Confluence Techniques for Dependent Type Theory with Typed Conversion

Thiago Felicissimo & Théo Winterhalter

27 August 2026 @ ICFP'26

INRIA

Dependent type theory

The foundation of many popular proof assistants:

(FKA Coq)

Dependent type theory

The foundation of many popular proof assistants:

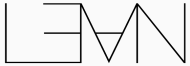
(FKA Coq)

Defining feature Types can depend on terms

$$[0, 1, 2] : \mathsf{Vec}\ \mathbb{N}\ 3$$

Dependent type theory

The foundation of many popular proof assistants:



(FKA Coq)

Defining feature Types can depend on terms

$$[0, 1, 2] : \mathsf{Vec}\ \mathbb{N}\ 3$$

Conversion Equational theory $\equiv$ dictates which terms are indiscernible (eg, $3 \equiv 1 + 2$)

Dependent type theory

The foundation of many popular proof assistants:

(FKA Coq)

Defining feature Types can depend on terms

$$[0, 1, 2] : \mathsf{Vec}\ \mathbb{N}\ 3$$

Conversion Equational theory $\equiv$ dictates which terms are indiscernible (eg, $3 \equiv 1 + 2$)

Due to type dependency, conversion impacts typing:

$$\frac{[0, 1, 2] : \mathsf{Vec}\ \mathbb{N}\ 3 \qquad 3 \equiv 1 + 2}{[0, 1, 2] : \mathsf{Vec}\ \mathbb{N}\ (1 + 2)}$$

Dependent type theory

The foundation of many popular proof assistants:

(FKA Coq)

Defining feature Types can depend on terms

$$[0, 1, 2] : \mathsf{Vec}\ \mathbb{N}\ 3$$

Conversion Equational theory $\equiv$ dictates which terms are indiscernible (eg, $3 \equiv 1 + 2$)

Due to type dependency, conversion impacts typing:

$$\frac{[0, 1, 2] : \mathsf{Vec}\ \mathbb{N}\ 3 \qquad 3 \equiv 1 + 2}{[0, 1, 2] : \mathsf{Vec}\ \mathbb{N}\ (1 + 2)}$$

This interaction is the most intricate feature of dependent types!

(Un)Typed Conversion

Historically, two ways to define conversion

(Un)Typed Conversion

Historically, two ways to define conversion

Untyped Conversion Relation $t \equiv u$ on untyped terms (as in untyped λ-calculus)

$$\frac{}{(\lambda x.t)u \equiv t[u/x]} \qquad \frac{t \equiv u}{u \equiv t} \qquad \frac{t \equiv t'}{\lambda x.t \equiv \lambda x.t'} \qquad \dots$$

(Un)Typed Conversion

Historically, two ways to define conversion

Untyped Conversion Relation $t \equiv u$ on untyped terms (as in untyped λ-calculus)

$$\frac{}{(\lambda x.t)u \equiv t[u/x]} \qquad \frac{t \equiv u}{u \equiv t} \qquad \frac{t \equiv t'}{\lambda x.t \equiv \lambda x.t'} \qquad \dots$$

Used in first dependent type systems (MLTT'72, CoC, ...) but also today (eg, MetaRocq)

(Un)Typed Conversion

Historically, two ways to define conversion

Untyped Conversion Relation $t \equiv u$ on untyped terms (as in untyped λ-calculus)

$$\frac{}{(\lambda x.t)u \equiv t[u/x]} \qquad \frac{t \equiv u}{u \equiv t} \qquad \frac{t \equiv t'}{\lambda x.t \equiv \lambda x.t'} \qquad \dots$$

Used in first dependent type systems (MLTT'72, CoC, ...) but also today (eg, MetaRocq)

✓ Conversion can be defined *before* typing, no mutual dependency

(Un)Typed Conversion

Historically, two ways to define conversion

Untyped Conversion Relation $t \equiv u$ on untyped terms (as in untyped λ-calculus)

$$\frac{}{(\lambda x.t)u \equiv t[u/x]} \qquad \frac{t \equiv u}{u \equiv t} \qquad \frac{t \equiv t'}{\lambda x.t \equiv \lambda x.t'} \qquad \dots$$

Used in first dependent type systems (MLTT'72, CoC, ...) but also today (eg, MetaRocq)

- ✓ Conversion can be defined *before* typing, no mutual dependency
- ✓ Confluence techniques, allow showing *injectivity/non-confusion properties*

(Un)Typed Conversion

Historically, two ways to define conversion

Untyped Conversion Relation $t \equiv u$ on untyped terms (as in untyped λ-calculus)

$$\frac{}{(\lambda x.t)u \equiv t[u/x]} \qquad \frac{t \equiv u}{u \equiv t} \qquad \frac{t \equiv t'}{\lambda x.t \equiv \lambda x.t'} \qquad \dots$$

Used in first dependent type systems (MLTT'72, CoC, ...) but also today (eg, MetaRocq)

- ✓ Conversion can be defined *before* typing, no mutual dependency
- ✓ Confluence techniques, allow showing *injectivity/non-confusion properties*

$$\Pi(x : A).B \equiv \Pi(x : A').B' \quad \rightsquigarrow \quad A \equiv A' \wedge B \equiv B'$$

(Un)Typed Conversion

Historically, two ways to define conversion

Untyped Conversion Relation $t \equiv u$ on untyped terms (as in untyped λ-calculus)

$$\frac{}{(\lambda x.t)u \equiv t[u/x]} \qquad \frac{t \equiv u}{u \equiv t} \qquad \frac{t \equiv t'}{\lambda x.t \equiv \lambda x.t'} \qquad \dots$$

Used in first dependent type systems (MLTT'72, CoC, ...) but also today (eg, MetaRocq)

- ✓ Conversion can be defined *before* typing, no mutual dependency
- ✓ Confluence techniques, allow showing *injectivity/non-confusion properties*

$$\Pi(x : A).B \equiv \Pi(x : A').B' \quad \rightsquigarrow \quad A \equiv A' \wedge B \equiv B'$$

Required for establishing correctness of implementations of type theory

Typed Conversion

Typed conversion judgment $\Gamma \vdash t \equiv u : A$ defined mutually with typing $\Gamma \vdash t : A$

$$\frac{\Gamma, x : A \vdash t : B \qquad \Gamma \vdash u : A}{\Gamma \vdash (\lambda x.t)u \equiv t[u/x] : B[u/x]} \qquad \frac{\Gamma \vdash t \equiv u : A}{\Gamma \vdash u \equiv t : A} \qquad \frac{\Gamma, x : A \vdash t \equiv t' : B}{\Gamma \vdash \lambda x.t \equiv \lambda x.t' : \Pi(x : A).B} \qquad \ldots$$

Typed Conversion

Typed conversion judgment $\Gamma \vdash t \equiv u : A$ defined mutually with typing $\Gamma \vdash t : A$

$$\frac{\Gamma, x : A \vdash t : B \qquad \Gamma \vdash u : A}{\Gamma \vdash (\lambda x.t)u \equiv t[u/x] : B[u/x]} \qquad \frac{\Gamma \vdash t \equiv u : A}{\Gamma \vdash u \equiv t : A} \qquad \frac{\Gamma, x : A \vdash t \equiv t' : B}{\Gamma \vdash \lambda x.t \equiv \lambda x.t' : \Pi(x : A).B} \qquad \ldots$$

In 2026, seems to be the standard definition of conversion (maybe still controversial?)

Typed Conversion

Typed conversion judgment $\Gamma \vdash t \equiv u : A$ defined mutually with typing $\Gamma \vdash t : A$

$$\frac{\Gamma, x : A \vdash t : B \qquad \Gamma \vdash u : A}{\Gamma \vdash (\lambda x.t)u \equiv t[u/x] : B[u/x]} \qquad \frac{\Gamma \vdash t \equiv u : A}{\Gamma \vdash u \equiv t : A} \qquad \frac{\Gamma, x : A \vdash t \equiv t' : B}{\Gamma \vdash \lambda x.t \equiv \lambda x.t' : \Pi(x : A).B} \qquad \dots$$

In 2026, seems to be the standard definition of conversion (maybe still controversial?)

- ✓ Better connection with semantics: with typed conversion, syntax can be assembled into initial models for usual notions of categorical semantics (see discussion in §8)

Typed Conversion

Typed conversion judgment $\Gamma \vdash t \equiv u : A$ defined mutually with typing $\Gamma \vdash t : A$

$$\frac{\Gamma, x : A \vdash t : B \qquad \Gamma \vdash u : A}{\Gamma \vdash (\lambda x.t)u \equiv t[u/x] : B[u/x]} \qquad \frac{\Gamma \vdash t \equiv u : A}{\Gamma \vdash u \equiv t : A} \qquad \frac{\Gamma, x : A \vdash t \equiv t' : B}{\Gamma \vdash \lambda x.t \equiv \lambda x.t' : \Pi(x : A).B} \qquad \dots$$

In 2026, seems to be the standard definition of conversion (maybe still controversial?)

- ✓ Better connection with semantics: with typed conversion, syntax can be assembled into initial models for usual notions of categorical semantics (see discussion in §8)

  Important for reasoning in *internal languages* (set theory, synthetic maths, . . .)

Typed Conversion

Typed conversion judgment $\Gamma \vdash t \equiv u : A$ defined mutually with typing $\Gamma \vdash t : A$

$$\frac{\Gamma, x : A \vdash t : B \qquad \Gamma \vdash u : A}{\Gamma \vdash (\lambda x.t)u \equiv t[u/x] : B[u/x]} \qquad \frac{\Gamma \vdash t \equiv u : A}{\Gamma \vdash u \equiv t : A} \qquad \frac{\Gamma, x : A \vdash t \equiv t' : B}{\Gamma \vdash \lambda x.t \equiv \lambda x.t' : \Pi(x : A).B} \qquad \dots$$

In 2026, seems to be the standard definition of conversion (maybe still controversial?)

✓ Better connection with semantics: with typed conversion, syntax can be assembled into initial models for usual notions of categorical semantics (see discussion in §8)

Important for reasoning in *internal languages* (set theory, synthetic maths, . . .)

✗ Most of the literature on confluence targets the untyped case, and exceptions are limited to rather basic type theories (Adams 2006 and Siles & Herbelin 2012)

Typed Conversion

Typed conversion judgment $\Gamma \vdash t \equiv u : A$ defined mutually with typing $\Gamma \vdash t : A$

$$\frac{\Gamma, x : A \vdash t : B \qquad \Gamma \vdash u : A}{\Gamma \vdash (\lambda x.t)u \equiv t[u/x] : B[u/x]} \qquad \frac{\Gamma \vdash t \equiv u : A}{\Gamma \vdash u \equiv t : A} \qquad \frac{\Gamma, x : A \vdash t \equiv t' : B}{\Gamma \vdash \lambda x.t \equiv \lambda x.t' : \Pi(x : A).B} \qquad \dots$$

In 2026, seems to be the standard definition of conversion (maybe still controversial?)

✓ Better connection with semantics: with typed conversion, syntax can be assembled into initial models for usual notions of categorical semantics (see discussion in §8)

Important for reasoning in *internal languages* (set theory, synthetic maths, . . .)

✗ Most of the literature on confluence targets the untyped case, and exceptions are limited to rather basic type theories (Adams 2006 and Siles & Herbelin 2012)

Injectivity properties are usually shown with complex logical relation arguments

This work: Combining confluence and typed conversion

This work: Combining confluence and typed conversion

We show confluence techniques can be scaled to rich theories with typed conversion

This work: Combining confluence and typed conversion

We show confluence techniques can be scaled to rich theories with typed conversion

Beyond universes and Π types (without η), as in prior work, our confluence proof handles:

This work: Combining confluence and typed conversion

We show confluence techniques can be scaled to rich theories with typed conversion

Beyond universes and Π types (without η), as in prior work, our confluence proof handles:

- Σ types (without η)

This work: Combining confluence and typed conversion

We show confluence techniques can be scaled to rich theories with typed conversion

Beyond universes and Π types (without η), as in prior work, our confluence proof handles:

- Σ types (without η)
- Lift types (**with** η), allow simulating weak form of cumulativity

This work: Combining confluence and typed conversion

We show confluence techniques can be scaled to rich theories with typed conversion

Beyond universes and Π types (without η), as in prior work, our confluence proof handles:

- Σ types (without η)
- Lift types (**with** η), allow simulating weak form of cumulativity
- A *definitionally proof-irrelevant universe* $\mathcal{U}^{\Omega}$ (Prop in Lean, SProp in Rocq)

$$\frac{\Gamma \vdash A : \mathcal{U}^{\Omega} \qquad \Gamma \vdash t : A \qquad \Gamma \vdash u : A}{\Gamma \vdash t \equiv u : A}$$

This work: Combining confluence and typed conversion

We show confluence techniques can be scaled to rich theories with typed conversion

Beyond universes and Π types (without η), as in prior work, our confluence proof handles:

- Σ types (without η)
- Lift types (**with** η), allow simulating weak form of cumulativity
- A *definitionally proof-irrelevant universe* $\mathcal{U}^{\Omega}$ (Prop in Lean, SProp in Rocq)

$$\frac{\Gamma \vdash A : \mathcal{U}^{\Omega} \qquad \Gamma \vdash t : A \qquad \Gamma \vdash u : A}{\Gamma \vdash t \equiv u : A}$$

- Some inductive types (Nat and Sum)

This work: Combining confluence and typed conversion

We show confluence techniques can be scaled to rich theories with typed conversion

Beyond universes and Π types (without η), as in prior work, our confluence proof handles:

- Σ types (without η)
- Lift types (**with** η), allow simulating weak form of cumulativity
- A *definitionally proof-irrelevant universe* $\mathcal{U}^{\Omega}$ (Prop in Lean, SProp in Rocq)

$$\frac{\Gamma \vdash A : \mathcal{U}^{\Omega} \qquad \Gamma \vdash t : A \qquad \Gamma \vdash u : A}{\Gamma \vdash t \equiv u : A}$$

- Some inductive types (Nat and Sum)

To illustrate extensibility of our proofs, we handle two flavours of proof-irrelevant equality:

This work: Combining confluence and typed conversion

We show confluence techniques can be scaled to rich theories with typed conversion

Beyond universes and Π types (without η), as in prior work, our confluence proof handles:

- Σ types (without η)
- Lift types (**with** η), allow simulating weak form of cumulativity
- A *definitionally proof-irrelevant universe* $\mathcal{U}^{\Omega}$ (Prop in Lean, SProp in Rocq)

$$\frac{\Gamma \vdash A : \mathcal{U}^{\Omega} \qquad \Gamma \vdash t : A \qquad \Gamma \vdash u : A}{\Gamma \vdash t \equiv u : A}$$

- Some inductive types (Nat and Sum)

To illustrate extensibility of our proofs, we handle two flavours of proof-irrelevant equality:

- McBride and Altenkirch's *observational equality*

This work: Combining confluence and typed conversion

We show confluence techniques can be scaled to rich theories with typed conversion

Beyond universes and Π types (without η), as in prior work, our confluence proof handles:

- Σ types (without η)
- Lift types (**with** η), allow simulating weak form of cumulativity
- A *definitionally proof-irrelevant universe* $\mathcal{U}^\Omega$ (Prop in Lean, SProp in Rocq)

$$\frac{\Gamma \vdash A : \mathcal{U}^\Omega \qquad \Gamma \vdash t : A \qquad \Gamma \vdash u : A}{\Gamma \vdash t \equiv u : A}$$

- Some inductive types (Nat and Sum)

To illustrate extensibility of our proofs, we handle two flavours of proof-irrelevant equality:

- McBride and Altenkirch's *observational equality*
- Lean's equality eliminator, with "*strong*" non-linear computation rule

$$\frac{\Gamma \vdash p : P[a/x] \qquad \Gamma \vdash e : a =_A a}{\Gamma \vdash \mathsf{transp}(x.P, p, a, a, e) \equiv p : P[a/x]}$$

The confluence proof

We introduce a typed *orthogonal (parallel) reduction* judgment $\Gamma \vdash t \Longrightarrow u : A$

The confluence proof

We introduce a typed *orthogonal (parallel) reduction* judgment $\Gamma \vdash t \Longrightarrow u : A$

Main points of definition:

The confluence proof

We introduce a typed *orthogonal (parallel) reduction* judgment $\Gamma \vdash t \Longrightarrow u : A$

Main points of definition:

- Need to keep type annotations and track their convertibility

$$\frac{\Gamma, x : A \vdash t \Longrightarrow t' : B \qquad \Gamma \vdash u \Longrightarrow u' : A}{\Gamma \vdash (\lambda x.t)u \Longrightarrow t'[u'/x] : B[u/x]}$$

The confluence proof

We introduce a typed *orthogonal (parallel) reduction* judgment $\Gamma \vdash t \Longrightarrow u : A$

Main points of definition:

- Need to keep type annotations and track their convertibility

$$\frac{\Gamma \vdash A \equiv A_0 : \mathcal{U}^{\ell} \qquad \Gamma, x : A \vdash B \equiv B_0 : \mathcal{U}^{\ell'} \qquad \Gamma, x : A \vdash t \Longrightarrow t' : B \qquad \Gamma \vdash u \Longrightarrow u' : A}{\Gamma \vdash (\lambda_{(x:A_0).B_0} x.t) @_{(x:A).B} u \Longrightarrow t'[u'/x] : B[u/x]}$$

The confluence proof

We introduce a typed *orthogonal (parallel) reduction* judgment $\Gamma \vdash t \Longrightarrow u : A$

Main points of definition:

- Need to keep type annotations and track their convertibility

$$\frac{\Gamma \vdash A \equiv A_0 : \mathcal{U}^{\ell} \qquad \Gamma, x : A \vdash B \equiv B_0 : \mathcal{U}^{\ell'} \qquad \Gamma, x : A \vdash t \Longrightarrow t' : B \qquad \Gamma \vdash u \Longrightarrow u' : A}{\Gamma \vdash (\lambda_{(x:A_0).B_0} x.t) @_{(x:A).B} u \Longrightarrow t'[u'/x] : B[u/x]}$$

  Moreover, annotations help with semantics and can be elaborated away

The confluence proof

We introduce a typed *orthogonal (parallel) reduction* judgment $\Gamma \vdash t \Longrightarrow u : A$

Main points of definition:

- Need to keep type annotations and track their convertibility
- Proof irrelevance handled in trivial manner

$$\frac{\Gamma \vdash^{\Omega} t : T \qquad \Gamma \vdash^{\Omega} u : T}{\Gamma \vdash^{\Omega} t \Longrightarrow u : T}$$

The confluence proof

We introduce a typed *orthogonal (parallel) reduction* judgment $\Gamma \vdash t \Longrightarrow u : A$

Main points of definition:

- Need to keep type annotations and track their convertibility
- Proof irrelevance handled in trivial manner
- η-equality for Lift treated as reduction

$$\frac{\Gamma \vdash A \equiv A_0 : \mathcal{U}^{\mathrm{n}} \qquad \Gamma \vdash t \Longrightarrow t' : \mathsf{Lift}^{\mathrm{n}}(A)}{\Gamma \vdash \mathsf{lift}_A(\mathsf{lower}_{A_0}(t)) \Longrightarrow t' : \mathsf{Lift}^{\mathrm{n}}(A)}$$

The confluence proof

We introduce a typed *orthogonal (parallel) reduction* judgment $\Gamma \vdash t \Longrightarrow u : A$

Main points of definition:

- Need to keep type annotations and track their convertibility
- Proof irrelevance handled in trivial manner
- η-equality for Lift treated as reduction
- Non-linear rule for transp treated as conditional rule

$$\frac{\Gamma \vdash A : \mathcal{U}^{\ell} \qquad \Gamma \vdash a \equiv b : A \qquad \Gamma, x : A \vdash P : \mathcal{U}^{\ell'} \qquad \Gamma \vdash p \Longrightarrow p' : P[a/x] \qquad \Gamma \vdash e : a =_A b}{\Gamma \vdash \mathsf{transp}^{\ell,\ell'}\,(A, x.P, p, a, b, e) \Longrightarrow p' : P[b/x]}$$

The confluence proof

We introduce a typed *orthogonal (parallel) reduction* judgment $\Gamma \vdash t \Longrightarrow u : A$

Main points of definition:

- Need to keep type annotations and track their convertibility
- Proof irrelevance handled in trivial manner
- η-equality for Lift treated as reduction
- Non-linear rule for transp treated as conditional rule

We have $\Longrightarrow$ included in $\equiv$ by construction

The confluence proof

We introduce a typed *orthogonal (parallel) reduction* judgment $\Gamma \vdash t \Longrightarrow u : A$

Main points of definition:

- Need to keep type annotations and track their convertibility
- Proof irrelevance handled in trivial manner
- η-equality for Lift treated as reduction
- Non-linear rule for transp treated as conditional rule

We have $\Longrightarrow$ included in $\equiv$ by construction

Main Theorem If $\Gamma \vdash t \equiv u : A$ then $\Gamma \vdash t \Longrightarrow^* v \ {}^*\!\Longleftarrow u : A$ for some v.

Follows from two lemmas: (1) diamond property for $\Longrightarrow$; (2) inclusion of $\equiv$ in $(\Longrightarrow \cup \Longleftarrow)^*$

The confluence proof

We introduce a typed *orthogonal (parallel) reduction* judgment $\Gamma \vdash t \Longrightarrow u : A$

Main points of definition:

- Need to keep type annotations and track their convertibility
- Proof irrelevance handled in trivial manner
- η-equality for Lift treated as reduction
- Non-linear rule for transp treated as conditional rule

We have $\Longrightarrow$ included in $\equiv$ by construction

Main Theorem If $\Gamma \vdash t \equiv u : A$ then $\Gamma \vdash t \Longrightarrow^* v \ {}^*\!\Longleftarrow u : A$ for some v.

Follows from two lemmas: (1) diamond property for $\Longrightarrow$; (2) inclusion of $\equiv$ in $(\Longrightarrow \cup \Longleftarrow)^*$

Corollary Injectivity and non-confusion of type formers, eg

$$\Gamma \vdash \Pi(x : A).B \equiv \Pi(x : A').B' : \mathcal{U}^\ell \quad \rightsquigarrow \quad \Gamma \vdash A \equiv A' : \mathcal{U}^{\ell_A} \quad \wedge \quad \Gamma, x : A \vdash B \equiv B' : \mathcal{U}^{\ell_B}$$

$$\Gamma \vdash \Pi(x : A).B \equiv \mathsf{Nat} : \mathcal{U}^\ell \quad \rightsquigarrow \quad \bot$$

Reaping the fruits: Correctness of conversion- and type-checking

Reaping the fruits: Correctness of conversion- and type-checking

Conversion-checking Reduce-and-compare, using algorithmic untyped full reduction $\searrow$

$$\frac{t \searrow t' \qquad u \searrow u' \qquad t' = u'}{t \equiv_? u}$$

Reaping the fruits: Correctness of conversion- and type-checking

Conversion-checking Reduce-and-compare, using algorithmic untyped full reduction $\searrow$

$$\frac{t \searrow t' \qquad u \searrow u' \qquad t' = u'}{t \equiv_? u}$$

But core syntax too verbose, unsuited for implementations

Reaping the fruits: Correctness of conversion- and type-checking

Conversion-checking Reduce-and-compare, using algorithmic untyped full reduction $\searrow$

$$\frac{t \searrow t' \qquad u \searrow u' \qquad t' = u'}{t \equiv_? u}$$

But core syntax too verbose, unsuited for implementations

Instead, defined over *erased syntax*; erasure $|-|$ removes proofs and most annotations

$$|p|_\Omega := \square \qquad |\lambda^{\ell,\ell'}_{(x:A).B} x.t|_{\mathrm{n}} := \lambda x.|t|_{\ell'} \qquad \ldots$$

Reaping the fruits: Correctness of conversion- and type-checking

Conversion-checking Reduce-and-compare, using algorithmic untyped full reduction $\searrow$

$$\frac{t \searrow t' \qquad u \searrow u' \qquad t' = u'}{t \equiv_? u}$$

But core syntax too verbose, unsuited for implementations

Instead, defined over *erased syntax*; erasure $|-|$ removes proofs and most annotations

$$|p|_\Omega := \square \qquad |\lambda^{\ell,\ell'}_{(x:A).B} x.t|_{\mathrm{n}} := \lambda x.|t|_{\ell'} \qquad \ldots$$

Given $\Gamma \vdash^\ell t, u : T$, if $|t|_\ell \equiv_? |u|_\ell$ then $\Gamma \vdash t \equiv u : T$; converse provided normalization

Reaping the fruits: Correctness of conversion- and type-checking

Conversion-checking Reduce-and-compare, using algorithmic untyped full reduction $\searrow$

$$\frac{t \searrow t' \qquad u \searrow u' \qquad t' = u'}{t \equiv_? u}$$

But core syntax too verbose, unsuited for implementations

Instead, defined over *erased syntax*; erasure $|-|$ removes proofs and most annotations

$$|p|_\Omega := \square \qquad |\lambda^{\ell,\ell'}_{(x:A).B} x.t|_{\mathrm{n}} := \lambda x.|t|_{\ell'} \qquad \ldots$$

Given $\Gamma \vdash^\ell t, u : T$, if $|t|_\ell \equiv_? |u|_\ell$ then $\Gamma \vdash t \equiv u : T$; converse provided normalization

Type-checking Bidirectional type-checking system, taking as input user-friendly syntax

Reaping the fruits: Correctness of conversion- and type-checking

Conversion-checking Reduce-and-compare, using algorithmic untyped full reduction $\searrow$

$$\frac{t \searrow t' \qquad u \searrow u' \qquad t' = u'}{t \equiv_? u}$$

But core syntax too verbose, unsuited for implementations

Instead, defined over *erased syntax*; erasure $|-|$ removes proofs and most annotations

$$|p|_\Omega := \square \qquad |\lambda^{\ell,\ell'}_{(x:A).B} x.t|_{\mathrm{n}} := \lambda x.|t|_{\ell'} \qquad \ldots$$

Given $\Gamma \vdash^\ell t, u : T$, if $|t|_\ell \equiv_? |u|_\ell$ then $\Gamma \vdash t \equiv u : T$; converse provided normalization

Type-checking Bidirectional type-checking system, taking as input user-friendly syntax

Once again, we prove soundness and partial completeness

Reaping the fruits: Correctness of conversion- and type-checking

Conversion-checking Reduce-and-compare, using algorithmic untyped full reduction $\searrow$

$$\frac{t \searrow t' \qquad u \searrow u' \qquad t' = u'}{t \equiv_? u}$$

But core syntax too verbose, unsuited for implementations

Instead, defined over *erased syntax*; erasure $|-|$ removes proofs and most annotations

$$|p|_\Omega := \square \qquad |\lambda^{\ell,\ell'}_{(x:A).B} x.t|_{\mathrm{n}} := \lambda x.|t|_{\ell'} \qquad \ldots$$

Given $\Gamma \vdash^\ell t, u : T$, if $|t|_\ell \equiv_? |u|_\ell$ then $\Gamma \vdash t \equiv u : T$; converse provided normalization

Type-checking Bidirectional type-checking system, taking as input user-friendly syntax

Once again, we prove soundness and partial completeness

Completeness result is *ecumenical*: we handle both French (Rocq) and Swedish (Agda) styles

Reaping the fruits: Correctness of conversion- and type-checking

Conversion-checking Reduce-and-compare, using algorithmic untyped full reduction $\searrow$

$$\frac{t \searrow t' \qquad u \searrow u' \qquad t' = u'}{t \equiv_? u}$$

But core syntax too verbose, unsuited for implementations

Instead, defined over *erased syntax*; erasure $|-|$ removes proofs and most annotations

$$|p|_\Omega := \square \qquad |\lambda^{\ell,\ell'}_{(x:A).B} x.t|_{\mathrm{n}} := \lambda x.|t|_{\ell'} \qquad \ldots$$

Given $\Gamma \vdash^\ell t, u : T$, if $|t|_\ell \equiv_? |u|_\ell$ then $\Gamma \vdash t \equiv u : T$; converse provided normalization

Type-checking Bidirectional type-checking system, taking as input user-friendly syntax

Once again, we prove soundness and partial completeness

Completeness result is *ecumenical*: we handle both French (Rocq) and Swedish (Agda) styles

See §4 and §5 of paper for more!

Conclusion

We scaled confluence techniques to rich theories with typed conversion, all formalized in Rocq!

Conclusion

We scaled confluence techniques to rich theories with typed conversion, all formalized in Rocq!

Many interesting features Proof irrelevance, η for Lift, non-linear rule for transp, etc

Conclusion

We scaled confluence techniques to rich theories with typed conversion, all formalized in Rocq!

Many interesting features Proof irrelevance, η for Lift, non-linear rule for transp, etc

Application Correctness of conversion/type-checking, on usual non-annotated syntaxes

Conclusion

We scaled confluence techniques to rich theories with typed conversion, all formalized in Rocq!

Many interesting features Proof irrelevance, η for Lift, non-linear rule for transp, etc

Application Correctness of conversion/type-checking, on usual non-annotated syntaxes

Limitations (see §7) No $\underbrace{\eta \text{ for } \Pi}$, Σ and Unit

Difference with the untyped case!

Conclusion

We scaled confluence techniques to rich theories with typed conversion, all formalized in Rocq!

Many interesting features Proof irrelevance, η for Lift, non-linear rule for transp, etc

Application Correctness of conversion/type-checking, on usual non-annotated syntaxes

Limitations (see §7) No $\underbrace{\eta \text{ for } \Pi}_{\text{Difference with the untyped case!}}$, Σ and Unit

Takeaway Confluence is not at odds with typed conversion, but rather with η

Conclusion

We scaled confluence techniques to rich theories with typed conversion, all formalized in Rocq!

Many interesting features Proof irrelevance, η for Lift, non-linear rule for transp, etc

Application Correctness of conversion/type-checking, on usual non-annotated syntaxes

Limitations (see §7) No η for Π, Σ and Unit

Difference with the untyped case!

Takeaway Confluence is not at odds with typed conversion, but rather with η

Question: Can we go beyond η for Lift with these techniques?

Conclusion

We scaled confluence techniques to rich theories with typed conversion, all formalized in Rocq!

Many interesting features Proof irrelevance, η for Lift, non-linear rule for transp, etc

Application Correctness of conversion/type-checking, on usual non-annotated syntaxes

Limitations (see §7) No η for Π, Σ and Unit

Difference with the untyped case!

Takeaway Confluence is not at odds with typed conversion, but rather with η

Question: Can we go beyond η for Lift with these techniques?

Thank you for your attention!