

2025 Activity Report

RESEARCH CENTRE: Inria Centre at Rennes University

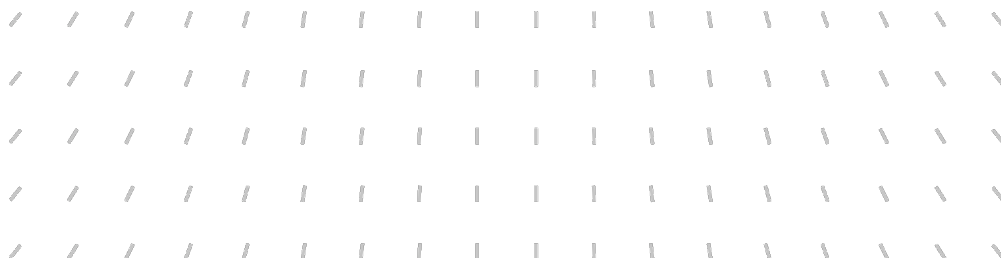
IN PARTNERSHIP WITH: Ecole Nationale Supérieure Mines-Télécom Atlantique
Bretagne Pays de la Loire, Nantes Université

Project-Team

GALLINETTE

Gallinette: developing a new generation of proof assistants

In collaboration with Laboratoire des Sciences du numérique de Nantes



Project-Team GALLINETTE

Creation of the Project-Team: 2018 June 01

Each year, Inria research teams publish an Activity Report presenting their work and results over the reporting period. These reports follow a common structure, with some optional sections depending on the specific team. They typically begin by outlining the overall objectives and research programme, including the main research themes, goals, and methodological approaches. They also describe the application domains targeted by the team, highlighting the scientific or societal contexts in which their work is situated. The reports then present the highlights of the year, covering major scientific achievements, software developments, or teaching contributions. When relevant, they include sections on software, platforms, and open data, detailing the tools developed and how they are shared. A substantial part is dedicated to new results, where scientific contributions are described in detail, often with subsections specifying participants and associated keywords. Finally, the Activity Report addresses funding, contracts, partnerships, and collaborations at various levels, from industrial agreements to international cooperations. It also covers dissemination and teaching activities, such as participation in scientific events, outreach, and supervision. The document concludes with a presentation of scientific production, including major publications and those produced during the year.

Keywords

Computer sciences and digital sciences

- A2.1.1. – Semantics of programming languages
- A2.1.2. – Imperative programming
- A2.1.3. – Object-oriented programming
- A2.1.4. – Functional programming
- A2.1.11. – Proof languages
- A2.2.3. – Memory management
- A4.5.3. – Program proof
- A7.2.3. – Interactive Theorem Proving
- A7.2.4. – Mechanized Formalization of Mathematics
- A8.4. – Computer Algebra

Other research topics and application domains

- B6.1. – Software industry

Contents

Project-Team GALLINETTE	1
1 Team members, visitors, external collaborators	5
2 Overall objectives	6
3 Research program	6
3.1 Scientific Context	6
3.2 Enhance the computational and logical power of proof assistants	8
3.2.1 Multiverse and Sort Polymorphism	8
3.2.2 Extensional Equalities	8
3.2.3 Adding Effects in Type Theory	8
3.3 Tools for Improving Proof Assistants	8
3.3.1 MetaProgramming in Rocq	8
3.3.2 Automatic Transport of Libraries	8
3.3.3 Logical Frameworks for Proof Assistants	9
3.4 Formal Verification and Semantics of Real World Programming Languages	9
3.4.1 Semantic foundations of resource management in programming languages	9
3.4.2 Interactive semantics	9
3.5 Formal Verification of Computer Assisted Certification	9
3.5.1 Certification of the Trusted Code Base of Rocq	9
3.5.2 Formally Verified Symbolic Computations	10
3.5.3 Erasure/Extraction of Certified Programs	10
4 Application domains	10
5 Social and environmental responsibility	10
6 Highlights of the year	11
7 Latest software developments, platforms, open data	11
7.1 Latest software developments	11
7.1.1 Ltac2	11
7.1.2 Equations	12
7.1.3 Math-Components	13
7.1.4 Math-comp-analysis	13
7.1.5 MetaRocq	14
7.1.6 Rocq	16
7.1.7 memprof-limits	16
7.1.8 ocaml-boxroot	17
7.1.9 LogRel-Coq	18
7.1.10 Trocq	18
8 New results	18
8.1 Type Theory	18
8.2 Proof Assistants	20
8.3 Logical Foundations of Programming Languages	21
8.4 Program Certifications and Formalisation of Mathematics	23
9 Bilateral contracts and grants with industry	24
9.1 Bilateral Contracts with Industry	24

10 Partnerships and cooperations	27
10.1 International initiatives	27
10.1.1 Associate Teams in the framework of an Inria International Lab or in the framework of an Inria International Program	27
10.2 International research visitors	28
10.2.1 Visits of international scientists	28
10.2.2 Visits to international teams	28
10.3 European initiatives	28
10.3.1 H2020 projects	28
10.4 National initiatives	29
11 Dissemination	30
11.1 Promoting scientific activities	30
11.1.1 Scientific events: organisation	30
11.1.2 Scientific events: selection	31
11.1.3 Journal	31
11.1.4 Invited talks	32
11.1.5 Scientific expertise	32
11.1.6 Research administration	32
11.2 Teaching - Supervision - Juries - Educational and pedagogical outreach	32
11.2.1 Supervision	32
11.2.2 Juries	33
11.2.3 Educational and pedagogical outreach	33
11.3 Popularization	33
11.3.1 Productions (articles, videos, podcasts, serious games, ...)	33
12 Scientific production	33
12.1 Major publications	33
12.2 Publications of the year	34
12.3 Cited publications	36

1 Team members, visitors, external collaborators

Research Scientists

- Nicolas Tabareau [Team leader, INRIA, Senior Researcher, HDR]
- Valentin Blot [INRIA, Researcher]
- Assia Mahboubi [INRIA, Senior Researcher, HDR]
- Kenji Maillard [INRIA, ISFP]
- Guillaume Munch-Maccagnoni [INRIA, Researcher]
- Pierre-Marie Pedrot [INRIA, Researcher]
- Matthieu Sozeau [INRIA, Researcher]

Faculty Members

- Rémi Douence [IMT ATLANTIQUE, Associate Professor, HDR]
- Guilhem Jaber [UNIV NANTES, Associate Professor]

Post-Doctoral Fellows

- Reinis Cirpons [INRIA, Post-Doctoral Fellow, from Nov 2025]
- Thiago Felicissimo Cesar [INRIA, Post-Doctoral Fellow]
- Axel Kerinec [INRIA, Post-Doctoral Fellow, until Aug 2025]
- Mateo Spadetto [UNIV NANTES, Post-Doctoral Fellow, from Apr 2025]

PhD Students

- Jonathan Arnoult [LS2N, from Sep 2025]
- Sidney Congard [INRIA]
- Tomas Diaz Troncoso [UNIV CHILI, from Sep 2025]
- Hamza Jaafar [INRIA, until Jan 2025]
- Thomas Lamiaux [UNIV NANTES]
- Yann Leray [UNIV NANTES]
- Josselin Poirret [UNIV NANTES]
- Leo Soudant [UNIV NANTES, from Oct 2025]
- Vojtech Stepancik [INRIA]
- Tomas Javier Vallejos Parada [INRIA]

Technical Staff

- Martin Baillon [INRIA, Engineer, until Jun 2025]
- Francois Michelon [INRIA, Engineer, from Nov 2025]

Interns and Apprentices

- Henrique Botelho Guerra [ULISBOA, from Mar 2025 until Jul 2025]
- Jean Caspar [ENS PARIS-SACLAY, Intern, from Mar 2025 until Aug 2025]
- Lucie Lahaye [ENS PARIS-SACLAY, Intern, from Oct 2025]
- Virgil Marionneau [INRIA, Intern, until Jan 2025]
- Johann Rosain [UNIV NANTES, Intern, from Feb 2025 until Jul 2025]
- Leo Soudant [ENS PARIS-SACLAY, Intern, from Mar 2025 until Aug 2025]

Administrative Assistant

- Anne-Claire Binetruy [INRIA]

2 Overall objectives

The EPI Gallinette aims at developing a new generation of proof assistants, with the belief that practical experiments must go in pair with foundational investigations:

- The goal is to advance proof assistants both as certified programming languages and mechanised logical systems. Advanced programming and mathematical paradigms must be integrated, notably dependent types and effects. The distinctive approach is to implement new programming and logical paradigms on top of Rocq by considering the latter as a target language for compilation.
- The aim of foundational investigations is to extend the boundaries of the Curry-Howard correspondence. It is seen both as providing foundations for programming languages and logic, and as a purveyor of techniques essential to the development of proof assistants. Under this perspective, the development of proof assistants is seen as a full-fledged experiment using the correspondence in every aspect: programming languages, type theory, proof theory, rewriting and algebra.

3 Research program

3.1 Scientific Context

Software quality is a requirement that is becoming more and more prevalent, by now far exceeding the traditional scope of embedded systems. The development of tools to construct software that respects a given specification is a major challenge in computer science. *Proof assistants* such as Rocq [46] provide a formal method whose central innovation is to produce *certified programs* by transforming the very activity of programming. Programming and proving are merged into a single development activity, informed by an elegant but rigid mathematical theory inspired by the correspondence between programming, logic and algebra: the *Curry-Howard correspondence*. For the certification of programs, this approach has shown its effectiveness in the development of important pieces of certified software such as the C compiler of the CompCert project [51]. The extracted CompCert compiler is reliable and efficient, running only 15% slower than GCC 4 at optimisation level 2 (`gcc -O2`), a level of optimisation that was considered before to be unreliable from critical applications such as embedded systems.

Proof assistants can also be used to *formalise mathematical theories*: they not only provide a means of representing mathematical theories in a form amenable to computer processing, but their internal logic provides a language for reasoning about such theories. In the last decade, proof assistants have been used to verify extremely large and complicated proofs of recent mathematical results, sometimes requiring either intensive computations [48, 50] or intricate combinations of a multitude of mathematical theories [49]. But formalised mathematics is more than just proof checking and proof assistants can help with the organisation of mathematical knowledge or even with the discovery of new constructions and proofs.

Unfortunately, the rigidity of the theory behind proof assistants restricts their expressiveness both as programming languages and as logical systems. For instance, a program extracted from Rocq only uses a purely functional subset of OCaml, leaving behind important means of expression such as side-effects and objects. Limitations also appear in the formalisation of advanced mathematics: proof assistants do not cope well with classical axioms such as excluded middle and choice which are sometimes used crucially. The fact of the matter is that the development of proof assistants cannot be dissociated from a reflection on the nature of programs and proofs coming from the Curry-Howard correspondence. In the EPC Gallinette, we propose to address several limitations of proof assistants by pushing the boundaries of this correspondence.

In the 1970's, the Curry-Howard correspondence was seen as a perfect match between functional programs, intuitionistic logic, and Cartesian closed categories. It received several generalisations over the decades, and now it is more widely understood as a fertile correspondence between computation, logic, and algebra.

Nowadays, the Curry-Howard correspondence is not perceived as a perfect match anymore, but rather as a collection of theories meant to explain similar structures at work in logic and computation, underpinned by mathematical abstractions. By relaxing the requirement of a perfect match between programs and proofs, and instead emphasising the common foundations of both, the insights of the Curry-Howard correspondence may be extended to domains for which the requirements of programming and mathematics may in fact be quite different.

Consider the following two major theories of the past decades, which were until recently thought to be irreconcilable:

- **(Martin-Löf) Type theory:** introduced by Martin-Löf in 1971, this formalism [52] is both a programming language and a logical system. The central ingredient is the use of *dependent types* to allow fine-grained invariants to be expressed in program types. In 1985, Coquand and Huet developed a similar system called the *calculus of constructions*, which served as logical foundation of the first implementation of Coq, now called the Rocq Prover. This kind of systems is still under active development, especially with the recent advent of homotopy type theory (HoTT) [56] that gives a new point of view on types and the notion of equality in type theory.
- **The theory of effects:** starting in the 1980's, Moggi [53] and Girard [47] put forward monads and co-monads as describing various compositional notions of computation. In this theory, programs can have side-effects (state, exceptions, input-output), logics can be non-intuitionistic (linear, classical), and different computational universes can interact (modal logics). Recently, the safe and automatic management of resources has also seen a coming of age (Rust, Modern C++) confirming the importance of linear logic for various programming concepts. It is now understood that the characteristic feature of the theory of effects is sensitivity to *evaluation order*, in contrast with type theory which is built around the assumption that evaluation order is irrelevant.

We now outline a series of scientific challenges aimed at understanding of type theory, effects, and their combination.

More precisely, three key axes of improvement have been identified:

- Making the notion of equality closer to what is usually assumed when doing proofs on black board, with a balance between irrelevant equality for simple structures and equality up-to equivalences for more complex ones (Section 3.2). Such a notion of equality should allow one to implement traditional model transformations that enhance the logical power of the proof assistant using distinct compilation phases.
- Advancing the foundations of effects within the Curry-Howard approach. The objective is to pave the way for the integration of effects in proof assistants and to prototype the corresponding implementation. This integration should allow for not only certified programming with effects, but also the expression of more powerful logics (Section 3.3).
- Making more programming features (notably, object polymorphism) available in proof assistants, in order to scale to practical-sized developments. The objective is to enable programming styles closer to common practices. One of the key challenges here is to leverage gradual typing to dependent programming (Section 3.4).

To validate the new paradigms, we propose in Section 3.5 three particular application fields in which members of the team already have a strong expertise: code refactoring, constraint programming and symbolic computation.

3.2 Enhance the computational and logical power of proof assistants

3.2.1 Multiverse and Sort Polymorphism

The experience of the team on various extensions of type theory (definitional proof irrelevant propositions, observational type theory, gradual type theory, opetopic type theory) begs naturally the question of the integration of these distinct flavours of type theory in a single type theory. At a theoretical level, we will investigate type theories with multiple universe hierarchies hosting theories with potentially incompatible principles able to express efficiently a variety of mathematical situations. At a practical level, we will develop a version of the Rocq Prover with multiple sorts, generalizing the existing situation where the sorts of types, propositions and definitionally proof-irrelevant propositions cohabit de facto. An important challenge in that direction is to design a sound mechanism of sort polymorphism to factor away the constructions common to multiple sorts and prevent the combinatorial explosion induced by a naive implementation. A somewhat related line of research is designing an efficient decision procedure for universe level constraints, following the work of [44].

3.2.2 Extensional Equalities

In the long quest towards a practical and extensional notion of equality, observational type theory, first introduced by [43] and further developed and studied in [55] and [54], represents an important milestone that should now be implemented in practice. We will pursue in parallel other extensionality principles to enhance the expressivity of type theories.

3.2.3 Adding Effects in Type Theory

The investigation of extensions of CIC with side effects, in particular that of exceptions and the addition of a case analysis operator on types, yields important insights to give sound models of the cast calculi behind gradual dependent types. We plan to go beyond these relatively simple extensions and consider other widely used side effects, for instance the addition of global state to a type theory. These type theories with a primitive support for effectful operations could provide a new approach to the verification of programs exhibiting side-effects. Extending our previous work on classical sequent calculus with dependent types, we will study the integration of classical axioms such as excluded middle and choice in rich type theory. One goal is to better integrate insights of (classical) proof theory in the state of art of type theory (or in an alternative approach thereof). We also aim to look at concrete issues met in formalized mathematics stemming from the classical/intuitionist divide.

3.3 Tools for Improving Proof Assistants

3.3.1 MetaProgramming in Rocq

The MetaRocq project currently provides bare-bones meta-programming facilities of quotation and denotation. We plan to improve this to provide a full-feature meta-programming facility, and explore the possibility to give strong specifications and verify our meta-programs. A prime example of this is the development of support for verified parametricity translations that can have many uses during formalization (elimination principles, automatic transport, etc.).

3.3.2 Automatic Transport of Libraries

We aim at pursuing the study of representation independence principles and the implementation of corresponding tools, so as to dramatically reduce the practical cost of library development. The mid-term expected outcome concerns the design of refinements libraries, which connect proof-oriented with computation-oriented data-structures, and better transport instruments for formalized mathematics, *e.g.*, automating reasoning modulo structure isomorphisms.

3.3.3 Logical Frameworks for Proof Assistants

The porting of the development of logical relations for MLTT of Abel *et al.* from Agda to Rocq paves the way to a much more modular library. We would like to extend this work by developing a generic framework for dependently-typed logical relations, and use it for a wide variety of new dependent type theories. The main goal is to establish strong metatheoretical properties: normalization, but also suitable forms of interoperability. Ultimately, we believe this framework could interface with the MetaRocq project.

3.4 Formal Verification and Semantics of Real World Programming Languages

3.4.1 Semantic foundations of resource management in programming languages

We will keep investigating the semantic foundations of features of systems programming languages from a mathematical point of view. Based on our earlier work [45] showing a link between resources and *ordered logic*, we will study resources management in the context of the formal theory of side-effects and linearity. Existing theorems will need to be generalized in many ways (extension of the notions of effect and resource modalities, handling of order, etc.). A link with linear logic will help make tighter connections between systems programming and “linear” approaches to program semantics (ownership, linear types, etc.). The notion of “borrowing” will be studied from the angle of linear logic, with possible applications to program verification. This study should also be extended to notions of fault tolerance (exception-safety and isolation) which might show links with modal logics. The anticipated outcome is an understanding of advanced notions in programming languages that better align with proven concepts from systems programming, compared to experimental type systems originating from pure theory, while providing clear distinctions between essential and accidental aspects of these real-world languages. As concrete experiments, we will keep researching ways to integrate systems programming concepts such as resources and fault tolerance in functional programming languages (notably OCaml and the OCaml-Rust interface).

3.4.2 Interactive semantics

We will continue our work on game semantics for programming languages, with the aim of studying interoperability and compilation between languages. Indeed, these semantics are particularly well suited to studying the interaction between a program and an environment written in different languages. We believe this approach will make it possible to overcome major open problems concerning interoperability between languages equipped with abstraction properties statically enforced by parametric polymorphism, and untyped languages where such abstractions properties are enforced dynamically. We will also continue studying the automation of reasoning on these semantics, along the lines of the CAVOC project. To do this, we want to apply abstract interpretation techniques, in particular the Abstracting Abstract Machine methodology, to automatically check accessibility properties on programs, such as unverified assertions.

As part of the CANofGAS project, we also plan to apply these interactive semantics to develop compositional cost models for programs. This would provide compositional reasoning on time and space complexity for higher-order programs.

3.5 Formal Verification of Computer Assisted Certification

3.5.1 Certification of the Trusted Code Base of Rocq

The MetaRocq project’s Achille’s heel is that it relies on an assumption of strong normalization for the calculus: there is ongoing work in the team on defining powerful logical relations in Rocq without relying on inductive-recursion, that gives hope that a strong-normalization model for a large fragment of MetaRocq can be constructed in the future. The main scientific obstacle is to specify the syntactic guard/productivity condition at the heart of termination checking in such a way that it can be reduced to an eliminator-based definition of (co-)inductive types, which is how they are usually modelled. We anticipate difficulties with nested and indexed inductive types, which might be currently accepted by Rocq but difficult to emulate with eliminators. However, this can only lead to a better understanding of the theory. As part of the ReCiProg project, we also plan to establish formal links between the validation criteria derived from circular proofs and this guard condition.

3.5.2 Formally Verified Symbolic Computations

The benefits of formally verified symbolic computations is twofold: increase the trust in computer-produced mathematics and expand the automation available for users of proof assistants. The main challenge is to enable the formal verification of efficient programs, whose correctness proofs involve sophisticated mathematical ingredients rather than subtle memory or parallelism issues. This involves in particular scaling up the automatic transport of libraries, as well as the formal verification of existing imperative code from computer algebra systems (typically written in C).

3.5.3 Erasure/Extraction of Certified Programs

The MetaRocq erasure pipeline, targeting C or OCaml, provides a guarantee that the evaluation of the compiled program gives a semantically correct result. However, in general extracted programs are linked to larger programs of the target language, where we lose guarantees of correctness in most non-trivial cases of interoperability. We are hence interested in developing techniques to show interoperability results between code that is extracted through our certified compilation pipeline and external code, *e.g.*, in OCaml or C. In [6], we developed a complete verified extraction pipeline from Rocq to OCaml. The goal for the future is to scale this work to allow more scenarios of interoperability with effectful target programs, using a formal semantics for the target language. In particular, we should be able to soundly interpret the primitive constructs that are already part of Rocq, which are fixed-width integers, IEEE-754 floating point numbers and applicative arrays. There is a point of synergy here with the previous goal of enabling the development of efficient, formally verified symbolic computation.

4 Application domains

Programming

- Correct and certified software engineering through the development and the advancement of Rocq (e.g. gradualizing type theory, MetaRocq) and practical experiments for its application.
- More general contributions to programming languages: theoretical works advancing semantic techniques (e.g. deciding equivalence between programs, abstract syntaxes and rewriting, models of effects and resources), and practical works for functional programming (e.g. related to OCaml and Rust).

Foundations of mathematics

- Formalisation of mathematics
- Contributions to mathematical logic: type theory (e.g. dependent types and univalence), proof theory (e.g. constructive classical logic), categorical logic (e.g. higher algebra, models of focusing and linear logic)

5 Social and environmental responsibility

The team actively participates in open science initiatives, sharing tools and results to avoid redundant efforts, further reducing the collective computational and material costs associated with formal verification research. It spends efforts on the usability of these tools.

The team promotes a socially healthy governance for the tools it develops, taking into account the voices of its stakeholders, in accordance with the principles outlined in the ACM Code of Ethics. For instance, this includes the organisation of development of the Rocq prover which is in various ways open to stakeholders. Another example is that we have been actively involved in the renaming process of the Coq proof assistant, which is now called the Rocq Prover. This renaming initiative originated from a request by the Coq user community. The wider issue of social responsibility and impact of the governance of software platforms, in particular towards the stakeholders, remains an under-explored question in computer science. The team has

been a driving force behind the creation of the [Undone Computer Science](#) interdisciplinary conference and journal series, where such topics can be explored.

The research activities of the Gallinette team at Inria have a relatively low environmental footprint compared to other computationally intensive fields, as their work primarily involves theoretical development, formal proofs, and software tool design.

6 Highlights of the year

- [The Rocq Prover](#) has been officially renamed (formerly known as the Coq proof assistant) and released in March 2025.
- Assia Mahboubi gave an [interview](#) at France Culture.
- Assia Mahboubi and Pierre-Marie Pédrot gave invited presentations at Collège de France under Thierry Coquand’s annual chair.

Awards

- Yann Leray got a distinguished paper award at POPL’26.

7 Latest software developments, platforms, open data

7.1 Latest software developments

7.1.1 Ltac2

Keywords: Coq, Proof assistant

Functional Description: Ltac2 is a member of the ML family of languages, in the sense that it is an effectful call-by-value functional language, with static typing à la Hindley-Milner. It is commonly accepted that ML constitutes a sweet spot in PL design, as it is relatively expressive while not being either too lax (unlike dynamic typing) nor too strict (unlike, say, dependent types).

The main goal of Ltac2 is to serve as a meta-language for Coq. As such, it naturally fits in the ML lineage, just as the historical ML was designed as the tactic language for the LCF prover. It can also be seen as a general-purpose language, by simply forgetting about the Coq-specific features.

Sticking to a standard ML type system can be considered somewhat weak for a meta-language designed to manipulate Coq terms. In particular, there is no way to statically guarantee that a Coq term resulting from an Ltac2 computation will be well-typed. This is actually a design choice, motivated by backward compatibility with Ltac1. Instead, well-typedness is deferred to dynamic checks, allowing many primitive functions to fail whenever they are provided with an ill-typed term.

The language is naturally effectful as it manipulates the global state of the proof engine. This allows to think of proof-modifying primitives as effects in a straightforward way. Semantically, proof manipulation lives in a monad, which allows to ensure that Ltac2 satisfies the same equations as a generic ML with unspecified effects would do, e.g. function reduction is substitution by a value.

Contact: Pierre-Marie Pedrot

Participants: Pierre-Marie Pedrot, Gaëtan Gilbert

7.1.2 Equations

Keywords: Coq, Dependent Pattern-Matching, Proof assistant, Functional programming, Rocq

Scientific Description: Equations is a tool designed to help with the definition of programs in the setting of dependent type theory, as implemented in the Rocq prover. Equations provides a syntax for defining programs by dependent pattern-matching and well-founded recursion and compiles them down to the core type theory of Rocq, using the primitive eliminators for inductive types, accessibility and equality. In addition to the definitions of programs, it also automatically derives useful reasoning principles in the form of propositional equations describing the functions, and an elimination principle for calls to this function. It realizes this using a purely definitional translation of high-level definitions to core terms, without changing the core calculus in any way, or using axioms.

The main features of Equations include:

Dependent pattern-matching in the style of Agda/Epigram, with inaccessible patterns, with and where clauses. The use of the K axiom or a proof of K is configurable, and it is able to solve unification problems without resorting to the K rule if not necessary.

Support for well-founded and mutual recursion using measure/well-foundedness annotations, even on indexed inductive types, using an automatic derivation of the subterm relation for inductive families.

Support for mutual and nested structural recursion using with and where auxilliary definitions, allowing to factor multiple uses of the same nested fixpoint definition. It proves the expected elimination principles for mutual and nested definitions.

Automatic generation of the defining equations as rewrite rules for every definition.

Automatic generation of the unfolding lemma for well-founded definitions (requiring only functional extensionality).

Automatic derivation of the graph of the function and its elimination principle. In case the automation fails to prove these principles, the user is asked to provide a proof.

A new dependent elimination tactic based on the same splitting tree compilation scheme that can advantageously replace dependent destruction and sometimes inversion as well. The as clause of dependent elimination allows to specify exactly the patterns and naming of new variables needed for an elimination.

A set of Derive commands for automatic derivation of constructions from an inductive type: its signature, no-confusion property, well-founded subterm relation and decidable equality proof, if applicable.

Functional Description: Equations is a function definition plugin for Rocq (supporting Coq 8.18 to 8.20, and Rocq version 9.0 and 9.1 with special support for the HoTT library), that allows the definition of functions by dependent pattern-matching and well-founded, mutual or nested structural recursion and compiles them into core terms. It automatically derives the clauses equations, the graph of the function and its associated elimination principle.

Equations is based on a simplification engine for the dependent equalities appearing in dependent eliminations that is also usable as a separate tactic, providing an axiom-free variant of dependent destruction.

Release Contributions: This is a minor update of Equations, now compatible with The Rocq Prover 9.0 and 9.1. The main changes are fixes in the funelim tactic and simplification engine to avoid trying to simplify unrelated hypotheses (those under "block" markers). which sometimes led to slowing down the tactic. Also includes performance improvements by @ppedrot.

See <https://github.com/mattam82/Coq-Equations/releases> for details

News of the Year: The plugin has been updated to adapt to the renaming from Coq to Rocq, to support the now standard dune build system and has been refactored significantly with the goal of integrating its code in Rocq core in the future.

URL: <http://mattam82.github.io/Coq-Equations/>

Publications: [hal-01671777](#), [hal-01248807](#), [inria-00628862](#)

Contact: Matthieu Sozeau

Participant: Matthieu Sozeau

7.1.3 Math-Components

Name: Mathematical Components library

Keywords: Proof assistant, Coq, Formalisation

Functional Description: The Mathematical Components library is a set of Coq libraries that cover the prerequisites for the mechanization of the proof of the Odd Order Theorem.

Release Contributions: This release is compatible with Coq 8.20 and Rocq 9.0 and 9.1.

The main changes are:

multiple refinements of the algebraic hierarchies (this may lead to a few small breakages, in which case we recommend using Rocq 9.1 that should minimize such breakages in the future) package `mathcomp-ssreflect` got split into `mathcomp-boot` and `mathcomp-order`, if you weren't using the `order.v` file, replacing your `mathcomp-ssreflect` dependency by `mathcomp-boot` may save you some compilation time

See the `CHANGELOG.md` file for more details.

URL: <https://math-comp.github.io/>

Contact: Assia Mahboubi

Participants: Assia Mahboubi, Cyril Cohen, Enrico Tassi, Georges Gonthier, Laurent Théry, Yves Bertot, 9 anonymous participants

7.1.4 Math-comp-analysis

Name: Mathematical Components Analysis

Keywords: Proof assistant, Coq, Formalisation

Functional Description: This library adds definitions and theorems to the Math-components library for real numbers and their mathematical structures.

Release Contributions: Compatible with Coq 8.20, Rocq 9.0 and 9.1 and MathComp 2.4.0–2.5.0.

URL: <https://github.com/math-comp/analysis>

Publications: [hal-02463336](#), [hal-03917948](#), [hal-01719918](#)

Contact: Cyril Cohen

Participant: 12 anonymous participants

Partners: Ecole Polytechnique, AIST Tsukuba, Onera

7.1.5 MetaRocq

Keyword: Rocq

Scientific Description: The MetaRocq project aims to provide a certified meta-programming environment in Rocq. It builds on Template-Rocq, a plugin for Rocq originally implemented by Malecha (Extensible proof engineering in intensional type theory, Harvard University, 2014), which provided a reifier for Rocq terms and global declarations, as represented in the Rocq kernel, as well as a denotation command. Recently, it was used in the CertiRocq certified compiler project (Anand et al., in: RocqPL, Paris, France, 2017), as its front-end language, to derive parametricity properties (Anand and Morrisett, in: RocqPL'18, Los Angeles, CA, USA, 2018). However, the syntax lacked semantics, be it typing semantics or operational semantics, which should reflect, as formal specifications in Rocq, the semantics of Rocq's type theory itself. The tool was also rather bare bones, providing only rudimentary quoting and unquoting commands. MetaRocq generalizes it to handle the entire polymorphic calculus of cumulative inductive constructions, as implemented by Rocq, including the kernel's declaration structures for definitions and inductives, and implement a monad for general manipulation of Rocq's logical environment. The MetaRocq framework allows Rocq users to define many kinds of general purpose plugins, whose correctness can be readily proved in the system itself, and that can be run efficiently after extraction. Examples of implemented plugins include a parametricity translation and a certified extraction to call-by-value lambda-calculus. The meta-theory of Rocq itself is verified in MetaRocq along with verified conversion, type-checking and erasure procedures providing highly trustable alternatives to the procedures in Rocq's OCaml kernel. MetaRocq is hence a foundation for the development of higher-level certified tools on top of Rocq's kernel. A meta-programming and proving framework for Rocq.

MetaRocq is made of 4 main components:

- The entry point of the project is the Template-Rocq quoting and unquoting library for Rocq which allows quotation and denotation of terms between three variants of the Rocq AST: the OCaml one used by Rocq's kernel, the Rocq one defined in MetaRocq and the one defined by the extraction of the MetaRocq AST, allowing to extract OCaml plugins from Rocq implementations.
- The PCUIC component is a full formalization of Rocq's typing and reduction rules, along with proofs of important metatheoretic properties: weakening, substitution, validity, subject reduction and principality. The PCUIC calculus differs slightly from the Template-Rocq one and verified translations between the two are provided.
- The checker component contains verified implementations of weak-head reduction, conversion and type inference for the PCUIC calculus, along with a verified checker for Rocq theories.
- The erasure component contains a verified implementation of erasure/extraction from PCUIC to untyped (call-by-value) lambda calculus extended with a dummy value for erased terms.

Functional Description: MetaRocq is a framework containing a formalization and verified implementation of Rocq's kernel in Rocq along with a verified erasure procedure. It provides tools for manipulating Rocq terms and developing certified plugins (i.e. translations, compilers or tactics) in Rocq.

Release Contributions: This release adapts MetaRocq 1.4 to Rocq 9.1. See the previous release notes for this version's main changes.

Fix test suite overriding COQEXTRAFLAGS by @SkySkimmer in #1141 Adapt w.r.t. rocq-prover/rocq#20165. by @ppedrot in #1144 Adapt to rocq-prover/rocq#20178 (UContext.to_context returns qvar set not quality set) by @SkySkimmer in #1145 Improve evar handling in tmUnquote/tmUnquoteTyped by @MathisBD in #1113 Adapt w.r.t. rocq-prover/rocq#20278. by @ppedrot in #1147 Adapt to rocq-prover/rocq#20360 (pr_universe_context_set renamed) by @SkySkimmer in #1150 CI stop testing ocaml 4.09 (equations needs >= 4.10 now) by @SkySkimmer in #1142 Update CI for main by @SkySkimmer in #1162 Adapt for rocq-prover/rocq#20415 by @yannl35133 in #1161 Fix the proof using a hole in the guard checker by @yannl35133 in #1160 Adapt to rocq-prover/rocq#20391 (Ltac redef isn't cancelled by importing original module) by @SkySkimmer in #1159 Take relevance into account for typing by @yannl35133 in #1163 Nix CI for Rocq-9.0 by @4ever2 in #1164

Nix CI for main branch by @4ever2 in #1167 Merge 9.0 in main by @mattam82 in #1170 Bump cachix/cachix-action from 15 to 16 by @dependabot[bot] in #1168 Bump cachix/install-nix-action from 30 to 31 by @dependabot[bot] in #1169 Don't overwrite COQEXTRAFLAGS in test suite by @SkySkimmer in #1171 Adapt to rocq-prover/rocq#20397 (removal of sort families) by @jrosain in #1172 Eval erase by @mattam82 in #1173 adapt to rocq-prover/rocq#20496 (changed bound_names) by @LeoAlexElouan in #1175 Adapt to rocq-prover/rocq#20603 (moved Lib.section_segment APIs) by @SkySkimmer in #1178 Adapt to rocq-prover/rocq#20605 (cleaned up projection flags) by @SkySkimmer in #1177 translations/minihott remove incorrect priority by @SkySkimmer in #1183 adapt to rocq-prover/rocq#20707 by @gares in #1190 Adapt to rocq-prover/rocq#20839 (classifier sees ätts) by @gares in #1191 New Contributors @LeoAlexElouan made their first contribution in #1175

The preprint "Correct and Complete Type Checking and Certified Erasure for Coq, in Coq" presents the development of the sound and complete type checker based on bidirectional typing, the meta-theoretical results (subject reduction, standardization, canonicity and consistency) and the verified erasure procedure of this version of MetaRocq.

MetaRocq integrates Template-Rocq, a reification and denotation plugin for Rocq terms and global declarations, a Template monad for metaprogramming (including the ability to extract these meta-programs to OCaml for efficiency), a formalisation of Rocq's calculus PCUIC in Rocq, a relatively efficient, sound and complete type checker for PCUIC, a verified type and proof erasure procedure from PCUIC to untyped lambda calculus and a quotation library. MetaRocq provides a low-level interface to develop certified plugins like translations, compilers or tactics in Rocq itself.

You can install MetaRocq directly from sources or using opam install rocq-metarocq. This release will be included in an upcoming Rocq Platform.

The current release includes several subpackages, which can be compiled and installed separately if desired:

the utils library contains extensions to the standard library (notably for reasoning with All/All-n type-valued predicates) (in directory utils, and as rocq-metarocq-utils). the common libraries of basic definitions for the abstract syntax trees shared by multiple languages (common, rocq-metarocq-common) the Template-Rocq quoting library and plugin (template-rocq / rocq-metarocq-template) a formalisation of meta-theoretical properties of PCUIC, the calculus underlying Rocq (pcuic / rocq-metarocq-pcuic) a verified equivalence between Template-Rocq and PCUIC typing (in directory template-pcuic and as rocq-metarocq-template-pcuic) a total verified type-checker for Rocq (safechecker / rocq-metarocq-safechecker), usable inside Rocq. a plugin interfacing with the extracted type-checker in OCaml, providing the MetaRocq SafeCheck <term> command (safechecker-plugin, rocq-metarocq-safechecker-plugin) a verified type and proof erasure function for Rocq (erasure / rocq-metarocq-erasure), usable inside Rocq. a plugin interfacing with the extracted erasure pipeline in OCaml, providing the MetaRocq Erase <term> command (erasure-plugin, rocq-metarocq-erasure-plugin) a quoting library, allowing the quotation of terms and type derivations along with associated data structures as ASTs/terms (quotation / rocq-metarocq-quotation). a set of example translations from Type Theory to Type Theory (translation/ rocq-metarocq-translations). A good place to start are the files demo.v, safechecker_test.v, erasure_test.v in the test-suite directory.

This version of MetaRocq was developed by Yannick Forster, Jason Gross, Yann Leray, Matthieu Sozeau and Nicolas Tabareau with contributions from Yishuai Li. You are welcome to contribute by opening issues and PRs. A MetaRocq Zulip stream is also available.

The MetaRocq Team

News of the Year: This year's work was mostly concentrated on improving the erasure/extraction pipeline, integrating and unifying parts of the CertiRocq and coq-malfunction projects to allow producing code that is amenable to compilation by a functional programming language compiler. The new pipeline supports the proof of new interoperability theorems, improving the formal guarantees provided by MetaRocq's extraction.

URL: <https://metarocq.github.io>

Publications: [hal-04077552](#), [hal-04329663](#), [hal-03516619](#), [hal-02901011](#), [hal-02380196](#), [hal-02167423](#), [hal-01809681](#)

Contact: Matthieu Sozeau

Participants: Abhishek Anand, Danil Annenkov, Meven Lennon-Bertrand, Jakob Botsch Nielsen, Simon Boulrier, Cyril Cohen, Yannick Forster, Kenji Maillard, Gregory Malecha, Matthieu Sozeau, Nicolas Tabareau, Theo Winterhalter

Partners: Concordium Blockchain Research Center, Saarland University

7.1.6 Rocq

Name: The Rocq Prover

Keyword: Proof assistant

Scientific Description: Rocq is an interactive proof assistant based on the Calculus of (Co-)Inductive Constructions, extended with universe polymorphism. This type theory features inductive and co-inductive families, an impredicative sort and a hierarchy of predicative universes, making it a very expressive logic. The calculus allows to formalize both general mathematics and computer programs, ranging from theories of finite structures to abstract algebra and categories to programming language metatheory and compiler verification. Rocq is organised as a (relatively small) kernel including efficient conversion tests on which are built a set of higher-level layers: a powerful proof engine and unification algorithm, various tactics/decision procedures, a transactional document model and, at the very top an integrated development environment (IDE).

Functional Description: The Rocq Prover provides both a dependently-typed functional programming language and a logical formalism, which, altogether, support the formalisation of mathematical theories and the specification and certification of properties of programs. The Rocq Prover also provides a large and extensible set of automatic or semi-automatic proof methods. Rocq's programs are extractible to OCaml, Haskell, Scheme, ...

Release Contributions: An overview of the new features and changes, along with the full list of contributors is available at <https://rocq-prover.org/releases/9.1.0>

News of the Year: The Rocq Prover was renamed at the beginning of 2025 (see <https://rocq-prover.org/about#Name> for details on the name change).

Its current version is Rocq 9.1, which integrates changes to the Rocq kernel, performance improvements, and a few new features. See the detailed changes at <https://rocq-prover.org/releases/9.1.0> for an overview, along with the full list of contributors.

URL: <http://rocq-prover.org/>

Contact: Matthieu Sozeau

Participants: Yves Bertot, Frédéric Besson, Tej Chajed, Cyril Cohen, Pierre Corbineau, Pierre Courtieu, Maxime Dénès, Jim Fehrle, Julien Forest, Emilio Jesús Gallego Arias, Gaëtan Gilbert, Georges Gonthier, Benjamin Grégoire, Jason Gross, Hugo Herbelin, Vincent Laporte, Olivier Laurent, Assia Mahboubi, Kenji Maillard, Erik Martin Dorel, Guillaume Melquiond, Pierre-Marie Pedrot, Clément Pit-Claudiel, Kazuhiko Sakaguchi, Vincent Semeria, Michael Soegtrop, Arnaud Spiwack, Matthieu Sozeau, Enrico Tassi, Laurent Théry, Anton Trunov, Li-Yao Xia, Theo Zimmermann

7.1.7 memprof-limits

Keyword: Library

Scientific Description: Memprof-limits is an implementation for OCaml of per-thread global memory limits, and per-thread allocation limits à la Haskell, and CPU-bound thread cancellation compatible with multiple threads.

Memprof-limits interrupts the execution by raising an asynchronous exception: an exception that can arise at almost any location in the program. It is provided resource-safety features and a guide on how to recover from asynchronous exceptions and other unexpected exceptions, summarising for the first time for the OCaml community the practical knowledge acquired by the Rocq prover as well as in other programming languages.

Memprof-limits is probabilistic, as it is based on the statistical memory profiler memprof. It is provided with a statistical analysis that the user that provides guarantees about the enforcement of limits.

Functional Description: Memprof-limits is an implementation of (per-thread) global memory limits, (per-thread) allocation limits, and cancellation of CPU-bound threads, for OCaml. Memprof-limits interrupts a computation by raising an exception asynchronously and offers features to recover from them such as interrupt-safe resources.

It is provided with an extensive documentation with examples which explains what must be done to ensure one recovers from an interrupt. This documentation summarises for the first time the experience acquired in OCaml in the Rocq prover, as well as in other situations in other programming languages.

Release Contributions: Support for OCaml 5 and multicore OCaml programs

URL: <https://gitlab.com/gadmm/memprof-limits>

Publication: hal-03517592

Contact: Guillaume Munch

Participant: Guillaume Munch

7.1.8 ocaml-boxroot

Keywords: Interoperability, Library, Ocaml, Rust

Scientific Description: Boxroot is an implementation of roots for the OCaml GC based on performant concurrent allocation techniques. These roots are designed to support a calling convention to interface between Rust and OCaml code that reconciles the latter's foreign function interface with the idioms from the former.

Functional Description: Boxroot implements fast movable roots for OCaml in C, Rust, and C++. A root is a data type which contains an OCaml value, and interfaces with the OCaml GC to ensure that this value and its transitive children are kept alive while the root exists. This can be used to write programs in other languages that interface with programs written in OCaml.

Release Contributions: Support for OCaml ≥ 5.3 . Add support for including the headers from C++, and add a C++ smart pointer Boxroot based on `unique_ptr` in `cpp/boxroot.h`. Simplified constraints for calling `boxroot_setup` with OCaml 4 thanks to a new linking method.

URL: <https://gitlab.com/ocaml-rust/ocaml-boxroot>

Publication: hal-03910313

Contact: Guillaume Munch

Participants: Guillaume Munch, an anonymous participant

7.1.9 LogRel-Coq

Keyword: Proof assistant

Functional Description: This Coq library develop the metatheory of Martin-Löf Type Theory with a universe and some inductive types in order to establish consistency, normalisation, canonicity and decidability of a core theory close to that of Coq.

URL: <https://github.com/CoqHott/logrel-coq>

Publications: [hal-04379245](#), [hal-04214008](#), [hal-04160858](#)

Contact: Kenji Maillard

Participant: 6 anonymous participants

7.1.10 Trocq

Keywords: Proof synthesis, Proof transfer, Coq, Elpi, Logic programming, Parametricity, Univalence

Functional Description: Trocq is a modular parametricity plugin for Rocq (prototype), aimed at proof transfer. It translates a user goal into a related variant using the target data structures, along with a rich parametricity witness from which a justifying substitution function can be extracted.

The plugin features a hierarchy of parametricity witness types, ranging from structure-less relations to a novel formulation of type equivalence. This gathers several pre-existing parametricity translations (including univalent parametricity and CoqEAL) in a unified framework.

This modular translation performs fine-grained analysis and generates sufficiently rich witnesses to preprocess the goal — without always requiring full type equivalence. It enables proof transfer with the power of univalent parametricity while avoiding the univalence axiom when unnecessary.

The translation is implemented in Rocq-Elpi and features transparent, readable code aligned with a sequent-style theoretical presentation.

Release Contributions: Bug fixes, syncing with most recent Rocq and Elpi

URL: <https://github.com/coq-community/trocq>

Publication: [hal-04177913](#)

Contact: Cyril Cohen

Participants: Cyril Cohen, Enzo Crance, Assia Mahboubi

Partner: Mitsubishi Electric R&D Centre Europe, France

8 New results

8.1 Type Theory

Participants: Martin Baillon, Gaëtan Gilbert, Yann Leray, Assia Mahboubi, Kenji Maillard, Guillaume Munch, Josselin Poiret, Pierre-Marie Pédrot, Matthieu Sozeau, Nicolas Tabareau.

A Zoo of Continuity Properties in Constructive Type Theory Continuity principles stating that all functions are continuous play a central role in some schools of constructive mathematics. However, there are different ways to formalise the property of being continuous in constructive foundations. In [25], we analyse these continuity properties from the perspective of constructive reverse mathematics. We work in constructive type theory, which can be seen as a minimal foundation for constructive reverse mathematics. We treat continuity of functions $F : (Q \rightarrow A) \rightarrow R$, i.e. with question type Q , answer type A , and result type R . Concretely, we discuss continuity defined via moduli, making the relevant list $L : LQ$ of questions explicit, dialogue trees, making the question answer process explicit as inductive tree, and tree functions, making the question answer process explicit as function. We prove equivalences where possible and isolate necessary and sufficient axioms for equivalence proofs. Many of the results we discuss are already present in the works of Hancock, Pattinson, Ghani, Kawai, Fujiwara, Brede, Herbelin, Escardó, and others. Our main contribution is their formulation over a uniform foundation, the observation that no choice axioms are necessary, the generalisation to arbitrary types from natural numbers where possible, and a mechanisation in the Coq proof assistant.

In Cantor Space No One Can Hear You Stream We revisit in [26] the famous notion of sheaves through the lens of type theory and side-effects. Using the language of MLTT, we show that they inductively approximate idealized functional objects as decision trees, realizing a generalized form of continuity. We materialize this intuition in $MLTT_f$, a case-study sheaf extension of MLTT with a Cohen real and leverage it to show uniform continuity of all MLTT functionals of type $(N \rightarrow B) \rightarrow N$. The latter results were mechanized in Rocq.

All Your Base Are Belong to Us: Sort Polymorphism for Proof Assistants Proof assistants based on dependent type theory, such as Coq, Lean and Agda, use different universes to classify types, typically combining a predicative hierarchy of universes for computationally-relevant types, and an impredicative universe of proof-irrelevant propositions. In general, a universe is characterized by its sort, such as Type or Prop, and its level, in the case of a predicative sort. Recent research has also highlighted the potential of introducing more sorts in the type theory of the proof assistant as a structuring means to address the coexistence of different logical or computational principles, such as univalence, exceptions, or definitional proof irrelevance. This diversity raises concrete and subtle issues from both theoretical and practical perspectives. In particular, in order to avoid duplicating definitions to inhabit all (combinations of) universes, some sort of polymorphism is needed. Universe level polymorphism is well-known and effective to deal with hierarchies, but the handling of polymorphism between sorts is currently ad hoc and limited in all major proof assistants, hampering reuse and extensibility. In [22], we develop sort polymorphism and its metatheory, studying in particular monomorphization, large elimination, and parametricity. Sort polymorphism is a natural solution that effectively addresses the limitations of current approaches and prepares the ground for future multi-sorted type theories.

Observational Equality Meets CIC The notion of equality is at the heart of dependent type theory, as it plays a fundamental role in program specifications and mathematical reasoning. In mainstream proof assistants such as Agda, Lean, and Coq, equality is usually defined using Martin-Löf's identity type, an elegant and simple approach that has stood the test of time since the 1970s. However, this definition also comes with serious downsides: the intensional nature of Martin-Löf's identity type means that it is impractical for reasoning about functions and predicates, and it is impossible to define quotient types. Recently, observational equality has garnered attention as an alternative method for encoding equality, particularly in proof assistants supporting definitionally proof-irrelevant propositions. However, it has yet to be integrated in any of the three proof assistants mentioned above, as it is not fully compatible with another important feature of type theory: indexed inductive types. In [23], we propose a systematic approach to reconcile observational equality with indexed inductive types, using a type coercion operator that computes on reflexive identity proofs. The second contribution of this article is a formal proof that this additional computation rule can be integrated to the system without compromising the decidability of conversion. Finally, we provide an implementation of our observational equality in an extension of Coq. This extension is based on the recently introduced rewrite rules and provides new extensionality principles while remaining fully backward-compatible.

"Upon This Quote I Will Build My Church Thesis" The internal Church thesis (CT) is a logical principle stating that one can associate to any function $f : \mathbb{N} \rightarrow \mathbb{N}$ a concrete code, in some Turing-complete language, that computes f . While the compatibility of CT in simpler systems has been long known, its compatibility with dependent type theory is still an open question. In [21], we answer this question positively. We define "MLTT", a type theory extending MLTT with quote operators in which CT is derivable. We furthermore prove that "MLTT" is consistent, strongly normalizing and enjoys canonicity using a rather standard logical relation model. All the results in this paper have been mechanized in Coq.

AdapTT: Functoriality for Dependent Type Casts The ability to cast values between related types is a leitmotiv of many flavors of dependent type theory, such as observational type theories, subtyping, or cast calculi for gradual typing. These casts all exhibit a common structural behavior that boils down to the pervasive functoriality of type formers. In [15], we propose and extensively study a type theory, called AdapTT, which makes systematic and precise this idea of functorial type formers, with respect to an abstract notion of adapters relating types. Leveraging descriptions for functorial inductive types in AdapTT, we derive structural laws for type casts on general inductive type formers.

8.2 Proof Assistants

Participants: Tomas Diaz, Thomas Lamaiaux, Yann Leray, Assia Mahboubi, Kenji Maillard, Pierre-Marie Pédro, Matthieu Sozeau, Nicolas Tabareau.

Trocq: Proof Transfer for Free, Beyond Equivalence and Univalence In [17], we present Trocq, a new proof transfer framework for dependent type theory. Trocq is based on a novel formulation of type equivalence, used to generalize the univalent parametricity translation. This framework takes care of avoiding dependency on the axiom of univalence when possible, and may be used with more relations than just equivalences. We have implemented a corresponding plugin for the Rocq/Coq interactive theorem prover, in the Coq-Elpi meta-language.

Bounded Sort Polymorphism with Elimination Constraints Proof assistants based on dependent type theory—such as Agda, Lean, and Rocq—employ different universes to classify types, typically combining a predicative tower for computationally relevant types with a possibly impredicative universe for proof-irrelevant propositions. Several other universes with specific logical and computational principles have been explored in the literature. In general, a universe is characterized by its sort (e.g., Type, Prop, or SProp) and, in the predicative case, by its level. To improve modularity and better avoid code duplication, sort polymorphism has recently been introduced and integrated in the Rocq prover.

In [24], we observe that, due to its unbounded formulation, sort polymorphism is currently insufficiently expressive to abstract over valid definitions with a single polymorphic schema. Indeed, to ensure soundness of a multi-sorted type theory, the interaction between different sorts must be carefully controlled, as exemplified by the forbidden elimination of irrelevant terms to produce relevant ones. As a result, generic functions that eliminate values of inductive types from one sort to another cannot be made polymorphic; dually, polymorphic records that encapsulate attributes of different sorts cannot be defined. This lack of expressiveness also breaks the possibility to infer principal types, which is highly desirable for both metatheoretical and practical reasons. To address these issues, we extend sort polymorphism with bounds that reflect the required elimination constraints on sort variables. We present the metatheory of bounded sort polymorphism, paying particular attention to the consistency of the resulting constraint graph. We implement bounded sort polymorphism in Rocq and illustrate its benefits through concrete examples. Bounded sort polymorphism with elimination constraints is a natural and general solution that effectively addresses current limitations and fosters the development of, and practical experimentation with, multi-sorted type theories.

Encode the Cake and Eat it Too Proof assistants based on dependent type theory such as Agda, Lean and Rocq identify objects up to computation during proof checking. This takes away some of the proof burden from the user and even provides a way to get very efficient automation. Recently, Agda and Rocq

have been extended to support user-defined computation. While they already prove very useful, user-defined computation rules are global: once they are added, they are here to stay. Importing a development that makes use of those rules then means relying on them, whether we want it or not, which can lead to unwanted incompatibilities. In [19], we design LRTT, a type theory with support for local abstraction over user-defined computation rules. This takes the form of a prenex quantification at the definition level. This quantification is supplemented with the possibility to provide one or several instantiations that verify the equations definitionally. We show that a procedure inlining definitions abstracting over definitional equality is possible, in the style of monomorphisation or of C++ templates. In the process we get a conservativity result over more conventional Martin-Löf type theories. There are several benefits to such a system. First, it provides encapsulation for user-defined computation rules, which is important to avoid unwanted bad interactions and limits the scope in which invariants of type theory (such as termination, confluence, type preservation and consistency) are broken. Second, abstraction lets users factorise code that crucially relies on definitional equality, as well as hide implementation details that are irrelevant in some settings. Finally, it gives a way to encode certain features without paying the price of the encoding. We showcase such examples in a prototype implementation as an extension of the Rocq Prover. Additionally, all the results in this have been formalised in Rocq.

Babel-formal: Translation of Proofs between Lean and Rocq In [35], we investigate using proof terms (the low-level representation of formal proofs) as a pivot language for translating proof scripts between proof assistants and across tactic sets. Unlike direct proof translation, this approach does not require an aligned training corpus; it only needs aligned context at inference time so that both systems elaborate comparable terms. We compare two strategies: (1) direct script-to-script translation with an off-the-shelf LLM (GPT-5), and (2) proof term translation, where an LLM turns a proof term into a proof script in the target language. We build a small benchmark of aligned sources (14 files, 117 lemmas) across Lean and Rocq, and train models to map Lean and Rocq proof terms back to their native proof scripts. Our experiments show that proof term translation works for cross-assistant translation and for translation between tactic sets. It is complementary to using a SoTA off-the-shelf LLM for direct proof script translation (combining both performs best), scales easily in terms of training data, and handles tactic-set translation better (e.g., vanilla Rocq \rightarrow SSReflect).

Nested Inductive Types Inductive types are a fundamental abstraction mechanism in type theory and proof assistants, supporting the definition of data structures and rich specifications. Nested inductive types extend this mechanism by allowing constructors to use parametric types instantiated with the type being defined (e.g., lists or trees of the type to be defined). They are widely used in large verification projects -including CompCert, Iris, Verinum, and MetaRocq -to express complex, structured specifications. Despite this widespread use, the treatment of nested inductive types in both Lean and Rocq is unsatisfactory. Lean rejects many practical definitions while Rocq accepts definitions for which no usable elimination principle can be defined. Neither system provides reliable automatic generation of elimination principles. As a result, developers must define custom eliminators by hand, leading to fragility, duplication, and significant proof engineering overhead. In [38], we introduce a novel validity criterion for nested inductive types that guarantees that they can be elaborated into well-formed mutual inductive types. Under this criterion, the elimination principle for the original nested definition is provably equivalent to that of its elaborated mutual form. Our condition strictly generalizes Lean’s current check while ruling out exactly the problematic cases accepted in Rocq. Using this foundation, we give a systematic method for automatically generating correct elimination principles for nested inductive types, and we provide an implementation integrated into Rocq, along with an implementation plan for Lean.

8.3 Logical Foundations of Programming Languages

Participants: Jean Caspar, Sidney Congard, Tomas Diaz, Rémi Douence, Thomas Lami-aux, Guillaume Munch, Nicolas Tabareau.

S4 modal sequent calculus as intermediate logic and intermediate language In [34], we advocate for the idea that continuation-based intermediate languages correspond to intermediate logics. The goal of

intermediate languages is to serve as a basis for compiler intermediate representations, allowing to represent expressive program transformations for optimisation and compilation, while preserving the properties that make programs compilable efficiently in the first place, such as the “stackability” of continuations. Intermediate logics are logics between intuitionistic and classical logic in terms of provability. Second-class continuations used in CPS-based intermediate languages correspond to a classical modal logic S4 with the added restriction that implications may only return modal types. This indeed corresponds to an intermediate logic, owing to the Gödel-McKinsey-Tarski theorem which states the intuitionistic nature of the modal fragment of S4. We introduce a three-kinded polarised sequent calculus for S4, together with an operational machine model that separates a heap from a stack. With this model we study a stackability property for the modal fragment of S4.

Classical notions of computation and the Hasegawa-Thielecke theorem In the spirit of the Curry-Howard correspondence between proofs and programs, we define and study a syntax and semantics for classical logic equipped with a computationally involutive negation, using a polarised effect calculus, the linear classical L-calculus. A main challenge in designing a denotational semantics for the calculus is to accommodate both call-by-value and call-by-name evaluation strategies, which leads to a failure of associativity of composition. In order to tackle this issue, we define in [20] a notion of adjunction between graph morphisms on non-associative categories, which we use to formulate polarized and non-associative notions of symmetric monoidal closed duploid and of dialogue duploid. We show that they provide a direct style counterpart to adjunction models: linear effect adjunctions for the (linear) call-by-push-value calculus and dialogue chiralities for linear continuations, respectively. In particular, we show that the syntax of the linear classical L-calculus can be interpreted in any dialogue duploid, and that it defines in fact a syntactic dialogue duploid. As an application, we establish, by semantic as well as syntactic means, the Hasegawa-Thielecke theorem, which states that the notions of central map and of thinkable map coincide in any dialogue duploid (in particular, for any double negation monad on a symmetric monoidal category).

Linear Effects, Exceptions, and Resource Safety In [29], we analyse the problem of combining linearity, effects, and exceptions, in abstract models of programming languages, as the issue of providing some kind of strength for a monad $T(_ \oplus E)$ in a linear setting. We consider in particular for T the *allocation monad*, which we introduce to model and study resource-safety properties. We apply these results to a series of two linear effectful calculi for which we establish their resource-safety properties. The first calculus is a linear call-by-push-value language with two allocation effects new and delete. The resource-safety properties follow from the linear (and even ordered) character of the typing rules. We then explain how to integrate exceptions on top of linearity and effects by adjoining default destruction actions to types, as inspired by C++/Rust destructors. We see destructors as objects $\delta : A \rightarrow TI$ in the slice category over TI . This construction gives rise to a second calculus, an *affine* ordered call-by-push-value language with exceptions and destructors, in which the weakening rule performs a side-effect. As in C++/Rust, a “move” operation is necessary to allow random-order release of resources, as opposed to last-in-first-out order. Moving resources is modelled as an exchange rule that performs a side-effect.

An abstract, certified account of operational game semantics Operational game semantics (OGS) is a method for interpreting programs as strategies in suitable games, or more precisely as labelled transition systems over suitable games, in the sense of Levy and Staton. Such an interpretation is called sound when, for any two given programs, weak bisimilarity of associated strategies entails contextual equivalence. OGS has been applied to a variety of languages, with rather tedious soundness proofs. In [27], we contribute to the unification and mechanisation of OGS. Indeed, we propose an abstract notion of language with evaluator, for which we construct a generic OGS interpretation, which we prove sound. Our framework covers a variety of simply-typed and untyped lambda-calculi with various evaluation strategies. These calculi notably feature recursive definitions, first-class continuations, and a wide variety of datatypes. All constructions and proofs are entirely mechanised in the Rocq proof assistant.

2-Functoriality of Initial Semantics, and Applications Initial semantics aims to model inductive structures and their properties, and to provide them with recursion principles respecting these properties. An ubiquitous example is the fold operator for lists. In [16], we are concerned with initial semantics that model languages

with variable binding and their substitution structure, and that provide substitution-safe recursion principles. There are different approaches to implementing languages with variable binding depending on the choice of representation for contexts and free variables, such as unscoped syntax, or well-scoped syntax with finite or infinite contexts. Abstractly, each approach corresponds to choosing a different monoidal category to model contexts and binding, each choice yielding a different notion of "model" for the same abstract specification (or "signature"). In this work, we provide tools to compare and relate the models obtained from a signature for different choices of monoidal category. We do so by showing that initial semantics naturally has a 2-categorical structure when parametrized by the monoidal category modeling contexts. We thus can relate models obtained from different choices of monoidal categories provided the monoidal categories themselves are related. In particular, we use our results to relate the models of the different implementation - de Bruijn vs locally nameless, finite vs infinite contexts -, and to provide a generalized recursion principle for simply-typed syntax.

Robust Dynamic Embedding for Gradual Typing Gradual typing has long been advocated as a means to bridge the gap between static and dynamic typing disciplines, enabling a range of use cases such as the gradual migration of existing dynamically typed code to more statically typed code, as well as making advanced static typing disciplines more accessible. To assess whether a given gradual language can effectively support these use cases, several formal properties have been proposed, most notably the refined criteria set forth by Siek et al. One criterion asserts that the dynamic extreme of the spectrum should be expressible in the gradual language, formalized by the existence of an adequate embedding from the corresponding dynamic language.

In [18], we observe that the existing dynamic embedding criterion does not capture the desirable property of being able to ascribe embedded code to a static type that it semantically satisfies, and ensure reliable interactions with other components within the gradual language. Specifically, we introduce the notion of robustness for gradual terms, meaning that when interacting with any gradual context, runtime failures that may occur ought to be caused by the context, not by the robust term itself. We then formulate the robust dynamic embedding criterion: if a dynamic component semantically satisfies a given static type, then its embedding subsequently ascribed to that static type should be a robust term. We demonstrate that robust dynamic embedding is not implied by any existing metatheoretical property from the literature, and is not upheld by various existing gradual languages. We show that robust dynamic embedding is achievable with a gradualized simply-typed language. All the results are formalized in the Rocq proof assistant. This novel criterion complements the set of criteria for gradual languages and opens several venues for further exploration, in particular for typing disciplines that enforce rich semantic properties.

Operational Game Semantics for Generative Algebraic Effects and Handlers In [32], we present a sound operational game semantics model (w.r.t. contextual equivalence) of a typed language with algebraic effects and handlers and dynamic generation of effect instances. We exhibit the interactive aspect of effect propagation, and to address it precisely, we identify the adequate granularity of the operational semantics and the decomposition of normal forms into their interactive and their observational part. To account for this additional form of interaction, we extend the standard pure interaction interface of game semantics consisting of questions and answers with effectful moves that involve the propagation of effects and the yielding of delimited continuations. Finally, we extend the well-bracketed constraint on the behavior of the environment, from the use of continuations as is standard in game semantics, to fragments of captured delimited continuations.

8.4 Program Certifications and Formalisation of Mathematics

Participants: Reinis Cirpons, Assia Mahboubi, Thiago Felicissimo, Kenji Maillard, Nicolas Tabareau.

Certifying the Decidability of the Word Problem in Monoids at Large While the word problem for monoids is undecidable in general, having a decision procedure for some finitely presented monoid of interest has numerous applications. In [28], we present a toolbox for the Rocq proof assistant that can be used to verify

the decidability of the word problem for a given monoid and, in some cases, to produce the corresponding decision procedure. As this verification can be computationally intensive, the toolbox heavily relies on proofs by reflection guided by an external oracle. This approach has been successfully used on several large presentations from the literature, as well as on a database of one million 1-relation monoids.

The huge size of this database forced some unusual considerations onto the Rocq formalization, so that the formal proofs could be checked in a reasonable amount of time.

Incremental Certified Programming Certified programming, as carried out in proof assistants and dependently-typed programming languages, ensures that a software meets its requirements by supporting the definition of both specifications and proofs. However, proofs easily break with partial definitions and incremental changes because specifications are not designed to account for the intermediate incomplete states of programs. In [30], we advocate for proper support for incremental certified programming by analyzing its objectives and inherent challenges, and propose a formal framework for achieving incremental certified programming in a principled manner. The key idea is to define appropriate notions of completion refinement and completeness to capture incrementality, and to systematically produce specifications that are valid at every stage of development while preserving the intent of the original statements. We provide a prototype implementation in the Rocq Prover, called IncRease, which exploits typeclasses for automation and extensibility, and is independent of any specific mechanism used to handle incompleteness. We illustrate its use with both an incremental textbook formalization of the simply-typed λ -calculus, and a more complex case study of incremental certified programming for an existing dead-code elimination optimization pass of the CompCert project. We show that the approach is compatible with randomized property-based testing as provided by QuickChick. Finally we study how to combine incremental certified programming with deductive synthesis, using a novel incrementality-friendly adaptation of the Fiat library. This work provides theoretical and practical foundations towards systematic support for incremental certified programming, highlighting challenges and perspectives for future developments.

Geometric Reasoning in Lean: from Algebraic Structures to Presheaves Algebraic theories such as semigroups, monoids, rings or heyting algebras can swiftly be described in dependent type theories, such as that of Lean, by packing together sorts, operations and equations. Abstractly, these theories should be interpretable in many different settings beyond that of bare types. Leveraging semantics of geometric logic in presheaf categories, i.e. categories of "varying sets", we explore in [36] the potential of interpreting such algebraic theories in these extended settings.

Sort-Based Confluence Criteria for Non-Left-Linear Higher-Order Rewriting Powerful confluence criteria for higher-order rewriting exist for left-linear systems, even in the presence of critical pairs and non-termination. On the other hand, confluence criteria that allow for mixing non-termination and non-left-linearity are either extremely limited or hardly usable in practice. In [31], we study confluence criteria which explore sort information to make proving higher-order confluence possible, even in the presence of non-termination and non-left-linearity. We give many interesting examples of systems covered by our results, including a (confluent) variant of Klop's counterexample, and a calculus issuing from a dependent type theory with cumulative universes.

9 Bilateral contracts and grants with industry

9.1 Bilateral Contracts with Industry

Contract Extension. The bilateral contracts listed below ended in 2023, but the associated overhead continued for an additional 12 months.

CoqExtra

Participants: Pierre-Marie Pédro, Matthieu Sozeau, Nicolas Tabareau, Yannick Forster, Pierre Giraud, Kazuhiko Sakaguchi.

Title: A Formally Verified Extraction Mechanism using Precise Type Specifications

Duration: 2020 - 2023 (extended to 2024)

Coordinator: Nicolas Tabareau

Partners:

- Inria
- Nomadic Labs

Inria contact: Nicolas Tabareau

Summary: The extraction mechanism from Coq to OCaml can be seen as a compilation phase, from a functional language with dependent types to a functional language with a weaker type system. It is very useful to be able to run and link critical pieces of code that have been certified with the rest of a software system. For instance, for Tezos, it is important to certify the Michelson language for smart contracts and then to be able to extract it to OCaml so that it interacts with the rest of the code that has been developed. Unfortunately, the current extraction mechanism of Coq suffers from two major flaws that prevent extraction from being used in complex situations—and in particular for the Michelson language. First, the extraction mechanism does not make use of new features of OCaml type system, such as Generalized Abstract Data Types (GADTs). This prevents code using indexed inductive types (Coq’s generalization of GADTs) to be extracted to code using GADTs. Therefore, in the case of Michelson, the extracted code does not correspond at all to the seminal implementation of Michelson in OCaml as it jeopardizes its type specification. The second flaw comes from the fact that extraction sometimes produces ill-typed pieces of code (even if it uses Obj.magic to cheat the type system), for instance when the arity of a function depends on some value. Therefore, the extracted program fails to type-checked in OCaml and cannot be used.

Expected Impact: This project proposes to remedy to the situation so that the formalized Michelson implementation can be extracted to OCaml in a satisfactory and certified way. But this project is also of great interest outside Nomadic Labs as it will allow Coq users to use a better extraction mechanism and, on a longer term, it will allow OCaml developers to prove their OCaml programs using a formal semantics of (a fragment of) OCaml defined in Coq.

CIFRE PhD grant, funded by Mitsubishi Electric R&D Centre Europe (MERCE)

Participants: Assia Mahboubi, Enzo Crance.

Title: Automated theorem proving and dependent types: automated reasoning for interactive proof assistants

Duration: 2020 - 2023 (extended to 2024)

Coordinator: Denis Cousineau (MERCE), Assia Mahboubi (Inria)

Partners:

- Inria
- Mitsubishi Electric R&D Centre Europe (MERCE)

Inria contact: Assia Mahboubi

Summary: The aim of this project is to vastly improve the automated reasoning skills of proof assistants based on dependent type theory, and in particular of the Coq proof assistant. Automated provers, like SAT solvers or SMT solvers, can provide fast decision answers on large formulas, typically quantifier-free first order statements generated by code analysis instruments like static analyzers. Modern provers are moreover able to produce additional data, called certificates, which contain enough

information for an a posteriori verification of their results, e.g., using a formal proof. In this project, we would like to use this feature to expand the automation available to users of proof assistants. The main motivation here is thus to increase the class of goals that can be proved formally and automatically by the interactive proof assistant, rather than to work on the formal verification of specific albeit large decision problems. In this case, the central research problem is to bridge the gap between the rich specification language of the proof assistant, and the restricted fragment handled by the automated prover. This project will thus investigate the design, and the implementation, of the corresponding translation phase. This translation transforms a logical statement possibly featuring user-defined data structures and higher-order quantifications, into another statement, logically stronger, that can be sent to the automated prover. We thus aim at a triple objective: expressivity, extensibility and efficiency. This grant is funding the PhD of Enzo Crance.

Expected Impact: Enhancing the automated reasoning skills of proof assistants based on dependent type theory will be key to their wider usage in industry. As of today, they are considered too expensive to be used in the large outside of specific niches.

OCaml-Rust

Participants: Guillaume Munch-Maccagnoni.

Title: OCaml/Rust bindings

Duration: 2021-2023 (extended to 2024)

Coordinator: Gabriel Scherer (INRIA Saclay, EPI Partout)

Participants:

- Guillaume Munch-Maccagnoni (INRIA Rennes, EPI Gallinette),
- Jacques-Henri Jourdan (CNRS, LRI)

Partners: Inria, Nomadic Labs

Inria contact: Gabriel Scherer

Summary: We often want to write programs with components in several different programming languages. Interfacing two languages typically goes through low-level, unsafe interfaces. The OCaml/Rust project studies safer interfaces between OCaml and Rust.

Expected Impact: We investigated safe low-level representations of OCaml values on the Rust side, representing GC ownership, and developed a calling convention that reconciles the OCaml FFI idioms with Rust idioms. We also developed Boxroot, a new API to register values with the OCaml GC, for use when interfacing with Rust (and other programming languages) and possibly when writing concurrent programs. This resulted in novel techniques which can benefit other pairs of languages in the future. These works are now integrated in the ocaml-rs interface between OCaml and Rust used in the industry.

CAVOC

Participants: Guilhem Jaber, Hamza Jaafar.

Title: Compositional Automated Verification for OCaml

Duration: 2021-2024

Coordinator: Guilhem Jaber

Partners:

- Inria
- Nomadic Labs

Inria contact: Guilhem Jaber

Summary: This project aims to develop a *sound and precise static analyzer* for OCaml, that can catch large classes of bugs represented by uncaught exceptions. It will deal with both user-defined exceptions, and built-in ones used to represent *error behaviors*, like the ones triggered by `failwith`, `assert`, or a match failure. Via “assert-failure” detection, it will thus be able to check that invariants annotated by users hold. The analyzer will reason *compositionally* on programs, in order to analyze them at the granularity of a function or of a module. It will be *sound* in a strong way: if an OCaml module is considered to be correct by the analyzer, then one will have the guarantee that no OCaml code interacting with this module can trigger uncaught exceptions coming from the code of this module. In order to be *precise*, it will take into account the abstraction properties provided by the type system and the module system of the language: local values, abstracted definition of types, parametric polymorphism. The goal being that most of the interactions taken into account correspond to typeable OCaml code (that do not use unsafe features of the Obj Module, or the Foreign Function Interface to some external code).

Expected Impact: Being modular the analyzer should be able to automatically check the absence of bugs of a large base of code written in the considered subset of OCaml. This subset will include most of the codebase developed by Nomadic Labs, which is an heavy user of GADT, for example to enforce subject reduction in the implementation of Michelson. We would then be able to get a higher degree of trust in its codebase, and possibly to find undetected bugs in it. The impact of this project could be large for the OCaml ecosystem in general, where automated analysis of programs to check soundness properties of the code could be really useful (for example for the Coq proof assistant, whose full analysis would be nonetheless too ambitious for this project).

10 Partnerships and cooperations

10.1 International initiatives

10.1.1 Associate Teams in the framework of an Inria International Lab or in the framework of an Inria International Program

GRAPA

Title: Gradual Proof Assistant

Duration: 2023 -> 2025

Coordinator: Eric Tanter (etanter@dcc.uchile.cl)

Partners:

- Universidad de Chile (Chili)

Inria contact: Nicolas Tabareau

Summary: The GRAPA project aims at improving the usability and ease-of-adoption of proof assistants such as Coq via gradual typing. Indeed, while certified programming with proof assistants is undoubtedly an extremely appealing and powerful approach towards the systematic construction of correct, robust and secure software systems, it suffers from a proportionally large complexity problem that prevents wide adoption beyond very specialized crowds. Gradual dependent types aim to lower the entry cost to proofs assistants, providing the flexibility of a dynamically typed language together with a transition path to statically, formally verified code. The project explores the design space of gradual dependent types and their practical integration in the Coq proof assistant.

10.2 International research visitors

10.2.1 Visits of international scientists

Inria International Chair

Participants: Nicolas Tabareau, Éric Tanter.

Éric Tanter received a Inria International Chair for 5 years.

10.2.2 Visits to international teams

Research stays abroad

Gullaume Munch-Maccagnoni

Visited institution: School of Computer Science, University of Birmingham

Country: United Kingdom

Dates: 22nd to 28th June 2025

Context of the visit: Collaboration with Anupam Das and Abhishek De

Mobility program/type of mobility: Research stay

10.3 European initiatives

10.3.1 H2020 projects

FRESCO [FRESCO project on cordis.europa.eu](https://cordis.europa.eu/project/FRESCO)

Title: Fast and Reliable Symbolic Computation

Duration: From November 1, 2021 to October 31, 2027

Partners:

- INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET AUTOMATIQUE (INRIA), France

Inria contact: Assia Mahboubi

Coordinator:

Summary: The use of computers for formulating conjectures, but also for substantiating proof steps, pervades mathematics, even in its most abstract fields. Most computer proofs are produced by symbolic computations, using computer algebra systems. Sadly, these systems suffer from severe, intrinsic flaws, key to their amazing efficiency, but preventing any flavor of post-hoc verification.

But can computer algebra become reliable while remaining fast? Bringing a positive answer to this question represents an outstanding scientific challenge per se, which this project aims at solving.

Our starting point is that interactive theorem provers are the best tools for representing mathematics in silico. But we intend to disrupt their architecture, shaped by decades of applications in computer science, so as to dramatically enrich their programming features, while remaining compatible with their logical foundations.

We will then design a novel generation of mathematical software, based on the firm grounds of modern programming language theory. This environment will feature a new, high-level, performance-oriented programming language, devised for writing efficient and correct code easily, and for serving the

frontline of research in computational mathematics. Users will have access to fast implementations, and to powerful proving technologies for verifying any component à la carte, with high productivity. Logic- and computer-based formal proofs will prevent run-time errors, and incorrect mathematical semantics.

We will maintain a close, continuous collaboration with interested high-profile mathematicians, on the verification of cutting-edge research results, today beyond the reach of formal proofs. We ambition to empower mathematical journals to install high-quality artifact evaluation, when peer-reviewing falls short of assessing computer proofs. This project will eventually impact the use of formal methods in engineering, in areas like cryptography or signal-processing.

10.4 National initiatives

ReCiProg

Participants: Guilhem Jaber, Guillaume Munch-Maccagnoni, Pierre-Marie Pédrot, Matthieu Sozeau.

Title: Reasoning on Circular proofs for Programming

Program: ANR AAPG2021,

Type: PRC, CES 48

Duration: Jan 2022 - Jan 2025

Coordinator: UMR CNRS - IRIF - Université de Paris

Local Contact: Guilhem Jaber

Summary: ReCiProg is a collaborative project (Lyon-Marseille-Nantes-Paris) aiming at extending the proofs-as-programs correspondence (also known as Curry-Howard correspondence) to recursive programs and circular proofs for logic and type systems using induction and coinduction. The project will contribute both to the necessary theoretical foundations of circular proofs and to the software development allowing to enhance the use of coinductive types and coinductive reasoning in the Coq proof assistant: such coinductive types present, in the current state of the art serious defects that the project will aim at solving.

CANofGAS

Participants: Guilhem Jaber.

Title: Cost Analysis of Game Semantics

Program: Inria Exploratory Action,

Duration: Sep 2022 - Sep 2025

Coordinator: Beniamino Accattoli (CR Inria, LIX, PARTOUT Team) and Guilhem Jaber (MCF, LS2N, Gallinette Team)

Local Contact: Guilhem Jaber

Summary: CANofGAS aims at capturing the time and space cost of the evaluation of higher-order programs at the semantic level. The directions we plan to explore are using the advances in reasonable cost models to develop a cost-based understanding of game semantics. In particular, we aim at modelling the efficient call-by-need evaluation scheme, at work for instance in the Haskell language and in the Coq proof assistant.

Rocqola

Participants: Gaëtan Gilbert, Guillaume Munch-Maccagnoni, Pierre-Marie Pédrot.

Title: Rocq optimisé par son langage

Program: ANR AAPG2025

Duration: Oct 2025 - Oct 2029

Coordinator: Fabrice Le Fessant (OCamlPro), Pierre-Marie Pédrot and Gabriel Scherer (INRIA Paris, PICUBE)

Local Contact: Pierre-Marie Pédrot

Summary: This proposal explores research at the interplay between the Rocq proof assistant and the OCaml programming language. The goal of Rocqola is to reunite experts of the Rocq implementation and experts of the OCaml compiler and runtime, to research improvements to the Rocq proof assistant that require language expertise, and improvements to OCaml that benefit Rocq and other symbolic manipulation systems. This area of confluence is fertile ground for new programming-languages and proof-assistant research.

11 Dissemination

Participants: Valentin Blot, Rémi Douence, Guilhème Jaber, Assia Mahboubi, Kenji Maillard, Guillaume Munch, Pierre-Marie Pédrot, Matthieu Sozeau, Nicolas Tabareau.

11.1 Promoting scientific activities

11.1.1 Scientific events: organisation

General chair, scientific chair

- Guillaume Munch-Maccagnoni is general co-chair and organiser of the 2nd conference on Undone Science in Computer Science (23-25 March 2026 in Luxembourg).

Steering committee

- Nicolas Tabareau is a member of the steering committee of the ACM Certified Programs and Proofs (CPP) conference.
- Assia Mahboubi is a member of the steering committee of the international conference on Interactive Theorem Proving (ITP).
- Guillaume Munch-Maccagnoni is a member of the steering committee of Undone Computer Science.
- Valentin Blot is a member of the steering committee of the Logic in Computer Science (LICS) international conference.
- Assia Mahboubi has co-organized, with Nicolas Brisebarre (Cnrs, ENS de Lyon) the closing meeting of the ANR Nuscap in Nantes in June 2025.
- Assia Mahboubi has co-organized a workshop in the honor of Georges Gonthier, with Enrico Tassi (Inria, Sophia Antipolis Méditerranée).

- Guilhem Jaber has co-organized the École de Printemps d’Informatique Théorique (EPIT 2025) at Aussois, on (Co)inductive & circular reasoning applied to programming, formal proofs and software verification.
- Matthieu Sozeau is a member of the steering committee of the Rocq workshops: Rocqshop collocated with ITP/FLoC since 20 years and RocqPL collocated with POPL since 10 years.

11.1.2 Scientific events: selection

Chair of conference program committees

- Nicolas Tabareau has served as co-chair of the ACM Certified Programs and Proofs (CPP) conference 2026
- Guilhem Jaber has served as co-chair of the HOPE’25 workshop (part of ICFP/SPLASH 2025 conference)

Member of the conference program committees

- Pierre-Marie Pédrot was a member of the TLLA’25 PC.
- Nicolas Tabareau was a member of the LICS’25 PC.
- Guilhem Jaber was a member of LICS’25 PC.
- Assia Mahboubi was a member of the CSL’25 PC, FSCD’25 PC and FOSSACS’25 PC.
- Matthieu Sozeau was a member of the POPL’26 and JFLA’26 PCs.
- Kenji Maillard was a member of the POPL’26 PC.

Reviewer

- Pierre-Marie Pédrot reviewed for LICS’25, FSCD’25, FoSSaCS’25, POPL’25.
- Assia Mahboubi served as external reviewer for the CPP’26 and TACAS’26 conferences.
- Guilhem Jaber served as external reviewer for POPL’25, FoSSaCS’25 and ICFP’25.
- Guillaume Munch-Maccagnoni served as external reviewer for LICS’25 and FSCD’25.
- Valentin Blot served as external reviewer for the CPP’26 conference.

11.1.3 Journal

Member of the editorial boards

- Assia Mahboubi serves on the editorial boards of the Journal of Automated Reasoning, Logical Methods in Computer Science and Annals of Formalized Mathematics.
- Guillaume Munch-Maccagnoni serves as guest editor for the special issue of Philosophia Scientiæ on “Undone Computer Science”.

Reviewer - reviewing activities

- Pierre-Marie Pédrot and Assia Mahboubi reviewed for the JACM.
- Guilhem Jaber reviewed for the journals Science of Computer Programming and Logical Methods in Computer Science.
- Valentin Blot reviewed for the journal of the ACM (JACM).
- Matthieu Sozeau reviewed for the Journal of Functional Programming (JFP) and the Journal of Automated Reasoning (JAR).

11.1.4 Invited talks

Pierre-Marie Pédrot gave an invited talk at the Collège de France for the colloquium "Formalisation des mathématiques et types dépendants", an invited talk at the conference "Synthetic mathematics, logic-affine computation and efficient proof systems" at CIRM, and an invited talk at the "International Workshop on Programs from Proofs" in Bath.

Assia Mahboubi gave an invited talk at the Collège de France for the colloquium "Formalisation des mathématiques et types dépendants". She was an invited (remote) speaker at the international conference CICM'2025. She has given a colloquium talk in Marseilles at the fédération de recherche en mathématiques de Méditerranée (Frunam) and two invited talks for students of the Ecole polytechnique and at the Ecole Normale Supérieure de Paris.

Guillaume Munch-Maccagnoni has given an invited talk at the 10-year anniversary event of "Codes Sources" organised by LIP6, IRILL, Cnam and CNRS.

11.1.5 Scientific expertise

Pierre-Marie Pédrot gave a lecture about the foundations of the Rocq proof assistant in front of various representatives of the Spanish police forces at the Basque Police School in Vitoria. Rocq is indeed used to formalize a piece of software that checks the compliance of truck tachographs with the European law, developed by the FormalVindications company.

11.1.6 Research administration

- Assia Mahboubi is an elected member of the Commission d'Évaluation Inria and member of the conseil de laboratoire of the Laboratoire des Sciences du Numérique (LS2N).

11.2 Teaching - Supervision - Juries - Educational and pedagogical outreach

11.2.1 Supervision

- Guilhem Jaber has co-supervised with Tom Hirschowitz (DR CNRS, LAMA) and Yannick Zakowski (CR Inria, Cambium) the PhD of Peio Borthelle, who has defended in March 2025.
- Assia Mahboubi is supervising the PhD of Léo Soudant, Jonathan Arnoult, Vojtech Stepancik and Tomas Vallejos Parada.
- Assia Mahboubi is supervising the long internship of Lucie Lahaye, as part of her 4th year at ENS de Lyon.
- Pierre-Marie Pédrot is supervising the PhD of Léo Soudant, and has also supervised his internship for his 4th year at ENS Paris-Saclay.
- Guillaume Munch-Maccagnoni is co-supervising with Paul-André Melliès (DR CNRS, IRIF) the PhD of Éléonore Mangel.
- Guillaume Munch-Maccagnoni is co-supervising with Rémi Douence the PhD of Sidney Congard.
- Guillaume Munch-Maccagnoni has supervised the M2 internship of Jean Caspar.
- Valentin Blot is co-supervising with Catherine Dubois (professor, ENSIIE) the PhD of Amélie Ledein.
- Matthieu Sozeau and Nicolas Tabareau are co-supervising with Yannick Forster (CR INRIA, Cambium) the PhD of Thomas Lamiaux.
- Matthieu Sozeau and Nicolas Tabareau are co-supervising with Théo Winterhalter (CR INRIA, Deducteam) the PhD of Yann Leray.
- Kenji Maillard and Nicolas Tabareau are co-supervising the PhD of Josselin Poiret.
- Matthieu Sozeau has supervised the M2 internship of Johann Rosain (ENS Lyon, 4 month).

- Guilhem Jaber and Nicolas Tabareau are co-supervising the PhD of Hamza Jaafar.
- Nicolas Tabareau is co-supervising with Éric Tanter (Full Prof at UChile) the PhD of Tomas Diaz Troncoso.

11.2.2 Juries

- Assia Mahboubi has served as reviewer for the PhD of Luc Chabassier and for the HDR of Pierre Roux.
- Assia Mahboubi has served as president for the PhD of Quentin Canu, Emile Oléon, David Julien and Vincent Kowalski.
- Assia Mahboubi has served in the PhD jury of Emilie Grienenberger.
- Assia Mahboubi has served in the selection process of the COGENT ITN network.
- Guillaume Munch-Maccagnoni has served in the PhD jury of Hector Suzanne.

11.2.3 Educational and pedagogical outreach

- Assia Mahboubi has lectured at Master level in the Mastermath program in fundamental mathematics in Amsterdam.
- Assia Mahboubi has given a course at the Marktoberdorf international summer school.

11.3 Popularization

11.3.1 Productions (articles, videos, podcasts, serious games, ...)

- Pierre-Marie Pédrot gave an interview at the Type Theory Forall podcast hosted by Pedro Abreu.
- Assia Mahboubi has been interviewed in the program La Science CQFD on the French national radio France Culture.
- Assia Mahboubi has been interviewed for an article published in Science & Vie.
- Assia Mahboubi has written a chapter in the book "Le Calcul à Découvert", published by the Editions du Cnrs.

12 Scientific production

12.1 Major publications

- [1] R. Affeldt, C. Cohen, M. Kerjean, A. Mahboubi, D. Rouhling and K. Sakaguchi. ‘Competing inheritance paths in dependent type theory: a case study in functional analysis’. In: *IJCAR 2020 - International Joint Conference on Automated Reasoning*. Paris, France, June 2020, pp. 1–19. URL: <https://hal.inria.fr/hal-02463336>.
- [2] L. Birkedal, T. Dinsdale-Young, A. Guéneau, G. Jaber, K. Svendsen and N. Tzevelekos. ‘Theorems for free from separation logic specifications’. In: *Proceedings of the ACM on Programming Languages* 5.ICFP (22nd Aug. 2021), pp. 1–29. DOI: [10.1145/3473586](https://doi.org/10.1145/3473586). URL: <https://hal.archives-ouvertes.fr/hal-03510684>.
- [3] J. Cockx, N. Tabareau and T. Winterhalter. ‘The Taming of the Rew: A Type Theory with Computational Assumptions’. In: *Proceedings of the ACM on Programming Languages*. POPL 2021 (2021). DOI: [10.1145/3434341](https://doi.org/10.1145/3434341). URL: <https://hal.archives-ouvertes.fr/hal-02901011>.
- [4] S. Congard, G. Munch-Maccagnoni and R. Douence. ‘Linear Effects, Exceptions, and Resource Safety: A Curry-Howard Correspondence for Destructors’. In: *LNCS. ESOP 2026 - 35th European Symposium on Programming*. Turin, Italy: Springer, 2026. DOI: [10.48550/arXiv.2510.23517](https://doi.org/10.48550/arXiv.2510.23517). URL: <https://inria.hal.science/hal-05430763>.

- [5] E. Finster, A. Allieux and M. Sozeau. ‘Types are internal infinity-groupoids’. In: LICS 2021. Rome, Italy, 21st June 2021. URL: <https://hal.inria.fr/hal-03133144>.
- [6] Y. Forster, M. Sozeau and N. Tabareau. ‘Verified Extraction from Coq to OCaml’. In: *Proceedings of the ACM on Programming Languages* 8.PLDI (20th June 2024), pp. 52–75. DOI: [10.1145/3656379](https://doi.org/10.1145/3656379). URL: <https://inria.hal.science/hal-04329663> (cit. on p. 10).
- [7] G. Jaber. ‘SyTeCi: Automating Contextual Equivalence for Higher-Order Programs with References’. In: *Proceedings of the ACM on Programming Languages* 28 (2020), pp. 1–28. DOI: [10.1145/3371127](https://doi.org/10.1145/3371127). URL: <https://hal.archives-ouvertes.fr/hal-02388621>.
- [8] É. Mangel, P.-A. Melliès and G. Munch-Maccagnoni. ‘Classical notions of computation and the Hasegawa-Thielecke theorem’. In: *Proceedings of the ACM on Programming Languages* 10.POPL (8th Jan. 2026), p. 73. DOI: [10.1145/3776715](https://doi.org/10.1145/3776715). URL: <https://hal.science/hal-05006242>.
- [9] P.-M. Pédrot. ‘Russian Constructivism in a Prefascist Theory’. In: *LICS 2020 - Thirty-Fifth Annual ACM/IEEE Symposium on Logic in Computer Science*. Saarbrücken, Germany: IEEE, July 2020, pp. 1–14. DOI: [10.1145/3373718.3394740](https://doi.org/10.1145/3373718.3394740). URL: <https://hal.inria.fr/hal-02548315>.
- [10] P.-M. Pédrot and N. Tabareau. ‘The Fire Triangle’. In: *Proceedings of the ACM on Programming Languages* (Jan. 2020), pp. 1–28. DOI: [10.1145/3371126](https://doi.org/10.1145/3371126). URL: <https://hal.archives-ouvertes.fr/hal-02383109>.
- [11] L. Pujet and N. Tabareau. ‘Impredicative Observational Equality’. In: *POPL 2023 Proceedings of the 50th ACM SIGPLAN Symposium on Principles of Programming Languages*. POPL 2023 - 50th ACM SIGPLAN Symposium on Principles of Programming Languages. Vol. 7. Proceedings of the ACM on programming languages. Boston, United States, 15th Jan. 2023, p. 74. DOI: [10.1145/3571739](https://doi.org/10.1145/3571739). URL: <https://hal.archives-ouvertes.fr/hal-03857705>.
- [12] L. Pujet and N. Tabareau. ‘Observational Equality: Now For Good’. In: POPL. Philadelphie, United States, 17th Jan. 2022. URL: <https://hal.inria.fr/hal-03367052>.
- [13] M. Sozeau, S. Boulier, Y. Forster, N. Tabareau and T. Winterhalter. ‘Coq Coq Correct! Verification of Type Checking and Erasure for Coq, in Coq’. In: *Proceedings of the ACM on Programming Languages* (Jan. 2020), pp. 1–28. DOI: [10.1145/3371076](https://doi.org/10.1145/3371076). URL: <https://hal.archives-ouvertes.fr/hal-02380196>.
- [14] M. Sozeau, Y. Forster, M. Lennon-Bertrand, J. B. Nielsen, N. Tabareau and T. Winterhalter. ‘Correct and Complete Type Checking and Certified Erasure for Coq, in Coq’. In: *Journal of the ACM (JACM)* (27th Nov. 2024), pp. 1–76. DOI: [10.1145/3706056](https://doi.org/10.1145/3706056). URL: <https://inria.hal.science/hal-04077552>.

12.2 Publications of the year

International journals

- [15] A. Adjedj, M. Lennon-Bertrand, T. Benjamin and K. Maillard. ‘AdapTT: Functoriality for Dependent Type Casts’. In: *Proceedings of the ACM on Programming Languages* 10.POPL (8th Jan. 2026), pp. 628–658. DOI: [10.1145/3776664](https://doi.org/10.1145/3776664). URL: <https://hal.science/hal-05167997> (cit. on p. 20).
- [16] B. Ahrens, A. Lafont and T. Lamiaux. ‘2-Functoriality of Initial Semantics, and Applications’. In: *Proceedings of the ACM on Programming Languages* 9.ICFP (5th Aug. 2025), pp. 643–674. DOI: [10.1145/3747527](https://doi.org/10.1145/3747527). URL: <https://hal.science/hal-05201321> (cit. on p. 22).
- [17] C. Cohen, E. Crance and A. Mahboubi. ‘Troq: Proof Transfer for Free, Beyond Equivalence and Univalence’. In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* (26th May 2025), pp. 1–40. DOI: [10.1145/3737283](https://doi.org/10.1145/3737283). URL: <https://inria.hal.science/hal-05192017> (cit. on p. 20).
- [18] K. Jacobs, M. Toro, N. Tabareau and É. Tanter. ‘Robust Dynamic Embedding for Gradual Typing *’. In: *Proceedings of the ACM on Programming Languages* 9.ICFP (21st Sept. 2025), pp. 1–26. DOI: [10.1145/3747507](https://doi.org/10.1145/3747507). URL: <https://inria.hal.science/hal-05162302> (cit. on p. 23).

- [19] Y. Leray and T. Winterhalter. ‘Encode the Cake and Eat it Too: Controlling computation in type theory, locally’. In: *Proceedings of the ACM on Programming Languages* 10.62 (14th Jan. 2026). DOI: [10.1145/3776704](https://doi.org/10.1145/3776704). URL: <https://hal.science/hal-05160846> (cit. on p. 21).
- [20] É. Mangel, P.-A. Melliès and G. Munch-Maccagnoni. ‘Classical notions of computation and the Hasegawa-Thielecke theorem’. In: *Proceedings of the ACM on Programming Languages* 10.POPL (8th Jan. 2026), p. 73. DOI: [10.1145/3776715](https://doi.org/10.1145/3776715). URL: <https://hal.science/hal-05006242> (cit. on p. 22).
- [21] P.-M. Pédrot. ‘Upon This Quote I Will Build My Church Thesis’. In: *Communications of the ACM* (29th May 2025), pp. 1–8. DOI: [10.1145/3715707](https://doi.org/10.1145/3715707). URL: <https://inria.hal.science/hal-05444840> (cit. on p. 20).
- [22] J. Poiret, G. Gilbert, K. Maillard, P.-M. Pédrot, M. Sozeau, N. Tabareau and É. Tanter. ‘All Your Base Are Belong to Us: Sort Polymorphism for Proof Assistants’. In: *Proceedings of the ACM on Programming Languages* 9 (Jan. 2025), pp. 1–34. DOI: [10.1145/3704912](https://doi.org/10.1145/3704912). URL: <https://nantes-universite.hal.science/hal-04801739> (cit. on p. 19).
- [23] L. Pujet, Y. Leray and N. Tabareau. ‘Observational Equality Meets CIC’. In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 47.2 (22nd Apr. 2025), pp. 1–35. DOI: [10.1145/3719342](https://doi.org/10.1145/3719342). URL: <https://hal.science/hal-05085635> (cit. on p. 19).
- [24] J. Rosain, T. Díaz, K. Maillard, M. Sozeau, N. Tabareau, É. Tanter and T. Winterhalter. ‘Bounded Sort Polymorphism with Elimination Constraints’. In: *Proceedings of the ACM on Programming Languages* (13th Jan. 2026). URL: <https://hal.science/hal-05372721> (cit. on p. 20).

International peer-reviewed conferences

- [25] M. Baillon, Y. Forster, A. Mahboubi, P.-M. Pédrot and M. Piquerez. ‘A Zoo of Continuity Properties in Constructive Type Theory’. In: 10th International Conference on Formal Structures for Computation and Deduction (FSCD 2025). Birmingham, France: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. DOI: [10.4230/LIPIcs.FSCD.2025.9](https://doi.org/10.4230/LIPIcs.FSCD.2025.9). URL: <https://inria.hal.science/hal-04908282> (cit. on p. 19).
- [26] M. Baillon, A. Mahboubi and P.-M. Pédrot. ‘In Cantor Space No One Can Hear You Stream’. In: ESOP2026 - 35th European Symposium on Programming and Systems. Turin, Italy, 11th Apr. 2026. URL: <https://hal.science/hal-05495450> (cit. on p. 19).
- [27] P. Borthelle, T. Hirschowitz, G. Jaber and Y. Zakowski. ‘An abstract, certified account of operational game semantics’. In: ESOP 2025 - 34th European Symposium on Programming. Vol. 15694. Hamilton, Ontario, Canada: Springer, 2025, pp. 172–199. DOI: [10.1007/978-3-031-91118-7_7](https://doi.org/10.1007/978-3-031-91118-7_7). URL: <https://hal.science/hal-04583895> (cit. on p. 22).
- [28] R. Cirpons, F. Hivert, A. Mahboubi, G. Melquiond, J. D. Mitchell and F. Smith. ‘Certifying the Decidability of the Word Problem in Monoids at Large’. In: *CPP 2026: Proceedings of the 15th ACM SIGPLAN International Conference on Certified Programs and Proofs*. CPP 2026 - 15th ACM SIGPLAN International Conference on Certified Programs and Proofs. Rennes, France: ACM, Jan. 2026, pp. 128–142. DOI: [10.1145/3779031.3779101](https://doi.org/10.1145/3779031.3779101). URL: <https://hal.science/hal-05448783> (cit. on p. 23).
- [29] S. Congard, G. Munch-Maccagnoni and R. Douence. ‘Linear Effects, Exceptions, and Resource Safety: A Curry-Howard Correspondence for Destructors’. In: *LNCS*. ESOP 2026 - 35th European Symposium on Programming. Turin, Italy: Springer, 2026. DOI: [10.48550/arXiv.2510.23517](https://doi.org/10.48550/arXiv.2510.23517). URL: <https://inria.hal.science/hal-05430763> (cit. on p. 22).
- [30] T. Díaz, K. Maillard, N. Tabareau and É. Tanter. ‘Incremental Certified Programming’. In: *ACM Digital Library*. OOPSLA 2025 - ACM Conference on Object Oriented Programming Systems Languages and Applications. Vol. 9. OOPSLA2. Singapore, Singapore: ACM, Oct. 2025, pp. 1–28. DOI: [10.1145/3763068](https://doi.org/10.1145/3763068). URL: <https://hal.science/hal-05256780> (cit. on p. 24).
- [31] T. Felicissimo and J.-P. Jouannaud. ‘Sort-Based Confluence Criteria for Non-Left-Linear Higher-Order Rewriting’. In: 30th International Conference on Automated Deduction. Stuttgart, Germany, 2025. URL: <https://hal.science/hal-04973508> (cit. on p. 24).

- [32] H. Jaafar and G. Jaber. ‘Operational Game Semantics for Generative Algebraic Effects and Handlers’. In: *Proceedings of the 27th International Symposium on Principles and Practice of Declarative Programming*. PDP 2025 - 27th International Symposium on Principles and Practice of Declarative Programming. 17. Rende, Italy, 2025, pp. 1–21. DOI: [10.1145/3756907.3756924](https://hal.science/hal-05166091). URL: <https://hal.science/hal-05166091> (cit. on p. 23).

Conferences without proceedings

- [33] M. Bouverot-Dupuis, T. Winterhalter, K. Stark and K. Maillard. ‘Sulfur: a Reflective Tactic for Substitution Simplification’. In: *RocqPL 2026 - Rocq for Programming Languages*. Rennes, France, 2026. URL: <https://hal.science/hal-05482084>.
- [34] J. Caspar and G. Munch-Maccagnoni. ‘S4 modal sequent calculus as intermediate logic and intermediate language’. In: *PEPM 2026 - ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation*. Rennes, France: ACM, 2026. URL: <https://inria.hal.science/hal-05430766> (cit. on p. 21).
- [35] T. Stoskopf, C. Cohen and N. Tabareau. ‘Babel-formal: Translation of Proofs between Lean and Rocq’. In: *The 5th Workshop on Mathematical Reasoning and AI at NeurIPS 2025*. San Diego, United States, 6th Dec. 2025. URL: <https://hal.science/hal-05342510> (cit. on p. 21).
- [36] Y. Xu and K. Maillard. ‘Geometric Reasoning in Lean: from Algebraic Structures to Presheaves’. In: *TYPES 2025 - 31st International Conference on Types for Proofs and Programs*. Glasgow, United Kingdom, 2025. URL: <https://hal.science/hal-05448271> (cit. on p. 24).

Reports & preprints

- [37] T. Felicissimo, Y. Leray, L. Pujet, N. Tabareau, É. Tanter and T. Winterhalter. *Definitional Proof Irrelevance Made Accessible*. 2026. URL: <https://hal.science/hal-05474391>.
- [38] T. Lamiaux, Y. Forster, M. Sozeau and N. Tabareau. *Nested Inductive Types: Justified and Usable Nested Inductive Types in Lean and Rocq*. 2025. URL: <https://hal.science/hal-05366368> (cit. on p. 21).
- [39] Y. Leray, G. Gilbert, N. Tabareau and T. Winterhalter. *The Rewster: Type Preserving Rewrite Rules for the Rocq Prover: Extended version of the ITP 2024 paper*. Oct. 2025. URL: <https://inria.hal.science/hal-05294553>.
- [40] J. Poiret, K. Maillard and N. Tabareau. *Divide and Check: Logical Relations, No Algorithms Attached*. 5th Feb. 2026. URL: <https://nantes-universite.hal.science/hal-05495420>.

Software

- [41] [SW] R. Affeldt, Y. Bertot, A. Bruni, C. Cohen, M. Kerjean, A. Mahboubi, D. Rouhling, P. Roux, K. Sakaguchi, Z. Stone, P.-Y. Strub and L. Théry, *Mathematical Components Analysis* 18th Apr. 2025. LIC: CeCILL-C Free Software License Agreement. HAL: [hal-05038721](https://inria.hal.science/hal-05038721), URL: <https://inria.hal.science/hal-05038721>, SWHID: [sw:1:dir:5fe4b61acc38fea6066881d466ac64524e90a4cd](https://sw.hal.science/sw/1/dir/5fe4b61acc38fea6066881d466ac64524e90a4cd).
- [42] [SW] C. Cohen, E. Crance and A. Mahboubi, *Trocq* 18th Apr. 2025. LIC: GNU Lesser General Public License v3.0 or later. DOI: [10.5281/zenodo.10492403](https://zenodo.org/doi/10.5281/zenodo.10492403), HAL: [hal-05038716](https://inria.hal.science/hal-05038716), URL: <https://inria.hal.science/hal-05038716>, SWHID: [sw:1:dir:21f2e18a8e117e0615eb19d5554ad4a5f828f6b1](https://sw.hal.science/sw/1/dir/21f2e18a8e117e0615eb19d5554ad4a5f828f6b1).

12.3 Cited publications

- [43] T. Altenkirch, C. McBride and W. Swierstra. ‘Observational equality, now!’ In: *Proceedings of the ACM Workshop on Programming Languages meets Program Verification (PLPV 2007)*. Freiburg, Germany, Oct. 2007, pp. 57–68 (cit. on p. 8).

- [44] M. Bezem and T. Coquand. ‘Loop-checking and the uniform word problem for join-semilattices with an inflationary endomorphism’. In: *Theor. Comput. Sci.* 913 (2022), pp. 1–7. DOI: [10.1016/J.TCS.2022.01.017](https://doi.org/10.1016/j.tcs.2022.01.017). URL: <https://doi.org/10.1016/j.tcs.2022.01.017> (cit. on p. 8).
- [45] G. Combette and G. Munch-Maccagnoni. *A resource modality for RAI (abstract)*. Tech. rep. LOLA 2018: Workshop on Syntax and Semantics of Low-Level Languages. INRIA, Apr. 2018. URL: <https://hal.inria.fr/hal-01806634> (cit. on p. 9).
- [46] Coq Development Team, The. *The Coq proof assistant reference manual*. Version 8.5. 2015. URL: <http://coq.inria.fr> (cit. on p. 6).
- [47] J.-Y. Girard. ‘Linear Logic’. In: *Theoretical Computer Science* 50 (1987), pp. 1–102 (cit. on p. 7).
- [48] G. Gonthier. ‘Formal proofs—the four-colour theorem’. In: *Notices of the AMS* 55.11 (2008), pp. 1382–1393 (cit. on p. 6).
- [49] G. Gonthier, A. Asperti, J. Avigad, Y. Bertot, C. Cohen, F. Garillot, S. Roux, A. Mahboubi, R. O’Connor, S. Ould Biha, I. Pasca, L. Rideau, A. Solovyev, E. Tassi and L. Théry. ‘A Machine-Checked Proof of the Odd Order Theorem’. In: *Interactive Theorem Proving*. Ed. by S. Blazy, C. Paulin-Mohring and D. Pichardie. Vol. 7998. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 163–179. DOI: [10.1007/978-3-642-39634-2_14](https://doi.org/10.1007/978-3-642-39634-2_14). URL: http://dx.doi.org/10.1007/978-3-642-39634-2_14 (cit. on p. 6).
- [50] T. C. Hales, M. Adams, G. Bauer, D. T. Dang, J. Harrison, T. L. Hoang, C. Kaliszzyk, V. Magron, S. McLaughlin, T. T. Nguyen, T. Q. Nguyen, T. Nipkow, S. Obua, J. Pleso, J. Rute, A. Solovyev, A. H. T. Ta, T. N. Tran, D. T. Trieu, J. Urban, K. K. Vu and R. Zumkeller. ‘A formal proof of the Kepler conjecture’. In: *CoRR* abs/1501.02155 (2015). URL: <http://arxiv.org/abs/1501.02155> (cit. on p. 6).
- [51] X. Leroy. ‘Formal certification of a compiler back-end or: programming a compiler with a proof assistant’. In: *ACM SIGPLAN Notices* 41.1 (2006), pp. 42–54 (cit. on p. 6).
- [52] P. Martin-Löf. ‘An intuitionistic theory of types: predicative part’. In: *Logic Colloquium ’73 Studies in Logic and the Foundations of Mathematics*.80 (1975), pp. 73–118 (cit. on p. 7).
- [53] E. Moggi. ‘Computational lambda-calculus and monads’. In: *Proceedings of the Fourth Annual IEEE Symposium on Logic in Computer Science (LICS 1989)*. Pacific Grove, CA, USA: IEEE Computer Society Press, June 1989, pp. 14–23 (cit. on p. 7).
- [54] L. Pujet and N. Tabareau. ‘Impredicative Observational Equality’. In: *Proceedings of the ACM on Programming Languages*. Proceedings of the ACM on programming languages 7.POPL (Jan. 2023), p. 74. DOI: [10.1145/3571739](https://doi.org/10.1145/3571739). URL: <https://hal.science/hal-03857705> (cit. on p. 8).
- [55] L. Pujet and N. Tabareau. ‘Observational Equality: Now For Good’. In: *Proceedings of the ACM on Programming Languages* 6.POPL (Jan. 2022), pp. 1–29. DOI: [10.1145/3498693](https://doi.org/10.1145/3498693). URL: <https://inria.hal.science/hal-03367052> (cit. on p. 8).
- [56] Univalent Foundations Project. *Homotopy Type Theory: Univalent Foundations for Mathematics*. <http://homotopytypetheory.org/book>, 2013 (cit. on p. 7).