

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 η -conversion

```
Goal f = (fun x => f x).
```

```
  reflexivity.
```

```
Qed.
```

Also, primitive records/strong sums/negative sums.

Coq has η -conversion

```
Goal f = (fun x => 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

Coq has η -conversion

```
Goal f = (fun x => 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

CONVERSION(S) IN METACoQ

MetaCoq

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

Declarative conversion

$$\frac{\Gamma \vdash t_1 \equiv t_2 \quad \Gamma \vdash t_2 \equiv t_3}{\Gamma \vdash t_1 \equiv t_3}$$

$$\frac{\Gamma \vdash t \equiv t' \quad \Gamma \vdash u \equiv u'}{\Gamma \vdash t u \equiv t' u'}$$

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

...

CONVERSION(S) IN METACoQ

MetaCoq

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

Declarative conversion

$$\frac{\Gamma \vdash t_1 \equiv t_2 \quad \Gamma \vdash t_2 \equiv t_3}{\Gamma \vdash t_1 \equiv t_3}$$

$$\frac{\Gamma \vdash t \equiv t' \quad \Gamma \vdash u \equiv u'}{\Gamma \vdash t u \equiv t' u'}$$

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

...

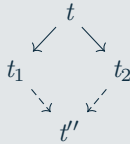
Algorithmic conversion

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

THE PLAN

Step 1: Sweat

- confluence:



- simulation:

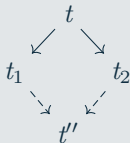


- transitivity of \Downarrow and equivalence of \Downarrow and \equiv on all terms

THE PLAN

Step 1: Sweat

- confluence:



- simulation:



- 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

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

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

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...

$\lambda x : A. f x \rightarrow f$ if x is not free in f

$\lambda x : A.f x \rightarrow f$ if x is not free in f

Bad interaction with annotations and cumulativity:

$$\begin{array}{ccc}
 & \lambda x : \square_0. (\lambda y : \square_1. y) x & \\
 \swarrow \beta & & \searrow \eta \\
 \lambda x : \square_0. x & & \lambda y : \square_1. y
 \end{array}$$

$\lambda x : A.f x \rightarrow f$ if x 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 $\not\vdash \lambda y : \square_1.y : \square_0 \rightarrow \square_1$

Take inspiration from the implementation:

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

Take inspiration from the implementation:

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

Goes some way, but hits a wall again:

$$\begin{array}{ccc} \text{if true then 0 else 1} & =_{\alpha} & \text{if } (\lambda x : \square_0. \text{true } x) \text{ then 0 else 1} \\ \downarrow & & \downarrow \\ 0 & & x \\ & & \downarrow \end{array}$$

Take inspiration from the implementation:

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

Goes some way, but hits a wall again:

$$\begin{array}{ccc} \text{if true then 0 else 1} & =_{\alpha} & \text{if } (\lambda x : \square_0. \text{true } x) \text{ then 0 else 1} \\ \downarrow & & \downarrow \\ 0 & & x \\ & & \downarrow \end{array}$$

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 η -reduction to regain SR

Reduction

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

Equality

Find a way to cut the loop?

*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

*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!