Towards a certified proof assistant kernel

What it takes and what we have

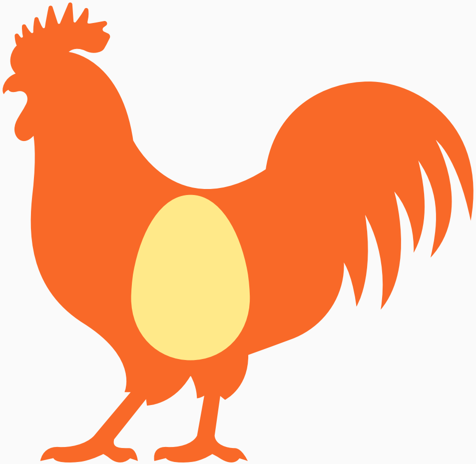
Meven Lennon-Bertrand

Deducteam Seminar – December 14th 2023

UNIVERSITY OF CAMBRIDGE
Department of Computer Science and Technology

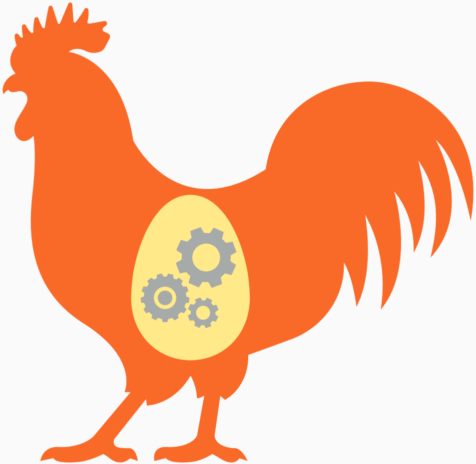
How Theorem Proving Should Feel Like

How Theorem Proving Too Often Feels Like



The de Bruijn architecture

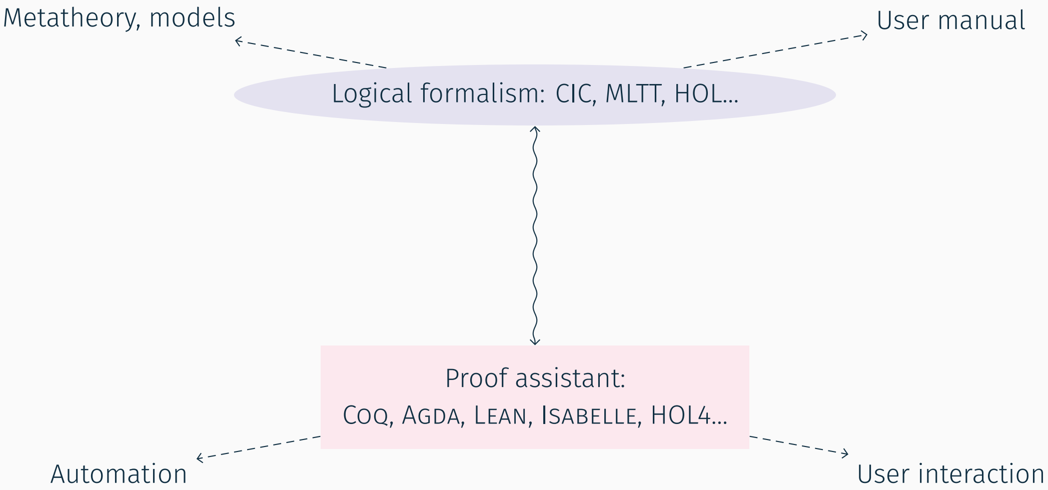
Trusting proof assistants

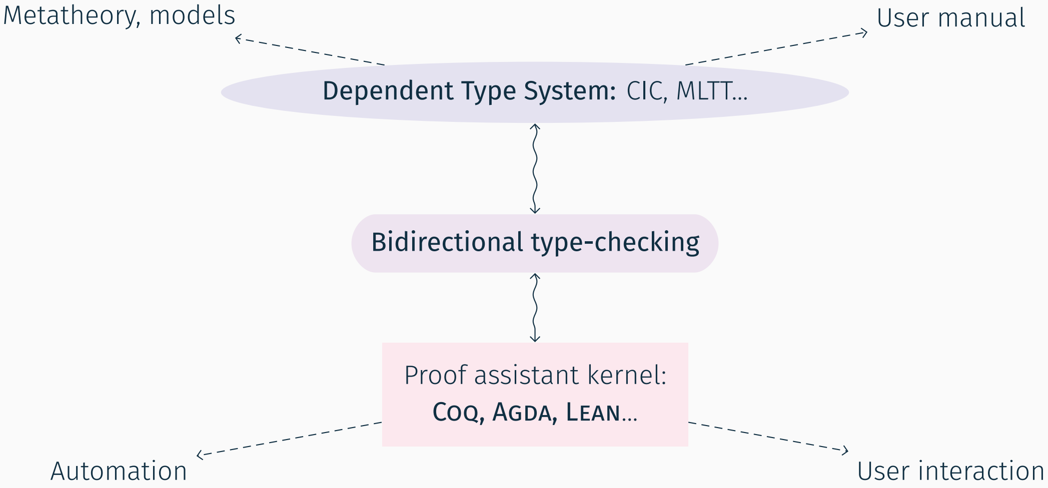


The de Bruijn architecture: a perfect target for certification!

Bidirectional typing

Specifying Proof Assistants




Metatheory, models
User manual
Dependent Type System: CIC, MLTT...
Bidirectional type-checking
Proof assistant kernel:
Coq, Agda, Lean...
Automation
User interaction

Boundaries and Invariants

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

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

Cautiousness: globally enforce well-formation

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

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

Cautiousness: globally enforce well-formation

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

Uncautiousness? Well-formation as an invariant

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

Well-Formation Must Flow

Inference and checking

$\Gamma \vdash t : T$ separates into

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

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

Well-Formation Must Flow

Inference and checking

$\Gamma \vdash t : T$ separates into

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

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

McBride says:

- *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

$$\frac{\vdash \Gamma \qquad (x : T \in \Gamma)}{\Gamma \vdash x : T} \qquad \frac{\vdash \Gamma}{\Gamma \vdash \square_i : \square_{i+1}} \qquad \frac{\Gamma \vdash A : \square_i \qquad \Gamma, x : A \vdash B : \square_j}{\Gamma \vdash \prod x : A.B : \square_{i \vee j}}$$

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

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

$\Gamma \vdash t \triangleright T$ type inference

$\Gamma \vdash t \triangleleft T$ type checking

$$\frac{\vdash \Gamma}{\Gamma \vdash \square_i : \square_{i+1}}$$

$$\frac{(x : T \in \Gamma)}{\Gamma \vdash x \triangleright T}$$

$$\boxed{\frac{}{\Gamma \vdash \square_i \triangleright \square_{i+1}}}$$

$$\frac{\Gamma \vdash A : \square_i \qquad \Gamma, x : A \vdash B : \square_j}{\Gamma \vdash \prod x : A.B : \square_{i \vee j}}$$

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

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

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

Turning dependent (CCω)

$\Gamma \vdash t \triangleright T$ type inference
$\Gamma \vdash t \triangleleft T$ type checking
$\Gamma \vdash t \triangleright_{\bullet} T$ constrained inference

$$\frac{\Gamma \vdash A : \square_i \qquad \Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda\, x : A. t : \Pi\, x : A.B}$$

$$\frac{(x : T \in \Gamma)}{\Gamma \vdash x \triangleright T} \qquad \frac{}{\Gamma \vdash \square_i \triangleright \square_{i+1}} \qquad \frac{\Gamma \vdash A \triangleright_{\square} \square_i \qquad \Gamma, x : A \vdash B \triangleright_{\square} \square_j}{\Gamma \vdash \Pi\, x : A.B \triangleright \square_{i \vee j}}$$

$$\boxed{\frac{\Gamma \vdash A \triangleright_{\square} \square_i \qquad \Gamma, x : A \vdash t \triangleright B}{\Gamma \vdash \lambda\, x : A.t \triangleright \Pi\, x : A.B}} \qquad \frac{\Gamma \vdash t : \Pi\, x : A.B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\ u : B[u]}$$

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

$\Gamma \vdash t \triangleright T$ type inference
$\Gamma \vdash t \triangleleft T$ type checking
$\Gamma \vdash t \triangleright_{\bullet} T$ constrained inference

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

$$\frac{(x : T \in \Gamma)}{\Gamma \vdash x \triangleright T} \qquad \frac{}{\Gamma \vdash \square_i \triangleright \square_{i+1}} \qquad \frac{\Gamma \vdash A \triangleright_{\square} \square_i \qquad \Gamma, x : A \vdash B \triangleright_{\square} \square_j}{\Gamma \vdash \Pi\, x : A.B \triangleright \square_{i \vee j}}$$

$$\frac{\Gamma \vdash A \triangleright_{\square} \square_i \qquad \Gamma, x : A \vdash t \triangleright B}{\Gamma \vdash \lambda\, x : A.t \triangleright \Pi\, x : A.B} \qquad \boxed{\frac{\Gamma \vdash t \triangleright_{\Pi} \Pi\, x : A.B \qquad \Gamma \vdash u \triangleleft A}{\Gamma \vdash t\ u \triangleright B[u]}}$$

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

Turning dependent (CCω)

$\Gamma \vdash t \triangleright T$ type inference
$\Gamma \vdash t \triangleleft T$ type checking
$\Gamma \vdash t \triangleright_{\bullet} T$ constrained inference

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

$$\frac{(x : T \in \Gamma)}{\Gamma \vdash x \triangleright T} \qquad \frac{}{\Gamma \vdash \square_i \triangleright \square_{i+1}} \qquad \frac{\Gamma \vdash A \triangleright_{\square} \square_i \qquad \Gamma, x : A \vdash B \triangleright_{\square} \square_j}{\Gamma \vdash \Pi\, x : A.B \triangleright \square_{i \vee j}}$$

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

$$\boxed{\frac{\Gamma \vdash t \triangleright T \qquad \Gamma \vdash T \cong T'}{\Gamma \vdash t \triangleleft T'} \qquad \frac{\Gamma \vdash t \triangleright T \qquad \Gamma \vdash T \to^{\star} \square_i}{\Gamma \vdash t \triangleright_{\square} \square_i} \qquad \frac{\Gamma \vdash t \triangleright T \qquad \Gamma \vdash T \to^{\star} \Pi\, x : A.B}{\Gamma \vdash t \triangleright_{\Pi} \Pi\, x : A.B}}$$

- Different modes command **different computation judgments** ($\to^{\star}$ vs $\cong$)
- **No free conversion** thanks to the judgments' structure

Bidirectional typing is correct

Nothing's changed

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

Bidirectional typing is correct

Nothing's changed

- 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 $\Gamma \vdash T' \cong T$

Nothing's changed

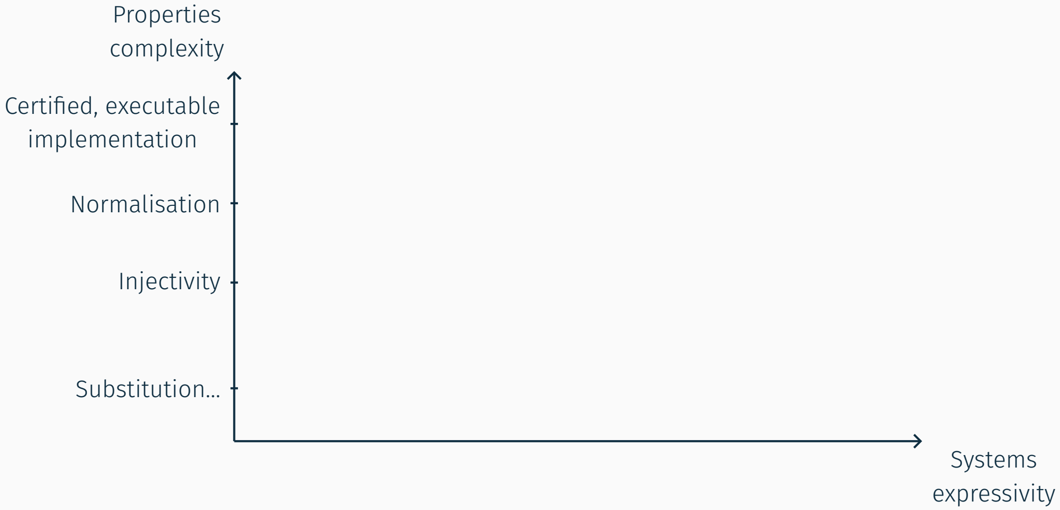
- Soundness: if $\vdash \Gamma$ and $\Gamma \vdash t \triangleright T$ then $\Gamma \vdash t : T$
- Completeness: if $\Gamma \vdash t : T$, there exists T' such that $\Gamma \vdash t \triangleright T'$ and $\Gamma \vdash T' \cong T$

Key properties ("injectivity"):

- reduction finds constructors: if $\Gamma \vdash T \cong \Pi\, x \colon A.\, B$ then $\Gamma \vdash T \to^\star \Pi\, x \colon A'.\, B'$
- if $\Gamma \vdash \Pi\, x \colon A.\, B \cong \Pi\, x \colon A'.\, B'$, then $\Gamma \vdash A \cong A'$ (and similarly for B)

Roadmap

Roadmap


Properties
complexity
Certified, executable
implementation
Normalisation
Injectivity
Substitution...
Systems
expressivity

Normalisation

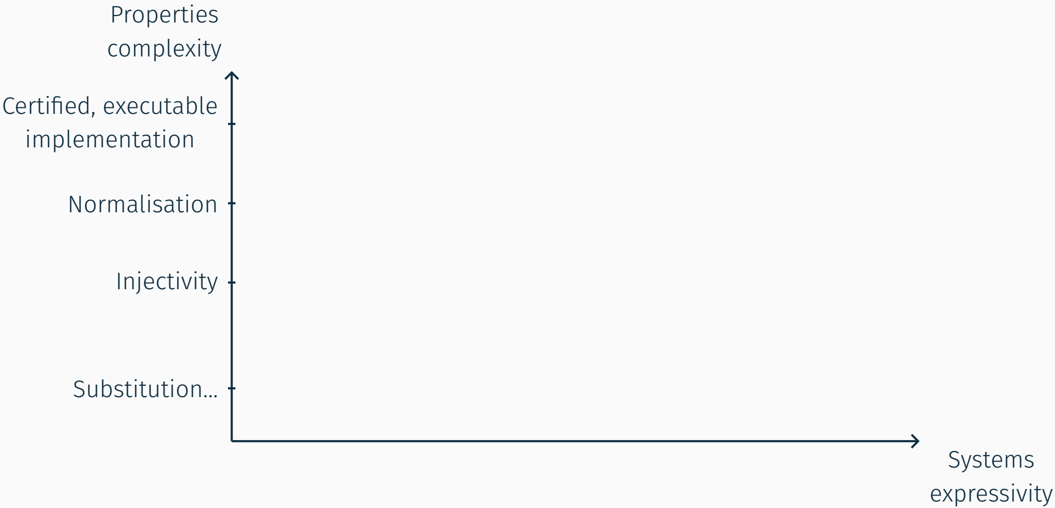
- every reduction path $t_0 \to t_1 \to t_2 \to \ldots$ is finite
- for every well-typed term t there is a normal form $\bar{t} \in \mathsf{Nf}$ s.t. $\Gamma \vdash t \cong \bar{t} : A$

- every reduction path $t_0 \to t_1 \to t_2 \to \ldots$ is finite
- for every well-typed term t there is a normal form $\bar{t} \in \mathsf{Nf}$ s.t. $\Gamma \vdash t \cong \bar{t} : A$

The mother of all properties for dependent type systems:

- decidability of conversion
- canonicity
- consistency ⚠

Roadmap


Properties complexity
Certified, executable implementation
Normalisation
Injectivity
Substitution...
Systems expressivity

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

Logical power VS expressivity

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

CoC is logically stronger than Agda's type theory, very close to Coq's. Time to change subject?

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

CoC is logically stronger than AGDA's type theory, very close to COQ's. Time to change subject?

Logical power is not the same as expressivity!

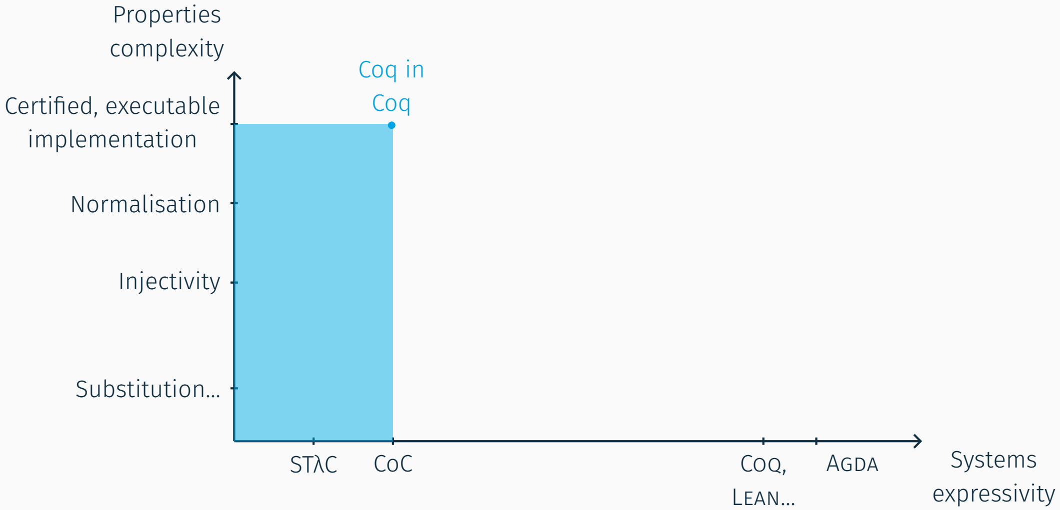
Logical power VS expressivity

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

CoC is logically stronger than Agda's type theory, very close to Coq's. Time to change subject?

Logical power is not the same as expressivity!

Turing-completeness vs "real" language.


Properties
complexity
Coq in
Coq
Certified, executable
implementation
Normalisation
Injectivity
Substitution…
STλC
CoC
Coq,
Lean…
Agda
Systems
expressivity

Coq in Coq?

Coq in Coq?

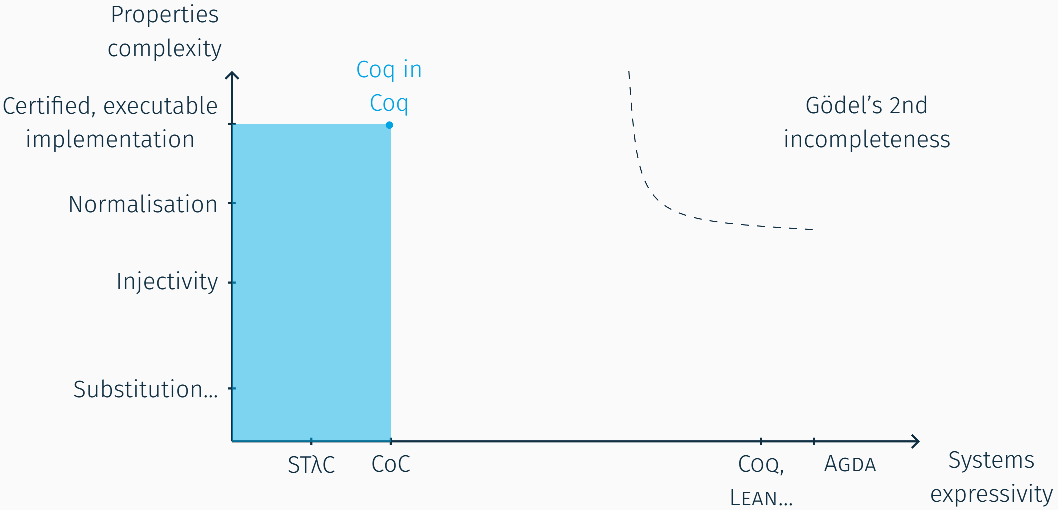
~~Coq in Coq?~~

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

~~COQ IN COQ?~~

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

Or: admit consistency/normalisation and concentrate on the rest.


Properties complexity
Coq in Coq
Certified, executable implementation
Normalisation
Injectivity
Substitution…
Gödel's 2nd incompleteness
STλC
CoC
Coq, Lean…
Agda
Systems expressivity

The MetaCoq project

Jww. Matthieu Sozeau, Yannick Forster,

Nicolas Tabareau, Théo Winterhalter…

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

CCω +

- Complex universes (impredicative propositions, algebraic expressions...)
- Very general (co-)inductive types
- Pattern-matching and fixed-points
- Cumulativity (subtyping)
- ...

MetaCoq in a nutshell

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

CCω +

- Complex universes (impredicative propositions, algebraic expressions…)
- Very general (co-)inductive types
- Pattern-matching and fixed-points
- Cumulativity (subtyping)
- …

Coq, in Coq (bis)

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

CCω +

- Complex universes (impredicative propositions, algebraic expressions...)
- Very general (co-)inductive types
- Pattern-matching and fixed-points
- Cumulativity (subtyping)
- ...

Coq, in Coq (bis)

- Formalized meta-theory of PCUIC
- Normalization axiom to implement a certified type-checker

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

CCω +

- Complex universes (impredicative propositions, algebraic expressions…)
- Very general (co-)inductive types
- Pattern-matching and fixed-points
- Cumulativity (subtyping)
- …

Coq, in Coq (bis)

- Formalized meta-theory of PCUIC
- Normalization axiom to implement a certified type-checker
- There's more: certified extraction, meta-programming…

MetaCoq in a nutshell

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

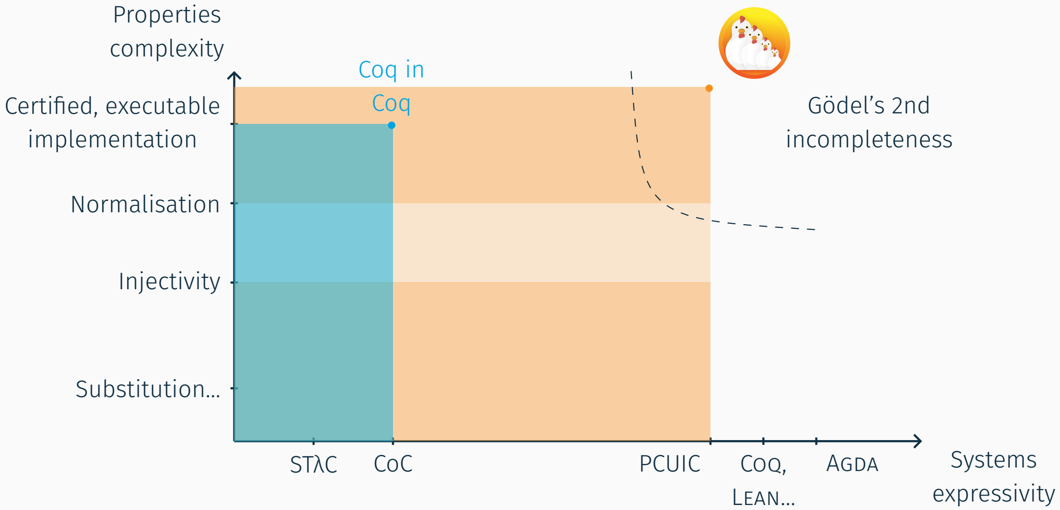
CCω +

- Complex universes (impredicative propositions, algebraic expressions...)
- Very general (co-)inductive types
- Pattern-matching and fixed-points
- Cumulativity (subtyping)
- ...

Coq, in Coq (bis)

- Formalized meta-theory of PCUIC
- Normalization axiom to implement a certified type-checker
- There's more: certified extraction, meta-programming...

Roadmap


Properties
complexity
Certified, executable
implementation
Normalisation
Injectivity
Substitution...
Coq in
Coq
Gödel's 2nd
incompleteness
STλC
CoC
PCUIC
Coq,
Lean...
Agda
Systems
expressivity

Meta-theory of PCUIC

- substitution lemmas (terms, universes)

Meta-theory of PCUIC

- substitution lemmas (terms, universes)
- confluence (“Parallel Reductions in λ-Calculus” (Takahashi 1995))

Meta-theory of PCUIC

- substitution lemmas (terms, universes)
- confluence (“Parallel Reductions in λ-Calculus” (Takahashi 1995))
- completeness of reduction and injectivity

Meta-theory of PCUIC

- substitution lemmas (terms, universes)
- confluence (“Parallel Reductions in λ-Calculus” (Takahashi 1995))
- completeness of reduction and injectivity
- subject reduction

- substitution lemmas (terms, universes)
- confluence ("Parallel Reductions in λ-Calculus" (Takahashi 1995))
- completeness of reduction and injectivity
- subject reduction

Works because conversion is untyped and purely computational.

Meta-theory of PCUIC

- substitut
- confluen
- complete
- subject r

Works becaus

1

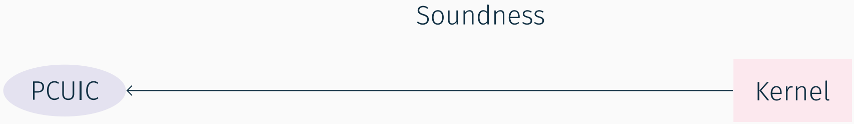
Correct and Complete Type Checking and Certified Erasure for Coq, in Coq

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

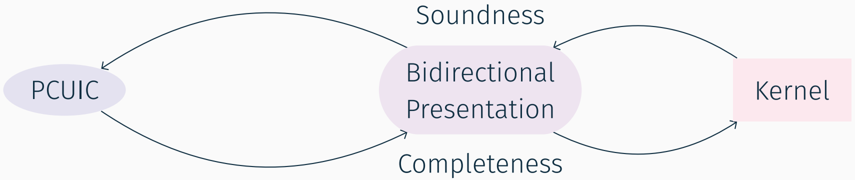
Coq is built around a well-delimited kernel that performs type checking for definitions in a variant of the Calculus of Inductive Constructions (CIC). Although the metatheory of CIC is very stable and reliable, the correctness of its implementation in Coq is less clear. Indeed, implementing an efficient 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**.

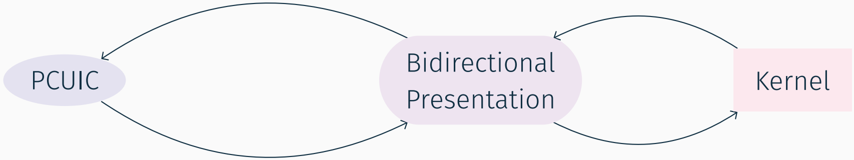
A Correct and Complete Kernel



A Correct and Complete Kernel

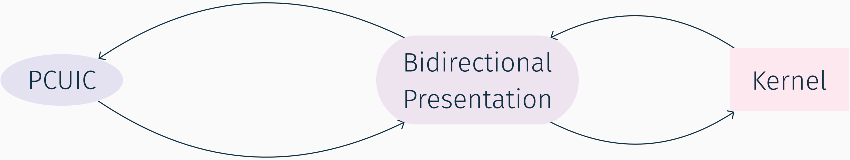


A Correct and Complete Kernel

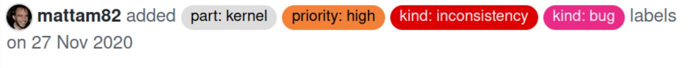


When starting the proof, we realized... it was false!

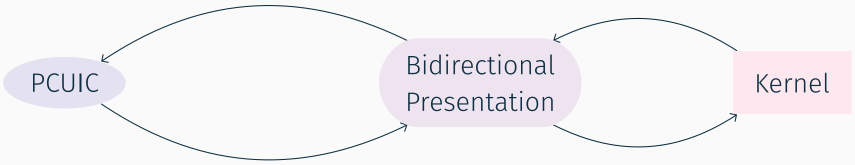
A Correct and Complete Kernel



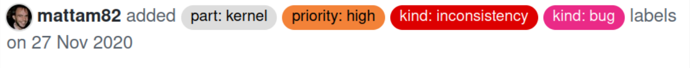
When starting the proof, we realized... it was false!



A Correct and Complete Kernel



When starting the proof, we realized… it was false!



Led to a complete re-design of pattern-matching in Coq.

MetaCoq's future?

MetaCoq is great, but:

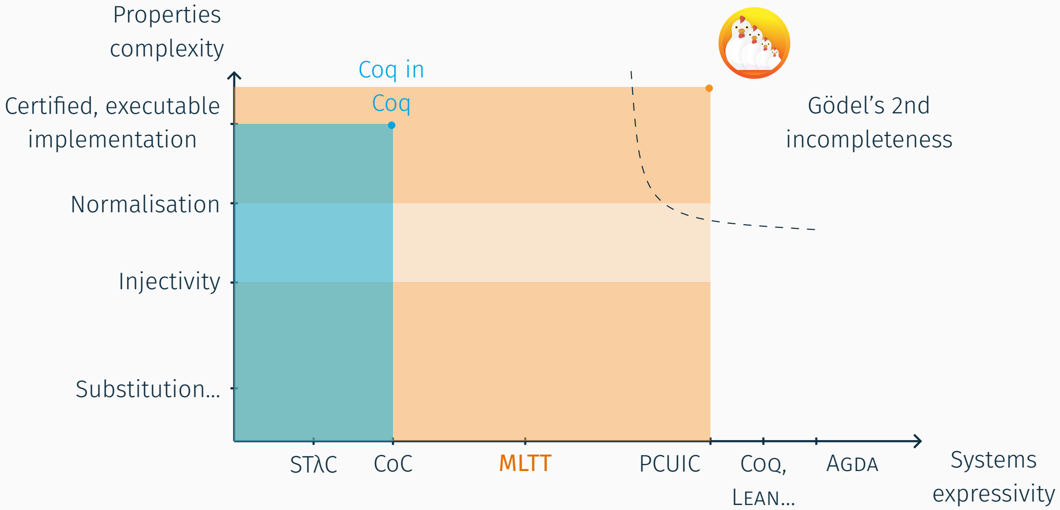
- it does not handle extensionality equations (η-laws);
- its current specification is *not* what semanticists use;
- it does not prove normalisation!

Martin-Löf *à la* Coq

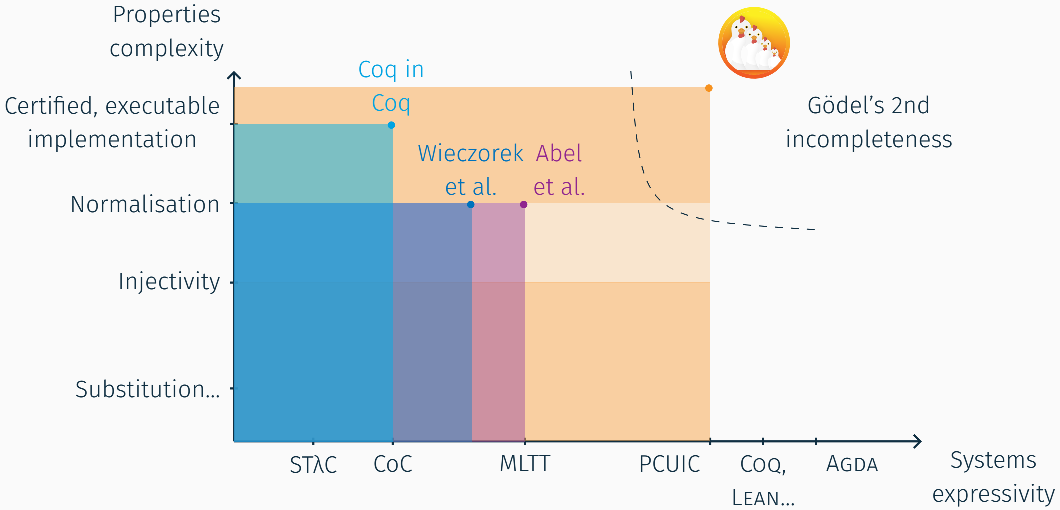
Jww. Arthur Adjedj, Kenji Maillard,

Pierre-Marie Pédrot and Loïc Pujet

Roadmap

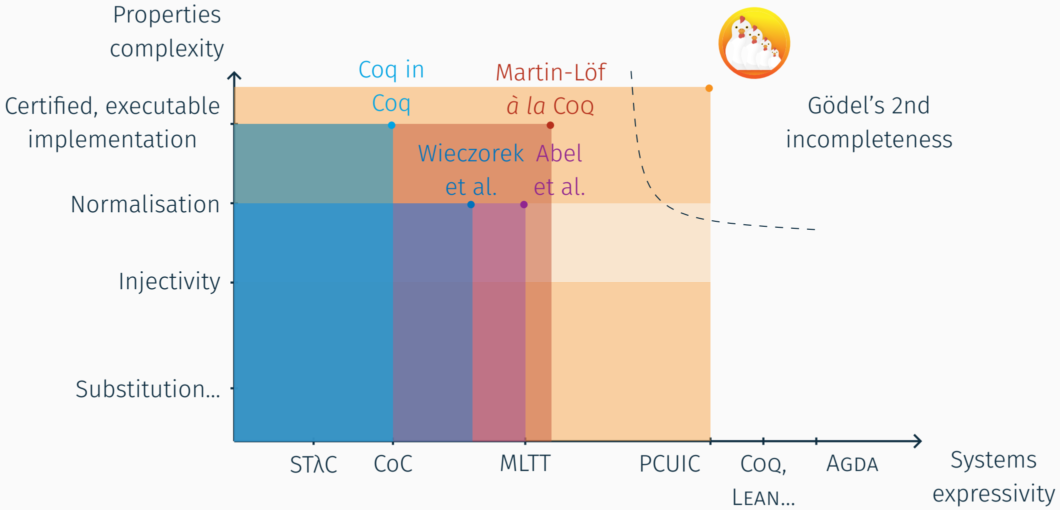

Properties complexity
Certified, executable implementation
Normalisation
Injectivity
Substitution…
Coq in Coq
Gödel's 2nd incompleteness
STλC
CoC
MLTT
PCUIC
Coq, Lean…
Agda
Systems expressivity

Roadmap



Properties complexity
Certified, executable implementation
Normalisation
Injectivity
Substitution...
Coq in Coq
Wieczorek et al.
Abel et al.
Gödel's 2nd incompleteness
STλC
CoC
MLTT
PCUIC
Coq, Lean...
Agda
Systems expressivity

Roadmap



Properties complexity
Certified, executable implementation
Normalisation
Injectivity
Substitution...
Coq in Coq
Martin-Löf à la Coq
Wieczorek et al.
Abel et al.
Gödel's 2nd incompleteness
STλC
CoC
MLTT
PCUIC
Coq, Lean...
Agda
Systems expressivity

Typed Conversion?

$t \cong u$ vs $\Gamma \vdash t \cong u : A$

Typed Conversion?

$t \cong u$ vs $\Gamma \vdash t \cong u : A$

It's bidirectional too!

Conversion checks, neutral comparison infers

Conversion $\cong$ checks

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

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

Conversion checks, neutral comparison infers

Conversion $\cong$ checks

$$\frac{\Gamma \vdash t \to^\star t' \qquad \Gamma \vdash u \to^\star u' \qquad \Gamma \vdash A \to^\star A' \qquad \Gamma \vdash t' \cong_h u' \triangleleft A'}{\Gamma \vdash t \cong u \triangleleft A}$$

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

Neutral comparison $\approx$ infers

$$\frac{\Gamma \vdash m \approx n \triangleright_\Pi \Pi x : A.\ B \qquad \Gamma \vdash t \cong u \triangleleft A}{\Gamma \vdash m\ t \approx n\ u \triangleright B[t]} \qquad \frac{(x : A) \in \Gamma}{\Gamma \vdash x \approx x \triangleright A}$$

Is algorithmic conversion correct?

Is algorithmic conversion correct?

Soundness

Injectivity needed to preserve invariants.

Is algorithmic conversion correct?

Soundness

Injectivity needed to preserve invariants.

Completeness

Transitivity: tricky but doable...

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

Is algorithmic conversion correct?

Soundness

Injectivity needed to preserve invariants.

Completeness

Transitivity: tricky but doable…

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

We need the power of logical relations.

Is algorithmic conversion correct?

Soundness

Injecti

Compl

Transit

Reflex

We nee

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:

Is algorithmic conversion correct?

Soundness

Injecti

Compl

Transit

Reflex

We nee

Decid

ANDRE
JOAKIM
ANDRE

Type theo
a verified
equality
a practica
numbers,
by a suita
once to c
algorithm
Thus, it is
for instan

CCS Con

Addition

ACM Re

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.

Wrapping up

Wrapping up

Two complementary approaches

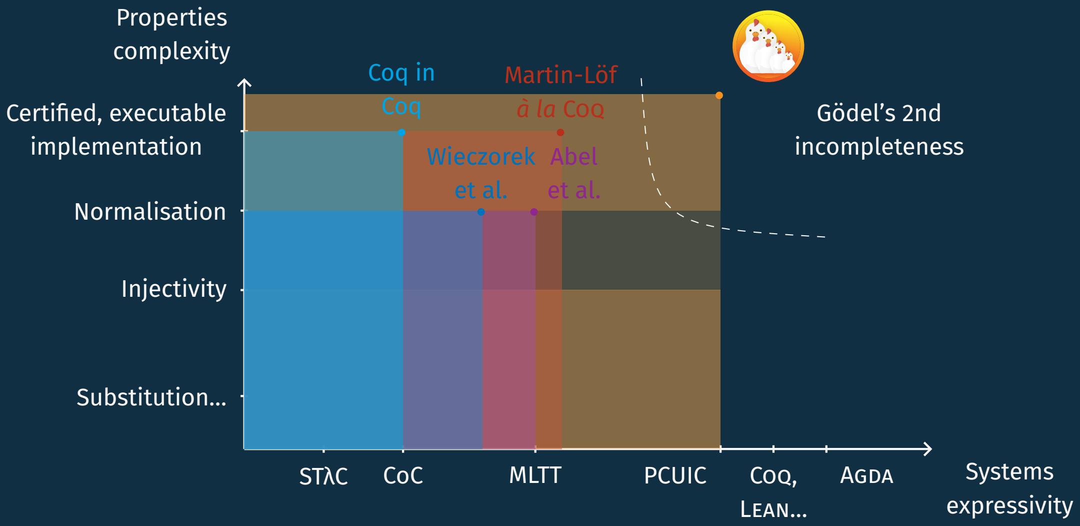
- MetaCoq: focus on **gory** issues of **realistic** systems
- MLTT *à la* Coq: try and go **as far as possible** in a **fully axiom-free way**

Two complementary approaches

- MetaCoq: focus on **gory** issues of **realistic** systems
- MLTT *à la* Coq: try and go **as far as possible** in a **fully axiom-free way**

What now?

- Typed conversion in MetaCoq? Can we do injectivity with η-laws, but without logical relations? Can we do all of Coq, and more?
- How far can we scale MLTT *à la* Coq? What are the required practical/theoretical tools we need?
- Can we bridge the gap between the two?


Properties complexity
Certified, executable implementation
Normalisation
Injectivity
Substitution…
Coq in Coq
Martin-Löf *à la* Coq
Wieczorek et al.
Abel et al.
Gödel's 2nd incompleteness
STλC
CoC
MLTT
PCUIC
Coq, Lean…
Agda
Systems expressivity

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`.

[Tak95] M. Takahashi. "Parallel Reductions in λ-Calculus". In: *Information and Computation* 118.1 (1995), pp. 120–127. ISSN: 0890-5401. DOI: `10.1006/inco.1995.1057`. URL: `https://www.sciencedirect.com/science/article/pii/S0890540185710577`.

[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`.