Rocq's History and Future

Google Zürich, Switzerland

January 8th 2026

Matthieu Sozeau, Inria & LS2N, University of Nantes, France

Outline

1. What is Rocq?
2. Coq/Rocq History
3. Rocq Today
4. Rocq's Future

What is Rocq?

What is Rocq?

- A proof assistant that evolved over 40 years of research in type theory and interactive theorem proving

What is Rocq?

- A proof assistant that evolved over 40 years of research in type theory and interactive theorem proving
- 1984: 1/2-developer research software project - 1 source file, interpreted by Le Lisp

What is Rocq?

- A proof assistant that evolved over 40 years of research in type theory and interactive theorem proving
- 1984: 1/2-developer research software project - 1 source file, interpreted by Le Lisp
- 2026: 100+ contributors, 150kLoC of OCaml, multi-platform, widespread use in CS and math.

What is Rocq?

- A proof assistant that evolved over 40 years of research in type theory and interactive theorem proving
- 1984: 1/2-developer research software project - 1 source file, interpreted by Le Lisp
- 2026: 100+ contributors, 150kLoC of OCaml, multi-platform, widespread use in CS and math.
- An implementation of the Polymorphic, Cumulative Calculus of Inductive Constructions

What is Rocq?

- A proof assistant that evolved over 40 years of research in type theory and interactive theorem proving
- 1984: 1/2-developer research software project - 1 source file, interpreted by Le Lisp
- 2026: 100+ contributors, 150kLoC of OCaml, multi-platform, widespread use in CS and math.
- An implementation of the Polymorphic, Cumulative Calculus of Inductive Constructions and then some!

The Curry-Howard “Isomorphism” at a higher level

- A unified language to express specifications and computations

The Curry-Howard "Isomorphism" at a higher level

- A unified language to express specifications and computations
- A unified language mixing logic and programing

The Curry-Howard "Isomorphism" at a higher level

- A unified language to express specifications and computations
- A unified language mixing logic and programing
- All based on the notion of (dependent) types: `nat` natural numbers, `Z` integers, `int31` numbers, real numbers (computable or not), propositions

The Curry-Howard "Isomorphism" at a higher level

- A unified language to express specifications and computations
- A unified language mixing logic and programing
- All based on the notion of (dependent) types: `nat` natural numbers, `Z` integers, `int31` numbers, real numbers (computable or not), propositions
- Propositions as types: `True`, `False`, `∧`, `∨`, `exists x : A, P`, `forall x : A, P`

The Curry-Howard “Isomorphism” at a higher level

- A unified language to express specifications and computations
- A unified language mixing logic and programing
- All based on the notion of (dependent) types: `nat` natural numbers, `Z` integers, `int31` numbers, real numbers (computable or not), propositions
- Propositions as types: `True`, `False`, `∧`, `∨`, `exists x : A, P`, `forall x : A, P`
- Dependency of types/propositions on terms:

```
Definition div : forall x y : Z, y <> 0 → { (q, r) | x = q * y + r }
```

The Curry-Howard "Isomorphism" at a higher level

- A unified language to express specifications and computations
- A unified language mixing logic and programing
- All based on the notion of (dependent) types: `nat` natural numbers, `Z` integers, `int31` numbers, real numbers (computable or not), propositions
- Propositions as types: `True`, `False`, `∧`, `∨`, `exists x : A, P`, `forall x : A, P`
- Dependency of types/propositions on terms:

```
Definition div : forall x y : Z, y <> 0 → { (q, r) | x = q * y + r }
```

- With (parametric) polymorphism (ML/Haskell/"Generics"):

```
Definition length : forall A : Type, list A → nat
```

Beyond Martin-Löf Type Theory

- Inductive and coinductive indexed types: e.g, sequences indexed by their length, streams, processes

Beyond Martin-Löf Type Theory

- Inductive and coinductive indexed types: e.g, sequences indexed by their length, streams, processes
- Primitive types: int, strings, IEEE754 floats, (persistent arrays)

Beyond Martin-Löf Type Theory

- Inductive and coinductive indexed types: e.g, sequences indexed by their length, streams, processes
- Primitive types: int, strings, IEEE754 floats, (persistent arrays)
- Universe polymorphism and cumulativity

Beyond Martin-Löf Type Theory

- Inductive and coinductive indexed types: e.g, sequences indexed by their length, streams, processes
- Primitive types: int, strings, IEEE754 floats, (persistent arrays)
- Universe polymorphism and cumulativity
- An ML-style module system

Beyond Martin-Löf Type Theory

- Inductive and coinductive indexed types: e.g, sequences indexed by their length, streams, processes
- Primitive types: int, strings, IEEE754 floats, (persistent arrays)
- Universe polymorphism and cumulativity
- An ML-style module system
- Extraction to usual functional languages (Scheme, OCaml, Haskell)

Beyond Martin-Löf Type Theory

- Inductive and coinductive indexed types: e.g, sequences indexed by their length, streams, processes
- Primitive types: int, strings, IEEE754 floats, (persistent arrays)
- Universe polymorphism and cumulativity
- An ML-style module system
- Extraction to usual functional languages (Scheme, OCaml, Haskell)
- A tactic language to write proof scripts

Coq/Rocq History

CONSTR

- 1983-84 – **v1**: Research prototype CONSTR for the Theory of Constructions developed by Gérard Huet, theory and examples by Thierry Coquand.

CONSTR

- 1983-84 – **v1**: Research prototype CONSTR for the Theory of Constructions developed by Gérard Huet, theory and examples by Thierry Coquand.
- 1985-86 – **v2**: Predicative universes, abstract machine for conversion, unification (Huet) and consistency proof for the Calculus of Constructions (Coquand).

CONSTR

- 1983-84 – **v1**: Research prototype CONSTR for the Theory of Constructions developed by Gérard Huet, theory and examples by Thierry Coquand.
- 1985-86 – **v2**: Predicative universes, abstract machine for conversion, unification (Huet) and consistency proof for the Calculus of Constructions (Coquand).
- 1986-87 – **v3**: auto (Coquand), based on CAML (running on Le Lisp)

CONSTR

- 1983-84 – **v1**: Research prototype CONSTR for the Theory of Constructions developed by Gérard Huet, theory and examples by Thierry Coquand.
- 1985-86 – **v2**: Predicative universes, abstract machine for conversion, unification (Huet) and consistency proof for the Calculus of Constructions (Coquand).
- 1986-87 – **v3**: auto (Coquand), based on CAML (running on Le Lisp)
- 1987-88 – **v4**: Extraction and Impredicative encodings of Inductives (Christine Paulin-Möhring), Vernacular language (Gilles Dowek), Section mechanism.

CONSTR

- 1983-84 – **v1**: Research prototype CONSTR for the Theory of Constructions developed by Gérard Huet, theory and examples by Thierry Coquand.
- 1985-86 – **v2**: Predicative universes, abstract machine for conversion, unification (Huet) and consistency proof for the Calculus of Constructions (Coquand).
- 1986-87 – **v3**: auto (Coquand), based on CAML (running on Le Lisp)
- 1987-88 – **v4**: Extraction and Impredicative encodings of Inductives (Christine Paulin-Möhring), Vernacular language (Gilles Dowek), Section mechanism.
- 1989 – **v4.9-4.11** with “elastic” universes, extensive documentation.

Coq

The Calculus of Inductive Constructions

- 1989-95 – **v5**: Primitive Inductive Types (Paulin, Coquand, Werner)
  - Move to Caml-light.
  - 1991: large rewrite by Chet Murthy for efficiency and user extensibility of the system. Proof engine with existential variables (Herbelin & Dowek).

Coq

The Calculus of Inductive Constructions

- 1989-95 – **v5**: Primitive Inductive Types (Paulin, Coquand, Werner)
  - Move to Caml-light.
  - 1991: large rewrite by Chet Murthy for efficiency and user extensibility of the system. Proof engine with existential variables (Herbelin & Dowek).
- 1996-99 – **v6**:
  - Port to OCaml by Bruno Barras.
  - Tactics, decision procedures, proof engine, automation, coercions, parsing and notations, ..

Coq + Extraction

- Rewrite using a modular architecture by Jean-Christophe Filliâtre and Hugo Herbelin.
- let-ins and pattern-matching compilation (Herbelin),
- Improved reduction, based on a stack machine (Barras)
- New extraction (Pierre Letouzey and Filliâtre)
- Many new tactics and decision procedures.

The 8.x Era

The Predicative Calculus of (Co-)Inductive Constructions

- 2004 – **v8.0**: revamped syntax and extensibility by notations with scopes (Herbelin, Barras), drop impredicative **`Set`** as default, reworked module system. Congruence closure tactic, solidified Standard Library.
- 2006 – **v8.1**: `vm_compute`, refined inductive types, setoid rewriting, reflexive `ring` and `tauto` tactics (Grégoire, Mahboubi, Théry), `field` tactic, connection to ATPs, **`Program`** (Sozeau), `Function` (Courtieu, Bertot), natural proof language.
- 2008 – **v8.2**: Typeclasses, `BigN`/number libraries and primitive `int31` numbers, `omega` on arbitrary rings, separate `coqchk` checker and `coqdoc` tool, many many tactic and high-level improvements.

The 8.x Era

The Predicative Calculus of (Co-)Inductive Constructions, v2

- 2010 – **v8.3**: module system generalization (`Include`, `<+`), `nsatz` tactic, finite sets and maps rework, refinements of features.
- 2011 – **v8.4**: η-conversion (Herbelin), modular number library (Letouzey), new proof engine (Arnaud Spiwack) supporting (dependent) goal manipulation and structured proof scripts, protocol for `CoqIDE` (Vincent Gross).

The 8.5 Version (2015)

The Predicative, Polymorphic Calculus of (Co-)Inductive Constructions

- A new asynchronous evaluation and compilation mode by Enrico Tassi with help from Bruno Barras and Carst Tankink.
- Full integration of the new proof engine by Arnaud Spiwack helped by Pierre-Marie Pédrot,
- Addition of conversion and reduction based on native compilation by Maxime Dénès and Benjamin Grégoire.
- Full universe polymorphism for definitions and inductive types by Matthieu Sozeau.
- An implementation of primitive projections with η-conversion bringing significant performance improvements when using records by Matthieu Sozeau.

The 8.6 Version (2016)

Coq version 8.6 contains the result of refinements, stabilization of 8.5's features and cleanups of the internals of the system. Over the year of (now time-based) development, about 450 bugs were resolved and over 100 contributions integrated. The main user visible changes are:

- A new, faster state-of-the-art universe constraint checker, by Jacques-Henri Jourdan.
- In CoqIDE and other asynchronous interfaces, more fine-grained asynchronous processing and error reporting by Enrico Tassi, making Coq capable of recovering from errors and continue processing the document.
- More access to the proof engine features from Ltac: goal management primitives, range selectors and a `typeclasses eauto` engine handling multiple goals and multiple successes, by Cyprien Mangin, Matthieu Sozeau and Arnaud Spiwack.
- Tactic behavior uniformization and specification, generalization of intro-patterns by Hugo Herbelin and others.
- A brand new warning system allowing to control warnings, turn them into errors or ignore them selectively by Maxime Dénès, Guillaume Melquiond, Pierre-Marie Pédrot and others.
- Irrefutable patterns in abstractions, by Daniel de Rauglaudre.
- The ssreflect subterm selection algorithm by Georges Gonthier and Enrico Tassi is now accessible to tactic writers through the ssrmatching plugin.
- Integration of LtacProf, a profiler for Ltac by Jason Gross, Paul Steckler, Enrico Tassi and Tobias Tebbi.

The 8.7 Version (2017)

The Predicative, Polymorphic and Cumulative
Calculus of (Co-)Inductive Constructions

- New tactics: variants of tactics supporting existential variables **`eassert`**, **`eenough`**, etc... by Hugo Herbelin. Tactics `extensionality in H` and **`inversion_sigma`** by Jason Gross, `specialize with ...` accepting partial bindings by Pierre Courtieu.
- `Cumulative Polymorphic Inductive` types, allowing cumulativity of universes to go through applied inductive types, by Amin Timany and Matthieu Sozeau.
- Integration of the SSReflect plugin and its documentation in the reference manual, by Enrico Tassi, Assia Mahboubi and Maxime Dénès.
- The `coq_makefile` tool was completely redesigned to improve its maintainability and the extensibility of generated Makefiles, and to make `_CoqProject` files more palatable to IDEs by Enrico Tassi.

The 8.8-8.18 Versions (2018-2023)

“Consolidation”

Rationalized development model:

- Move to git/github/gitlab

The 8.8-8.18 Versions (2018-2023)

"Consolidation"

Rationalized development model:

- Move to git/github/gitlab
- Formalized release management and release cycles (10 versions in 6 years!)

The 8.8-8.18 Versions (2018-2023)

“Consolidation”

Rationalized development model:

- Move to git/github/gitlab
- Formalized release management and release cycles (10 versions in 6 years!)
- Bug triaging and minimizing

The 8.8-8.18 Versions (2018-2023)

“Consolidation”

Rationalized development model:

- Move to git/github/gitlab
- Formalized release management and release cycles (10 versions in 6 years!)
- Bug triaging and minimizing
- Critical issue tracking

The 8.8-8.18 Versions (2018-2023)

“Consolidation”

Rationalized development model:

- Move to git/github/gitlab
- Formalized release management and release cycles (10 versions in 6 years!)
- Bug triaging and minimizing
- Critical issue tracking
- Contribution guide

The 8.8-8.18 Versions (2018-2023)

“Consolidation”

Rationalized development model:

- Move to git/github/gitlab
- Formalized release management and release cycles (10 versions in 6 years!)
- Bug triaging and minimizing
- Critical issue tracking
- Contribution guide
- Community management (Zulip, Discourse) and code of conduct team.

The 8.8-8.18 Versions (2018-2023)

- Development of the Coq Consortium led by Yves Bertot, supporting research engineers Maxime Dénes and Gaëtan Gilbert.

The 8.8-8.18 Versions (2018-2023)

- Development of the Coq Consortium led by Yves Bertot, supporting research engineers Maxime Dénes and Gaëtan Gilbert.
- Development of the Coq Platform: multi-platform TeX-Live style distributions of Coq together with essential packages, led by Michael Soegtrop (Intel, Apple).

The 8.8-8.18 Versions (2018-2023)

- Development of the Coq Consortium led by Yves Bertot, supporting research engineers Maxime Dénes and Gaëtan Gilbert.
- Development of the Coq Platform: multi-platform TeX-Live style distributions of Coq together with essential packages, led by Michael Soegtrop (Intel, Apple).
- Main implementation changes: efficiency, efficiency, maintainability, correctness (Pierre-Marie Pédrot and Gaëtan Gilbert).

The 8.8-8.18 Versions (2018-2023)

Main theory changes:

- Primitive types (ints, floats, strings, persistent arrays)
- `SProp` (definitional proof-irrelevance).

The 8.8-8.18 Versions (2018-2023)

Main user-level changes:

- Ltac2: ML-style language on top of the tactic monad

The 8.8-8.18 Versions (2018-2023)

Main user-level changes:

- Ltac2: ML-style language on top of the tactic monad
- Improved notations/grammar extensibility, `String` and `Number` notations, binders

The 8.8-8.18 Versions (2018-2023)

Main user-level changes:

- Ltac2: ML-style language on top of the tactic monad
- Improved notations/grammar extensibility, `String` and `Number` notations, binders
- Reversible coercions and improved coherence checking (enabling elpi/hb)

The 8.8-8.18 Versions (2018-2023)

Main user-level changes:

- Ltac2: ML-style language on top of the tactic monad
- Improved notations/grammar extensibility, `String` and `Number` notations, binders
- Reversible coercions and improved coherence checking (enabling elpi/hb)
- Proof language: focusing, ssr tactics, improved `Search`, tactic deprecations

The 8.8-8.18 Versions (2018-2023)

Main user-level changes:

- Ltac2: ML-style language on top of the tactic monad
- Improved notations/grammar extensibility, `String` and `Number` notations, binders
- Reversible coercions and improved coherence checking (enabling elpi/hb)
- Proof language: focusing, ssr tactics, improved `Search`, tactic deprecations
- Stdlib: reorganizations, deprecations, extensions, split of stdlib from main repo

The 8.8-8.18 Versions (2018-2023)

Main user-level changes:

- Ltac2: ML-style language on top of the tactic monad
- Improved notations/grammar extensibility, `String` and `Number` notations, binders
- Reversible coercions and improved coherence checking (enabling elpi/hb)
- Proof language: focusing, ssr tactics, improved `Search`, tactic deprecations
- Stdlib: reorganizations, deprecations, extensions, split of stdlib from main repo
- Support for attributes, warning system

The 8.8-8.18 Versions (2018-2023)

Main user-level changes:

- Ltac2: ML-style language on top of the tactic monad
- Improved notations/grammar extensibility, `String` and `Number` notations, binders
- Reversible coercions and improved coherence checking (enabling elpi/hb)
- Proof language: focusing, ssr tactics, improved `Search`, tactic deprecations
- Stdlib: reorganizations, deprecations, extensions, split of stdlib from main repo
- Support for attributes, warning system
- Parallel separate compilation, asynchronous processing, LSPs

The 8.8-8.18 Versions (2018-2023)

Main user-level changes:

- Ltac2: ML-style language on top of the tactic monad
- Improved notations/grammar extensibility, `String` and `Number` notations, binders
- Reversible coercions and improved coherence checking (enabling elpi/hb)
- Proof language: focusing, ssr tactics, improved `Search`, tactic deprecations
- Stdlib: reorganizations, deprecations, extensions, split of stdlib from main repo
- Support for attributes, warning system
- Parallel separate compilation, asynchronous processing, LSPs
- `opam`, `dune`, `nix` and `Docker` support.

The 8.19-8.20 Versions (2024)

The Sort and Universe Polymorphic, Cumulative
Calculus of (Co-)Inductive Constructions

- Sort polymorphism (Poiret, Gilbert, Maillard, Pédrot, Sozeau, Tabareau, Tanter)
  Parameterize definitions over `Prop`/`SProp`/`Type`/…
  - Improve elaboration of definitions (less eager commitment to `Type` for example)
  - Allow deep extensibility/customization of the logic by (expert) users:
    Embedded Domain-Specific Logics
    - Observational Type Theory
    - Exceptional Type Theory, …

The 8.19-8.20 Versions (2024)

The Sort and Universe Polymorphic, Cumulative
Calculus of (Co-)Inductive Constructions

- Sort polymorphism (Poiret, Gilbert, Maillard, Pédrot, Sozeau, Tabareau, Tanter)
  Parameterize definitions over `Prop`/`SProp`/`Type`/…
  - Improve elaboration of definitions (less eager commitment to `Type` for example)
  - Allow deep extensibility/customization of the logic by (expert) users:
    Embedded Domain-Specific Logics
    - Observational Type Theory
    - Exceptional Type Theory, …
- User-defined rewrite rules (Leray, Winterhalter): e.g. natural numbers with definitionally associative and neutral addition and multiplication.

The 8.19-8.20 Versions (2024)

- Ltac2 consolidation (Pédrot, Gilbert, Gross), profiling and debugging support

The 8.19-8.20 Versions (2024)

- Ltac2 consolidation (Pédrot, Gilbert, Gross), profiling and debugging support
- Relocatability, interuptibility, use of memprof-limits, flambda

The 8.19-8.20 Versions (2024)

- Ltac2 consolidation (Pédrot, Gilbert, Gross), profiling and debugging support
- Relocatability, interuptibility, use of memprof-limits, flambda
- VSRocq LSP development (Maxime Dénès, Enrico Tassi and Romain Tetley)

The 8.19-8.20 Versions (2024)

- Ltac2 consolidation (Pédrot, Gilbert, Gross), profiling and debugging support
- Relocatability, interuptibility, use of memprof-limits, flambda
- VSRocq LSP development (Maxime Dénès, Enrico Tassi and Romain Tetley)
- SERapi/JSCoq/coq-lsp/Pétanque (Emilio Jesus Gallego Arias)

The 8.19-8.20 Versions (2024)

- Ltac2 consolidation (Pédrot, Gilbert, Gross), profiling and debugging support
- Relocatability, interuptibility, use of memprof-limits, flambda
- VSRocq LSP development (Maxime Dénès, Enrico Tassi and Romain Tetley)
- SERapi/JSCoq/coq-lsp/Pétanque (Emilio Jesus Gallego Arias)
- Alectryon (Clément Pit-Claudel)

The Rocq Prover - 9.0-9.1 (2025)

- Renaming, new identity and website
- Splitting the “Coq” Stdlib into a Rocq Corelib and Rocq Stdlib
- A single `rocq` binary dispatching to various commands
- Consolidation and improvements of Sort Polymorphism (extraction, elaboration), Ltac2, Notations

Tools and Ressources

Tools:

- Interfaces: Proof General, RocqIDE, LSPs, Waterproof, JSCoq
- Documentation: coqdoc, Alectryon

Ressources:

- Software Foundations Series (Pierce et al)
- Computer Arithmetic and Formal Proofs (Boldo & Melquiond)
- Ssreflect / Mathematical Components (Gonthier, Mahboubi, Tassi)
- DTP (Chlipala)
- Tutorials

See the new website gathering all this!

Selected Applications of Rocq

Libraries, Frameworks:

- Mathematical Components: Four Colour Theorem, Feit-Thompson, …
- CompCert and its variants
- Hoare Type Theory, Iris and its variants
- DeepSpec: Quickchick, VST, CertiKOS, VeLLVM, Kami, DataCert, REMS, CertiCoq, …
- Interaction Trees, FreeSpec, Paco, Relation Algebra, …
- HoTT-Coq, UniMath
- MetaRocq, logrel-tt, WasmCert-Coq, …
- …

Rocq Today

The Rocq Prover - 9.2 (ETA March 2026)

- Support for sort elimination constraints (Rosain et al, POPL'26)

The Rocq Prover - 9.2 (ETA March 2026)

- Support for sort elimination constraints (Rosain et al, POPL'26)
- Rewriting with arbitrary relations supporting the Leibniz principle (in any appropriate sort).

The Rocq Prover - 9.2 (ETA March 2026)

- Support for sort elimination constraints (Rosain et al, POPL'26)
- Rewriting with arbitrary relations supporting the Leibniz principle (in any appropriate sort).
- `Scheme` tables for robust scheme/derived principle lookups (deprecating fragile lookups "by name")

The Rocq Prover - 9.2 (ETA March 2026)

- Support for sort elimination constraints (Rosain et al, POPL'26)
- Rewriting with arbitrary relations supporting the Leibniz principle (in any appropriate sort).
- `Scheme` tables for robust scheme/derived principle lookups (deprecating fragile lookups “by name”)
- Clean support for `abstract` / environment extension in proofs / obligations

The Rocq Prover - 9.2 (ETA March 2026)

- Support for sort elimination constraints (Rosain et al, POPL'26)
- Rewriting with arbitrary relations supporting the Leibniz principle (in any appropriate sort).
- `Scheme` tables for robust scheme/derived principle lookups (deprecating fragile lookups "by name")
- Clean support for `abstract` / environment extension in proofs / obligations
- Extended Ltac2 APIs & notation support

The Rocq Prover - 9.2 (ETA March 2026)

- Support for sort elimination constraints (Rosain et al, POPL'26)
- Rewriting with arbitrary relations supporting the Leibniz principle (in any appropriate sort).
- `Scheme` tables for robust scheme/derived principle lookups (deprecating fragile lookups "by name")
- Clean support for `abstract` / environment extension in proofs / obligations
- Extended Ltac2 APIs & notation support
- Performance improvements with OCaml 5, on par or improving w.r.t 4.14 in many cases now

Rocq Development Teams

- **Rocq Core**: 11 researchers/engineers led by a coordinator: Huet 1984-1996, Paulin 1996-2006, Herbelin 2006-2016, Sozeau 2016-2026?
- **Rocq Maintainers** (of The Rocq Prover projects): ~50 researchers/engineers
- **Rocq Consortium**: backed by Inria and contributors, supports 2 engineers: G. Gilbert (Core) and S. Borgogno (Rocq Platform, Docker-Rocq)
- **Rocq Community**: open source community effort of package maintance

Rocq Ecosystem (focusing on the core team members interests)

- `Ltac2`: small/medium-scale metaprogramming 'done right'
- elpi & Hierarchy Builder: high-level, large scale metaprogramming and working with hierarchies of structures
- Mathematical Components: largest Rocq library of formalized mathematics (Algebra, Analysis, Physics…)
- MetaRocq: metaprogramming, verified metatheory, type-checker and erasure for Rocq
- CertiCoq/CertiCoq-WASM/Verified Extraction: verified extraction to C/Clight/WASM/OCaml/…
- … too many to name here!

Rocq's Future

Current R&D

- **Sort Polymorphism with Elimination Constraints** (Rosain et al, POPL'26)
  Generic eliminators, generic theories working over `Prop/SProp/Type/ ..`
- **Universe Level Polymorphism** (Sozeau and Bezem, RocqPL'26)
  Algebraic universes everywhere based to a new, verified, constraint satisfaction and validity algorithms for the $(\mathbb{N}, \max, \leq)$ semilattice

Current R&D: Rewrite Rules

Encode the Cake and Eat it Too, Leray and Winterhalter, POPL'26.

Extend definitional equality with semi-decidable theories locally and modularly.

- Axiomatic quotients with a computation rule (~ Lean's theory)
- Work up-to associative and neutral operations, e.g., indexing by numbers, with less dependent type hassles.

Current R&D: Rewrite Rules

- More comfortable deep-shallow embeddings.

Example Decidable substitution calculus on AST definitions

```
Module Type Renaming.

  Parameter renaming : Type.

  Symbol ren_var : renaming → nat → nat.
  Symbol up_ren : renaming → renaming.
  Symbol shift_ren : renaming.

  Symbol comp_ren : renaming → renaming → renaming.
  Symbol id_ren : renaming.

  Rewrite Rules ren_rew :=
  | ren_var (up_ren ?σ) O => O
  | ren_var (up_ren ?σ) (S ?n) => S (ren_var ?σ ?n)
  | ren_var shift_ren ?n => S ?n
  | ren_var (comp_ren ?σ ?τ) ?n => ren_var ?σ (ren_var ?τ ?n)
  | ren_var id_ren ?n => ?n
  | comp_ren id_ren ?σ => ?σ
  | comp_ren ?σ id_ren => ?σ
  | comp_ren (comp_ren ?σ ?σ') ?σ'' => comp_ren ?σ (comp_ren ?σ' ?σ'').

End Renaming.
```

```
Definition rename (r : renaming) (t : term) : term :=
  ..

Goal forall (t : term) r, comp_ren (shift_ren r) id_ren = r.
Proof. reflexivity. Qed.
```

Current R&D: Custom sorts

Observational Equality Meets CIC

Pujet, Leray and Tabareau, TOPLAS'25

Leibniz equality (in `Prop`) $\Rightarrow$ Observational Equality (in `SProp`)

- Definitionally Proof-Irrelevant Equality: UIP is built-in
- Functional and propositional extensionality hold by definition
- Quotients by propositional relations without breaking the metatheory!

OTT provides axiomatic freedom: extension with consistent axioms in `SProp` is safe ($\neq$ Homotopy Type Theory).

For example, canonicity for $\mathbb{N}$ still holds even in presence of classical axioms like excluded middle or choice.

Current R&D: Custom sorts

Ghost Type Theory / Ghost Sort

Winterhalter, ICFP'24 / Rosain, Sozeau and Winterhalter, TYPES'25

A sort for proof-relevant **and** computationally irrelevant data (~ "ghost" variables used in program logics).

- Erased at extraction time (or even conversion time)
- "Inductive Families Need Not Store Their Indices" (Brady, McBride and McKinna)
- Clean basis to internalise Hoare Type Theory and variants.

Ghost Sort Example: Indexed Types I

Wasteful representation

```
(* Total lookup function *)

Inductive fin : nat → Set :=
| finO {n} : fin (S n)
| finS {n} : fin n → fin (S n).

Fixpoint lookup {A} (l : list A) :
  fin (length l) → A :=
  match l return fin (length l) → A with
  | nil ⇒ fun (f : fin 0) ⇒
      False_rect (match f in (fin 0) with end)
  | cons a l ⇒ fun f : fin (S (length l)) ⇒
    match f with
    | finO ⇒ a
    | finS f' ⇒ lookup l f'
    end
  end.
```

```
(* Extraction to OCaml *)

type fin =
| Fin0 of nat
| FinS of nat * fin

let rec lookup l f =
  match l with
  | Nil → assert false
  | Cons (a, l0) →
    match f with
    | FinO _ → a
    | FinS (_, f') → lookup l0 f'
```

Ghost Sort Example: Indexed Types II

Optimal representation

```
(* Total lookup function *)

Inductive fin : nat@{Ghost; 0} → Set :=
| finO {n} : fin (S n)
| finS {n} : fin n → fin (S n).

Fixpoint lookup {A} (l : list A)
  : fin (length@{Ghost; 0} l) → A :=
  match l return fin (length@{Ghost} l) → A with
  | nil ⇒ fun (f : fin 0) ⇒
      False_rect (match f in (fin 0) with end)
  | cons a l ⇒ fun f : fin (S (length l)) ⇒
    match f with
    | finO ⇒ a
    | finS f' ⇒ lookup l f'
    end
  end.
```

```
(* Extraction to OCaml: *)

type fin = nat =
| Fin0
| FinS of fin

let rec lookup l f =
  match l with
  | Nil → assert false
  | Cons (a, l0) →
    match f with
    | FinO → a
    | FinS f' → lookup l0 f'
```

Current R&D

- *A Lazy, Concurrent Convertibility Checker*, Nathanëlle Courant and Xavier Leroy, POPL'26

  "Optimal" conversion with sharing

- Programmable reduction and certificates for conversion, Quentin Corradi & Clément Pit-Claudel

  https://systemf.epfl.ch/blog/rocq-conversion-algorithm/

  Avoid the dreaded non-terminating `Qed`

MetaRocq Roadmap

MetaRocq: Reification of Rocq in Rocq + self-hosted type-checker and extraction

- Improved meta-programming environment (with verifiability) for expressive and efficient user-defined tactics.
- Peregrine (https://peregrine-project.github.io) Verified extraction middle-end from Rocq, Agda and Lean to C/Assembly/WASM/Rust/..