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
Coq has $\eta$-conversion

Goal $f = (\text{fun } x \Rightarrow f x)$.
  reflexivity.
Qed.

Also, primitive records/strong sums/negative sums.
Coq has $\eta$-conversion

\begin{align*}
\text{Goal } f & = (\text{fun } x \Rightarrow f \ x). \\
\text{reflexivity.} \\
\text{Qed.}
\end{align*}

Also, primitive records/strong sums/negative sums.

Under the hood

Untyped conversion testing:

\begin{itemize}
\item reduce to weak-head normal form
\item expand only neutrals against abstractions
\end{itemize}
### Coq has $\eta$-conversion

**Goal** $f = (\text{fun } x \Rightarrow f \; x)$.

reflexivity.

Qed.

Also, primitive 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!
### MetaCoq

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

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

### Declarative conversion

\[
\begin{align*}
\Gamma \vdash t_1 &\equiv t_2 & \Gamma \vdash t_2 &\equiv t_3 & \Gamma \vdash t &\equiv t' & \Gamma \vdash u &\equiv u' \\
\therefore \quad \Gamma \vdash t_1 &\equiv t_3 & \therefore \quad \Gamma \vdash t u &\equiv t' u' \\
\therefore \quad \Gamma \vdash (\lambda x : A. t) u &\equiv t\{x := u\} & \therefore \quad \ldots
\end{align*}
\]
### MetaCoq

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

### Declarative conversion

\[
\begin{align*}
\Gamma \vdash t_1 &\equiv t_2 & \Gamma \vdash t_2 &\equiv t_3 & \Gamma \vdash t_1 &\equiv t_3 \\
\Gamma \vdash t &\equiv t' & \Gamma \vdash u &\equiv u' & \Gamma \vdash t \; u &\equiv t' \; u'
\end{align*}
\]

\[
\Gamma \vdash (\lambda x : A . t) \; u \equiv t \{ x := u \}
\]

### Algorithmic conversion

\[
\Gamma \vdash t \downarrow u := t \downarrow u \downarrow t' =_{\alpha} u'
\]
Step 1: Sweat

- confluence:

- simulation:

- transitivity of $\downarrow$ and equivalence of $\downarrow$ and $\equiv$ on all terms
**The Plan**

### Step 1: Sweat

- **confluence:**
  
  ![Diagram](image)
  
  \[ t = \alpha u \]
  
  \[ t'' \]
  
- **simulation:**
  
  \[ t' = \alpha u' \]

- **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 $\eta$
Everybody knows it’s the way to go! But not without types…

\[ f \rightarrow \lambda x : A. f x \]
Everybody knows it’s the way to go!

\[ f \rightarrow \lambda x : A. f \ x \]
$f \rightarrow \lambda x : A. f x$

Everybody knows it’s the way to go!

But not without types...
\( \lambda x : A. f x \rightarrow f \) if \( x \) 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:

\[
\begin{align*}
\lambda x : \Box_0. (\lambda y : \Box_1. y) \ x \\
\lambda x : \Box_0. x & \quad \beta \quad \lambda y : \Box_1. y \\
& \quad \eta
\end{align*}
\]
\[
\lambda x : A. f x \rightarrow f \quad \text{if } x \text{ is not free in } f
\]

Bad interaction with annotations and cumulativity:

\[
\begin{align*}
\lambda x : \Box_0.(\lambda y : \Box_1.y) \ x & \\
\lambda y : \Box_1.y & \quad \beta \quad \eta
\end{align*}
\]

Breaks

- confluence
- subject reduction, since

\[\vdash \lambda x : \Box_0.(\lambda y : \Box_1.y) \ x : \Box_0 \rightarrow \Box_1\]

and \[\vdash \lambda y : \Box_1.y : \Box_1 \rightarrow \Box_1\]

but \[\not\vdash \lambda y : \Box_1.y : \Box_0 \rightarrow \Box_1\]
Take inspiration from the implementation:

\[
\frac{t \ x =_\alpha u}{t =_\alpha \lambda x : A. u}
\]

\[
\frac{t =_\alpha u \ x}{\lambda x : A. t =_\alpha u}
\]

Goes some way, but hits a wall again:

\[
\text{if true then } 0 \text{ else } 1 \\
\text{if } (\lambda x : \square 0. \text{true} x) \text{ then } 0 \text{ else } 1
\]

Still well-behaved on typed terms, but this is not enough for the plan.
Take inspiration from the implementation:

\[
\begin{align*}
\frac{t \ x = \alpha \ u}{t = \alpha \ \lambda \ x : A. \ u} \\
\frac{t = \alpha \ u \ x}{\lambda \ x : A. \ t = \alpha \ u}
\end{align*}
\]

Goes some way, but hits a wall again:

\[
\text{if true then 0 else 1} \quad =_\alpha \quad \text{if (} \lambda \ x : \Box_0. \ true \ x \text{) then 0 else 1}
\]

\[
\downarrow
0
\]

\[
\times
\downarrow
\]
Take inspiration from the implementation:

\[
\begin{align*}
\text{...} & \quad \frac{t x =_\alpha u}{t =_\alpha \lambda x : A. u} \\
\frac{t =_\alpha u x}{\lambda x : A. t =_\alpha u}
\end{align*}
\]

Goes some way, but hits a wall again:

\[
\begin{align*}
\text{if true then 0 else 1} & =_\alpha \text{if (} \lambda x : \square_0. \text{true } x \text{) then 0 else 1} \\
\downarrow & \quad \downarrow \\
0 & \quad \times
\end{align*}
\]

Still well-behaved on typed terms, but this is not enough for the plan.
HOW DO WE GET OUT?
**Reduction**

- Erase annotations to regain confluence
- Contravariant products or separate $\eta$-reduction to regain SR
**Reduction**

- Erase annotations to regain confluence
- Contravariant products or separate $\eta$-reduction to regain SR

**Equality**

Find a way to cut the loop?
The addition of $\eta$-conversion is justified by the confidence that the formulation of [CIC] based on typed equality [...] is applicable to the concrete implementation of Coq.
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!