Injectivity of type constructors

Implementation of type theory seminar – March 10th 2026

What am I trying to do?

Verifying the verifier

We keep telling the world they should verify their critical code...

Verifying the verifier

We keep telling the world they should verify their critical code...
A lot of the verification ecosystem relies on proof assistant kernels...

Verifying the verifier

We keep telling the world they should verify their critical code...
A lot of the verification ecosystem relies on proof assistant kernels...

Verifying the verifier

We keep telling the world they should verify their critical code...
A lot of the verification ecosystem relies on proof assistant kernels...

The programs are (relatively) simple...

Verifying the verifier

We keep telling the world they should verify their critical code...
A lot of the verification ecosystem relies on proof assistant kernels...

The programs are (relatively) simple...

But the reasons why they work are complicated!

Where I'm coming from

- MetaRocq: Rocq in Rocq
- *Martin-Löf à la Coq*: a place to experiment

Where I'm coming from

- MetaRocq: Rocq in Rocq
- *Martin-Löf à la Coq*: a place to experiment

Today: some things I've learned to care about there
Main reference: *What Does It Take to Certify a Conversion Checker?* (FSCD 2025)

Logical relations

Irrelevant
computational content

Meta-theory

Models

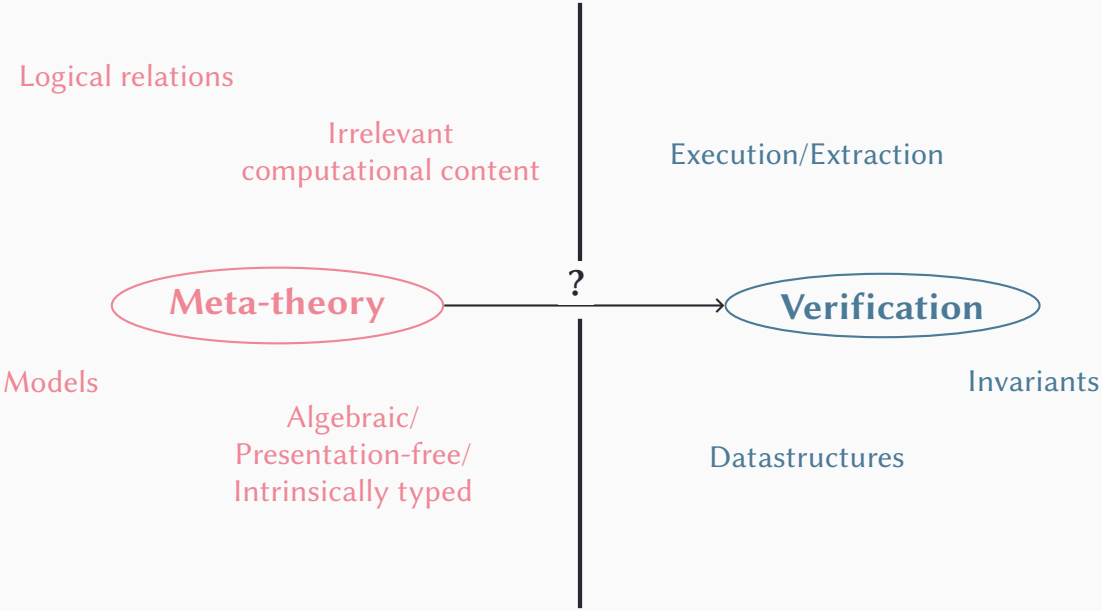
Algebraic/
Presentation-free/
Intrinsically typed

Execution/Extraction

Verification

Invariants

Datastructures


Logical relations
Irrelevant
computational content
Execution/Extraction
Meta-theory
?
Verification
Models
Algebraic/
Presentation-free/
Intrinsically typed
Invariants
Datastructures

The algorithms (and their specification)

Specification

Rules for each term/type former
$(\Pi, \Sigma, \mathrm{Id}, \mathcal{U}, \mathbb{N}, \bot \ldots)$ +

$$\frac{\Gamma \vdash t : T \qquad \Gamma \vdash T \cong T'}{\Gamma \vdash t : T'}$$

Specification

Rules for each term/type former $(\Pi, \Sigma, \mathrm{Id}, \mathcal{U}, \mathbb{N}, \bot \ldots)$ +

$$\frac{\Gamma \vdash t : T \qquad \Gamma \vdash T \cong T'}{\Gamma \vdash t : T'}$$

Algorithm

Bidirectional: rules for each term former integrate (some) conversion.

Information Must Flow

Inference and checking

$\Gamma \vdash t : A$ separates into

inference: $\Gamma \vdash t \triangleright A$ checking: $\Gamma \vdash t \triangleleft A$

Similar “meaning”, different operational modes: input/subject/output.

Structure!

$$\frac{\Gamma \vdash t : \Pi\, x{:}\, A.B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\ u : B[u]}$$

Structure!

$$\frac{\Gamma \vdash t : \Pi\, x{:}\, A.B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\ u : B[u]}$$

$$\frac{\Gamma \vdash t \triangleright_{\mathrm{h}} \Pi\, x{:}\, A.B \qquad \Gamma \vdash u \triangleleft A}{\Gamma \vdash t\ u \triangleright B[u]}$$

- Clear **information flow**

Structure!

$$\frac{\Gamma \vdash t : \Pi\, x{:}\, A.B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\ u : B[u]}$$

$$\frac{\Gamma \vdash t \triangleright_{\mathrm{h}} \Pi\, x{:}\, A.B \qquad \Gamma \vdash u \triangleleft A}{\Gamma \vdash t\ u \triangleright B[u]}$$

$$\frac{\Gamma \vdash t : T \qquad \Gamma \vdash T \cong T'}{\Gamma \vdash t : T'}$$

- Clear information flow
- ~~Unconstrained conversion~~ $\longrightarrow$ Computation ($\rightsquigarrow^\star$ or $\cong$) **guided by the mode**

Structure!

$$\frac{\Gamma \vdash t : \Pi\, x{:}\, A.B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\ u : B[u]} \qquad \frac{\Gamma \vdash t \triangleright_{\mathrm{h}} \Pi\, x{:}\, A.B \qquad \Gamma \vdash u \triangleleft A}{\Gamma \vdash t\ u \triangleright B[u]}$$

$$\frac{\Gamma \vdash t : T \qquad \Gamma \vdash T \cong T'}{\Gamma \vdash t : T'} \qquad \frac{\Gamma \vdash t \triangleright T \qquad \Gamma \vdash T \cong T' \triangleleft}{\Gamma \vdash t \triangleleft T'} \qquad \frac{\Gamma \vdash t \triangleright T \qquad T \rightsquigarrow^{\star} T'}{\Gamma \vdash t \triangleright_{\mathrm{h}} T'}$$

- Clear information flow
- ~~Unconstrained conversion~~ $\longrightarrow$ Computation ($\rightsquigarrow^{\star}$ or $\cong$) **guided by the mode**

Information Must Flow

Inference and checking

$\Gamma \vdash t : A$ separates into

inference: $\Gamma \vdash t \triangleright A$ checking: $\Gamma \vdash t \triangleleft A$

Similar "meaning", different operational modes: input/subject/output.

McBride: ***A rule is a server for its conclusion and a client for its premises.***

- Modes guide invariant preservation
- In a conclusion, you assume inputs are well-formed, and ensure outputs are
- In a premise, you ensure inputs are well-formed, and assume outputs are

Declarative specification

Arbitrarily mixing:

- Refl./Sym./Trans.
- Congruences
- Computation (β)
- Extensionality (η)

Typed!

Conversion

Declarative specification

Arbitrarily mixing:

- Refl./Sym./Trans.
- Congruences
- Computation (β)
- Extensionality (η)

Typed!

Type-directed algo.

Alternate

1. β-reduction to whnf
2. **Type**-directed η
3. **Head** congruences

Conversion

Declarative specification

Arbitrarily mixing:

- Refl./Sym./Trans.
- Congruences
- Computation (β)
- Extensionality (η)

Typed!

Type-directed algo.

Alternate

1. β-reduction to whnf
2. **Type**-directed η
3. Head congruences

“Untyped” algo.

Alternate

1. β-reduction to whnf
2. **Term**-directed η
3. Head congruences

Conversion

Declarative specification

Arbitrarily mixing:

- Refl./Sym./Trans.
- Congruences
- Computation (β)
- Extensionality (η)

Typed!

Type-directed algo.

Alternate

1. β-reduction to whnf
2. Type-directed η
3. Head congruences

\+ closer to specification
\+ supports fancier rules
\- slower

“Untyped” algo.

Alternate

1. β-reduction to whnf
2. Term-directed η
3. Head congruences

\+ faster
\+ simpler (?)
\- further from spec.

Type-directed conversion

$$x : \mathbb{N} \to \mathbb{N} \vdash x \cong x : \mathbb{N} \to \mathbb{N}$$

Type-directed conversion

$$\dfrac{\dfrac{}{x{:}\,\mathbb{N} \to \mathbb{N},\, y{:}\,\mathbb{N} \vdash x\ y \cong x\ y : \mathbb{N}}}{x{:}\,\mathbb{N} \to \mathbb{N} \vdash x \cong x : \mathbb{N} \to \mathbb{N}}$$

Type-directed conversion

$$\dfrac{\dfrac{}{x\colon \mathbb{N} \to \mathbb{N},\, y\colon \mathbb{N} \vdash x\ y \cong x\ y : \mathbb{N}}}{x\colon \mathbb{N} \to \mathbb{N} \vdash x \cong x : \mathbb{N} \to \mathbb{N}}$$

Type-directed conversion

$$\dfrac{\dfrac{}{x\colon \mathbb{N} \to \mathbb{N}, y\colon \mathbb{N} \vdash x\ y \approx x\ y : \mathbb{N}}}{x\colon \mathbb{N} \to \mathbb{N}, y\colon \mathbb{N} \vdash x\ y \cong x\ y : \mathbb{N}}}{x\colon \mathbb{N} \to \mathbb{N} \vdash x \cong x : \mathbb{N} \to \mathbb{N}}$$

Type-directed conversion

$$\dfrac{\dfrac{(x\colon \mathbb{N} \to \mathbb{N}) \in x\colon \mathbb{N} \to \mathbb{N}, y\colon \mathbb{N}}{x\colon \mathbb{N} \to \mathbb{N}, y\colon \mathbb{N} \vdash x \approx x : \mathbb{N} \to \mathbb{N}} \qquad \dfrac{\ldots}{x\colon \mathbb{N} \to \mathbb{N}, y\colon \mathbb{N} \vdash y \cong y : \mathbb{N}}}{\dfrac{x\colon \mathbb{N} \to \mathbb{N}, y\colon \mathbb{N} \vdash x\ y \approx x\ y : \mathbb{N}}{\dfrac{x\colon \mathbb{N} \to \mathbb{N}, y\colon \mathbb{N} \vdash x\ y \cong x\ y : \mathbb{N}}{x\colon \mathbb{N} \to \mathbb{N} \vdash x \cong x : \mathbb{N} \to \mathbb{N}}}}$$

Type-directed conversion

$$\dfrac{\dfrac{\dfrac{(x\colon \mathbb{N} \to \mathbb{N}) \in x\colon \mathbb{N} \to \mathbb{N}, y\colon \mathbb{N}}{x\colon \mathbb{N} \to \mathbb{N}, y\colon \mathbb{N} \vdash x \approx x : \mathbb{N} \to \mathbb{N}} \quad \dfrac{\ldots}{x\colon \mathbb{N} \to \mathbb{N}, y\colon \mathbb{N} \vdash y \cong y : \mathbb{N}}}{x\colon \mathbb{N} \to \mathbb{N}, y\colon \mathbb{N} \vdash x\ y \approx x\ y : \mathbb{N}}}{x\colon \mathbb{N} \to \mathbb{N}, y\colon \mathbb{N} \vdash x\ y \cong x\ y : \mathbb{N}}}{x\colon \mathbb{N} \to \mathbb{N} \vdash x \cong x : \mathbb{N} \to \mathbb{N}}$$

Term-directed conversion

$$\dfrac{\dfrac{}{x \approx x}}{x \cong x}$$

When is this valid?

The plan

What's in a decision procedure?

$$P : D \to \mathbb{P} \qquad \overset{?}{\Leftrightarrow} \qquad p : D \to \mathbb{B}$$

What's in a decision procedure?

$$P : D \to \mathbb{P} \qquad \overset{?}{\Leftrightarrow} \qquad p : D \to \mathbb{B}$$

0. decidability: $(p\ d = \text{true}) \vee (p\ d = \text{false})$

What's in a decision procedure?

$$P : D \to \mathbb{P} \qquad \overset{?}{\Leftrightarrow} \qquad p : D \to \mathbb{B}$$

0. decidability: $(p\ d = \text{true}) \vee (p\ d = \text{false})$

1. soundness: $p\ d = \text{true} \Rightarrow P\ d$

2. completeness: $P\ d \Rightarrow p\ d = \text{true}$

3. profit!

What's in a decision procedure?

$$P : D \to \mathbb{P} \qquad \overset{?}{\Leftrightarrow} \qquad p : D \to \mathbb{B}$$

0. decidability: $(p\ d = \text{true}) \vee (p\ d = \text{false})$

1. soundness: $p\ d = \text{true} \Rightarrow P\ d$

2. completeness: $P\ d \Rightarrow p\ d = \text{true}$

3. **profit?**

What's in a decision procedure?

$$P : D \to \mathbb{P} \qquad \overset{?}{\Leftrightarrow} \qquad p : D \to \mathbb{B}$$

0. **decidability**: $(p\ d = \text{true}) \vee (p\ d = \text{false})$
How do you know that the type-checker terminates?

1. soundness: $p\ d = \text{true} \Rightarrow P\ d$

2. completeness: $P\ d \Rightarrow p\ d = \text{true}$

3. profit?

What's in a decision procedure?

$$P : D \to \mathbb{P} \quad \overset{?}{\Leftrightarrow} \quad p : D \rightharpoonup \mathbb{B}$$

0. **decidability**: $(p\ d = \text{true}) \vee (p\ d = \text{false})$
   How do you know that the type-checker terminates?
1. **soundness**: $p\ d = \text{true} \Rightarrow P\ d$
   Look at the trace of the type-checker
2. completeness: $P\ d \Rightarrow p\ d = \text{true}$

3. profit?

What's in a decision procedure?

$$P : D \to \mathbb{P} \quad \overset{?}{\Leftrightarrow} \quad p : D \rightharpoonup \mathbb{B}$$

0. **decidability**: $(p\ d = \text{true}) \vee (p\ d = \text{false})$
   How do you know that the type-checker terminates?
1. **soundness**: $p\ d = \text{true} \Rightarrow P\ d$
   Look at the trace of the type-checker
2. **completeness**: $P\ d \Rightarrow p\ d = \text{true}$
   reflexivity $\simeq$ normalisation
3. profit?

Better abstract nonsense

$$P : D \to \mathbb{P} \qquad \overset{?}{\Leftrightarrow} \qquad p : D \rightharpoonup \mathbb{B}$$

1. **positive soundness**: $p\ d = \text{true} \Rightarrow P\ d$
   Look at the trace of the type-checker

Better abstract nonsense

$$P : D \to \mathbb{P} \qquad \overset{?}{\Leftrightarrow} \qquad p : D \rightharpoonup \mathbb{B}$$

1. **positive soundness**: $p\ d = \text{true} \Rightarrow P\ d$
   Look at the trace of the type-checker
2. **negative soundness**: $p\ d = \text{false} \Rightarrow \neg(P\ d)$
   Look (harder) at the trace of the type-checker

Better abstract nonsense

$$P : D \to \mathbb{P} \qquad \overset{?}{\Leftrightarrow} \qquad p : D \rightharpoonup \mathbb{B}$$

1. **positive soundness**: $p\ d = \text{true} \Rightarrow P\ d$
   Look at the trace of the type-checker
2. **negative soundness**: $p\ d = \text{false} \Rightarrow \neg(P\ d)$
   Look (harder) at the trace of the type-checker
3. **termination**: $(p\ d = \text{true}) \vee (p\ d = \text{false})$
   Still hard, of course...

Better abstract nonsense

$$P : D \to \mathbb{P} \quad \overset{?}{\Leftrightarrow} \quad p : D \rightharpoonup \mathbb{B}$$

1. **positive soundness**: $p\ d = \text{true} \Rightarrow P\ d$
   Look at the trace of the type-checker
2. **negative soundness**: $p\ d = \text{false} \Rightarrow \neg(P\ d)$
   Look (harder) at the trace of the type-checker
3. **termination**: $(p\ d = \text{true}) \vee (p\ d = \text{false})$
   Still hard, of course...

Now we have a plan

What meta-theory do we need?

Neutrals and normals

The least amount of information on a term that is stable by conversion

Neutrals and normals

The least amount of information on a term that is stable by conversion

(Weak-head) normal forms

Canonical forms

$$\boxed{\mathrm{can}\ f} \quad \stackrel{\text{def}}{=} \quad \mathcal{U} \mid \Pi\, x{:}t.t \mid \mathbb{N} \mid \Sigma\, x{:}t.t \mid \lambda\, x{:}t.t \mid 0 \mid \mathrm{S}(t) \mid (t,t) \mid \ldots$$

Neutrals and normals

The least amount of information on a term that is stable by conversion

(Weak-head) normal forms

Canonical forms, or neutrals

$$\boxed{\mathrm{can}\ f} \quad \stackrel{\text{def}}{=} \quad \mathcal{U} \mid \Pi\, x{:}t.t \mid \mathbb{N} \mid \Sigma\, x{:}t.t \mid \lambda\, x{:}t.t \mid 0 \mid \mathrm{S}(t) \mid (t,t) \mid \ldots$$

$$\boxed{\mathrm{nf}\ f} \quad \stackrel{\text{def}}{=} \quad \mathrm{can}\ f \vee \mathrm{ne}\ f$$

(Weak-head) neutrals

Elimination, blocked on a variable

$$\boxed{\mathrm{ne}\ n} \quad := \quad x \mid n\ t \mid \pi_1\ n \mid \pi_2\ n \mid \mathrm{rec}_{\mathbb{N}}(n, x.t, t, x.y.t) \mid \ldots$$

$x{:}\mathbb{N} \to \mathbb{N} \times \mathbb{N}, \ldots \vdash \mathrm{rec}_{\mathbb{N}}(\pi_1(x\ 7), P, b_0, b_S) : P$

Injectivity and no-confusion of type constructors

If $\Gamma \vdash T \cong T'$ and T, T' are weak-head normal form, then:

- $T = \mathbb{N} = T'$
- or $T = \Pi\, x{:}\, A.B$, $T' = \Pi\, x{:}\, A'.B'$, with $\Gamma \vdash A' \cong A$ and $\Gamma, x{:}\, A' \vdash B \cong B'$
- or ...
- or T, T' are both neutral, and $\Gamma \vdash T \cong T' : \mathcal{U}$

Any non-diagonal case is impossible (*no-confusion*).

Note: easy to break! Even in a consistent system!

Injectivity and no-confusion of type constructors

Injectivity and no-confusion at $\mathbb{N}$

If $\Gamma \vdash n \cong n' : \mathbb{N}$ and n, n' are weak-head normal forms, then:

- $n = 0 = n'$
- or $n = \mathrm{S}(t), n' = \mathrm{S}(t')$, with $\Gamma \vdash t \cong t' : \mathbb{N}$
- or n, n' are both neutral.

The good properties

Injectivity and no-confusion of type constructors

Injectivity and no-confusion at $\mathbb{N}$

Injectivity and no-confusion at $\mathcal{U}$

...

The good properties

Injectivity and no-confusion of type constructors

Injectivity and no-confusion at $\mathbb{N}$

Injectivity and no-confusion at $\mathcal{U}$

Injectivity of neutral eliminators*

If $\Gamma \vdash n \cong n' : T$ and n and n' are neutrals, then

- $n = x = n'$
- or $n = m\ u, n' = m'\ u'$ with $m \cong m'$ and $u \cong u'$
- or $n = \mathrm{rec}_{\mathbb{N}}(m, x.P, t_0, x.y.t_{\mathsf{S}}), n' = \mathrm{rec}_{\mathbb{N}}(m', x.P', t_0', x.y.t_{\mathsf{S}}')$, and ...

* More on this in a moment

The good properties

Injectivity and no-confusion of type constructors

Injectivity and no-confusion at $\mathbb{N}$

Injectivity and no-confusion at $\mathcal{U}$

Injectivity of neutral eliminators*

* More on this in a moment

Deep normalisation

Every well-typed term is deeply normalising at its type.
Every well-formed type is deeply normalising.

Positive soundness

Should be easy, right? All algorithmic rules are also valid declaratively and tada!

Positive soundness

Should be easy, right? All algorithmic rules are also valid declaratively and tada?
Problem: we don't want to re-check everything all the time...

Positive soundness

Should be easy, right? All algorithmic rules are also valid declaratively and tada!
Problem: we don't want to re-check everything all the time...

Invariants: *A rule is a server for its conclusion and a client for its premises*

+ no need to re-validate things all the time
- need to establish invariants

Positive soundness

Should be easy, right? All algorithmic rules are also valid declaratively and tada!
Problem: we don't want to re-check everything all the time...

Invariants: *A rule is a server for its conclusion and a client for its premises*

+ no need to re-validate things all the time
- need to establish invariants

Main lemma = subject reduction (also called preservation, etc.)

$$\Gamma \vdash T \wedge T \rightsquigarrow^\star T' \Rightarrow \Gamma \vdash T \cong T'$$

Why is SR hard?

Generation: eventually, you hit a proper structural rule

Have
$$\dfrac{\dfrac{\Gamma, x{:}\,A' \vdash t : B' \qquad \Gamma \vdash \Pi\, x{:}\,A'.B' \cong \Pi\, x{:}\,A.B}{\Gamma \vdash \lambda\, x.t : \Pi\, x{:}\,A.B} \qquad \Gamma \vdash u : A}{\Gamma \vdash (\lambda\, x.t)\, u : B[u]}$$

Want
$$\dfrac{\Gamma, x{:}\,A \vdash t : B \qquad \Gamma \vdash u : A}{\Gamma \vdash (\lambda\, x.t)\, u \cong t[u] : B[u]}$$

Why is SR hard?

Generation: eventually, you hit a proper structural rule

$$\text{Have}\quad \frac{\dfrac{\Gamma, x{:}\,A' \vdash t : B' \qquad \Gamma \vdash \Pi\, x{:}\,A'.B' \cong \Pi\, x{:}\,A.B}{\Gamma \vdash \lambda\, x.t : \Pi\, x{:}\,A.B} \qquad \Gamma \vdash u : A}{\Gamma \vdash (\lambda\, x.t)\, u : B[u]}$$

$$\text{Want}\quad \frac{\Gamma, x{:}\,A \vdash t : B \qquad \Gamma \vdash u : A}{\Gamma \vdash (\lambda\, x.t)\, u \cong t[u] : B[u]}$$

Note: only uses injectivity, no-confusion is used for progress: every term is a whnf or can reduce (ie there are no weird stuck things like $\pi_1\ (\lambda\, x.t)$ or $0\ u$)

Invariant for neutral inference

$$\Gamma \vdash n \approx n' \triangleright A$$

What are the invariants?

Precondition Γ is well-formed; n, n' are well-typed (but **not necessarily the same type**)

Postcondition $\Gamma \vdash n \cong n' : A$ **and** A **is the most general/unique type of** n

Invariant for neutral inference

$$\Gamma \vdash n \approx n' \triangleright A$$

What are the invariants?

Precondition Γ is well-formed; n, n' are well-typed (but **not necessarily the same type**)

Postcondition $\Gamma \vdash n \cong n' : A$ **and** A **is the most general/unique type of** n

Also needs injectivity...

Invariant for neutral inference

$$\Gamma \vdash n \approx n' \triangleright A$$

What are the invariants?

Precondition Γ is well-formed; n, n' are well-typed (but **not necessarily the same type**)

Postcondition $\Gamma \vdash n \cong n' : A$ **and** A **is the most general/unique type of** n

Also needs injectivity...

Positive soundness is already tricky

Negative soundness

Show every algorithmic step is **invertible**: the set of premises is necessary for the conclusion
If we reach a contradiction, the original problem also had no solution

Negative soundness

Show every algorithmic step is **invertible**: the set of premises is necessary for the conclusion
If we reach a contradiction, the original problem also had no solution

Won't show it again, but invariants all over the place

But now also injectivity for terms: all the congruences we apply are invertible!

Neutral injectivity

Injectivity of neutral eliminators?

If $\Gamma \vdash n \cong n' : T$ and n and n' are neutrals, then

- $n = x = n'$
- or $n = m\ u$, $n' = m'\ u'$ with $\Gamma \vdash m \cong m' : \Pi\, x{:}A.B$ and $\Gamma \vdash u \cong u' : A$
- or...

Injectivity of neutral eliminators?

If $\Gamma \vdash n \cong n' : T$ and n and n' are neutrals, then

- $n = x = n'$
- or $n = m\ u, n' = m'\ u'$ with $\Gamma \vdash m \cong m' : \Pi\, x{:}A.B$ and $\Gamma \vdash u \cong u' : A$
- or...

Does not always hold!

$$x, y{:}(\mathbb{N} \to \top) \times \top \vdash x \cong y : (\mathbb{N} \to \top) \times \top$$

Injectivity of neutral eliminators?

If $\Gamma \vdash n \cong n' : T$ and n and n' are neutrals, then

- $n = x = n'$
- or $n = m\ u, n' = m'\ u'$ with $\Gamma \vdash m \cong m' : \Pi\, x{:}A.B$ and $\Gamma \vdash u \cong u' : A$
- or...

Does not always hold!

$$x, y{:}(\mathbb{N} \to \top) \times \top \vdash x \cong y : (\mathbb{N} \to \top) \times \top$$

Neutral comparison is invertible only at certain types

Injectivity of neutral eliminators?

If $\Gamma \vdash n \cong n' : T$ and n and n' are neutrals, then

- $n = x = n'$
- or $n = m\ u, n' = m'\ u'$ with $\Gamma \vdash m \cong m' : \Pi\, x{:}A.B$ and $\Gamma \vdash u \cong u' : A$
- or...

Does not always hold!

$$x, y{:}(\mathbb{N} \to \top) \times \top \vdash x \cong y : (\mathbb{N} \to \top) \times \top$$

Neutral comparison is invertible only at certain types

Agda	Lean	Rocq
Type-directed	Term-directed	Term-directed
Short path for neutral functions	Detect unit-like types	Forbid unit-like types

	Positive soundness	Negative soundness (typed conversion)	Negative soundness (untyped conversion)	Termination
Injectivity of type constructors	×	×	×	×
Term-level injectivities		×*	×*	
Normalisation				×

* Subtlety for neutrals

	Positive soundness	Negative soundness (typed conversion)	Negative soundness (untyped conversion)	Termination
Injectivity of type constructors	×	×	×	×
Term-level injectivities		×*	×*	
Normalisation				×

Claim/conjecture: this analysis scales to realistic proof assistant kernels

How to prove the properties?

	Logical relation [AÖV17; PMT26]	**Rewriting/Confluence** [Tak95]/MetaRocq	**Gluing/Nf Model** [Ste21; BKS23]	**Domain model** [CH18]
Syntax	Raw	Raw	Intrinsic	Raw (Intrinsic?)
Weak ambiant th.	×	✓	×	✓
Normalisat°	✓	×	✓	×
Scaling	×	✓	?	?
η laws	✓	×	✓	✓
	Most explored	Insane scaling	Formalisation currently difficult	Very unexplored

Thank you!

Bibliography

[Len25] Meven Lennon-Bertrand. 'What Does It Take to Certify a Conversion Checker?' In: *10th International Conference on Formal Structures for Computation and Deduction (FSCD 2025)*. 2025. DOI: 10.4230/LIPIcs.FSCD.2025.27.

[AÖV17] Andreas Abel, Joakim Öhman and Andrea Vezzosi. 'Decidability of Conversion for Type Theory in Type Theory'. In: *Proc. ACM Program. Lang.* POPL (Dec. 2017). DOI: 10.1145/3158111.

[Adj+24] Arthur Adjedj et al. 'Martin-Löf à la Coq'. In: *Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs*. 2024. DOI: 10.1145/3636501.3636951.

[PMT26] Josselin Poiret, Kenji Maillard and Nicolas Tabareau. 'Divide and Check: Logical Relations, No Algorithms Attached'. Feb. 2026.

[Tak95] M. Takahashi. 'Parallel Reductions in λ-Calculus'. In: *Information and Computation* 1 (1995). DOI: 10.1006/inco.1995.1057.

[Soz+25] Matthieu Sozeau et al. 'Correct and Complete Type Checking and Certified Erasure for Coq, in Coq'. In: *Journal of the ACM* (Jan. 2025). DOI: 10.1145/3706056.

[Ste21] Jonathan Sterling. 'First Steps in Synthetic Tait Computability: The Objective Metatheory of Cubical Type Theory'. PhD thesis. Carnegie Mellon University, Nov. 2021. DOI: 10.5281/zenodo.6990769.

[BKS23] Rafaël Bocquet, Ambrus Kaposi and Christian Sattler. 'For the Metatheory of Type Theory, Internal Sconing Is Enough'. In: *8th International Conference on Formal Structures for Computation and Deduction, FSCD 2023*. 2023. DOI: 10.4230/LIPICS.FSCD.2023.18.

[CH18] Thierry Coquand and Simon Huber. 'An Adequacy Theorem for Dependent Type Theory'. In: *Theory of Computing Systems* 4 (July 2018). DOI: 10.1007/s00224-018-9879-9.