How (not) to prove typed type conversion transitive

Yann Leray

11th June 2025

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

Overarching goal

Completeness

$$\frac{}{\Gamma \vdash t \equiv u : T}$$

```
let t = whnf t in
let u = whnf u in
compare Γ ty t u
```

Correctness

Specification

Algorithm

Overarching goal

Completeness

$$\boxed{\dfrac{}{\Gamma \vdash t \equiv u : T}}$$

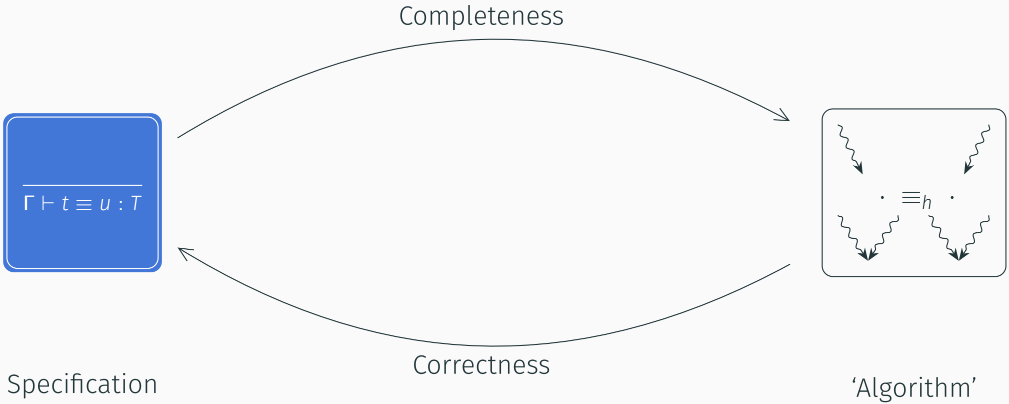
```
let t = whnf t in
let u = whnf u in
eq_syntax t u ||
compare Γ ty t u
```

Correctness

Specification

Algorithm

Overarching goal



Correction and completeness

Specification

- Equivalence relation
  - Partial reflexivity
  - Symmetry
  - Transitivity
- Congruence
- Type conversion
- Reduction rules
- η laws

Algorithm

- Congruence
- Type conversion
- Reduction rules and η laws (auxiliary reduction relation)

Correction and completeness

Specification

- Equivalence relation
  - Partial reflexivity
  - Symmetry
  - Transitivity
- Congruence
- Type conversion
- Reduction rules
- η laws

Algorithm

- Congruence ✓
- Type conversion ✓
- Reduction rules and η laws ✓
  (auxiliary reduction relation) ✓

Correction and completeness

Specification

- Equivalence relation
  - Partial reflexivity ✓
  - Symmetry ✓
  - Transitivity
- Congruence ✓
- Type conversion ✓
- Reduction rules ✓
- η laws ✓

Algorithm

- Congruence ✓
- Type conversion ✓
- Reduction rules and η laws ✓
  (auxiliary reduction relation) ✓

Correction and completeness

Specification

- Equivalence relation
  - Partial reflexivity ✓
  - Symmetry ✓
  - Transitivity
- Congruence ✓
- Type conversion ✓
- Reduction rules ✓
- η laws ✓

Algorithm

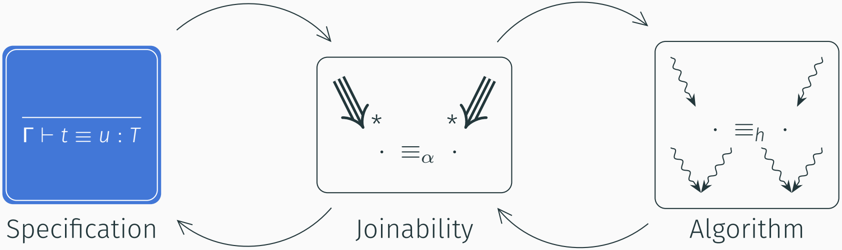
- Congruence ✓
- Type conversion ✓
- Reduction rules and η laws ✓ (auxiliary reduction relation) ✓

How to prove transitivity

1. By a simple induction?
   Case $(\lambda(x : A_0), t_0)\ u_0 \equiv_a (\lambda(x : A_1), t_1)\ u_1 \equiv_a t_2[x := u_2]$
   Substitution does not preserve sizes for $\equiv_a$

2. Through logical relations and deterministic reduction
   Transitivity is traded for reflexivity (need normalisation)

3. Through parallel relations and Hindley-Rosen technique



Joinability, parallel relations

The conversion relation is split between $\Rrightarrow$, $\Rrightarrow_\eta$ and $\equiv_\alpha$:

- Congruence
- Type conversion (yet to be determined conversion relation)
- Base blocks

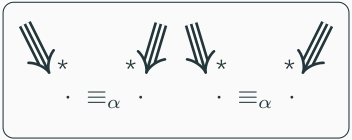
$$\frac{\Gamma, (x : A) \vdash t \Rrightarrow t' : B \qquad \Gamma \vdash u \Rrightarrow u' : A}{\Gamma \vdash (\lambda x, t)\ u \Rrightarrow t'[x := u'] : B[x := u]}\ \beta \qquad \frac{\Gamma \vdash t \Rrightarrow_\eta t' : \Pi(x : A), B}{\Gamma \vdash t \Rrightarrow_\eta \lambda x, t'\ x : \Pi(x : A), B}\ \eta$$

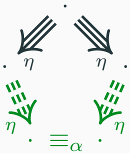
- All combined : $(\equiv_j) = [(\Rrightarrow + \Rrightarrow_\eta)^*; \equiv_\alpha; (\Lleftarrow + {}_\eta\Lleftarrow)^*]$

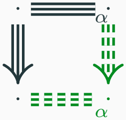
$$\begin{tikzcd} {} \arrow[dr, Rrightarrow, "*"] & & & {} \arrow[dl, Rrightarrow, "*"'] \\ & \cdot \arrow[r, phantom, "\equiv_\alpha"] & \cdot & \end{tikzcd}$$

Confluence

Required commutation bricks







(+ with $\Rrightarrow_\eta$)
(+ vertical symmetric)

Constraints on type conversion

0. Type uniqueness (or principality),
   To do any kind of derivation inversion.

1. Reflexivity and transitivity of type conversion (symmetry can also help greatly),
   To juggle with the different possible types that appear.

2. Inclusion of $\Rrightarrow$, $\Rrightarrow_\eta$ and $\equiv_\alpha$ in type conversion,
   Since types and contexts may reduce through the type annotations.

Constraints on type conversion

0. Type uniqueness
1. Type conversion PER
2. Inclusion of $\Rrightarrow$, $\Rrightarrow_\eta$ and $\equiv_\alpha$
3. During the $\Rrightarrow$ / $\Rrightarrow$ commutation, in the case β against congruence:

$$\begin{tikzcd}
& (\lambda x, t_0)u_0 \arrow[dl, Rrightarrow, "\text{CONGR}"'] \arrow[dr, Rrightarrow, "\beta"] & \\
(\lambda x, t_1)u_1 \arrow[dr, Rrightarrow, dashed, "\beta"'] & & t_2[x := u_2] \arrow[dl, Rrightarrow, dashed, "\text{SUBST}"] \\
& t_f[x := u_f] &
\end{tikzcd}$$

$\Rrightarrow$ needs to be substitutive (its whole point),

but this means that type conversion itself has to be substitutive.

Constraints on type conversion

0. Type uniqueness
1. Type conversion PER
2. Inclusion of $\Rrightarrow$, $\Rrightarrow_\eta$ and $\equiv_\alpha$
3. Type conversion substitutive

4. During the $\Rrightarrow$ / $\Rrightarrow_\eta$ commutation, in the case β against η:

$$
\begin{tikzcd}
& (\lambda x, t_0)u_0 \arrow[dl, Rrightarrow, "\beta"'] \arrow[dr, Rrightarrow, "\eta"] & \\
t_1[x := u_1] \arrow[dr, dashed, "\mathrm{SUBST}"'] & & (\lambda y, (\lambda x, t_2)y)u_2 \arrow[dl, dashed, "\beta^2"] \\
& t_f[x := u_f] &
\end{tikzcd}
$$

In a β-redex, $\lambda x, t : \Pi(x : A), B \equiv \Pi(x : A'), B'$ (application to u).
For a β-redex to be well typed, we need $A \equiv A'$ and $B \equiv B'$.
But here, we create a redex ex nihilo.

Therefore, we need injectivity of type conversion for type constructors.

Constraints on type conversion

0. Type uniqueness
1. Type conversion PER
2. Inclusion of $\Rrightarrow$, $\Rrightarrow_\eta$ and $\equiv_\alpha$
3. Type conversion substitutive
4. Injectivity of type constructors
5. During the $\Rrightarrow$ / $\Rrightarrow_\eta$ commutation, in the case β against η:

$$\begin{tikzcd}
 & (\lambda x, t_0) u_0 \arrow[dl, Rrightarrow, "\beta"'] \arrow[dr, Rrightarrow, "\eta"] & \\
t_1[x := u_1] \arrow[dr, dashed, Rrightarrow] & & \langle (\lambda x, t_2)_1, (\lambda x, t_2)_2 \rangle u_2 \arrow[dl, dashed, Rrightarrow] \\
 & ??? &
\end{tikzcd}$$

This η-expansion must be prevented: no term t may have both type $\Pi(x : A), B$ and $A' \times B'$.

For type conversion, this translates to non-confusion for type constructors ($\Pi(x : A), B \not\equiv A' \times B'$).

What definition for type conversion

0. Type uniqueness
1. Type conversion PER
2. Inclusion of $\Rrightarrow$, $\Rrightarrow_\eta$ and $\equiv_\alpha$
3. Type conversion substitutive
4. Injectivity of type constructors
5. Non-confusion for type conversion

The immediate candidates:

- $\equiv_s$ (1, 2, 3 but not 4, 5)
- $\equiv_j$ (2, 3, 4, 5 but not transitivity) (even by induction)
- $\equiv_a$ (same)

The possible side steps:

- 0 can be solved through either bidirectional typing or annotations
- Reflexivity and (2) can be waived, if transitivity changed into stability under $\Rrightarrow, \Rrightarrow_\eta, \equiv_\alpha$

Conclusion

1. When type conversion is typed and has η rules, injectivity and non-confusion of type constructors are central to the proof of adequacy between specification and implementation of a type theory

2. Transitivity is also required and difficult to side step

3. More work is needed to find the ideal definition for type conversion