*À bas l'*η

Coq's troublesome η-conversion

Meven Lennon-Bertrand
Inria — LS2N, Université de Nantes
meven.bertrand@univ-nantes.fr

Abstract

In this talk, I shall report on the difficulties presented by the treatment of η-conversion in the setting of the MetaCoq project. I will try to give insight on why the situation is surprisingly thorny in that specific setting, and what we could do to get out of this.

1 η is good

The η-equality rule for functions, stating that $\lambda\, x.f\ x$ is convertible to f whenever x is free in f, is present in one form or another in most modern proof assistants based upon type theory: Lean and Agda have always had it, Coq has it since version 8.4… It is also an important tool in practice, for instance to derive function extensionality from univalence [4]. So the case looks closed: η-conversion is desirable, and we know how to implement it. Why would it still be an issue? Sadly, trying to give it a good theoretical treatment in the setting of the MetaCoq project [3] proved surprisingly challenging – to the point that it is still not handled despite quite some efforts.

2 What are we talking about?

The MetaCoq project aims at describing a type theory for Coq's kernel. Without going into all gory details, let us mention its main characteristics of interest to us. Abstractions are Church-style, *e.g.,* the identity at type A is $\lambda\, x : A.A$. To relate different types for a same term, we rely not only on *conversion* $\equiv$, but also the restricted form of subtyping called *cumulativity*, written $\leq$. Their definitions rely on *reduction* $\rightsquigarrow$, rather than being presented as a (typed) judgement. Moreover, cumulativity is equivariant on domains for product types, meaning that $\Pi\, x : A.B \leq \Pi\, x : A'.B'$ whenever $A \equiv A'$ and $B \leq B'$.

The cornerstone property of MetaCoq is subject reduction (SR), which asserts that if $\Gamma \vdash t : T$ and $t \rightsquigarrow t'$ then $\Gamma \vdash t' : T$. To establish this, one first introduces $=_\alpha$, an extension of the usual α-equality to handle universe levels, then proves that $t \equiv t'$ if and only if there exists u and u' such that $t \rightsquigarrow u$, $t' \rightsquigarrow u'$ and $u =_\alpha u'$. The main lemmas here are confluence of $\rightsquigarrow$, and $=_\alpha$ being a simulation for $\rightsquigarrow$. From this many consequences follow, including injectivity of type constructors, the key ingredient of SR.

Now we wish to extend conversion with η-equality, *i.e.,* add the fact that $\lambda\, x.f\ x \equiv f$. If we wish to keep the previous strategy, we have three ways to go: we can incorporate η-equality into reduction, either by adding η-reduction $\lambda\, x : A.f\ x \rightsquigarrow f$ or η-expansion $f \rightsquigarrow \lambda\, x : A.f\ x$; or we can extend $=_\alpha$ instead.

3 η is hard

3.1 Expansion

Expansion is perhaps the most problematic strategy. First, it is non-normalizing, since one can alternate β-reduction and η-expansion steps. Moreover, inventing the type annotation on the λ binder on the fly without any type information is impossible, even more so if one wishes to preserve SR.

3.2 Reduction

Reduction seems more promising, as at least the reduct is clear. However, we hit subtler problems because of type annotations. Indeed, consider types A and A' such that $A \leq A'$. Then $f : A' \to A' \vdash \lambda x : A.f\ x : A \to A'$ because the variable has type A' by cumulativity, but this term η-reduces to f, and $f : A' \to B \nvdash f : A \to A'$ because cumulativity is equivariant for product types. Thus, SR is once again broken. In the same vein, the term $\lambda x : A.(\lambda y : A'.y)\ x$ η-reduces to $\lambda y : A'.y$ and β-reduces to $\lambda x : A.x$, so confluence does not hold, even on typed terms.

There are other, softer arguments against reduction. First it is non-local insofar as it requires a global variable freedom test, which makes it more complex to handle than other reduction rules. It is also quite different from Coq's conversion test – which performs a form of η-expansion, but only on well-chosen terms.

3.3 Equality

Inspired by the previous failures and by the way η-expansion is performed in the kernel only when comparing two terms, a last possibility is to extend $=_\alpha$ so that $f =_\alpha \lambda\, x : A.g$ if $f\ x =_\alpha g$. This goes some way, but again hits issues, this time because $=_\alpha$ fails to stay a simulation for $\rightsquigarrow$. Indeed, one would have

$$\texttt{if true then 0 else 1} =_\alpha \texttt{if}(\lambda\, x : \square.\texttt{true}\ x)\ \texttt{then 0 else 1}$$

and the left term reduces to 0 while the right one is stuck. This issue arises because of the possibility to block ι-redexes via an ill-typed η-expansion. Imposing some typing constraint to ban such pathological terms and regain the characterization of conversion is unfeasible, because one would need SR to know that this typability constraint is preserved by reduction during the proof of equivalence, but SR is a consequence of that equivalence.

4 Can we still have η?

In the current setting, it seems that extending MetaCoq with η-equality is at least complex, if not unfeasible. What ways out do we have to specify Coq's kernel?

A first possibility is to somewhat alter the type system, but keep the same general strategy, for instance by turning to contravariant product types and/or ignoring type annotations for conversion – as done in [2] –, so as to mitigate some of the issues we exposed.

Another option, more ambitious but also maybe more interesting, is to turn to typed conversion in order to completely remove the need to consider untyped terms. One could then try to follow [1] to get an equivalence between conversion and an algorithmic version thereof to which Coq's kernel could be compared – and hopefully proven correct and complete.

References

[1] Abel, A., Öhman, J., and Vezzosi, A. Decidability of conversion for type theory in type theory. *Proc. ACM Program. Lang. 2*, POPL (Dec. 2017).

[2] Barras, B., and Grégoire, B. On the role of type decorations in the calculus of inductive constructions. In *Computer Science Logic* (Berlin, Heidelberg, 2005), L. Ong, Ed., Springer Berlin Heidelberg, pp. 151–166.

[3] Sozeau, M., Boulier, S., Forster, Y., Tabareau, N., and Winterhalter, T. Coq Coq Correct! Verification of Type Checking and Erasure for Coq, in Coq. *Proceedings of the ACM on Programming Languages* (2020), 1–28.

[4] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. https://homotopytypetheory.org/book, 2013.