À bas l'η

Coq's troublesome η-conversion

Meven Lennon-Bertrand – Gallinette team, Inria
Joint with the MetaCoq team

WITS 2022

Setting up the scene

η-conversion in Coq

Coq has η-conversion

```
Goal f = (fun x ⇒ f x).
 reflexivity.
Qed.
```

Also, primivite records/strong sums/negative sums.

η-conversion in Coq

Coq has η-conversion

```
Goal f = (fun x ⇒ f x).
 reflexivity.
Qed.
```

Also, primivite records/strong sums/negative sums.

Under the hood

Untyped conversion testing:

- reduce to weak-head normal form
- expand only neutrals against abstractions

η-conversion in Coq

Coq has η-conversion

```
Goal f = (fun x ⇒ f x).
 reflexivity.
Qed.
```

Also, primivite records/strong sums/negative sums.

Under the hood

Untyped conversion testing:

- reduce to weak-head normal form
- expand only neutrals against abstractions

The implementation is not the troublesome part!

Conversion(s) in MetaCoq

MetaCoq

- metatheory of Coq, in Coq
- correct and complete type-checker

MetaCoq

- metatheory of Coq, in Coq
- correct and complete type-checker

Declarative conversion

$$\frac{\Gamma \vdash t_1 \equiv t_2 \qquad \Gamma \vdash t_2 \equiv t_3}{\Gamma \vdash t_1 \equiv t_3} \qquad \frac{\Gamma \vdash t \equiv t' \qquad \Gamma \vdash u \equiv u'}{\Gamma \vdash t\ u \equiv t'\ u'}$$

$$\frac{}{\Gamma \vdash (\lambda\, x : A.t)\ u \equiv t\{x := u\}} \qquad \dots$$

Conversion(s) in MetaCoq

MetaCoq

- metatheory of Coq, in Coq
- correct and complete type-checker

Declarative conversion

$$\frac{\Gamma \vdash t_1 \equiv t_2 \qquad \Gamma \vdash t_2 \equiv t_3}{\Gamma \vdash t_1 \equiv t_3} \qquad \frac{\Gamma \vdash t \equiv t' \qquad \Gamma \vdash u \equiv u'}{\Gamma \vdash t\ u \equiv t'\ u'}$$

$$\frac{}{\Gamma \vdash (\lambda\, x : A.t)\ u \equiv t\{x := u\}} \qquad \dots$$

Algorithmic conversion

$$\Gamma \vdash t \Downarrow u := \begin{array}{ccc} t & & u \\ \downarrow & & \downarrow \\ t' & =_\alpha & u' \end{array}$$

The plan

Step 1: Sweat

- confluence:

$$\begin{tikzcd}[column sep=small] & t \arrow[dl] \arrow[dr] & \\ t_1 \arrow[dr, dashed] & & t_2 \arrow[dl, dashed] \\ & t'' & \end{tikzcd}$$

- simulation:

$$\begin{tikzcd} t \arrow[d, dashed] \arrow[r, phantom, "=_\alpha"] & u \arrow[d] \\ t' \arrow[r, phantom, "=_\alpha"] & u' \end{tikzcd}$$

- transitivity of $\Downarrow$ and equivalence of $\Downarrow$ and $\equiv$ *on all terms*

Step 1: Sweat

- confluence:

$$\begin{tikzcd} & t \arrow[dl] \arrow[dr] & \\ t_1 \arrow[dr, dashed] & & t_2 \arrow[dl, dashed] \\ & t'' & \end{tikzcd}$$

- simulation:

$$\begin{tikzcd} t \arrow[d, dashed] \arrow[r, phantom, "=_\alpha"] & u \arrow[d] \\ t' \arrow[r, phantom, "=_\alpha"] & u' \end{tikzcd}$$

- transitivity of $\Downarrow$ and equivalence of $\Downarrow$ and $\equiv$ *on all terms*

Step 2: Enjoy!

- injectivity of type constructors
- subject reduction
- the conversion test and type-checker meet their specification

Extending MetaCoq with η

Expansion

$$f \to \lambda\, x : A. f\, x$$

Expansion

$$f \to \lambda\, x : A. f\, x$$

Everybody knows it's the way to go!

$$f \rightarrow \lambda\, x : A . f\, x$$

Everybody knows it's the way to go!

But not without types...

Reduction

$$\lambda x : A. f\, x \rightarrow f \quad \text{if } x \text{ is not free in } f$$

$$\lambda\, x : A. f\, x \rightarrow f \quad \text{if } x \text{ is not free in } f$$

Bad interaction with annotations and cumulativity:

$$\lambda\, x : \Box_0 . x \xleftarrow{\beta} \lambda\, x : \Box_0 . (\lambda\, y : \Box_1 . y)\ x \xrightarrow{\eta} \lambda\, y : \Box_1 . y$$

$$\lambda\, x : A. f\, x \rightarrow f \quad \text{if } x \text{ is not free in } f$$

Bad interaction with annotations and cumulativity:

$$\lambda\, x : \square_0 . x \xleftarrow{\ \beta\ } \lambda\, x : \square_0 . (\lambda\, y : \square_1 . y)\; x \xrightarrow{\ \eta\ } \lambda\, y : \square_1 . y$$

Breaks

- confluence
- subject reduction, since
  $\vdash \lambda\, x : \square_0 . (\lambda\, y : \square_1 . y)\; x : \square_0 \rightarrow \square_1$
  and $\vdash \lambda\, y : \square_1 . y : \square_1 \rightarrow \square_1$
  but $\nvdash \lambda\, y : \square_1 . y : \square_0 \rightarrow \square_1$

Equality

Take inspiration from the implementation:

$$\dots \qquad \frac{t\,x =_\alpha u}{t =_\alpha \lambda\, x : A.u} \qquad \frac{t =_\alpha u\,x}{\lambda\, x : A.t =_\alpha u}$$

Take inspiration from the implementation:

$$\dots \qquad \frac{t\,x =_\alpha u}{t =_\alpha \lambda\, x : A.u} \qquad \frac{t =_\alpha u\,x}{\lambda\, x : A.t =_\alpha u}$$

Goes some way, but hits a wall again:

$$\begin{array}{ccc} \texttt{if true then } 0 \texttt{ else } 1 & =_\alpha & \texttt{if } (\lambda\, x : \square_0.\texttt{true } x) \texttt{ then } 0 \texttt{ else } 1 \\ \downarrow & & \not\downarrow \\ 0 & & \end{array}$$

Equality

Take inspiration from the implementation:

$$\ldots \qquad \frac{t\,x =_\alpha u}{t =_\alpha \lambda\, x : A.u} \qquad \frac{t =_\alpha u\, x}{\lambda\, x : A.t =_\alpha u}$$

Goes some way, but hits a wall again:

$$\begin{array}{ccc} \texttt{if true then}\ 0\ \texttt{else}\ 1 & =_\alpha & \texttt{if}\ (\lambda\, x : \square_0.\texttt{true}\ x)\ \texttt{then}\ 0\ \texttt{else}\ 1 \\ \downarrow & & \not\downarrow \\ 0 & & \end{array}$$

Still well-behaved on typed terms, but this is not enough for the plan.

How do we get out?

Keeping an untyped conversion

Reduction

- Erase annotations to regain confluence
- Contravariant products or separate η-reduction to regain SR

Keeping an untyped conversion

Reduction

- Erase annotations to regain confluence
- Contravariant products or separate η-reduction to regain SR

Equality

Find a way to cut the loop?

Moving to typed conversion

The addition of η-conversion is justified by the confidence that the formulation of [CIC] based on typed equality [...] is applicable to the concrete implementation of Coq.

Coq 8.4 summary of changes

Moving to typed conversion

The addition of η-conversion is justified by the confidence that the formulation of [CIC] based on typed equality [...] is applicable to the concrete implementation of Coq.

Coq 8.4 summary of changes

Can we prove anything at all about such a type system ? In Coq?

Thank you!