Towards a certified proof assistant kernel

What it takes and what we have

Meven Lennon-Bertrand

EuroProofNet WG6 Meeting – April 5th 2024

University of Cambridge
Department of Computer Science and Technology

A plea for stronger frameworks

The power of dependent type theory: **Say what we mean.**

The power of dependent type theory: **Say what we mean.**

✨ Fancy pattern-matching

✨ (Strong) records

(Computable) univalence

(Co)Inductive types

Universes

Proof irrelevance ✨

The power of dependent type theory: **Say what we mean.**

✨ Fancy pattern-matching

✨ (Strong) records

(Computable) univalence

(Co)Inductive types

Universes

Proof irrelevance ✨

Modalities

✨ Subtyping

Observational equality

Gradual typing

...

The power of dependent type theory: **Say what we mean.**

✨ Fancy pattern-matching

✨ (Strong) records

(Computable) univalence

(Co)Inductive types

Universes

Proof irrelevance ✨

Modalities

✨ Subtyping

Observational equality

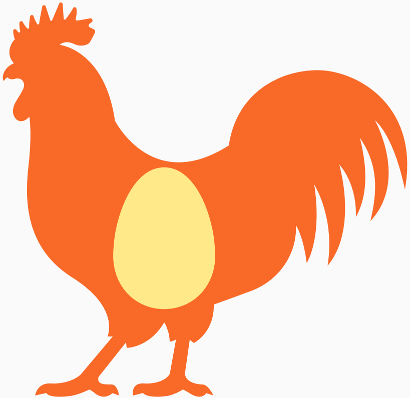
Gradual typing

...

We should **embrace** this...

… but also keep high safety guarantees.

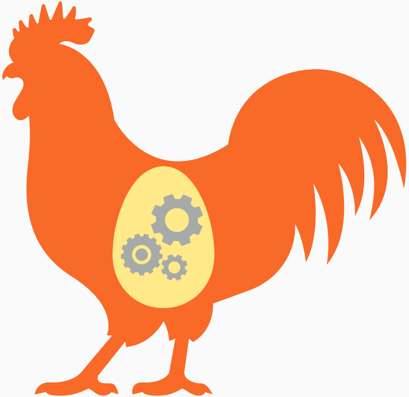
… but also keep high safety guarantees.



The de Bruijn architecture

Proof assistants should eat themselves

… but also keep high safety guarantees.



The de Bruijn architecture is a perfect target for certification!

What is so hard?

Coq's kernel is only ∼20kLoC of pure functional code. Surely it can't be that difficult?

What is so hard?

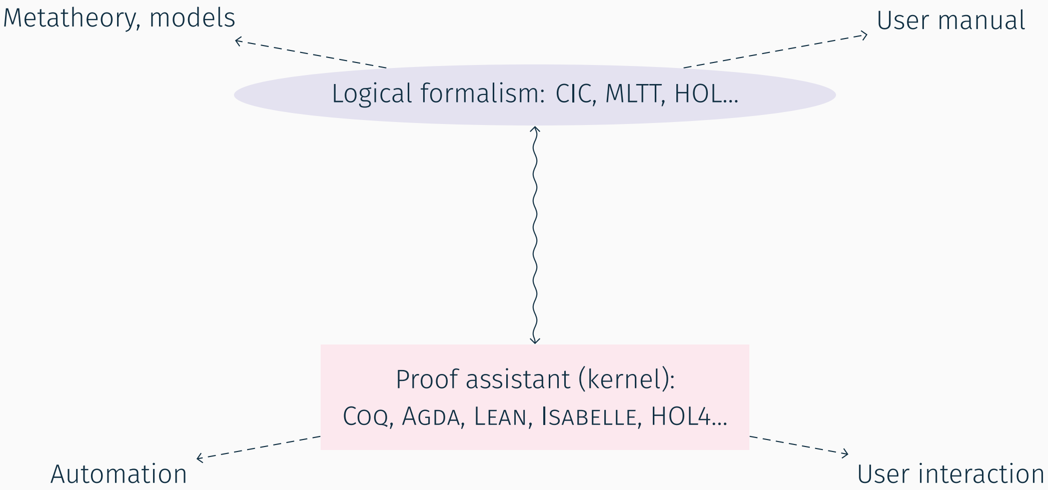
Coq's kernel is only ~20kLoC of pure functional code. Surely it can't be that difficult?

Dependent type theory + Invariants

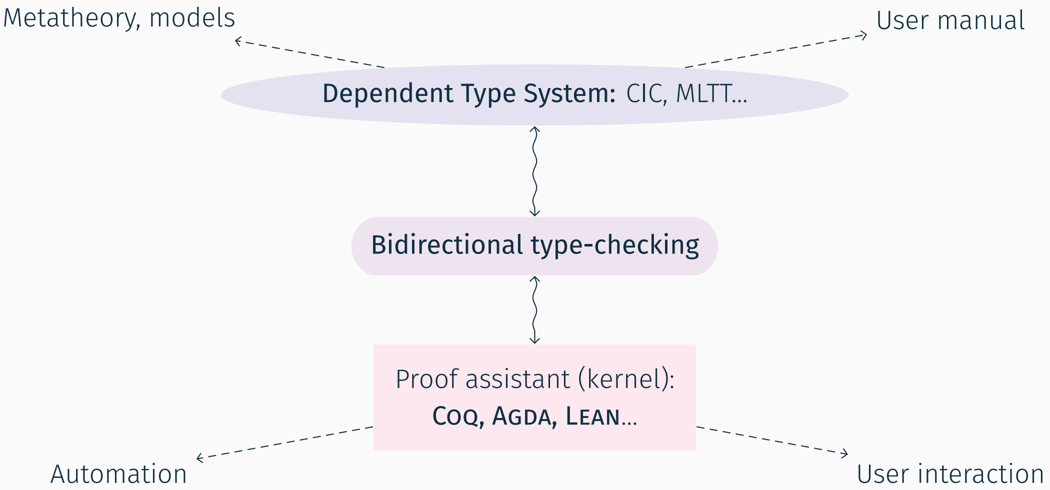
=

Bidirectional typing

Specifying Proof Assistants



Specifying Proof Assistants



A typing judgment $\Gamma \vdash t : A$ has *boundaries*. What about their well-formation?

A typing judgment $\Gamma \vdash t : A$ has *boundaries*. What about their well-formation?

Cautiousness: globally enforce well-formation

$$\frac{\vdash \Gamma \qquad (x : A) \in \Gamma}{\Gamma \vdash x : A} \qquad\qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda\, x : A.t : \prod x : A.B}$$

Boundaries and Invariants

A typing judgment $\Gamma \vdash t : A$ has *boundaries*. What about their well-formation?

Cautiousness: globally enforce well-formation

$$\frac{\vdash \Gamma \qquad (x : A) \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda\, x : A.t : \prod x : A.B}$$

Uncautiousness? Well-formation as an invariant

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x : A} \qquad \frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda\, x : A.t : \prod x : A.B}$$

Inference and checking

$\Gamma \vdash t : A$ separates into

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

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

Well-Formation 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 modes: input/subject/output.

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

- 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
- Modes guide invariant preservation

Structure!

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

Structure!

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

$$\frac{\Gamma \vdash t \rhd_{\mathrm{r}} \prod x{:}\, A.B \qquad \Gamma \vdash u \lhd A}{\Gamma \vdash t\ u \rhd B[u]}$$

- Clear information flow

Structure!

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

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

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

- Clear information flow

Structure!

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

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

- Clear information flow
- Different modes command different computation judgments ($\to^\star$ vs $\cong$)
- No free conversion thanks to the judgments' structure

Nothing has changed

Nothing has changed

Bidirectional typing is correct

Soundness: if $\vdash \Gamma$ and $\Gamma \vdash t \triangleright T$ then $\Gamma \vdash t : T$

Bidirectional typing is correct

Soundness: if $\vdash \Gamma$ and $\Gamma \vdash t \triangleright T$ then $\Gamma \vdash t : T$

Completeness*: if $\Gamma \vdash t : T$, there exists T' such that $\Gamma \vdash t \triangleright T'$ and $\Gamma \vdash T' \cong T$

*T&C apply

Bidirectional typing is correct

Soundness: if $\vdash \Gamma$ and $\Gamma \vdash t \triangleright T$ then $\Gamma \vdash t : T$

Completeness*: if $\Gamma \vdash t : T$, there exists T' such that $\Gamma \vdash t \triangleright T'$ and $\Gamma \vdash T' \cong T$

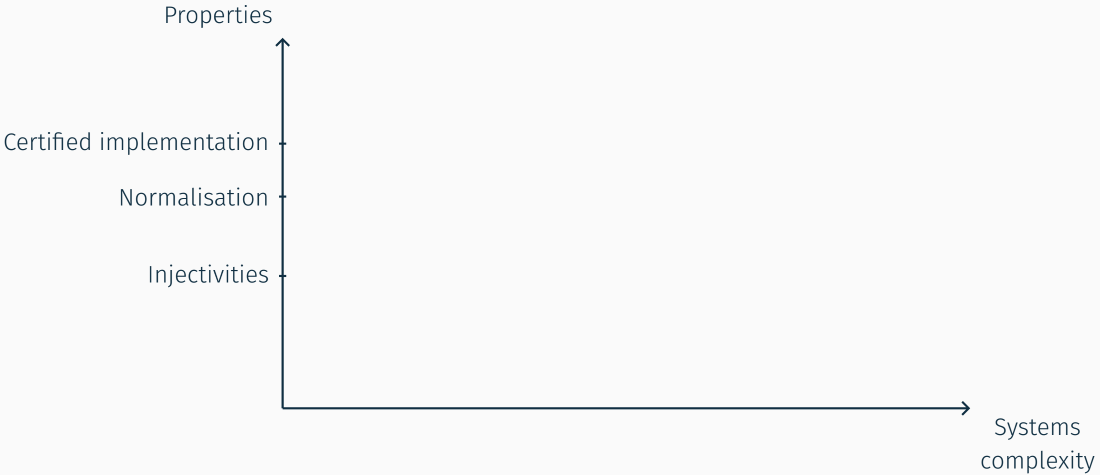
Key properties:

- injectivity: if $\Gamma \vdash \prod x : A.\ B \cong \prod x : A'.\ B'$, then $\Gamma \vdash A \cong A'$ and $\Gamma, x : A \vdash B \cong B'$
- reduction finds constructors: if $\Gamma \vdash T \cong \prod x : A.\ B$ then $\Gamma \vdash T \to^\star \prod x : A'.\ B'$

*T&C apply

Roadmap

Roadmap



Properties
Certified implementation
Normalisation
Injectivities
Systems
complexity

Normalisation

- every reduction path $t_0 \to t_1 \to t_2 \to \ldots$ is finite
- there is exactly one normal form $\bar{t} \in \mathrm{Nf}$ in each equivalence class for $\cong$

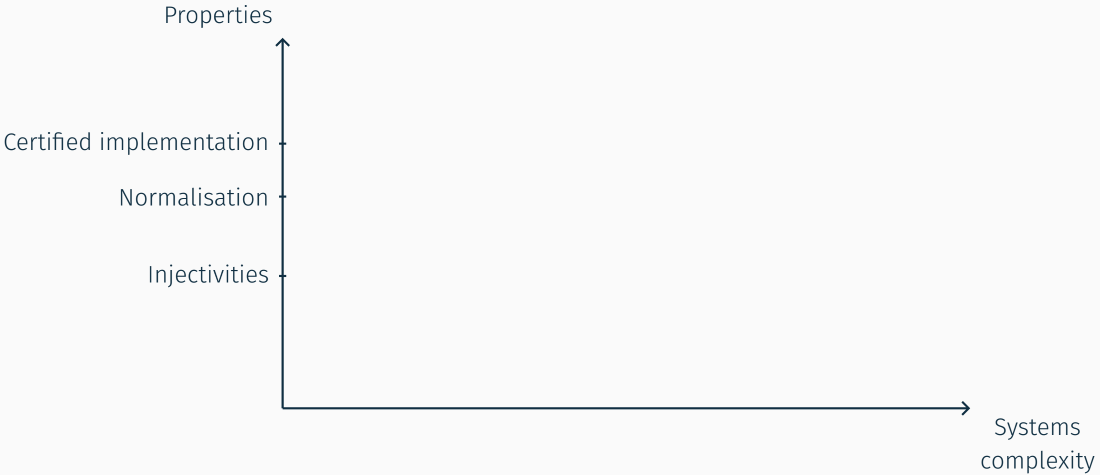
Normalisation

- every reduction path $t_0 \to t_1 \to t_2 \to \dots$ is finite
- there is exactly one normal form $\bar{t} \in \mathrm{Nf}$ in each equivalence class for $\cong$

The mother of all properties:

- decidability of conversion
- canonicity
- consistency ⚠

Roadmap



Properties
Certified implementation
Normalisation
Injectivities
Systems complexity

Coq in Coq (Barras et al. 1997): certified type-checker for the CoC, in Coq.

Proof-theoretic strength VS expressivity

Coq in Coq (Barras et al. 1997): certified type-checker for the CoC, in Coq.

CoC is proof-theoretically stronger than Agda, close to Coq. Time to change subject?

Proof-theoretic strength VS expressivity

Coq in Coq (Barras et al. 1997): certified type-checker for the CoC, in Coq.

CoC is proof-theoretically stronger than Agda, close to Coq. Time to change subject?

Proof-theoretic strength is not the same as expressivity!

Proof-theoretic strength VS expressivity

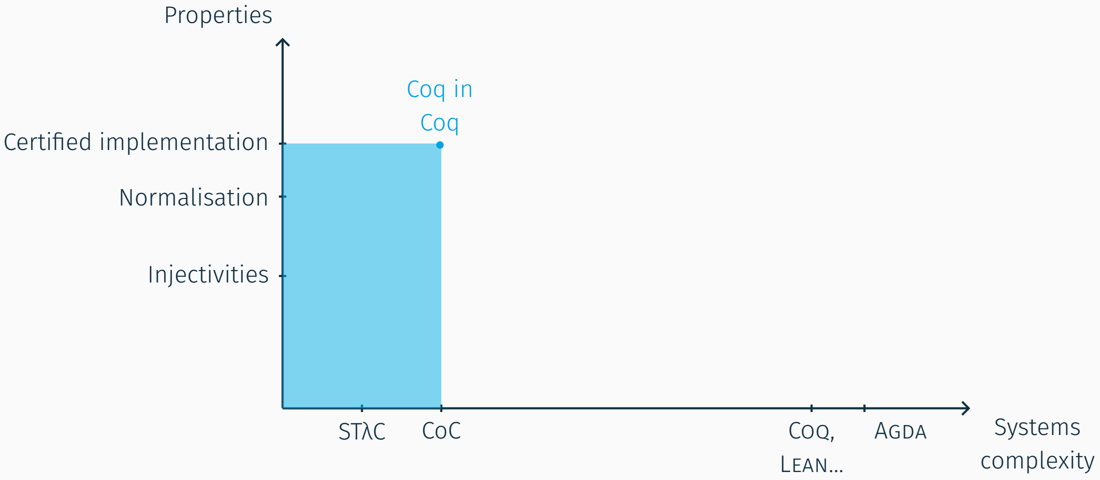
Coq in Coq (Barras et al. 1997): certified type-checker for the CoC, in Coq.

CoC is proof-theoretically stronger than Agda, close to Coq. Time to change subject?

Proof-theoretic strength is not the same as expressivity!

Turing-completeness vs "real" language.

Roadmap



Properties
Coq in Coq
Certified implementation
Normalisation
Injectivities
STλC
CoC
Coq, Lean...
Agda
Systems complexity

Gödel's 2^{nd} incompleteness theorem

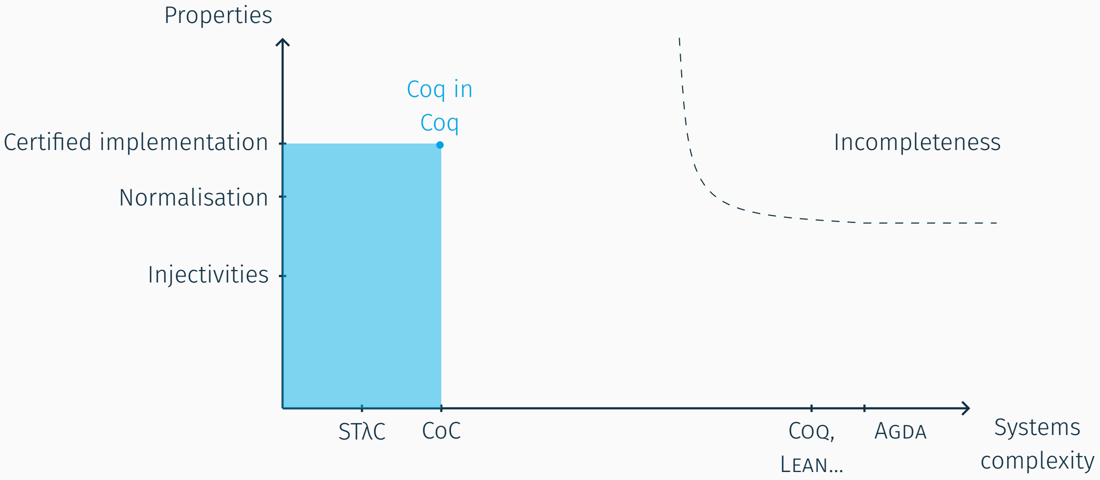
Coq in Coq?

Coq in Coq?

~~COQ in COQ?~~

An object type theory $\mathcal{T}$ in a (slightly) stronger meta type theory $\mathcal{T}'$.

Roadmap


Properties
Coq in
Coq
Certified implementation
Normalisation
Injectivities
Incompleteness
STλC
CoC
Coq,
Lean...
Agda
Systems
complexity

The MetaCoq project

Jww. M. Sozeau, Y. Forster, J. Botch Nielsen,

N. Tabareau, T. Winterhalter...

The Predicative Calculus of Universe-Polymorphic Inductive Constructions (PCUIC)

A dependent type theory with

- Very general (co-)inductive types
- Pattern-matching and fixed-points
- Complex universes + cumulativity
- …

```
Inductive term : Type :=
  | tRel (n : nat)
  | tVar (id : ident)
  | tEvar (ev : nat) (args : list term)
  | tSort (s : sort)
  | tCast (t : term) (kind : cast_kind) (v : term)
  | tProd (na : aname) (ty : term) (body : term)
  | tLambda (na : aname) (ty : term) (body : term)
  | tLetIn (na : aname) (def : term) (def_ty : term) (body : term)
  | tApp (f : term) (args : list term)
  | tConst (c : kername) (u : Instance.t)
  | tInd (ind : inductive) (u : Instance.t)
  | tConstruct (ind : inductive) (idx : nat) (u : Instance.t)
  | tCase (ci : case_info) (type_info : predicate term)
      (discr : term) (branches : list (branch term))
  | tProj (proj : projection) (t : term)
  | tFix (mfix : mfixpoint term) (idx : nat)
  | tCoFix (mfix : mfixpoint term) (idx : nat)
  | tInt (i : PrimInt63.int)
  | tFloat (f : PrimFloat.float)
  | tArray (u : Level.t) (arr : list term) (default : term) (type : term).
```

The Predicative Calculus of Universe-Polymorphic Inductive Constructions (PCUIC)

A dependent type theory with

- Very general (co-)inductive types
- Pattern-matching and fixed-points
- Complex universes + cumulativity
- ...

Coq, in Coq (bis)

- Formalized meta-theory of PCUIC
- Normalization axiom to implement a certified type-checker ($\mathcal{T}' = \mathcal{T} + \mathrm{Norm}(\mathcal{T})$)

MetaCoq in a nutshell

The Predicative Calculus of Universe-Polymorphic Inductive Constructions (PCUIC)

A dependent type theory with

- Very general (co-)inductive types
- Pattern-matching and fixed-points
- Complex universes + cumulativity
- ...

Coq, in Coq (bis)

- Formalized meta-theory of PCUIC
- Normalization axiom to implement a certified type-checker ($\mathcal{T}' = \mathcal{T} + \mathrm{Norm}(\mathcal{T})$)
- Certified extraction, meta-programming...

MetaCoq in a nutshell

The Predicative Calculus of Universe-Polymorphic Inductive Constructions (PCUIC)

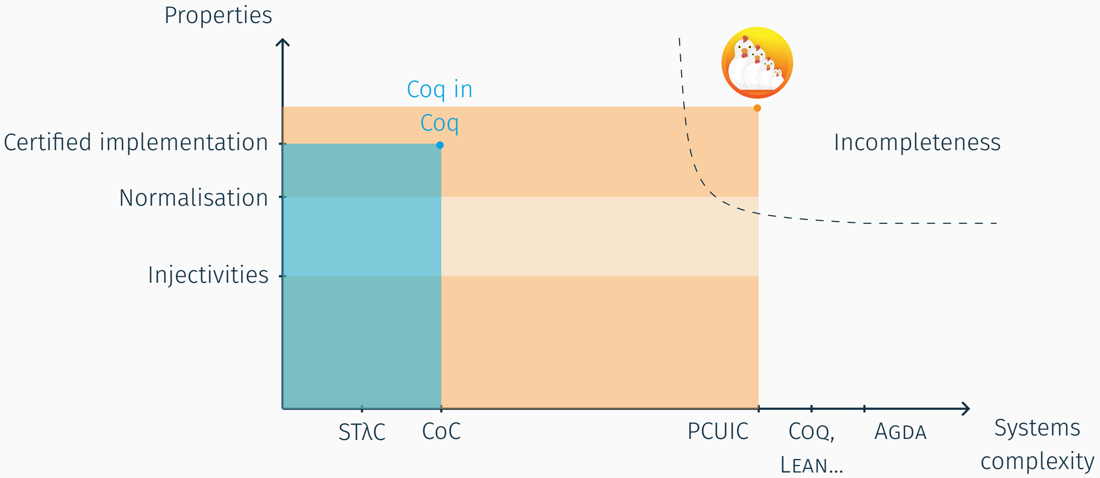
A dependent type theory with

- Very general (co-)inductive types
- Pattern-matching and fixed-points
- Complex universes + cumulativity
- …

Coq, in Coq (bis)

- Formalized meta-theory of PCUIC
- Normalization axiom to implement a certified type-checker ($\mathcal{T}$ … $\mathrm{n}(\mathcal{T})$)
- Certified extraction, meta-programming…

Roadmap


Properties
Coq in Coq
Certified implementation
Normalisation
Injectivities
Incompleteness
STλC
CoC
PCUIC
Coq, Lean...
Agda
Systems complexity

Meta-theory of PCUIC

- Substitution lemmas (terms, universes)
- **Confluence** (parallel reduction *à la* Tait-Martin-Löf, following Takahashi '95)
- Injectivities & reduction finds constructors
- Preservation & progress
- Bidirectional typing

Meta-theory of PCUIC

- Substitution lemmas (terms, universes)
- **Confluence** (parallel reduction *à la* Tait-Martin-Löf, following Takahashi ’95)
- Injectivities & reduction finds constructors
- Preservation & progress
- Bidirectional typing

Main challenge = **scaling** standard techniques

- Substitution lemmas (terms, universes)
- Confluence (parallel reduction *à la* Tait-Martin-Löf, following Takahashi ’95)
- Injectivities & reduction finds constructors
- Preservation & progress
- Bidirectional typing

Main challenge = scaling standard techniques

Works because cumulativity is untyped and purely computational:

$$\Gamma \vdash T \preceq U \quad \Leftrightarrow \quad \begin{array}{ccc} T & & U \\ \downarrow^{\star} & & \downarrow^{\star} \\ T' & \leq_{\mathcal{U}} & U' \end{array}$$

Meta-theory of PCUIC

- Substitut
- Confluen
- Injectiviti
- Preservat
- Bidirectio

Main challeng

Works becaus

'95)

1

Correct and Complete Type Checking and Certified Erasure for Coq, in Coq

MATTHIEU SOZEAU, Inria, France
YANNICK FORSTER, Inria, France
MEVEN LENNON-BERTRAND, University of Cambridge, United Kingdom
JAKOB BOTSCH NIELSEN, Concordium Blockchain Research Center, Denmark
NICOLAS TABAREAU, Inria, France
THÉO WINTERHALTER, Inria, France

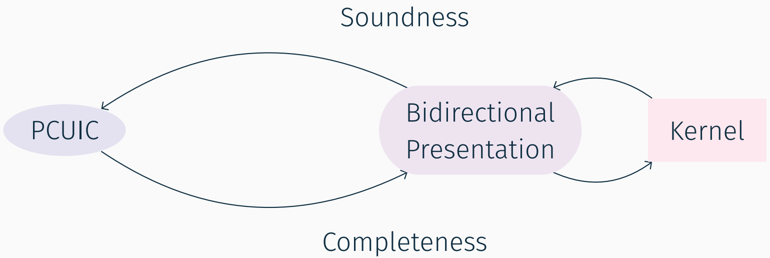
Coq is built around a well-delimited kernel that performs type checking for definitions in a variant of the Calculus of Inductive Constructions (CIC). Although the metatheory of CIC is very stable and reliable, the correctness of its implementation in Coq is less clear. Indeed, implementing an efficient type checker for CIC is a rather complex task, and many parts of the code rely on implicit invariants which can easily be broken by further evolution of the code. Therefore, on average, one critical bug has been found every year in Coq. This paper presents the first implementation of a type checker for the kernel of Coq (without the module system, template polymorphism and η-conversion), which is proven sound and complete in Coq with respect to its formal specification. Note that because of Gödel's second incompleteness theorem, there is no hope to prove completely the soundness of the specification of Coq inside Coq (in particular strong normalization), but it is possible to prove the correctness and completeness of the implementation assuming soundness of the specification, thus moving from a trusted code base (TCB) to a trusted theory base (TTB) paradigm. Our work is based on the MetaCoq project which provides meta-programming facilities to work with terms and declarations at the level of the kernel. We verify a relatively efficient type checker based on the specification of the typing relation of the Polymorphic, Cumulative Calculus of Inductive Constructions (PCUIC) at the basis of Coq. It is worth mentioning that during the verification process, we have found a source of incompleteness in Coq's official type checker, which has then been fixed in Coq 8.14 thanks to our work. In addition to the kernel implementation, another essential feature of Coq is the so-called *extraction* mechanism: the production of executable code in functional languages from Coq definitions. We present a verified version of this subtle type and proof erasure step, therefore enabling the verified extraction of a safe type checker for Coq in the future.

CCS Concepts: • **Theory of computation → Type theory**.

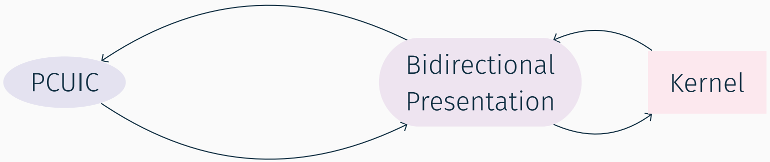
A Correct and Complete Kernel

Soundness

A Correct and Complete Kernel

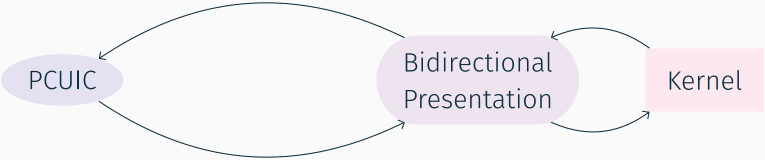


A Correct and Complete Kernel

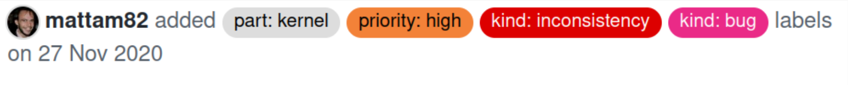


Deep in the proof, we realized... it was false!

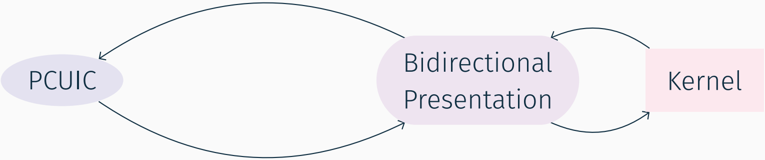
A Correct and Complete Kernel



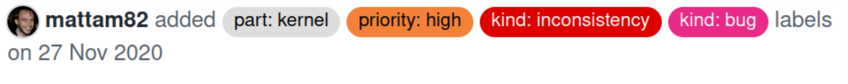
Deep in the proof, we realized... it was false!



A Correct and Complete Kernel



Deep in the proof, we realized... it was false!



→ re-design of pattern-matching in Coq, backed by MetaCoq.

And now?

We have a fully certified, extracted kernel!

And now?

We have a fully certified, extracted kernel!

But:

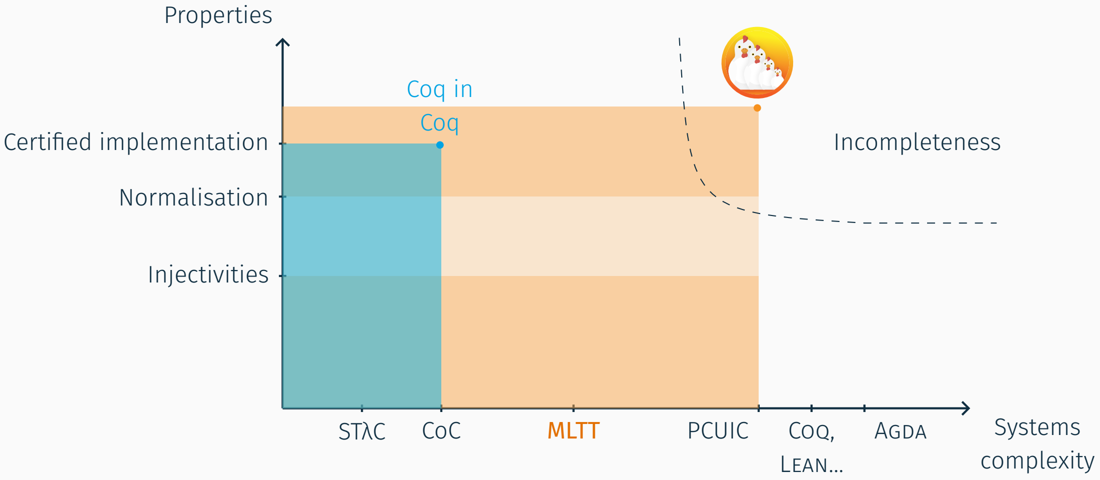
- no normalisation;
- untyped conversion (not what semanticists like);
- no extensionality equations (η-laws!).

Martin-Löf *à la* Coq

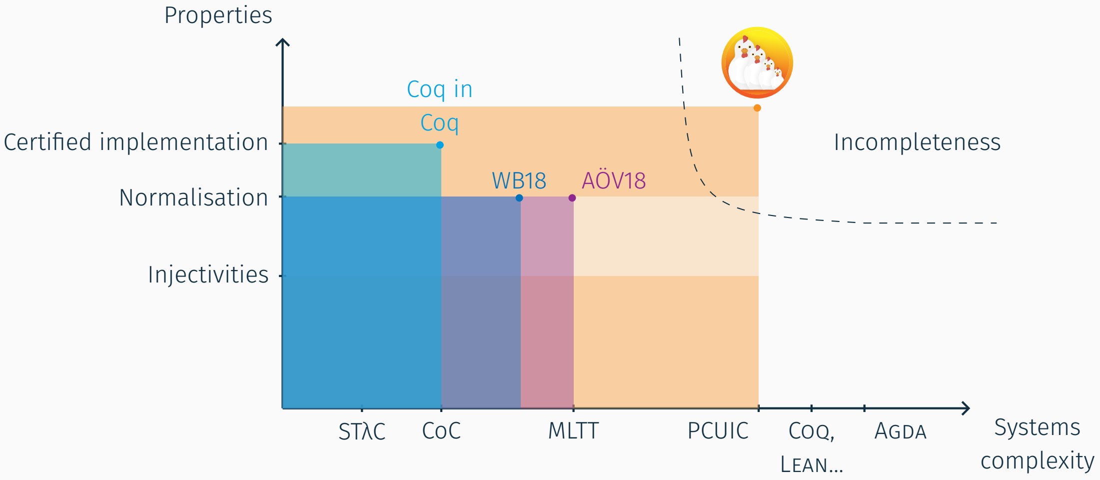
Jww. Arthur Adjedj, Kenji Maillard,

Pierre-Marie Pédrot and Loïc Pujet

Roadmap

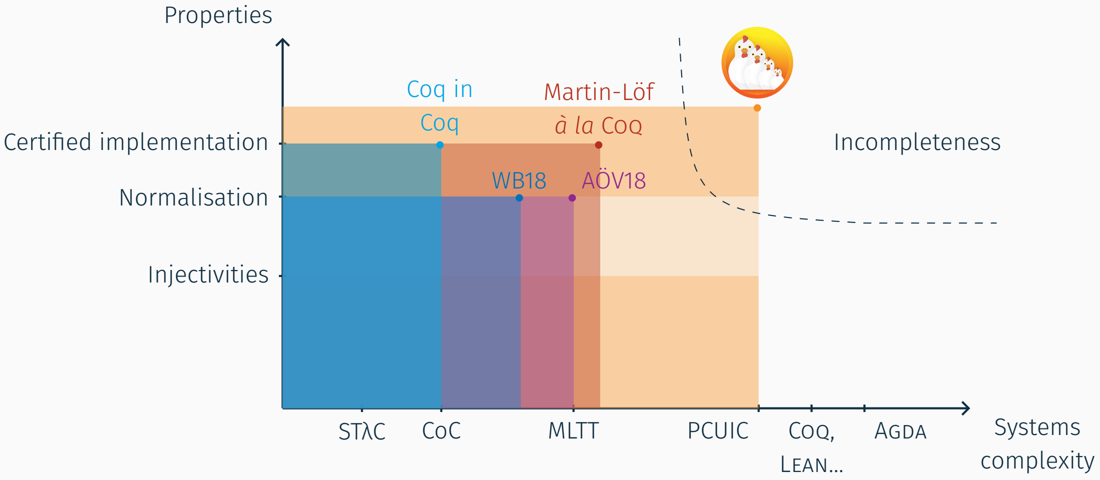

Properties
Coq in Coq
Certified implementation
Normalisation
Injectivities
Incompleteness
STλC
CoC
MLTT
PCUIC
Coq, Lean...
Agda
Systems complexity

Roadmap



Properties
Coq in Coq
Certified implementation
Normalisation
Injectivities
WB18
AÖV18
Incompleteness
STλC
CoC
MLTT
PCUIC
Coq, Lean...
Agda
Systems complexity

Roadmap


Properties
Certified implementation
Normalisation
Injectivities
Coq in Coq
Martin-Löf à la Coq
WB18
AÖV18
Incompleteness
STλC
CoC
MLTT
PCUIC
Coq, Lean...
Agda
Systems complexity

Typed conversion

$$\textsc{Refl}\ \frac{\Gamma \vdash t : A}{\Gamma \vdash t \cong t : A} \qquad \textsc{Sym}\ \frac{\Gamma \vdash t \cong u : A}{\Gamma \vdash u \cong t : A} \qquad \textsc{Trans}\ \frac{\Gamma \vdash t \cong u : A \qquad \Gamma \vdash u \cong v : A}{\Gamma \vdash t \cong v : A}$$

Typed conversion

$$\textsc{Refl}\ \frac{\Gamma \vdash t : A}{\Gamma \vdash t \cong t : A} \qquad \textsc{Sym}\ \frac{\Gamma \vdash t \cong u : A}{\Gamma \vdash u \cong t : A} \qquad \textsc{Trans}\ \frac{\Gamma \vdash t \cong u : A \qquad \Gamma \vdash u \cong v : A}{\Gamma \vdash t \cong v : A}$$

$$\textsc{AppCong}\ \frac{\Gamma \vdash t \cong t' : \prod x\colon A.B \qquad \Gamma \vdash u \cong u' : A}{\Gamma \vdash t\ u \cong t'\ u' : B[u]}$$

$$\beta\textsc{Fun}\ \frac{\begin{array}{c}\Gamma \vdash A \qquad \Gamma, x\colon A \vdash B \\ \Gamma, x\colon A \vdash t : B \qquad \Gamma \vdash u : A\end{array}}{\Gamma \vdash (\lambda\, x\colon A.t)\, u \cong t[u] : B[u]} \qquad \eta\textsc{Fun}\ \frac{\Gamma \vdash f : \prod x\colon A.B}{\Gamma \vdash f \cong \lambda x\colon A. f\ x : \prod x\colon A.B} \qquad \ldots$$

Typed conversion

$$\textsc{Refl}\ \frac{\Gamma \vdash t : A}{\Gamma \vdash t \cong t : A} \qquad \textsc{Sym}\ \frac{\Gamma \vdash t \cong u : A}{\Gamma \vdash u \cong t : A} \qquad \textsc{Trans}\ \frac{\Gamma \vdash t \cong u : A \qquad \Gamma \vdash u \cong v : A}{\Gamma \vdash t \cong v : A}$$

$$\textsc{AppCong}\ \frac{\Gamma \vdash t \cong t' : \Pi\, x{:}\, A.B \qquad \Gamma \vdash u \cong u' : A}{\Gamma \vdash t\ u \cong t'\ u' : B[u]}$$

$$\beta\textsc{Fun}\ \frac{\begin{array}{c}\Gamma \vdash A \qquad \Gamma, x{:}\, A \vdash B \\ \Gamma, x{:}\, A \vdash t : B \qquad \Gamma \vdash u : A\end{array}}{\Gamma \vdash (\lambda\, x{:}\, A.t)\, u \cong t[u] : B[u]} \qquad \eta\textsc{Fun}\ \frac{\Gamma \vdash f : \Pi\, x{:}\, A.B}{\Gamma \vdash f \cong \lambda x{:}\, A. f\ x : \Pi\, x{:}\, A.B} \qquad \dots$$

It's bidirectional too!

Conversion ≅ checks

$$\frac{\Gamma \vdash t \to^\star t' : A \qquad \Gamma \vdash u \to^\star u' : A \qquad \Gamma \vdash A \to^\star A' \qquad \Gamma \vdash t' \cong_{\mathrm{h}} u' \triangleleft A'}{\Gamma \vdash t \cong u \triangleleft A}$$

$$\frac{\Gamma, x\colon A \vdash f\ x \cong g\ x \triangleleft B}{\Gamma \vdash f \cong_{\mathrm{h}} g \triangleleft \Pi\, x\colon A.\, B} \qquad \frac{\Gamma \vdash t \cong t' \triangleleft \mathbf{N}}{\Gamma \vdash \mathrm{S}(t) \cong_{\mathrm{h}} \mathrm{S}(t') \triangleleft \mathbf{N}} \qquad \frac{\Gamma \vdash n \approx n' \triangleright T}{\Gamma \vdash n \cong_{\mathrm{h}} n' \triangleleft \mathbf{N}}$$

Conversion ≅ checks

$$\frac{\Gamma \vdash t \to^\star t' : A \qquad \Gamma \vdash u \to^\star u' : A \qquad \Gamma \vdash A \to^\star A' \qquad \Gamma \vdash t' \cong_{\mathrm{h}} u' \lhd A'}{\Gamma \vdash t \cong u \lhd A}$$

$$\frac{\Gamma, x\colon A \vdash f\ x \cong g\ x \lhd B}{\Gamma \vdash f \cong_{\mathrm{h}} g \lhd \Pi\, x\colon A.\ B} \qquad \frac{\Gamma \vdash t \cong t' \lhd \mathbf{N}}{\Gamma \vdash \mathrm{S}(t) \cong_{\mathrm{h}} \mathrm{S}(t') \lhd \mathbf{N}} \qquad \frac{\Gamma \vdash n \approx n' \rhd T}{\Gamma \vdash n \cong_{\mathrm{h}} n' \lhd \mathbf{N}}$$

Neutral comparison ≈ infers

$$\frac{\Gamma \vdash m \approx n \rhd_{\mathrm{r}} \Pi\, x\colon A.\ B \qquad \Gamma \vdash t \cong u \lhd A}{\Gamma \vdash m\ t \approx n\ u \rhd B[t]} \qquad \frac{(x\colon A) \in \Gamma}{\Gamma \vdash x \approx x \rhd A}$$

Correctness

Soundness

Injectivity to preserve invariants.

Correctness

Soundness

Injectivity to preserve invariants.

Completeness

Symmetry, transitivity, conversion: tricky but doable…

Reflexivity: $\Gamma \vdash t : A \Rightarrow \Gamma \vdash t \cong t : A \Rightarrow \Gamma \vdash t \cong t \lhd A$ is basically normalisation!

Correctness

Soundness

Injectivity to preserve invariants.

Completeness

Symmetry, transitivity, conversion: tricky but doable…

Reflexivity: $\Gamma \vdash t : A \Rightarrow \Gamma \vdash t \cong t : A \Rightarrow \Gamma \vdash t \cong t \triangleleft A$ is basically normalisation!

One word: logical relations.

Soundness

Injecti...

Compl...

Symm...

Reflex... ...normalisation!

One wo...

Decidability of Conversion for Type Theory in Type Theory

ANDREAS ABEL, Gothenburg University, Sweden
JOAKIM ÖHMAN, IMDEA Software Institute, Spain
ANDREA VEZZOSI, Chalmers University of Technology, Sweden

Type theory should be able to handle its own meta-theory, both to justify its foundational claims and to obtain a verified implementation. At the core of a type checker for intensional type theory lies an algorithm to check equality of types, or in other words, to check whether two types are convertible. We have formalized in Agda a practical conversion checking algorithm for a dependent type theory with one universe à la Russell, natural numbers, and η-equality for Π types. We prove the algorithm correct via a Kripke logical relation parameterized by a suitable notion of equivalence of terms. We then instantiate the parameterized fundamental lemma twice: once to obtain canonicity and injectivity of type formers, and once again to prove the completeness of the algorithm. Our proof relies on inductive-recursive definitions, but not on the uniqueness of identity proofs. Thus, it is valid in variants of intensional Martin-Löf Type Theory as long as they support induction-recursion, for instance, Extensional, Observational, or Homotopy Type Theory.

CCS Concepts: • **Theory of computation** → **Type theory**; *Proof theory*;

Additional Key Words and Phrases: Dependent types, Logical relations, Formalization, Agda

ACM Reference Format:

Correctness

Soundness

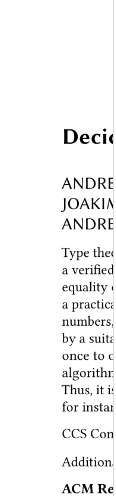
Injecti

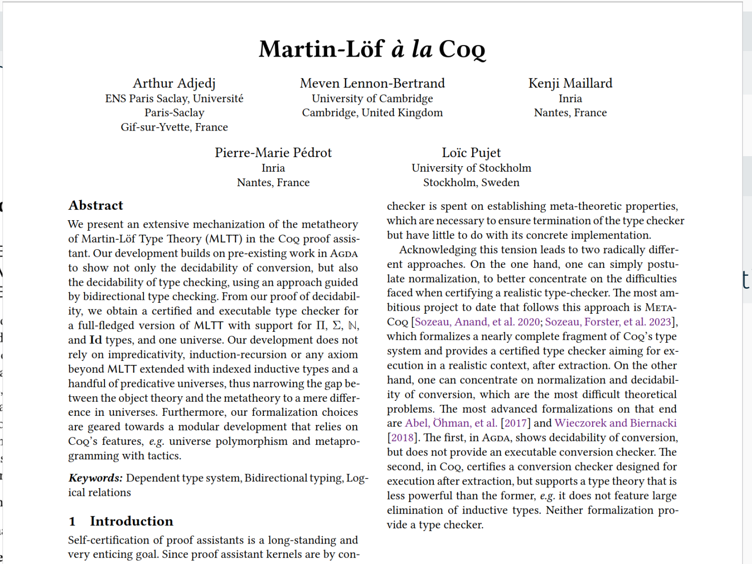
Compl

Symm

Reflex … tion!

One wo





Martin-Löf *à la* Coq

Arthur Adjedj
ENS Paris Saclay, Université Paris-Saclay
Gif-sur-Yvette, France

Meven Lennon-Bertrand
University of Cambridge
Cambridge, United Kingdom

Kenji Maillard
Inria
Nantes, France

Pierre-Marie Pédrot
Inria
Nantes, France

Loïc Pujet
University of Stockholm
Stockholm, Sweden

Abstract

We present an extensive mechanization of the metatheory of Martin-Löf Type Theory (MLTT) in the Coq proof assistant. Our development builds on pre-existing work in Agda to show not only the decidability of conversion, but also the decidability of type checking, using an approach guided by bidirectional type checking. From our proof of decidability, we obtain a certified and executable type checker for a full-fledged version of MLTT with support for Π, Σ, $\mathbb{N}$, and **Id** types, and one universe. Our development does not rely on impredicativity, induction-recursion or any axiom beyond MLTT extended with indexed inductive types and a handful of predicative universes, thus narrowing the gap between the object theory and the metatheory to a mere difference in universes. Furthermore, our formalization choices are geared towards a modular development that relies on Coq's features, *e.g.* universe polymorphism and metaprogramming with tactics.

Keywords: Dependent type system, Bidirectional typing, Logical relations

1 Introduction

Self-certification of proof assistants is a long-standing and very enticing goal. Since proof assistant kernels are by con-

checker is spent on establishing meta-theoretic properties, which are necessary to ensure termination of the type checker but have little to do with its concrete implementation.

Acknowledging this tension leads to two radically different approaches. On the one hand, one can simply postulate normalization, to better concentrate on the difficulties faced when certifying a realistic type-checker. The most ambitious project to date that follows this approach is MetaCoq [Sozeau, Anand, et al. 2020; Sozeau, Forster, et al. 2023], which formalizes a nearly complete fragment of Coq's type system and provides a certified type checker aiming for execution in a realistic context, after extraction. On the other hand, one can concentrate on normalization and decidability of conversion, which are the most difficult theoretical problems. The most advanced formalizations on that end are Abel, Öhman, et al. [2017] and Wieczorek and Biernacki [2018]. The first, in Agda, shows decidability of conversion, but does not provide an executable conversion checker. The second, in Coq, certifies a conversion checker designed for execution after extraction, but supports a type theory that is less powerful than the former, *e.g.* it does not feature large elimination of inductive types. Neither formalization provide a type checker.

Soundness

Injecti...

Compl...

Symm...

Reflex... ...tion!

One wo...

D...

Al...
JC...
Al...

Ty... a v... eq... a p... nu... by... on... alg... Th... fo...

CC...

Ac...

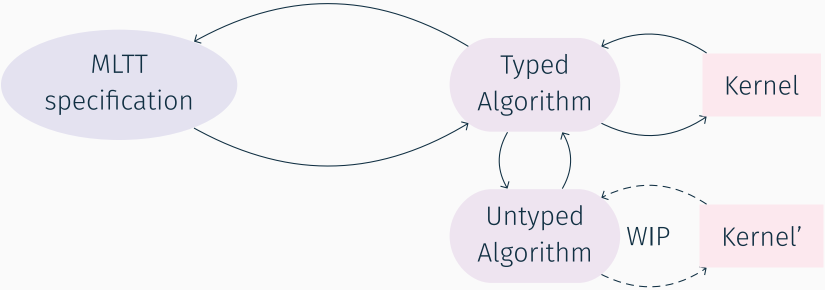
ACM Re...

Self-certification of proof assistants is a long-standing and very enticing goal. Since proof assistant kernels are by con-

Back to untyped conversion

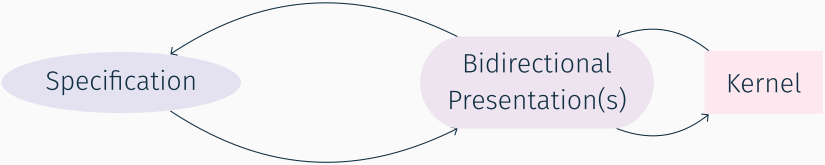
But Coq's cumulativity check is untyped?

But Coq's cumulativity check is untyped?

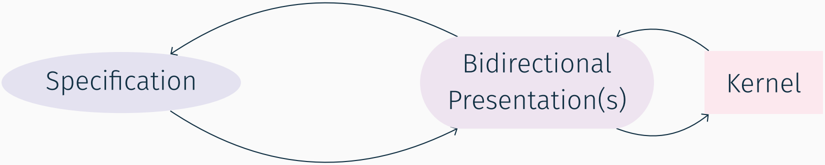


Wrapping up

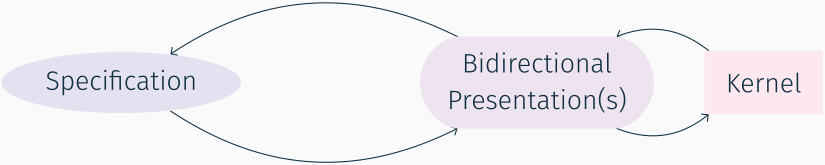
Wrapping up



Wrapping up



- MetaCoq: focus on **gory** issues of a **real** system
- MLTT *à la* Coq: go **as far as possible** in an **axiom-free** way

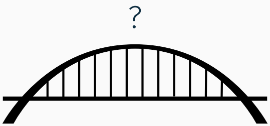


- MetaCoq: focus on **gory** issues of a **real** system
- MLTT *à la* Coq: go **as far as possible** in an **axiom-free** way

What now?

MetaCoq

Typed conversion?
Injectivity with η-laws?
All of Coq? And more?



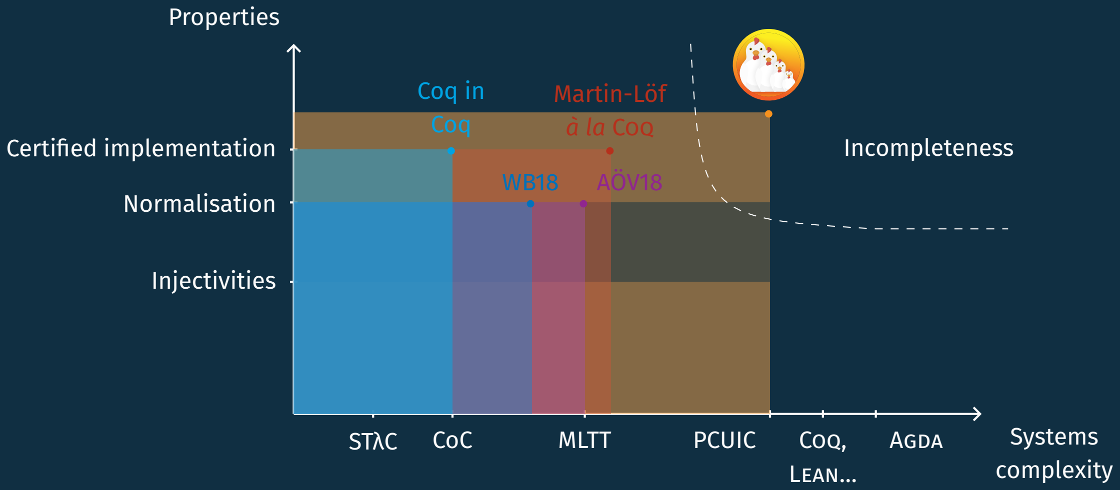
MLTT *à la* Coq

How far can we scale?
What practical/theoretical tools do we need?

A question for the audience

Take MLTT with typed conversion, Π with β and η, and Type : Type.

Can you show Π types are injective?


Properties
Coq in Coq
Martin-Löf à la Coq
Certified implementation
Normalisation
Injectivities
WB18
AÖV18
Incompleteness
STλC
CoC
MLTT
PCUIC
Coq, Lean…
Agda
Systems complexity

THANK YOU!

Bibliography

[AÖV18] Andreas Abel, Joakim Öhman, and Andrea Vezzosi. **"Decidability of Conversion for Type Theory in Type Theory".** In: *Proc. ACM Program. Lang.* (Jan. 2018). DOI: 10.1145/3158111.

[WB18] Paweł Wieczorek and Dariusz Biernacki. **"A Coq Formalization of Normalization by Evaluation for Martin-Löf Type Theory".** In: *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*. CPP 2018. Los Angeles, CA, USA: Association for Computing Machinery, 2018, pp. 266–279. ISBN: 9781450355865. DOI: 10.1145/3167091.

[BW97] Bruno Barras and Benjamin Werner. **"Coq in Coq".** 1997. URL: http://www.lix.polytechnique.fr/Labo/Bruno.Barras/publi/coqincoq.pdf.

[Soz+23] Matthieu Sozeau et al. **"Correct and Complete Type Checking and Certified Erasure for Coq, in Coq".** Preprint. Apr. 2023. URL: https://inria.hal.science/hal-04077552.

[Adj+24] Arthur Adjedj et al. **"Martin-Löf *à la* Coq".** In: *Certified Programs and Proofs* (2024). URL: https://inria.hal.science/hal-04214008.