. This bears the question of what notion of transport is going to be used. Transport can be defined from elimination of equality as follows:

Definition 3.1 (Transport). Given $\Gamma \vdash p : T_1 =_s T_2$ and $\Gamma \vdash t : T_1$ we define the transport of t along p, written $p_*\ t$, as $\mathsf{J}(s, T_1, X.e.\ T_1 \to X, \lambda(x : T_1).T_1.x, T_2, p)\ @_{T_1.T_2}\ t$ such that $\Gamma \vdash p_*\ t : T_2$.

However, in order not to confuse the transports added by the translation with the transports that were already present in the source, we consider p_* as part of the syntax in the reasoning. It will be unfolded to its definition only after the complete translation is performed. This idea is not novel as Hoffman's Subst operator that was part of his ITT (noted TT_I in his paper [Hofmann 1995]).

We first define the (purely syntactic) relation $\sqsubset$ between ETT terms and ITT terms in Figure 5 stating that the ITT term is simply a decoration of the first term by transports. Its purpose is to state how close to the original term its translation is. Then, in Figure 6 we extend this relation to $\sim$ that is meant to deal with two ITT terms (bascically $\sim\ =\ (\sqsupset \cdot \sqsubset)^*$). We remark easily that whenever $t \sqsubset \bar{t}$ we also have $t \sim \bar{t}$.

Congruence of type constructors.

$$\frac{\Gamma \vdash A_1 \equiv A_2 : s \qquad \Gamma, x : A_1 \vdash B_1 \equiv B_2 : s'}{\Gamma \vdash \Pi(x : A_1).\, B_1 \equiv \Pi(x : A_2).\, B_2 : s''}(s, s', s'')$$

$$\frac{\Gamma \vdash A_1 \equiv A_2 : s \qquad \Gamma, x : A_1 \vdash B_1 \equiv B_2 : s'}{\Gamma \vdash \Sigma(x : A_1).\, B_1 \equiv \Sigma(x : A_2).\, B_2 : s''}(s, s', s'')$$

$$\frac{\Gamma \vdash A_1 \equiv A_2 : s \qquad \Gamma \vdash u_1 \equiv u_2 : A_1 \qquad \Gamma \vdash v_1 \equiv v_2 : A_1}{\Gamma \vdash u_1 =_{A_1} v_1 \equiv u_2 =_{A_2} v_2 : s}$$

Congruence of λ-calculus terms.

$$\frac{\Gamma \vdash A_1 \equiv A_2 : s \qquad \Gamma, x : A_1 \vdash B_1 \equiv B_2 : s' \qquad \Gamma, x : A_1 \vdash t_1 \equiv t_2 : B_1}{\Gamma \vdash \lambda(x : A_1).B_1.t_1 \equiv \lambda(x : A_2).B_2.t_2 : \Pi(x : A_1).\, B_1}$$

$$\frac{\Gamma \vdash A_1 \equiv A_2 : s \qquad \Gamma, x : A_1 \vdash B_1 \equiv B_2 : s' \qquad \Gamma \vdash t_1 \equiv t_2 : \Pi(x : A_1).\, B_1 \qquad \Gamma \vdash u_1 \equiv u_2 : A_1}{\Gamma \vdash t_1 \mathbin{@}_{x:A_1.B_1} u_1 \equiv t_1 \mathbin{@}_{x:A_1.B_1} u_1 : B_1[x \leftarrow u_1]}$$

$$\frac{\Gamma \vdash A_1 \equiv A_2 : s \qquad \Gamma \vdash u_1 \equiv u_2 : A_1 \qquad \Gamma, x : A_1 \vdash B_1 \equiv B_2 : s' \qquad \Gamma \vdash v_1 \equiv v_2 : B_1[x \leftarrow u_1]}{\Gamma \vdash \langle u_1; v_1 \rangle_{x:A_1.B_1} \equiv \langle u_2; v_2 \rangle_{x:A_2.B_2} : \Sigma(x : A_1).\, B_1}$$

$$\frac{\Gamma \vdash A_1 \equiv A_2 : s \qquad \Gamma, x : A_1 \vdash B_1 \equiv B_2 : s' \qquad \Gamma \vdash p_1 \equiv p_2 : \Sigma(x : A_1).\, B_1}{\Gamma \vdash \pi_1^{x:A_1.B_1}\, p_1 \equiv \pi_1^{x:A_2.B_2}\, p_2 : A_1}$$

$$\frac{\Gamma \vdash A_1 \equiv A_2 : s \qquad \Gamma, x : A_1 \vdash B_1 \equiv B_2 : s' \qquad \Gamma \vdash p_1 \equiv p_2 : \Sigma(x : A_1).\, B_1}{\Gamma \vdash \pi_2^{x:A_1.B_1}\, p_1 \equiv \pi_2^{x:A_2.B_2}\, p_2 : B_1[x \leftarrow \pi_1^{x:A_1.B_1}\, p_1]}$$

Congruence of equality terms.

$$\frac{\Gamma \vdash A_1 \equiv A_2 : s \qquad \Gamma \vdash u_1 \equiv u_2 : A}{\Gamma \vdash \mathsf{refl}_{A_1}\, u_1 \equiv \mathsf{refl}_{A_2}\, u_2 : u_1 =_{A_1} u_1}$$

$$\frac{\begin{array}{c}\Gamma \vdash A_1 \equiv A_2 : s \qquad \Gamma \vdash u_1 \equiv u_2 : A_1 \qquad \Gamma \vdash v_1 \equiv v_2 : A_1 \qquad \Gamma, x : A_1, e : u_1 =_{A_1} x \vdash P_1 \equiv P_2 : s' \\ \Gamma \vdash p_1 \equiv p_2 : u_1 =_{A_1} v_1 \qquad \Gamma \vdash w_1 \equiv w_2 : P_1[x \leftarrow u_1, e \leftarrow \mathsf{refl}_{A_1}\, u_1]\end{array}}{\Gamma \vdash \mathsf{J}(A_1, u_1, x.e.P_1, w_1, v_1, p_1) \equiv \mathsf{J}(A_2, u_2, x.e.P_2, w_2, v_2, p_2) : P[x \leftarrow v_1, e \leftarrow p_1]}$$

$$\frac{\begin{array}{c}\Gamma \vdash A_1 \equiv A_2 : s \qquad \Gamma, x : A_1 \vdash B_1 \equiv B_2 : s' \qquad \Gamma \vdash f_1 \equiv f_2 : \Pi(x : A_1).\, B_1 \\ \Gamma \vdash g_1 \equiv g_2 : \Pi(x : A_1).\, B_1 \qquad \Gamma \vdash e_1 \equiv e_2 : \Pi(x : A_1).\, f_1 \mathbin{@}_{x:A_1.B_1} x =_{B_1} g_1 \mathbin{@}_{x:A_1.B_1} x\end{array}}{\Gamma \vdash \mathsf{funext}(x : A_1, B_1, f_1, g_1, e_1) \equiv \mathsf{funext}(x : A_2, B_2, f_2, g_2, e_2) : f_1 = g_1}$$

$$\frac{\begin{array}{c}\Gamma \vdash A_1 \equiv A_2 \\ \Gamma \vdash u_1 \equiv u_2 : A_1 \qquad \Gamma \vdash v_1 \equiv v_2 : A_2 \qquad \Gamma \vdash p_1 \equiv p_2 : u_1 =_{A_1} v_1 \qquad \Gamma \vdash q_1 \equiv q_2 : u_1 =_{A_1} v_1\end{array}}{\Gamma \vdash \mathsf{uip}(A_1, u_1, v_1, p_1, q_1) \equiv \mathsf{uip}(A_2, u_2, v_2, p_2, q_2) : p_1 = q_1}$$

Fig. 4. Congruence rules

$$\frac{}{x \sqsubset x} \qquad \frac{t_1 \sqsubset t_2}{t_1 \sqsubset p_* \, t_2} \qquad \frac{A_1 \sqsubset A_2 \qquad B_1 \sqsubset B_2}{\Pi(x : A_1).\, B_1 \sqsubset \Pi(x : A_2).\, B_2} \qquad \frac{A_1 \sqsubset A_2 \qquad B_1 \sqsubset B_2}{\Sigma(x : A_1).\, B_1 \sqsubset \Sigma(x : A_2).\, B_2}$$

$$\frac{A_1 \sqsubset A_2 \qquad u_1 \sqsubset u_2 \qquad v_1 \sqsubset v_2}{u_1 =_{A_1} v_1 \sqsubset u_2 =_{A_2} v_2} \qquad \frac{}{s \sqsubset s} \qquad \frac{A_1 \sqsubset A_2 \qquad B_1 \sqsubset B_2 \qquad t_1 \sqsubset t_2}{\lambda(x : A_1).B_1.t_1 \sqsubset \lambda(x : A_2).B_2.t_2}$$

$$\frac{t_1 \sqsubset t_2 \qquad A_1 \sqsubset A_2 \qquad B_1 \sqsubset B_2 \qquad u_1 \sqsubset u_2}{t_1 \; @_{x:A_1.B_1} \; u_1 \sqsubset t_2 \; @_{x:A_2.B_2} \; u_2} \qquad \frac{A_1 \sqsubset A_2 \qquad B_1 \sqsubset B_2 \qquad t_1 \sqsubset t_2 \qquad u_1 \sqsubset u_2}{\langle t_1; u_1 \rangle_{x:A_1.B_1} \sqsubset \langle t_2; u_2 \rangle_{x:A_2.B_2}}$$

$$\frac{A_1 \sqsubset A_2 \qquad B_1 \sqsubset B_2 \qquad p_1 \sqsubset p_2}{\pi_1^{x:A_1.B_1} \, p_1 \sqsubset \pi_1^{x:A_2.B_1} \, p_2} \qquad \frac{A_1 \sqsubset A_2 \qquad B_1 \sqsubset B_2 \qquad p_1 \sqsubset p_2}{\pi_2^{x:A_1.B_1} \, p_1 \sqsubset \pi_2^{x:A_2.B_2} \, p_2}$$

$$\frac{A_1 \sqsubset A_2 \qquad u_1 \sqsubset u_2}{\mathsf{refl}_{A_1} \, u_1 \sqsubset \mathsf{refl}_{A_2} \, u_2} \qquad \frac{A_1 \sqsubset A_2 \qquad B_1 \sqsubset B_2 \qquad f_1 \sqsubset f_2 \qquad g_1 \sqsubset g_2 \qquad e_1 \sqsubset e_2}{\mathsf{funext}(x : A_1, B_1, f_1, g_1, e_1) \sqsubset \mathsf{funext}(x : A_2, B_2, f_2, g_2, e_2)}$$

$$\frac{A_1 \sqsubset A_2 \qquad u_1 \sqsubset u_2 \qquad v_1 \sqsubset v_2 \qquad p_1 \sqsubset p_2 \qquad q_1 \sqsubset q_2}{\mathsf{uip}(A_1, u_1, v_1, p_1, q_1) \sqsubset \mathsf{uip}(A_2, u_2, v_2, p_2, q_2)}$$

$$\frac{A_1 \sqsubset A_2 \qquad u_1 \sqsubset u_2 \qquad P_1 \sqsubset P_2 \qquad w_1 \sqsubset w_2 \qquad v_1 \sqsubset v_2 \qquad p_1 \sqsubset p_2}{\mathsf{J}(A_1, u_1, x.e.P_1, w_1, v_1, p_1) \sqsubset \mathsf{J}(A_2, u_2, x.e.P_2, w_2, v_2, p_2)}$$

Fig. 5. Relation $\sqsubset$

The goal is to prove that two terms in this relation, that are well-typed in the target type theory, are heterogeneously equal. As for this notion, we define $t \; {}_T\cong_U \; u := \Sigma(p : T = U).\, p_* \; t = u$. This definition of heterogeneous equality can be shown to be reflexive, symmetric and transitive. Because of UIP, heterogeneous equality collapses to equality when taken on the same type.

Lemma 3.2. *If* $\Gamma \vdash e : u \; {}_A\cong_A \; v$ *then there exists* p *such that* $\Gamma \vdash p : u =_A v$.

Proof. This holds thanks to UIP on equality, which implies K, and so the proof of $A = A$ can be taken to be reflexivity. □

Note. *In particular,* $\cong$ *on types corresponds to equality. This is not as trivial as it sounds, one might be concerned about what happens if we have* $\Gamma \vdash e : A \; {}_s\cong_{s'} \; B$ *with two distinct sorts* s *and* s'*. We would thus have* $s = s'$*, however, for this to be well-typed, we need to give a common type to* s *and* s'*, which can only be achieved if* s *and* s' *are actually the same sort.*

Before we can prove the fundamental lemma stating that two terms in relation are heterogeneously equal, we need to consider another construction. As explained in the introduction, when proving the property by induction on terms, we introduce variables in the context that are equal only with the heterogeneous equality, and thus cannot not be taken to be syntactically the same. This phenomenon is similar to what happens in the parametricity translation [Bernardy et al. 2012]. Therefore, a context will not only be a telescope of variables, but rather a telescope of triples consisting of two variables of possibly different types, and a witness that they are heterogeneously

$$\frac{}{x \sim x} \qquad \frac{t_1 \sim t_2}{p_* \; t_1 \sim t_2} \qquad \frac{t_1 \sim t_2}{t_1 \sim p_* \; t_2} \qquad \frac{A_1 \sim A_2 \qquad B_1 \sim B_2}{\Pi(x : A_1).\, B_1 \sim \Pi(x : A_2).\, B_2}$$

$$\frac{A_1 \sim A_2 \qquad B_1 \sim B_2}{\Sigma(x : A_1).\, B_1 \sim \Sigma(x : A_2).\, B_2} \qquad \frac{A_1 \sim A_2 \qquad u_1 \sim u_2 \qquad v_1 \sim v_2}{u_1 =_{A_1} v_1 \sim u_2 =_{A_2} v_2} \qquad \frac{}{s \sim s}$$

$$\frac{A_1 \sim A_2 \qquad B_1 \sim B_2 \qquad t_1 \sim t_2}{\lambda(x : A_1).B_1.t_1 \sim \lambda(x : A_2).B_2.t_2} \qquad \frac{t_1 \sim t_2 \qquad A_1 \sim A_2 \qquad B_1 \sim B_2 \qquad u_1 \sim u_2}{t_1 \; @_{x:A_1.B_1} \; u_1 \sim t_2 \; @_{x:A_2.B_2} \; u_2}$$

$$\frac{A_1 \sim A_2 \qquad B_1 \sim B_2 \qquad t_1 \sim t_2 \qquad u_1 \sim u_2}{\langle t_1; u_1 \rangle_{x:A_1.B_1} \sim \langle t_2; u_2 \rangle_{x:A_2.B_2}} \qquad \frac{A_1 \sim A_2 \qquad B_1 \sim B_2 \qquad p_1 \sim p_2}{\pi_1^{x:A_1.B_1} \; p_1 \sim \pi_1^{x:A_2.B_1} \; p_2}$$

$$\frac{A_1 \sim A_2 \qquad B_1 \sim B_2 \qquad p_1 \sim p_2}{\pi_2^{x:A_1.B_1} \; p_1 \sim \pi_2^{x:A_2.B_2} \; p_2} \qquad \frac{A_1 \sim A_2 \qquad u_1 \sim u_2}{\mathsf{refl}_{A_1} \; u_1 \sim \mathsf{refl}_{A_2} \; u_2}$$

$$\frac{A_1 \sim A_2 \qquad B_1 \sim B_2 \qquad f_1 \sim f_2 \qquad g_1 \sim g_2 \qquad e_1 \sim e_2}{\mathsf{funext}(x : A_1, B_1, f_1, g_1, e_1) \sim \mathsf{funext}(x : A_2, B_2, f_2, g_2, e_2)}$$

$$\frac{A_1 \sim A_2 \qquad u_1 \sim u_2 \qquad v_1 \sim v_2 \qquad p_1 \sim p_2 \qquad q_1 \sim q_2}{\mathsf{uip}(A_1, u_1, v_1, p_1, q_1) \sim \mathsf{uip}(A_2, u_2, v_2, p_2, q_2)}$$

$$\frac{A_1 \sim A_2 \qquad u_1 \sim u_2 \qquad P_1 \sim P_2 \qquad w_1 \sim w_2 \qquad v_1 \sim v_2 \qquad p_1 \sim p_2}{\mathsf{J}(A_1, u_1, x.e.P_1, w_1, v_1, p_1) \sim \mathsf{J}(A_2, u_2, x.e.P_2, w_2, v_2, p_2)}$$

Fig. 6. Relation $\sim$

equal. To make this precise, we define the following macro:

$$\mathsf{Pack}\; A_1\; A_2 := \Sigma(x : A_1).\, \Sigma(y : A_2).\, x \cong y$$

together with its projections

$$\mathsf{Proj}_1\; p := \pi_1^{\cdot}\; p \qquad \mathsf{Proj}_2\; p := \pi_1^{\cdot}\; \pi_2^{\cdot}\; p \qquad \mathsf{Proj}_\mathsf{e}\; p := \pi_2^{\cdot}\; \pi_2^{\cdot}\; p.$$

We can then extend this notion canonically to contexts of the same length that are well formed using the same sorts:

$$\mathsf{Pack}\; (\Gamma_1, x : A_1)\; (\Gamma_2, x : A_2) := (\mathsf{Pack}\; \Gamma_1\; \Gamma_2), x : \mathsf{Pack}\; (A_1\; \uparrow)\; (A_2\; \uparrow) \qquad \mathsf{Pack}\; \bullet\; \bullet := \bullet.$$

When we pack contexts, we also need to apply the correct projections for the terms in that context to still make sense. Assuming γ is the common length of Γ_1 and Γ_2 (and hence $\mathsf{Pack}\; \Gamma_1\; \Gamma_2$), we have $\Gamma, \mathsf{Pack}\; \Gamma_1\; \Gamma_2 \vdash t\; \uparrow^\gamma : A\; \uparrow^\gamma$ whenever $\Gamma, \Gamma_1 \vdash t : A$ (resp. $\Gamma, \mathsf{Pack}\; \Gamma_1\; \Gamma_2 \vdash t\; \uparrow^\gamma : A\; \uparrow^\gamma$ whenever $\Gamma, \Gamma_2 \vdash t : A$). We often omit γ for readability. Similarly, when Γ_1 and Γ_2 are understood we will write Γ_p for $\mathsf{Pack}\; \Gamma_1\; \Gamma_2$.

Implicitely, whenever we use the notation $\mathsf{Pack}\; \Gamma_1\; \Gamma_2$ it means that the two contexts are of the same length and well-formed with the same sorts. We can now state the fundamental lemma.

Lemma 3.3 (Fundamental lemma). *Let t_1 and t_2 be two terms. If $\Gamma, \Gamma_1 \vdash t_1 : T_1$ and $\Gamma, \Gamma_2 \vdash t_2 : T_2$ and $t_1 \sim t_2$ then there exists p such that $\Gamma, \mathsf{Pack}\ \Gamma_1\ \Gamma_2 \vdash p : t_1 \restriction_{T_1\downarrow} \cong_{T_2\uparrow} t_2 \restriction$.*

Proof. The proof is by induction on the derivation of $t_1 \sim t_2$. It is given in Appendix A. □

We can also prove that $\sim$ preserves substitution and is an equivalence relation.

Lemma 3.4. *If $t_1 \sim t_2$ and $u_1 \sim u_2$ then $t_1[x \leftarrow u_1] \sim t_2[x \leftarrow u_2]$.*

Proof. We proceed by induction on the derivation of $t_1 \sim t_2$. □

Lemma 3.5 ($\sim$ is an equivalence relation). *$\sim$ is reflexive, symmetric and transitive.*

Proof. By induction on the derivation. □

4 TRANSLATING ETT TO ITT

We now define the translations (let us stress the plural here) of an extensional judgment. We extend $\sqsubset$ canonically to contexts ($\Gamma \sqsubset \overline{\Gamma}$ when they bind the same variables and the types are in relation for $\sqsubset$).

Before defining the translation, we define a set $[\![\Gamma \vdash_{\mathsf{x}} t : A]\!]$ of typing judgments in ITT associated to a typing judgment $\Gamma \vdash_{\mathsf{x}} t : A$ in ETT. The idea is that this set describes all the possible translations that lead to the expected property. When $\overline{\Gamma} \vdash \overline{t} : \overline{A} \in [\![\Gamma \vdash_{\mathsf{x}} t : A]\!]$, we say that $\overline{\Gamma} \vdash \overline{t} : \overline{A}$ realises $\Gamma \vdash_{\mathsf{x}} t : A$. The translation will be given by showing that this set is inhabited by induction on the derivation.

Definition 4.1 (Characterisation of possible translations).

- For any $\vdash_{\mathsf{x}} \Gamma$ we define $[\![\vdash_{\mathsf{x}} \Gamma]\!]$ as a set of valid judgments (in ITT) such that $\vdash \overline{\Gamma} \in [\![\vdash_{\mathsf{x}} \Gamma]\!]$ if and only if $\Gamma \sqsubset \overline{\Gamma}$.
- Similarly, $\overline{\Gamma} \vdash \overline{t} : \overline{A} \in [\![\Gamma \vdash_{\mathsf{x}} t : A]\!]$ iff $\vdash \overline{\Gamma} \in [\![\vdash_{\mathsf{x}} \Gamma]\!]$ and $A \sqsubset \overline{A}$ and $t \sqsubset \overline{t}$.

In order to better master the shape of the produced realiser, we state the following lemma which allows to assume that it has the same head type constructor. This is important for instance for the case of an application, where do not know a priori if the translated function has the type of a dependent product, which is required to be able to use the typing rule for application.

Lemma 4.2. *We can always* choose *types $\overline{T}$ that have the same head constructor as T.*

Proof. Assume we have $\overline{\Gamma} \vdash \overline{t} : \overline{T} \in [\![\Gamma \vdash_{\mathsf{x}} t : T]\!]$. By definition of $\sqsubset$, $T \sqsubset \overline{T}$ means that $\overline{T}$ is shaped $p_*\ q_*\ \ldots\ r_*\ \overline{T}'$ with $\overline{T}'$ having the same head constructor as T. By inversion (2.3), the subterms are typable, including $\overline{T}'$. Actually, from inversion, we even get that the type of is $\overline{T}'$ in a universe. Then, using lemma 3.3 and lemma 3.2, we get $\overline{\Gamma} \vdash e : \overline{T} = \overline{T}'$. We conclude with $\overline{\Gamma} \vdash e_*\ \overline{t} : \overline{T}' \in [\![\Gamma \vdash_{\mathsf{x}} t : T]\!]$. □

Finally, in order for the induction to go through, we need to know that when we have a realiser of a derivation $\Gamma \vdash_{\mathsf{x}} t : T$, we can pick an arbitrary other type realising $\Gamma \vdash_{\mathsf{x}} T$ and still get a new derivation realising $\Gamma \vdash_{\mathsf{x}} t : T$ with that type. This is important for instance for the case of an application, where the type of the domain of the translated function may differ from the type of the translated argument. So we need to be able to change it *a posteriori*.

Lemma 4.3. *When we have $\overline{\Gamma} \vdash \overline{t} : \overline{T} \in [\![\Gamma \vdash_{\mathsf{x}} t : T]\!]$ and $\overline{\Gamma} \vdash \overline{T}' \in [\![\Gamma \vdash_{\mathsf{x}} T]\!]$ then we also have $\overline{\Gamma} \vdash \overline{t}' : \overline{T}' \in [\![\Gamma \vdash_{\mathsf{x}} t : T]\!]$ for some $\overline{t}'$.*

Proof. By definition we have $T \sqsubset \overline{T}$ and $T \sqsubset \overline{T}'$ and thus $T \sim \overline{T}$ and $T \sim \overline{T}'$, implying $\overline{T} \sim \overline{T}'$ by transitivity (3.5). By lemma 3.3 (in the case $\Gamma_1 \equiv \Gamma_2 \equiv \bullet$) we get $\overline{\Gamma} \vdash p : \overline{T} \cong \overline{T}'$ for some p. By lemma 3.2 (and lemma 4.2 to give universes as types to $\overline{T}$ and $\overline{T}'$) we can assume $\overline{\Gamma} \vdash p : \overline{T} = \overline{T}'$. Then $\overline{\Gamma} \vdash p_* \, \overline{t} : \overline{T}'$ is still a translation since $\sqsubset$ ignores transports. □

We can now define the translation. This is done by mutual induction on context well-formedness, typing and conversion derivations. Indeed, in order to be able to produce a realiser by induction, we need to show that every conversion in ETT is translated as an heterogeneous equality in ITT.

Theorem 4.4 (Translation).

- *If* $\vdash_{\mathsf{x}} \Gamma$ *then there exists* $\vdash \overline{\Gamma} \in [\![\vdash_{\mathsf{x}} \Gamma]\!]$,
- *If* $\Gamma \vdash_{\mathsf{x}} t : T$ *then for any* $\vdash \overline{\Gamma} \in [\![\vdash_{\mathsf{x}} \Gamma]\!]$ *there exist* $\overline{t}$ *and* $\overline{T}$ *such that* $\overline{\Gamma} \vdash \overline{t} : \overline{T} \in [\![\Gamma \vdash_{\mathsf{x}} t : T]\!]$,
- *If* $\Gamma \vdash_{\mathsf{x}} u \equiv v : A$ *then for any* $\vdash \overline{\Gamma} \in [\![\vdash_{\mathsf{x}} \Gamma]\!]$ *there exist* $A \sqsubset \overline{A}, A \sqsubset \overline{A}', u \sqsubset \overline{u}, v \sqsubset \overline{v}$ *and* $\overline{e}$ *such that* $\overline{\Gamma} \vdash \overline{e} : \overline{u} \; {}_{\overline{A}}{\cong}_{\overline{A}'} \; \overline{v}$.

Proof. We prove the theorem by induction on the derivation in the extensional type theory. To keep the flow of the presentation, the proof is given in Appendix B. □

We can check that all ETT theorems whose type are typable in ITT have proofs in ITT as well:

Corollary 4.5 (Preservation of ITT). *If* $\vdash_{\mathsf{x}} t : T$ *and* $\vdash T$ *then there exist* $\overline{t}$ *such that* $\vdash \overline{t} : T \in [\![\vdash_{\mathsf{x}} t : T]\!]$.

Proof. Since $\vdash \bullet \in [\![\vdash_{\mathsf{x}} \bullet]\!]$, by Theorem (4.4), there exists $\overline{t}$ and $\overline{T}$ such that $\vdash \overline{t} : \overline{T} \in [\![\vdash_{\mathsf{x}} t : T]\!]$ But as $\vdash T$, we have $\vdash T \in [\![\vdash_{\mathsf{x}} T]\!]$, and, using Lemma 4.3, we obtain $\vdash \overline{t} : T \in [\![\vdash_{\mathsf{x}} t : T]\!]$. □

Corollary 4.6 (Relative consistency). *Assuming ITT is consistent, there is no term* t *such that* $\vdash_{\mathsf{x}} t : \Pi(A : \square_0).\, A$.

Proof. Assume such a t exists. By the Corollary 4.5, because $\vdash \Pi(A : \square_0).\, A$, there exists $\overline{t}$ such that $\vdash \overline{t} : \Pi(A : \square_0).\, A$ which contradicts the assumed consistency of ITT. □

5 FORMALISATION WITH TEMPLATE-COQ

We have formalised the translation (provided as an anonymous supplementary material) in the setting of TemplateCoq [Anand et al. 2018] in order to have a more precise proof, but also to evidence the fact that the translation is indeed constructive and can be used to perform computations.

TemplateCoq is a Coq library that has a representation of Coq terms as they are in Coq kernel (in particular using de Bruijn indices for variables), a (partial) implementation of the type checking algorithm and so on. It comes with a Coq plugin that allows to quote Coq terms into their representations, and to produce Coq terms from their representation (if they indeed denote well-typed terms). We decided to integrate our formalisation within that framework in order to ensure we had a theory close to Coq's but also to take advantage of the quoting mechanism to produce terms using interactive mode (in particular we get to use tactics). Note that we also rely on Mangin and Sozeau's Equation [Mangin and Sozeau 2017] plugin to derive nice dependent induction principles. You can see the organisation of the formalisation in Table 1.

Our formalisation takes full advantage of its easy interfacing with TemplateCoq: we define two theories, namely ETT and ITT, but ITT enjoys a lot of syntactic sugar by having things such as transport, heterogeneous equality and packing as part of the syntax. The operations regarding them—in particular the tedious ones—are written in Coq and then quoted to be used in the translation from ITT to TemplateCoq.

Table 1. Organisation of the files

SAst.v	Common syntax to ETT and ITT
SLiftSubst.v	Lifting and substitution on this syntax, together with helpful lemmata
SCommon.v	Common definitions such as contexts
XTyping.v	Typing rules of ETT
ITyping.v	Typing rules of ITT and lemmata
PackLifts.v	Notion of pack and associated lifts and lemmata
Translation.v	Main lemmata and theorem (core of the formalisation)
Reduction.v	Simplification process on ITT terms
Quotes.v	Definition and quoting of terms to realise the ones assumed in ITT
FinalTranslation.v	Translation from ITT to TemplateCoq
Example.v	Examples of translations from ETT to Coq

ITT to TemplateCoq. Note that the translation from ITT to TemplateCoq is not proven correct, but it is not really important as it can just be seen as a feature to observe the produced terms in a nicer setting. In any case, TemplateCoq does not yet provide a complete formalization of CIC rules, as guard checking of recursive definitions and strict positivity of inductive type declarations are not formalized yet.

5.1 Representation of ETT and ITT

The file SAst.v contains the definition of sorts and inductive types representing the syntax of ETT and ITT. We chose de Bruijn indices for variables as it is already like this in Coq and TemplateCoq. For sorts, we simply use the type of natural numbers to setup the hierarchy.

```
Definition sort := nat.

Inductive sterm : Type :=
| sRel (n : nat)
| sSort (s : sort)
| sProd (nx : name) (A B : sterm)
| sLambda (nx : name) (A B t : sterm)
| sApp (u : sterm) (nx : name) (A B v : sterm)
(* Homogenous equality *)
| sEq (A u v : sterm)
| sRefl (A u : sterm)
| (* ... *) .
```

Because we use de Bruijn indices, we also need to implement lifting operations and substitution as is done in SLiftSubst.v. This allows us to write `lift0 n t` to add `n` to all variables in `t` and `t{ n := u }` to substitute the `n`th variable in `t` by `u`.

SCommon.v introduces notions that are also common to ETT and ITT such as contexts and associated notations. A context is a list of declarations and can be either `nil` or the extension of another context `Γ ,, svass na A` where `na` is a name (or identifier—it is only used for printing) and `A` is a type.

We then introduce in ITyping.v (and similarly in XTyping.v for ETT) the typing rules of ITT. They are represented by three mutual inductive types for typing, well-formedness of contexts and conversion respectively. Note that all the derivations inductive families are in **Type**, i.e. are considered as computational, syntactic objects: they are the central structures in this work, and the translation is pattern-matching on their shape, hence the cannot be put in **Prop** as usual. Moreover,

to derive useful elimination principles on derivations, we sometimes employ orders including the size measure of a derivation, which can only be defined if they are informative.

For compatibility with TemplateCoq we have a `global_context` that basically contains named definitions (of inductives and constants or axioms). The `safe_nth` function retrieves the `n`th element of a list, when provided with a proof that it exists. Figure 7 shows excerpts of this mutual inductive type for ITT.

In `XTyping.v` we also have the reflection rule as one of the conversion rules.

```
with eq_term Σ : scontext -> sterm -> sterm -> sterm -> Type :=
(* ... *)
| reflection Γ A u v e :
  Σ ;;; Γ |-x e : sEq A u v ->
  Σ ;;; Γ |-x u = v : A
```

Similarly, we define $\sqsubset$ and $\sim$ using inductive types.

5.2 Main results

`Translation.v` is where all the interesting things happen. First is the fundamental lemma (3.3) that reconstructs heterogeneous equalities between terms in relation.

```
Lemma trel_to_heq' :
  forall {Σ t1 t2},
    t1 ∼ t2 ->
    forall {Γ Γ1 Γ2 Γm T1 T2},
      ismix Σ Γ Γ1 Γ2 Γm ->
      Σ ;;; Γ ,,, Γ1 |-i t1 : T1 ->
      Σ ;;; Γ ,,, Γ2 |-i t2 : T2 ->
      ∑ p,
        Σ ;;; Γ ,,, Γm |-i p : sHeq (llift0 #|Γm| T1)
                                    (llift0 #|Γm| t1)
                                    (rlift0 #|Γm| T2)
                                    (rlift0 #|Γm| t2).
```

The `ismix Σ Γ Γ1 Γ2 Γm` predicate states that `Γm` is a proper packing of contexts `Γ1` and `Γ2` above `Γ` (the latter is necessary to check well-formedness of types in the same sorts). Also note that $\sum$ is a notation for the existential quantifier of `Type` and `#|Γm|` is a notation for the length of `Γm`, while `llift` and `rlift` correspond to the operations _ ↑ and _ ↑ defined in Section 3. They are defined in `PackLifts.v`.

Before we have a look at the translation, we define the notion of translation, similarly to their theoretical counterparts in Section 4.

```
Definition trans Σ Γ A t Γ' A' t' := Γ ⊂ Γ' * A ⊏ A' * t ⊏ t' * Σ ;;; Γ' |-i t' : A'.

Notation " Σ ;;;; Γ' |--- [ t' ] : A' # ⟦ Γ |--- [ t ] : A ⟧ " :=
  (trans Σ Γ A t Γ' A' t') (at level 7) : i_scope.

Definition ctxtrans Σ Γ Γ' := Γ ⊂ Γ' * (wf Σ Γ').

Notation " Σ |--i Γ' # ⟦ Γ ⟧ " := (ctxtrans Σ Γ Γ') (at level 7) : i_scope.

(* ... *)

Definition eqtrans Σ Γ A u v Γ' A' A'' u' v' p' :=
  Γ ⊂ Γ' * A ⊏ A' * A ⊏ A'' * u ⊏ u' * v ⊏ v' * (Σ ;;; Γ' |-i p' : sHeq A' u' A'' v').
```

```
Inductive typing (Σ : global_context) : scontext -> sterm -> sterm -> Type :=
| type_Rel Γ n :
  wf Σ Γ ->
  forall (isdecl : n < List.length Γ),
    Σ ;;; Γ |-i (sRel n)
    : lift0 (S n) (safe_nth Γ (exist _ n isdecl)).(sdecl_type)

| type_Sort Γ s :
  wf Σ Γ ->
  Σ ;;; Γ |-i (sSort s) : sSort (succ_sort s)

| type_Prod Γ n t b s1 s2 :
  Σ ;;; Γ |-i t : sSort s1 ->
  Σ ;;; Γ ,, svass n t |-i b : sSort s2 ->
  Σ ;;; Γ |-i (sProd n t b) : sSort (max_sort s1 s2)

| type_conv Γ t A B s :
  Σ ;;; Γ |-i t : A ->
  Σ ;;; Γ |-i B : sSort s ->
  Σ ;;; Γ |-i A = B : sSort s ->
  Σ ;;; Γ |-i t : B

(* ... *)

where " Σ ;;; Γ '|-i' t : T " := (@typing Σ Γ t T) : i_scope

with wf (Σ : global_context) : scontext -> Type :=
| wf_nil :
  wf Σ nil

| wf_snoc s Γ x A :
  wf Σ Γ ->
  Σ ;;; Γ |-i A : sSort s ->
  wf Σ (Γ ,, svass x A)

with eq_term Σ : scontext -> sterm -> sterm -> sterm -> Type :=
| eq_beta Γ s1 s2 n A B t u :
  Σ ;;; Γ |-i A : sSort s1 ->
  Σ ;;; Γ ,, svass n A |-i B : sSort s2 ->
  Σ ;;; Γ ,, svass n A |-i t : B ->
  Σ ;;; Γ |-i u : A ->
  Σ ;;; Γ |-i sApp (sLambda n A B t) n A B u = t{ 0 := u } : B{ 0 := u }

(* ... *)
```

Fig. 7. Inductive types for typing of ITT in TemplateCoq

Finally we state the main theorem as a mutal induction over context well-formedness, typing and conversion derivations.

```
Theorem complete_translation {Σ} :
  (forall Γ (h : XTyping.wf Σ Γ), ∑ Γ', Σ |--i Γ' # ⟦ Γ ⟧ ) *
  (forall {Γ t A} (h : Σ ;;; Γ |-x t : A)
          {Γ'} (hΓ : Σ |--i Γ' # ⟦ Γ ⟧),
    ∑ A' t', Σ ;;;; Γ' |--- [t'] : A' # ⟦ Γ |--- [t] : A ⟧) *
  (forall { Γ u v A} (h : Σ ;;; Γ |-x u = v : A)
          {Γ'} (hΓ : Σ |--i Γ' # ⟦ Γ ⟧),
    ∑ A' A'' u' v' p',
      eqtrans Σ Γ A u v Γ' A' A'' u' v' p').
```

The fact that the theorem holds in Coq ensures we can actually compute a translated term and type out of a derivation in ETT.

```
Definition type_translation {Σ Γ t A} h {Γ'} hΓ :=
  pi2_ _ _ (pi1_ _ _ (@complete_translation Σ)) Γ t A h Γ' hΓ.

(* type_translation
       : ?Σ;;; ?Γ |-x ?t : ?A ->
         forall Γ' : scontext,
         ctxtrans ?Σ ?Γ Γ' -> ∑ A' t' : sterm, trans ?Σ ?Γ ?A ?t Γ' A' t'
 *)
```

5.3 Challenges of the formalisation

Our paper version differs from Oury's in several aspects. One is of course the type annotations that free us from the need of an axiom. Another is the use of a different (but equivalent) notion of heterogeneous equality that fits best in homotopy type theory while making more obvious the uses of axiom K. There are however some changes that are due to the formalisation, namely the use of packing to deal with diverging dependencies. Oury had a different way of dealing with them by using choice on a set of pairs of variables that were heterogeneously equal. This doesn't work well with De Bruijn indices and isn't really suited to a constructive presentation.

The packed extensions of context are very similar to the binary variant of the parametricity translation [Bernardy et al. 2012]. It would be interesting to investigate this more.

Another point worth mentioning is the fact that for the translation from ITT to TemplateCoq, for efficiency purposes, we couldn't reuse the whole graph of universe constraints of Coq and instead rely on the 'Type in Type' option. This does not affect in any way the translation from ETT to ITT that represents the core of our work.

5.4 Simplification for reduced usage of axioms

As in the paper proof, the formalised translation refers a lot to the fundamental lemma (3.3) (mostly indirectly via Lemmata 4.2 and 4.3). For instance, in the application rule, we want to make sure that the translated function is indeed a function, *i.e.,* its type is indeed a product; in order to make sure of that we transport the function into a Π-type. In most instances (e.g. when translating ITT derivations—which are valid ETT derivations), this will result in transport along equalities between the same type and useless transports poluting the final term. Even worse, this might result in the use of axiom K on reflexivity and unfortunately even in such case, the axiom (in Coq) doesn't reduce to reflexivity. For this we introduce an extra step (in `Reduction.v`) that simplifies the generated terms, in particular removing useless transports and making axioms compute on reflexivity. At the time of writing, this operation is not yet proven correct and should rather be considered as a prism to observe the resulting terms.

5.5 Examples

A few examples are shown in `Example.v`. They consist of ETT derivations that we translate into an ITT term that gets reduced and then translated to a TemplateCoq term and finally into a Coq term.

The first example we consider is a simple use of reflection, turning a type equality into a coercion: $\lambda\, A\, B\, e\, x \Rightarrow x : \forall\, (A\, B : \square_0), A = B \to A \to B$. This corresponds to the ETT term

```
Definition tm :=
  sLambda (nNamed "A") _ _
    (sLambda (nNamed "B") _ _
      (sLambda (nNamed "e") _ _
        (sLambda (nNamed "x") _ _ (sRel 0))))
```

where we use _ for type annotations in order to keep the term legible. We give it the following type with the corresponding derivation using reflection exactly once.

```
Definition ty :=
  sProd (nNamed "pppp") (sSort 0)
    (sProd (nNamed "pppp") (sSort 0)
      (sProd (nNamed "pppp") (sEq (sSort 0) (sRel 1) (sRel 0))
        (sProd (nNamed "pppp") (sRel 2) (sRel 2))))

Fact tmty : Σ ;;; [] |-x tm : ty.
Proof. (* ... *) Defined.
```

We then translate the derivation to ITT using our certified translation of Section 5.2.

```
Definition itt_tm : sterm.
  destruct (type_translation tmty istrans_nil) as [A [t h]].
  exact t.
Defined.
```

We do not print the result as it takes more that 100,000 characters... This is because the translation introduces a lot of transports that happen to be unnecessary in the end. We can remove those useless transports using the `reduce` procedure.

```
Definition red_itt_tm := reduce itt_tm.
```

This produces the following ITT term.

```
sLambda (nNamed "A") _ _
  (sLambda (nNamed "B") _ _
    (sLambda (nNamed "e") _ _
      (sLambda (nNamed "x") _ _
        (sTransport _ _ (sRel 1) (sRel 0)))))
```

We can then translate it to TemplateCoq. The `2^18` integer is the fuel needed for the typing inference that is done by the reduction procedure, which returns a value in the `tsl_result` monad (which is coarsely the option monad) because the type inference procedure may fail in general and does not always return a value.

```
Definition tc_red_tm : tsl_result term :=
  tsl_rec (2 ^ 18) Σ [] red_itt_tm.
```

Once we have the term in TemplateCoq, we can use the plugin to compute its denotation in Coq.

```
Make Definition coq_red_tm :=
  ltac:(
    let t := eval lazy in
             (match tc_red_tm with
```

```
              | Success t => t
              | _ => tSort Universe.type0
              end)
      in exact t
  ).
```

Finally, `coq_red_tm` computes to

```
fun (A B : Type) (e : A = B) (x : A) =>
  transport e x
: forall A B : Type, A = B -> A -> B
```

that is to say: $\lambda\ A\ B\ e\ x \Rightarrow e_*\ x : \forall\ (A\ B : \square_0), A = B \to A \to B$. This corresponds exactly to the naive translation of ETT to ITT that just replaces uses of reflection by transports above the equality witnesses.

Another example is that of the identity function $\lambda\ A\ x \Rightarrow x : \forall\ (A : \square_0), A \to A$, provided with its usual ITT derivation (without any use of reflection). The translation (composed with the simplication procedure) is the term itself. This is also reassuring as it tends to show experimentally that the translation picks up the right term (as defined in Corollary (4.5)) in all those in the relation.

5.6 Further work: Inductive types

We firmly believe that this translation can be extended to inductive types. More generally it should scale to the whole Calculus of Inductive Constructions [Bertot and Castéran 2004] (CIC), the theory behind Coq. The main obstacle is the generation of lemmata such as congruences for heterogeneous equality (see `Quotes.v`) that would need to be automatised.

5.7 Towards a plugin for a fragment of ETT

As we already mentioned we can already turn an ETT derivation into a Coq term. Besides the inductive types, what's left to do before we can have a complete plugin is to offer a frontend to write such ETT derivations in Coq syntax. One such way would be close to the 'Program' construction of Coq. The idea is that the user writes a term as usual in Coq, except that during typechecking, when conversion fails, instead of failing globally it can instead produce corresponding obligations of equality using the reflection rule. In this mode, the refinement engine would mark such uses in the term, in order to be able to reconstruct a derivation. For instance, let us consider the case of vectors:

```
Inductive vec A : nat -> Type :=
| vnil : vec A 0
| vcons : A -> forall n, vec A n -> vec A (S n).

Arguments vnil {_}.
Arguments vcons {_} _ _ _.

Fixpoint rev {A n m} (v : vec A n) (acc : vec A m) : vec A (n + m) :=
  match v with
  | vnil => acc
  | vcons a n' v' => rev v' (vcons a m acc)
  end.
```

If the user types this, Coq will complain that `rev v' (vcons a m acc)` has type `vec A (n' + S m)` whereas `vec A (S n' + m)` was expected. To avoid this, we need to transport along a proof of the equality `n' + S m = S n' + m`. In ETT, thanks to the reflection rule, the terms become convertible

and such a definition can be accepted as is. We cannot expect Coq to find it on its own everytime as type checking is undecidable, thus the plugin would allow something closer to the following.

```
Extensional Fixpoint rev {A n m} (v : vec A n) (acc : vec A m) : vec A (n+m) :=
  match v with
  | vnil => acc
  | vcons a n' v' => rev v' (vcons a m acc)
  end.
Next Obligation. omega. (* Proof that n' + S m = S n' + m *) Qed.
```

This is however not so simple as it would rely on a typed conversion checking and not on the comparision of normal forms present in Coq. This would be closer to how Agda [Norell 2007] works with an algorithmic equality [Abel 2011; Harper and Pfenning 2005]. Furthermore we would need type checker to issue a certificate that would then be used to reconstruct the ETT derivation.

Additionally, this wouldn't capture the whole of ETT as the types might not even make sense in ITT and it isn't clear what to do in those cases. In practice this would be enough to deal with examples such as the one above which might be good enough in a lot of cases.

5.8 Composition with other transations

This translation also allows for the formalisation of translations that target ETT rather than ITT and still get mechanised proofs of (relative) consistency by composition with this ETT to ITT translation. This could also be used to implement plugins based on the composition of translations. In particular if the translation targetting ETT is able to construct a derivation without having a derivation in the source, the composition could retain decidability of type checking. This would provide a simple way to justify the consistency of CoqMT for example, seeing it as an extensional type theory where reflection is restricted to equalities on a specific domain whose theory is decidable [Jouannaud and Strub 2017].

6 RELATED WORKS AND CONCLUSION

The seminal works on the precise connection between ETT and ITT go back to Streicher [Streicher 1993] and Hofmann [Hofmann 1995, 1997]. In particular, the work of Hofmann provides a categorical answer to the question of consistency and conservativity of ETT over ITT with UIP and functional extensionality. Ten years later, Oury [Oury 2005, 2006] provided a translation from ETT to ITT with UIP and functional extensionality and other axioms (mainly due to technical difficulties). Although a first step towards a move from categorical semantics to a syntactic translation, his work does not stress any constructive aspect of the proof and shows that there merely exist translations in ITT to a typed term in ETT.

van Doorn *et al.* [van Doorn et al. 2013] have later proposed and formalised a similar translation between a PTS with and without explicit conversion. This does not entail anything about ETT to ITT but we can find similarities in that any there is a witness of conversion between any term and itself under an explicit conversion, which corresponds in a way to a witness of uniqueness of explicit conversions.

In this paper we provide the first effective translation from ETT to ITT with UIP and functional extensionality. The translation has been formalised in Coq using TemplateCoq, a plugin of Coq to do metaprogramming in Coq. This translation is also effective in the sense that we can produce in the end a Coq term using the TemplateCoq denotation machinery. With ongoing work to extend the translation to the inductive fragment of Coq, we are paving the way to an extensional version of the Coq proof assistant which could be translated back to its intensional version, allowing the user to navigate between the two modes, and in the end produce a proof term checkable in the intensional fragment.

REFERENCES

Andreas Abel. 2011. Irrelevance in type theory with a heterogeneous equality judgement. In *International Conference on Foundations of Software Science and Computational Structures*. Springer, 57–71.

Abhishek Anand, Simon Boulier, Nicolas Tabareau, and Matthieu Sozeau. 2018. Typed Template Coq – Certified Meta-Programming in Coq. In *The Fourth International Workshop on Coq for Programming Languages*. Los Angeles, CA, United States. https://hal.inria.fr/hal-01671948

Andrej Bauer, Gaëtan Gilbert, Philipp G. Haselwarter, Matija Pretnar, and Chris Stone. 2016. The 'Andromeda' prover. (2016). http://www.andromeda-prover.org/

Jean-philippe Bernardy, Patrik Jansson, and Ross Paterson. 2012. Proofs for free: Parametricity for dependent types. *Journal of Functional Programming* 22, 2 (2012), 107–152.

Yves Bertot and Pierre Castéran. 2004. Interactive Theorem Proving and Program Development. (2004).

Marc Bezem, Thierry Coquand, and Simon Huber. 2013. A Model of Type Theory in Cubical Sets. (December 2013). http://www.cse.chalmers.se/~coquand/mod1.pdf

Simon Boulier, Pierre-Marie Pédrot, and Nicolas Tabareau. 2017. The Next 700 Syntactical Models of Type Theory. In *Certified Programs and Proofs – CPP 2017*. 182–194.

Robert L. Constable and Joseph L. Bates. 2014. The NuPrl system, PRL project. (2014). http://www.nuprl.org/

The Coq development team. 2017. *The Coq proof assistant reference manual*. LogiCal Project. http://coq.inria.fr Version 8.7.

Robert Harper and Frank Pfenning. 2005. On equivalence and canonical forms in the LF type theory. *ACM Transactions on Computational Logic (TOCL)* 6, 1 (2005), 61–101.

Martin Hofmann. 1995. Conservativity of equality reflection over intensional type theory. In *International Workshop on Types for Proofs and Programs*. Springer, 153–164.

Martin Hofmann. 1997. *Extensional constructs in intensional type theory*. Springer.

Martin Hofmann and Thomas Streicher. 1998. The Groupoid Interpretation of Type Theory. In *Twenty-five years of constructive type theory (Venice, 1995)*. Oxford Logic Guides, Vol. 36. Oxford Univ. Press, New York, 83–111. http://www.tcs.informatik.uni-muenchen.de/lehre/SS97/types-vl/venedig.ps

Jean-Pierre Jouannaud and Pierre-Yves Strub. 2017. Coq without Type Casts: A Complete Proof of Coq Modulo Theory. In *LPAR-21, 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Maun, Botswana, May 7-12, 2017 (EPiC Series in Computing)*, Thomas Eiter and David Sands (Eds.), Vol. 46. EasyChair, 474–489. http://www.easychair.org/publications/paper/340342

Chris Kapulkin and Peter LeFanu Lumsdaine. 2012. The simplicial model of univalent foundations. *arXiv preprint arXiv:1211.2851* (2012).

Cyprien Mangin and Matthieu Sozeau. 2017. Equations - a function definition plugin. (2017). http://mattam82.github.io/Coq-Equations/

Conor McBride. 2000. *Dependently typed functional programs and their proofs*. Ph.D. Dissertation. University of Edinburgh.

Ulf Norell. 2007. *Towards a practical programming language based on dependent type theory*. Vol. 32. Citeseer.

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

Nicolas Oury. 2006. *Egalité et filtrage avec types dépendants dans le calcul des constructions inductives*. Ph.D. Dissertation. http://www.theses.fr/2006PA112136 ThÃšse de doctorat dirigÃ©e par Paulin-Mohring, Christine Informatique Paris 11 2006.

Thomas Streicher. 1993. *Investigations into intensional type theory*.

The Univalent Foundations Program. 2013. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study.

Floris van Doorn, Herman Geuvers, and Freek Wiedijk. 2013. Explicit convertibility proofs in pure type systems. In *Proceedings of the Eighth ACM SIGPLAN international workshop on Logical frameworks & meta-languages: theory & practice*. ACM, 25–36.

A PROOF OF THE FUNDAMENTAL LEMMA

Lemma A.1 (Fundamental lemma). *Let t_1 and t_2 be two terms. If $\Gamma, \Gamma_1 \vdash t_1 : T_1$ and $\Gamma, \Gamma_2 \vdash t_2 : T_2$ and $t_1 \sim t_2$ then there exists p such that $\Gamma, \mathsf{Pack}\ \Gamma_1\ \Gamma_2 \vdash p : t_1\ \lceil_{T_1\rceil} \cong_{T_2\rceil}\ t_2\ \rceil$.*

Proof. We prove it by induction on the derivation of $t_1 \sim t_2$.

- $$\frac{}{x \sim x}$$

If x belongs to Γ, we apply reflexivity—together with uniqueness of typing (2.2)—to conclude. Otherwise, $\mathsf{Proj}_{\mathsf{e}}\ x$ has the expected type (since $x \downarrow \equiv \mathsf{Proj}_1\ x$ and $x \uparrow \equiv \mathsf{Proj}_2\ x$).

- $$\frac{t_1 \sim t_2}{p_*\ t_1 \sim t_2}$$

  We have $\Gamma, \Gamma_1 \vdash p_*\ t_1 : T_1$ and $\Gamma, \Gamma_2 \vdash t_2 : T_2$. By inversion (2.3) we have $\Gamma, \Gamma_1 \vdash p : T_1' = T_1$ and $\Gamma, \Gamma_1 \vdash t_1 : T_1'$. Then by induction hypothesis we have e such that $\Gamma, \Gamma_{\mathsf{p}} \vdash e : t_1 \downarrow \cong t_2 \uparrow$. From transitivity and symmetry we only need to provide a proof of $t_1 \downarrow \cong p \downarrow_*\ t_1 \downarrow$ which is inhabited by $\langle p \downarrow ; \mathsf{refl}\ (p \downarrow_*\ t_1 \downarrow)\rangle_{\_,\_}$.

- $$\frac{t_1 \sim t_2}{t_1 \sim p_*\ t_2}$$

  Similarly.

- $$\frac{A_1 \sim A_2 \qquad B_1 \sim B_2}{\Pi(x : A_1).\, B_1 \sim \Pi(x : A_2).\, B_2}$$

  We have $\Gamma, \Gamma_1 \vdash \Pi(x : A_1).\, B_1 : T_1$ and $\Gamma, \Gamma_2 \vdash \Pi(x : A_2).\, B_2 : T_2$ so by inversion (2.3) we have $\Gamma, \Gamma_1 \vdash A_1 : s_1$ and $\Gamma, \Gamma_1, x : A_1 \vdash B_1 : s_1'$ and $\Gamma, \Gamma_1 \vdash s_1'' \equiv T_1$ for $(s_1, s_1', s_1'') \in \mathsf{R}$ (and similarly with 2s). By induction hypothesis we have $\Gamma, \Gamma_{\mathsf{p}} \vdash p_A : A_1 \downarrow \cong A_2 \uparrow$ and $\Gamma, \Gamma_{\mathsf{p}}, x : \mathsf{Pack}\ A_1\ A_2 \vdash p_B : B_1 \downarrow \cong B_2 \uparrow$ hence the result (using UIP and functional extensionality, refer to the formalisation and especially to the file `Quotes.v` for more details on how to realise this equality).

- $$\frac{A_1 \sim A_2 \qquad u_1 \sim u_2 \qquad v_1 \sim v_2}{u_1 =_{A_1} v_1 \sim u_2 =_{A_2} v_2}$$

  We have $\Gamma, \Gamma_1 \vdash u_1 =_{A_1} v_1 : T_1$ and $\Gamma, \Gamma_2 \vdash u_2 =_{A_2} v_2 : T_2$ so, by inversion (2.3), we have $\Gamma, \Gamma_1 \vdash A_1 : s_1$ and $\Gamma, \Gamma_1 \vdash u_1 : A_1$ and $\Gamma, \Gamma_1 \vdash v_1 : A_1$ as well as $\Gamma, \Gamma_1 \vdash s_1 \equiv T_1$ (and the same with 2s). By induction hypothesis we thus have $\Gamma, \Gamma_{\mathsf{p}} \vdash p_A : A_1 \cong A_2$ and $\Gamma, \Gamma_{\mathsf{p}} \vdash p_u : u_1 \cong u_2$ and $\Gamma, \Gamma_{\mathsf{p}} \vdash p_v : v_1 \cong v_2$. We can thus conclude.

- $$\frac{}{s \sim s}$$

  This one holds by reflexivity and uniqueness of typing (2.2) (indeed, $s \downarrow \equiv s$ and $s \uparrow \equiv s$).

- $$\frac{A_1 \sim A_2 \qquad B_1 \sim B_2 \qquad t_1 \sim t_2}{\lambda(x : A_1).B_1.t_1 \sim \lambda(x : A_2).B_2.t_2}$$

  We have $\Gamma, \Gamma_1 \vdash \lambda(x : A_1).B_1.t_1 : T_1$ and $\Gamma, \Gamma_2 \vdash \lambda(x : A_2).B_2.t_2 : T_2$, thus, by inversion 2.3 the subterms are well-typed and we can apply induction hypothesis. The conclusion follows similarly to the Π case.

- $$\frac{t_1 \sim t_2 \qquad A_1 \sim A_2 \qquad B_1 \sim B_2 \qquad u_1 \sim u_2}{t_1\ @_{x:A_1.B_1}\ u_1 \sim t_2\ @_{x:A_2.B_2}\ u_2}$$

  We have $\Gamma, \Gamma_1 \vdash t_1\ @_{x:A_1.B_1}\ u_1 : T_1$ and $\Gamma, \Gamma_2 \vdash t_2\ @_{x:A_2.B_2}\ u_2 : T_2$ which means by inversion (2.3) that the subterms are well-typed. We apply the induction hypothesis and then conclude.

- $$\frac{A_1 \sim A_2 \qquad u_1 \sim u_2}{\mathsf{refl}_{A_1}\ u_1 \sim \mathsf{refl}_{A_2}\ u_2}$$

  We have $\Gamma, \Gamma_1 \vdash \mathsf{refl}_{A_1}\ u_1 : T_1$ and $\Gamma, \Gamma_2 \vdash \mathsf{refl}_{A_2}\ u_2 : T_2$ so by inversion (2.3) we have $\Gamma, \Gamma_1 \vdash A_1 : s_1$ and $\Gamma, \Gamma_1 \vdash u_1 : A_1$ (same with 2s). By IH we have $A_1 \uparrow \cong A_2 \uparrow$ and $u_1 \uparrow_{A_1\uparrow} \cong_{A_2\uparrow} u_2 \uparrow$. The proof follows easily.
- $$\frac{A_1 \sim A_2 \qquad B_1 \sim B_2 \qquad f_1 \sim f_2 \qquad g_1 \sim g_2 \qquad e_1 \sim e_2}{\mathsf{funext}(x : A_1, B_1, f_1, g_1, e_1) \sim \mathsf{funext}(x : A_2, B_2, f_2, g_2, e_2)}$$

  Similar.
- $$\frac{A_1 \sim A_2 \qquad u_1 \sim u_2 \qquad v_1 \sim v_2 \qquad p_1 \sim p_2 \qquad q_1 \sim q_2}{\mathsf{uip}(A_1, u_1, v_1, p_1, q_1) \sim \mathsf{uip}(A_2, u_2, v_2, p_2, q_2)}$$

  Similar.
- $$\frac{A_1 \sim A_2 \qquad u_1 \sim u_2 \qquad P_1 \sim P_2 \qquad w_1 \sim w_2 \qquad v_1 \sim v_2 \qquad p_1 \sim p_2}{\mathsf{J}(A_1, u_1, x.e.P_1, w_1, v_1, p_1) \sim \mathsf{J}(A_2, u_2, x.e.P_2, w_2, v_2, p_2)}$$

  Similar.

□

B CORRECTNESS OF THE TRANSLATION

Theorem B.1 (Translation).

- *If* $\vdash_{\mathsf{x}} \Gamma$ *then there exists* $\vdash \overline{\Gamma} \in [\![\vdash_{\mathsf{x}} \Gamma]\!]$,
- *If* $\Gamma \vdash_{\mathsf{x}} t : T$ *then for any* $\vdash \overline{\Gamma} \in [\![\vdash_{\mathsf{x}} \Gamma]\!]$ *there exist* $\overline{t}$ *and* $\overline{T}$ *such that* $\overline{\Gamma} \vdash \overline{t} : \overline{T} \in [\![\Gamma \vdash_{\mathsf{x}} t : T]\!]$,
- *If* $\Gamma \vdash_{\mathsf{x}} u \equiv v : A$ *then for any* $\vdash \overline{\Gamma} \in [\![\vdash_{\mathsf{x}} \Gamma]\!]$ *there exist* $A \sqsubset \overline{A}, A \sqsubset \overline{A}', u \sqsubset \overline{u}, v \sqsubset \overline{v}$ *and* $\overline{e}$ *such that* $\overline{\Gamma} \vdash \overline{e} : \overline{u}\ {}_{\overline{A}}{\cong}_{\overline{A}'}\ \overline{v}$.

Proof. We prove the theorem by induction on the derivation in the extensional type theory. In most cases we need to assume some $\overline{\Gamma}$, translation of the context, we will implictely refer to $\overline{\Gamma}$ in such cases as the one given as hypothesis.

- $$\frac{}{\vdash_{\mathsf{x}} \bullet}$$

  We have $\vdash \bullet \in [\![\vdash_{\mathsf{x}} \bullet]\!]$.
- $$\frac{\vdash_{\mathsf{x}} \Gamma \qquad \Gamma \vdash_{\mathsf{x}} A}{\vdash_{\mathsf{x}} \Gamma, x : A}(x \notin \Gamma)$$

  By IH we have $\vdash \overline{\Gamma} \in [\![\vdash_{\mathsf{x}} \Gamma]\!]$ and, using $\overline{\Gamma}$ as well as lemma 4.2, $\overline{\Gamma} \vdash \overline{A} : s \in [\![\Gamma \vdash_{\mathsf{x}} A : s]\!]$. Thus $\vdash \overline{\Gamma}, x : \overline{A} \in [\![\vdash_{\mathsf{x}} \Gamma, x : A]\!]$.
- $$\frac{\vdash_{\mathsf{x}} \Gamma}{\Gamma \vdash_{\mathsf{x}} s : s'}(s, s')$$

  We have $\overline{\Gamma} \vdash s : s' \in [\![\Gamma \vdash_{\mathsf{x}} s : s']\!]$.

- $$\frac{\Gamma \vdash_{\mathsf{x}} A : s \qquad \Gamma, x : A \vdash_{\mathsf{x}} B : s'}{\Gamma \vdash_{\mathsf{x}} \Pi(x : A).\, B : s''}(s, s', s'')$$

  By IH and lemma 4.2 we have $\overline{\Gamma} \vdash \overline{A} : s$, meaning $\vdash \overline{\Gamma}, x : \overline{A} \in [\![\vdash_{\mathsf{x}} \Gamma, x : A]\!]$, and then $\overline{\Gamma}, x : \overline{A} \vdash \overline{B} : s'$. We thus conclude $\overline{\Gamma} \vdash \Pi(x : \overline{A}).\, \overline{B} : s'' \in [\![\Gamma \vdash_{\mathsf{x}} \Pi(x : A).\, B : s'']\!]$.

- $$\frac{\Gamma \vdash_{\mathsf{x}} A : s \qquad \Gamma, x : A \vdash_{\mathsf{x}} B : s'}{\Gamma \vdash_{\mathsf{x}} \Sigma(x : A).\, B : s''}(s, s', s'')$$

  Similar.

- $$\frac{\Gamma \vdash_{\mathsf{x}} A : s \qquad \Gamma \vdash_{\mathsf{x}} u : A \qquad \Gamma \vdash_{\mathsf{x}} v : A}{\Gamma \vdash_{\mathsf{x}} u =_A v : s}$$

  By IH and lemma 4.2 we have $\overline{\Gamma} \vdash \overline{A} : s$, and—using lemma 4.3—we also have $\overline{\Gamma} \vdash \overline{u} : \overline{A}$ and $\overline{\Gamma} \vdash \overline{v} : \overline{A}$. Then $\overline{\Gamma} \vdash \overline{u} =_{\overline{A}} \overline{v} : s \in [\![\Gamma \vdash_{\mathsf{x}} u =_A v : s]\!]$.

- $$\frac{\vdash_{\mathsf{x}} \Gamma \qquad (x : A) \in \Gamma}{\Gamma \vdash_{\mathsf{x}} x : A}$$

  We have $\vdash \overline{\Gamma} \in [\![\vdash_{\mathsf{x}} \Gamma]\!]$ (as we assumed, this is not an instance of the induction hypothesis) and $(x : A) \in \Gamma$. By definition of $\Gamma \sqsubset \overline{\Gamma}$ we also have some $(x : \overline{A}) \in \overline{\Gamma}$ with $A \sqsubset \overline{A}$, thus $\overline{\Gamma} \vdash x : \overline{A} \in [\![\Gamma \vdash_{\mathsf{x}} x : A]\!]$.

- $$\frac{\Gamma \vdash_{\mathsf{x}} u : A \qquad \Gamma \vdash_{\mathsf{x}} A \equiv B}{\Gamma \vdash_{\mathsf{x}} u : B}$$

  By IH and lemma 3.2 we have $\overline{\Gamma} \vdash \overline{e} : \overline{A} = \overline{B}$ which implies $\overline{\Gamma} \vdash \overline{A} \in [\![\Gamma \vdash_{\mathsf{x}} A]\!]$ by inversion (2.3), thus, from lemma 4.3 and IH we get $\overline{\Gamma} \vdash \overline{u} : \overline{A}$, yielding $\overline{\Gamma} \vdash \overline{e}_* \, \overline{u} : \overline{B} \in [\![\Gamma \vdash_{\mathsf{x}} u : B]\!]$.

- $$\frac{\Gamma \vdash_{\mathsf{x}} A : s \qquad \Gamma, x : A \vdash_{\mathsf{x}} B : s' \qquad \Gamma, x : A \vdash_{\mathsf{x}} t : B}{\Gamma \vdash_{\mathsf{x}} \lambda(x : A).B.t : \Pi(x : A).\, B}$$

  By IH and lemma 4.2 we have $\overline{\Gamma} \vdash \overline{A} : s$ and thus $\vdash \overline{\Gamma}, x : \overline{A} \in [\![\vdash_{\mathsf{x}} \Gamma, x : A]\!]$, meaning we can apply IH and lemma 4.2 to the second hypothesis to get $\overline{\Gamma}, x : \overline{A} \vdash \overline{B} : s' \in [\![\Gamma, x : A \vdash_{\mathsf{x}} B : s']\!]$ and then IH and lemma 4.3 to get $\overline{\Gamma}, x : \overline{A} \vdash \overline{t} : \overline{B} \in [\![\Gamma, x : A \vdash_{\mathsf{x}} t : B]\!]$. All of this yields $\overline{\Gamma} \vdash \lambda(x : \overline{A}).\overline{B}.\overline{t} : \Pi(x : \overline{A}).\, \overline{B} \in [\![\Gamma \vdash_{\mathsf{x}} \lambda(x : A).B.t : \Pi(x : A).\, B]\!]$.

- $$\frac{\Gamma \vdash_{\mathsf{x}} A : s \qquad \Gamma, x : A \vdash_{\mathsf{x}} B : s' \qquad \Gamma \vdash_{\mathsf{x}} t : \Pi(x : A).\, B \qquad \Gamma \vdash_{\mathsf{x}} u : A}{\Gamma \vdash_{\mathsf{x}} t \mathbin{@}_{x:A.B} u : B[x \leftarrow u]}$$

  Using IH together with lemmata 4.2 and 4.3 we get $\overline{\Gamma} \vdash \overline{A} : s$ and $\overline{\Gamma}, x : \overline{A} \vdash \overline{B} : s'$ and $\overline{\Gamma} \vdash \overline{t} : \Pi(x : \overline{A}).\, \overline{B}$ and $\overline{\Gamma} \vdash \overline{u} : \overline{A}$ meaning we can conclude $\overline{\Gamma} \vdash \overline{t} \mathbin{@}_{x:\overline{A}.\overline{B}} \overline{u} : \overline{B}[x \leftarrow \overline{u}] \in [\![\Gamma \vdash_{\mathsf{x}} t \mathbin{@}_{x:A.B} u : B[x \leftarrow u]]\!]$.

- $$\frac{\Gamma \vdash_{\mathsf{x}} u : A \qquad \Gamma \vdash_{\mathsf{x}} A : s \qquad \Gamma, x : A \vdash_{\mathsf{x}} B : s' \qquad \Gamma \vdash_{\mathsf{x}} v : B[x \leftarrow u]}{\Gamma \vdash_{\mathsf{x}} \langle u; v \rangle_{x:A.B} : \Sigma(x : A).\, B}$$

  Using IH with lemmata 4.2 and 4.3 we translate all the hypotheses to conclude $\overline{\Gamma} \vdash \langle \overline{u}; \overline{v} \rangle_{x:\overline{A}.\overline{B}} : \Sigma(x : \overline{A}).\, \overline{B} \in [\![\Gamma \vdash_{\mathsf{x}} \langle u; v \rangle_{x:A.B} : \Sigma(x : A).\, B]\!]$.

- $$\frac{\Gamma \vdash_{\mathsf{x}} p : \Sigma(x : A).\, B}{\Gamma \vdash_{\mathsf{x}} \pi_1^{x:A.B}\ p : A}$$

  Similar.

- $$\frac{\Gamma \vdash_{\mathsf{x}} p : \Sigma(x : A).\, B}{\Gamma \vdash_{\mathsf{x}} \pi_2^{x:A.B}\ p : B[x \leftarrow \pi_1^{x:A.B}\ p]}$$

  Similar.

- $$\frac{\Gamma \vdash_{\mathsf{x}} A : s \qquad \Gamma \vdash_{\mathsf{x}} u : A}{\Gamma \vdash_{\mathsf{x}} \mathsf{refl}_A\ u : u =_A u}$$

  By IH we have $\overline{\Gamma} \vdash \overline{u} : \overline{A}$ and thus $\overline{\Gamma} \vdash \mathsf{refl}_{\overline{A}}\ \overline{u} : \overline{u} =_{\overline{A}} \overline{u} \in [\![\Gamma \vdash_{\mathsf{x}} \mathsf{refl}_A\ u : u =_A u]\!]$.

- $$\frac{\begin{array}{c}\Gamma \vdash_{\mathsf{x}} A : s \qquad \Gamma \vdash_{\mathsf{x}} u, v : A \\ \Gamma, x : A, e : u =_A x \vdash_{\mathsf{x}} P : s' \qquad \Gamma \vdash_{\mathsf{x}} p : u =_A v \qquad \Gamma \vdash_{\mathsf{x}} w : P[x \leftarrow u, e \leftarrow \mathsf{refl}_A\ u]\end{array}}{\Gamma \vdash_{\mathsf{x}} \mathsf{J}(A, u, x.e.P, w, v, p) : P[x \leftarrow v, e \leftarrow p]}$$

  By IH and lemma 4.2 we have $\overline{\Gamma} \vdash \overline{A} : s$. From this and IH and lemma 4.3 we have $\overline{\Gamma} \vdash \overline{u}, \overline{v} : \overline{A}$. We can thus deduce $\vdash \overline{\Gamma}, x : \overline{A}, e : \overline{u} =_{\overline{A}} x \in [\![\Gamma, x : A, e : u =_A x]\!]$ which in turn gives us $\overline{\Gamma}, x : \overline{A}, e : \overline{u} =_{\overline{A}} x \vdash \overline{P} : s'$. Similarly we also get $\overline{\Gamma} \vdash \overline{p} : \overline{u} =_{\overline{A}} \overline{v}$ and $\overline{\Gamma} \vdash \overline{w} : \overline{P}[x \leftarrow \overline{u}, e \leftarrow \mathsf{refl}_{\overline{A}}\ \overline{u}]$. All of this allows us to conclude $\overline{\Gamma} \vdash \mathsf{J}(\overline{A}, \overline{u}, x.e.\overline{P}, \overline{w}, \overline{v}, \overline{p}) : \overline{P}[x \leftarrow \overline{v}, e \leftarrow \overline{p}] \in [\![\Gamma \vdash_{\mathsf{x}} \mathsf{J}(A, u, x.e.P, w, v, p) : P[x \leftarrow v, e \leftarrow p]]\!]$.

- $$\frac{\Gamma \vdash f, g : \Pi(x : A).\, B \qquad \Gamma \vdash e : \Pi(x : A).\, f\ @_{x:A.B}\ x =_B g\ @_{x:A.B}\ x}{\Gamma \vdash \mathsf{funext}(x : A, B, f, g, e) : f = g}$$

  Similar.

- $$\frac{\Gamma \vdash_{\mathsf{x}} e_1, e_2 : u =_A v}{\Gamma \vdash_{\mathsf{x}} \mathsf{uip}(A, u, v, e_1, e_2) : e_1 = e_2}$$

  Similar.

- $$\frac{\Gamma \vdash_{\mathsf{x}} A : s \qquad \Gamma, x : A \vdash_{\mathsf{x}} B : s' \qquad \Gamma, x : A \vdash_{\mathsf{x}} t : B \qquad \Gamma \vdash_{\mathsf{x}} u : A}{\Gamma \vdash_{\mathsf{x}} (\lambda(x : A).B.t)\ @_{x:A.B}\ u \equiv t[x \leftarrow u] : B[x \leftarrow u]}$$

  From IH and the lemmata, we even get the conversion, we conclude using reflexivity.

- $$\frac{\Gamma \vdash_{\mathsf{x}} A : s \qquad \Gamma \vdash_{\mathsf{x}} u : A \qquad \Gamma, x : A \vdash_{\mathsf{x}} B : s' \qquad \Gamma \vdash_{\mathsf{x}} v : B[x \leftarrow u]}{\Gamma \vdash_{\mathsf{x}} \pi_1^{x:A.B}\ \langle u; v \rangle_{x:A.B} \equiv u : A}$$

  Likewise.

- $$\frac{\Gamma \vdash_{\mathsf{x}} A : s \qquad \Gamma \vdash_{\mathsf{x}} u : A \qquad \Gamma, x : A \vdash_{\mathsf{x}} B : s' \qquad \Gamma \vdash_{\mathsf{x}} v : B[x \leftarrow u]}{\Gamma \vdash_{\mathsf{x}} \pi_2^{x:A.B}\ \langle u; v \rangle_{x:A.B} \equiv v : B[x \leftarrow u]}$$

  Likewise.

- $$\frac{\Gamma \vdash_{\mathsf{x}} A : \mathsf{U}_i \qquad \Gamma \vdash_{\mathsf{x}} u : A \qquad \Gamma, x : A, e : u =_A x \vdash_{\mathsf{x}} P : \mathsf{U}_j \qquad \Gamma \vdash_{\mathsf{x}} w : P[x \leftarrow u, e \leftarrow \mathsf{refl}_A\, u]}{\Gamma \vdash_{\mathsf{x}} \mathsf{J}(A, u, x.e.P, w, u, \mathsf{refl}_A\, u) \equiv w : P[x \leftarrow u, e \leftarrow \mathsf{refl}_A\, u]}$$

  Likewise.
- $$\frac{\Gamma \vdash_{\mathsf{x}} u : A}{\Gamma \vdash_{\mathsf{x}} u \equiv u : A}$$

  We conclude from IH and reflexivity of $\cong$.
- $$\frac{\Gamma \vdash_{\mathsf{x}} u \equiv v : A}{\Gamma \vdash_{\mathsf{x}} v \equiv u : A}$$

  We conclude from IH and symmetry of $\cong$.
- $$\frac{\Gamma \vdash_{\mathsf{x}} u \equiv v : A \qquad \Gamma \vdash_{\mathsf{x}} v \equiv w : A}{\Gamma \vdash_{\mathsf{x}} u \equiv w : A}$$

  We conclude from IH and transitivity of $\cong$.
- $$\frac{\Gamma \vdash_{\mathsf{x}} t_1 \equiv t_2 : T_1 \qquad \Gamma \vdash_{\mathsf{x}} T_1 \equiv T_2}{\Gamma \vdash_{\mathsf{x}} t_1 \equiv t_2 : T_2}$$

  By IH (and lemma 3.2) we have $\overline{\Gamma} \vdash \overline{e} : \overline{t}_1 \;{}_{\overline{T}_1}{\cong}_{\overline{T}_1'}\; \overline{t}_2$ and $\overline{\Gamma} \vdash p : \overline{T}_1'' = \overline{T}_2$. Also from lemmata 3.3 and 3.2 we have $\overline{T}_1' = \overline{T}_1''$ and $\overline{T}_1 = \overline{T}_1''$, meaning we get $\overline{T}_1' = \overline{T}_2$ and $\overline{T}_1 = \overline{T}_2$. This allows us to conclude by transporting along the aforementioned equalities.
- $$\frac{\Gamma \vdash_{\mathsf{x}} A_1 \equiv A_2 : s \qquad \Gamma, x : A_1 \vdash_{\mathsf{x}} B_1 \equiv B_2 : s'}{\Gamma \vdash_{\mathsf{x}} \Pi(x : A_1).\, B_1 \equiv \Pi(x : A_2).\, B_2 : s''}(s, s', s'')$$

  We conclude exactly like we did in the proof of lemma 3.3.
- All congruences hold like in proof of lemma 3.3.
- $$\frac{\Gamma \vdash_{\mathsf{x}} e : u =_A v}{\Gamma \vdash_{\mathsf{x}} u \equiv v : A}$$

  By IH and lemma 4.2 we have $\overline{\Gamma} \vdash \overline{e} : \overline{u} =_{\overline{A}} \overline{v} \in [\![\Gamma \vdash_{\mathsf{x}} e : u =_A v]\!]$ which yields the conclusion we wanted.

□