How (not) to prove typed type conversion transitive

Yann Leray

Nantes Université, École Centrale Nantes, CNRS, INRIA, LS2N, UMR 6004, F-44000 Nantes, France

Type conversion is the central part of implementations of type theories. It is theoretically decidable thanks to the normalisation of all well-typed terms. However, most if not all proof assistants do not in fact normalise all terms to test for convertibility, instead trying to check if they are syntactically equal as a shortcut. Tangentially, many languages used in proof assistants are not proved to be normalising, and some even allow extensions which explicitly prevent normalisation.

Without the assumption of normalisation, many properties required for the adequacy between such an algorithm and the intended specification become brittle and are left to be justified, with the most central of these being the transitivity of the relation defined by the algorithm. There currently exist proofs of the soundness of the algorithm for an untyped specification of PCUIC (the idealised type theory of Rocq) [SFL+25], and with a typed specification of Pure Type Systems (PTSs) [Ada06, SH12] with no extensionality (η) rules. We present the difficulties of replicating these proofs in the context of a typed specification, with extensionality rules, as well as the paths we explored and why we stopped exploring some.

Specification and algorithm. Let us describe the two approaches to type conversion that we want to link. On one end, the specification for convertibility of terms (denoted by $\equiv$) will include freely all reductions, η-rules and congruence rules, such that it will be an equivalence relation. It however doesn't satisfy any type constructor injectivity results out of the box, due to its transitivity rule muddying everything. On the other end, we will abstract the behaviour of an algorithm by the relation $\cong$, which does repeated head-reductions and (typed) head-η-expansions on both sides arbitrarily, and at any point when the head constructors are the same can it stop and resume the operation on the subterms instead (i.e. apply term constructor congruence rules).

With these definitions, inclusion of $\cong$ in $\equiv$ is immediate and, for the converse inclusion, most rules are also already present. Reflexivity and symmetry are both obtained by simple inductions, leaving only transitivity remaining. One should first notice that transitivity cannot be proven with a simple induction, for the same reason that confluence of λ-calculus cannot be derived by induction using simply its weak confluence property, because of an increasing reduction length issue: when you do deal with the case $\Gamma \vdash (\lambda(x : A_0), t_0)\ u_0 \equiv (\lambda(x : A_1), t_1)\ u_1 : T$ (by congruence) against $\Gamma \vdash (\lambda(x : A_1), t_1)\ u_1 \equiv t_2[x := u_2] : T$, you can derive $\Gamma \vdash t_0[x := u_0] \equiv t_1[x := u_1] : T$, but since it isn't a structurally smaller derivation than the one you start with, the induction cannot go through.

How to prove transitivity. In a context where we intend to prove normalisation and can afford to work only with normalising terms (e.g. [AÖV17, HJP23, ALM+24]), there is a simple proof of transitivity for $\cong$. It relies on the observation that only congruence rules are applicable to normal forms, which makes the relation transitive on such terms. It then only remains to show that all shortcuts that can be taken are admissible.

However, in our context where no proof of normalisation is accessible, we need another approach. Since our issue at hand is similar to what happens in λ-calculus, we can resort to the solution that is used there and was already extended to work for PCUIC and PTSs: parallel

reduction [Tak89] and the Hindley-Rosen technique [BKv98]. We will split the rules of $\cong$ into multiple relations which will be parallel reduction ($\Rrightarrow$, reduction rules and congruences), parallel η-expansion ($\Rrightarrow_\eta$, η-expansion and congruences) and syntactic equality ($=_s$, α-equivalence and type-annotation irrelevance). Our complete conversion relation will now be $(\Rrightarrow + \Rrightarrow_\eta)^*; =_s ; (\Lleftarrow +_\eta \Lleftarrow)^*$ (arbitrary reductions on both sides using both parallel reduction and parallel η-expansion, closed by syntactic equality).

The idea behind this split is that, starting from a concatenation of two conversions, we reorder the individual steps using some swapping lemmas and hope to reach a state where it has a the shape of a single conversion. Since we're working with parallel relations, we are looking for simulations between these relations ($R; R' \subseteq R'; R$ or sometimes $R; R' \subseteq R'; =_s; R$)

Compatibilities, constraints on type conversion. The proofs of the different swapping lemmas will require establishing some properties on terms, themselves further requiring some properties on the type conversion used in the parallel relations (abstract until now).

First, during the inversions that happen on the examined parallel relations, we need to relate the types of corresponding subterms, so we require that terms have principal types and that type conversion be reflexive and transitive. Second, since typing is based on the left term, our relations have to be stable under type and context change (for the equivalent context-level relations), which means that typing and type conversion has to be stable under such changes; here on top of that, type conversion also needs to be stable on both sides by all relations in both directions (e.g. both $\Rrightarrow$ and $\Lleftarrow$).

For the proof of the strong confluence (self-simulation) of $\Rrightarrow$, we observe that it has to be stable under substitution, which means in turn that both typing and type conversion have to be stable under substitution.

For the simulation between $\Rrightarrow$ and $\Rrightarrow_\eta$, we must disallow for a term t to have both $t : \mathbb{B}$ and $t : \Pi(x : A), B$, or for a term $t : \mathrm{Type}$ to have either $t : \mathbb{B}$ or $t : \Pi(x : A), B$. Since we also assume that terms have principal types, this translates into a requirement of non-confusion of type conversion for type formers, that is, no two among $\mathbb{B}$, $\Pi(x : A), B$ and Type should be convertible. Simultaneously, we require that all possible β- and ι-redexes may be reduced, i.e. any redex well typed as a term has its additional redex typing constraints satisfied. These typing constraints are all provable if we require the injectivity of type conversion for type formers (when two function types are convertible, then so are their domains and so are their codomains).

However, this whole proof sketch of transitivity is not structural, so we can't use a simple induction hypothesis to satisfy both our transitivity requirement and our stability under reduction requirements. This means we have to find an existing relation which satisfies all of our constraints, yet the list is long enough that our needed conversion is already what we want to find.

Side-stepping some constraints. In the talk we will report on ongoing investigations and here present the current status of our research for a conclusive proof. Let us first focus over the requirements we can fulfil. For type principality, two ways exist which don't require any previous results: using a bidirectional type system [Len21] or using over-annotated terms (where for instance an application $(f)\ a$ becomes $(f)\{(x : A : s_0), B : s_1\}\ a$ to add type annotations for all subterms). While the former approach looks simpler, the latter approach allows for context changes which do not influence principal types (thanks to the type annotation on variables), so we will focus on that one.

Working with principal types also helps reduce the requirements on type conversion we need: when one term reduces to another (through either $\Rightarrow$ or $\Rightarrow_\eta$), then the same is true of their principal types. On the one hand, this means that our relations now work on two types instead of one; we also need redexes to preserve principal types, so we may need to introduce explicit definitional casts in some reductions, and a specific reduction relation to erase them. On the other hand, this waives the requirement of both reflexivity and stability by $\Leftarrow$ and ${}_\eta\Leftarrow$ for type conversion. At this stage, this means that instantiating type conversion with the false relation would be sound (but obviously not complete).

The requirements for transitivity become located to a certain number of redexes (namely, η-expansions and cast erasing), where it can be assumed locally instead of globally, with hopes of transporting the assumption instead of reconstructing it when needed).

This leaves only the requirements of non-confusion and injectivity of type constructors, and stability under substitution and all parallel relations. Once more, neither $\equiv$ nor $\cong$ satisfy both properties out of the box, so we need another solution. The currently envisioned solution would be to devise an induction with a very strong predicate, such that it can satisfy all of our requirements at the same time. We first need to make sure that we won't need to construct any proofs of it, which is the reason why we tried to eliminate the requirement of reflexivity in the first place. A lot of work remains to be done to conclude the proof nonetheless.

References

[Ada06] Robin Adams. Pure type systems with judgemental equality. *Journal of Functional Programming*, 16(2):219–246, March 2006.

[ALM+24] Arthur Adjedj, Meven Lennon-Bertrand, Kenji Maillard, Pierre-Marie Pédrot, and Loïc Pujet. Martin-Löf à la Coq. In *Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2024, pages 230–245, New York, NY, USA, January 2024. Association for Computing Machinery.

[AÖV17] Andreas Abel, Joakim Öhman, and Andrea Vezzosi. Decidability of conversion for type theory in type theory. *Proc. ACM Program. Lang.*, 2(POPL):23:1–23:29, December 2017.

[BKv98] Marc Bezem, Jan Willem Klop, and Vincent van Oostrom. Diagram Techniques for Confluence. *Information and Computation*, 141(2):172–204, March 1998.

[HJP23] Jason Z. S. Hu, Junyoung Jang, and Brigitte Pientka. Normalization by evaluation for modal dependent type theory. *Journal of Functional Programming*, 33:e7, January 2023.

[Len21] Meven Lennon-Bertrand. Complete Bidirectional Typing for the Calculus of Inductive Constructions. In Liron Cohen and Cezary Kaliszyk, editors, *12th International Conference on Interactive Theorem Proving (ITP 2021)*, volume 193 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 24:1–24:19, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[SFL+25] Matthieu Sozeau, Yannick Forster, Meven Lennon-Bertrand, Jakob Nielsen, Nicolas Tabareau, and Théo Winterhalter. Correct and Complete Type Checking and Certified Erasure for Coq, in Coq. *J. ACM*, 72(1):8:1–8:74, January 2025.

[SH12] Vincent Siles and Hugo Herbelin. Pure Type System conversion is always typable. *Journal of Functional Programming*, 22(2):153–180, March 2012.

[Tak89] Masako Takahashi. Parallel reductions in λ-calculus. *Journal of Symbolic Computation*, 7(2):113–123, February 1989.