

# 2025 Activity Report

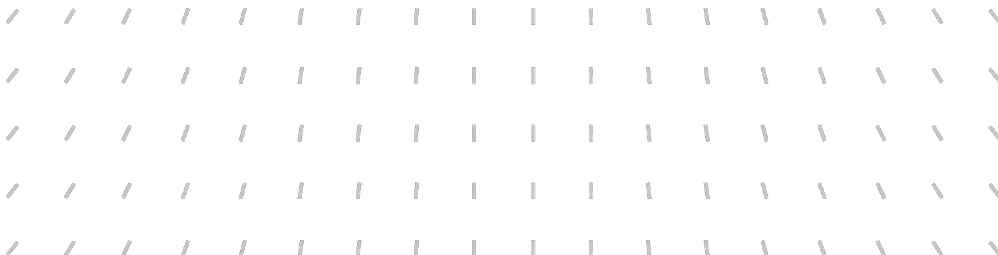
RESEARCH CENTRE: Inria Centre at Université Côte d'Azur

  
Project-Team

## STAMP

Safety Techniques based on Formalized Mathematical  
Proofs





## **Project-Team STAMP**

*Creation of the Project-Team: 2019 November 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

- A2.1.11. – Proof languages
- A4.5.3. – Program proof
- A7.2. – Logic in Computer Science
- A7.2.3. – Interactive Theorem Proving
- A7.2.4. – Mechanized Formalization of Mathematics

### Other research topics and application domains

- B6.1. – Software industry
- B9.5.1. – Computer science
- B9.5.2. – Mathematics

## Contents

<b>Project-Team STAMP</b>	<b>1</b>
<b>1 Team members, visitors, external collaborators</b>	<b>5</b>
<b>2 Overall objectives</b>	<b>5</b>
<b>3 Research program</b>	<b>6</b>
3.1 Theoretical background . . . . .	6
<b>4 Application domains</b>	<b>6</b>
4.1 Mathematical Components . . . . .	6
4.2 Proofs for robotics . . . . .	7
<b>5 Latest software developments, platforms, open data</b>	<b>7</b>
5.1 Latest software developments . . . . .	7
5.1.1 Rocq . . . . .	7
5.1.2 rocq-elpi . . . . .	8
5.1.3 ELPI . . . . .	8
5.1.4 Hierarchy Builder . . . . .	9
5.1.5 VsRocq . . . . .	9
5.1.6 Mastic . . . . .	10
<b>6 New results</b>	<b>10</b>
6.1 Determinacy checker . . . . .	10
6.2 Rocq formalization of the determinacy checker . . . . .	11
6.3 Formal Semantics for Hierarchy Builder . . . . .	11
6.4 Subsets and Subtypes in Hierarchy Builder . . . . .	11
6.5 Improving Rocq unification . . . . .	11
6.6 Recursive functions on real numbers with more than one argument . . . . .	12
6.7 A tool to recognize differences between terms . . . . .	12
6.8 Describing positions in terms . . . . .	12
6.9 Formalized introductory course on trigonometry . . . . .	12
6.10 Elementary programming constructs for teaching . . . . .	13
6.11 Computing safe trajectories between straight line obstacles . . . . .	13
6.12 Formalization of the CAD in the Rocq prover . . . . .	13
6.13 Formal study of the Fast Fourier Transform . . . . .	13
6.14 Formalizing Recreative Mathematics . . . . .	13
<b>7 Partnerships and cooperations</b>	<b>14</b>
7.1 International initiatives . . . . .	14
7.1.1 Inria associate team not involved in an ILL or an international program . . . . .	14
7.2 National initiatives . . . . .	14
7.2.1 ANR . . . . .	14
7.2.2 Inria Challenges . . . . .	14
<b>8 Dissemination</b>	<b>14</b>
8.1 Promoting scientific activities . . . . .	14
8.1.1 Scientific events: organization . . . . .	14
8.1.2 Invited talks . . . . .	15
8.1.3 Research administration . . . . .	15
8.2 Teaching - Supervision - Juries - Educational and pedagogical outreach . . . . .	15
8.2.1 Teaching . . . . .	15
8.2.2 Supervision . . . . .	15
8.2.3 Juries . . . . .	15

<b>9 Scientific production</b>	<b>15</b>
9.1 Publications of the year . . . . .	15
9.2 Cited publications . . . . .	17

## 1 Team members, visitors, external collaborators

### Research Scientists

- Yves Bertot [Team leader, INRIA, Senior Researcher, HDR]
- Enrico Tassi [INRIA, Researcher]
- Laurent Théry [INRIA, Researcher]

### PhD Students

- Davide Fissore [UNIV COTE D'AZUR]
- Thomas Portet [INRIA]
- Quentin Vermande [UNIV COTE D'AZUR]

### Technical Staff

- Romain Tetley [INRIA, Engineer]

### Administrative Assistant

- Christine Foggia [INRIA]

### Visiting Scientist

- Matteo Calosci [UNIV FLORENCE, from Apr 2025 until Aug 2025]

### External Collaborator

- Julien Puydt [Ministère Armées, from Jun 2025]

## 2 Overall objectives

Computers and programs running on these computers are powerful tools for many domains of human activities. In some of these domains, program errors can have enormous consequences. It will become crucial for all stakeholders that the best techniques are used when designing these programs.

We advocate using higher-order logic proof assistants as tools to obtain better quality programs and designs. These tools make it possible to build designs where all decisive arguments are explicit, ambiguity is alleviated, and logical steps can be verified precisely. In practice, we are intensive users of the Rocq system and we participate actively to the development of this tool, in collaboration with other teams at Inria, and we also take an active part in promoting its usage by academic and industrial users around the world.

Many domains of modern computer science and engineering make a heavy use of mathematics. If we wish to use proof assistants to avoid errors in designs, we need to develop corpora of formally verified mathematics that are adapted to these domains. Developing libraries of formally verified mathematics is the main motivation for our research. In these libraries, we wish to capture not only the knowledge that is usually recorded in definitions and theorems, but also the practical knowledge that is recorded in mathematical practice, idioms, and work habits. Thus, we are interested in logical facts, algorithms, and notation habits. Also, the very process of developing an ambitious library is a matter of organization, with design decisions that need to be evaluated and improved. Refactoring of libraries is also an important topic. Among all higher-order logic based proof assistants, we contend that those based on Type theory are the best suited for this work on libraries, thanks to their strong capabilities for abstraction and modular re-use.

The interface between mathematics, computer science and engineering is large. To focus our activities, we will concentrate on applications of proof assistants to robotics.

## 3 Research program

### 3.1 Theoretical background

The proof assistants that we consider provide both a programming language, where users can describe algorithms performing tasks in their domain of interest, and a logical language to reason about the programs, thus making it possible to ensure that the algorithms do solve the problems for which they were designed. Trustability is gained because algorithms and logical statements provide multiple views of the same topic, thus making it possible to detect errors coming from a mismatch between expected and established properties. The verification process is itself a logical process, where the computer can bring rigor in aligning expectations and guarantees.

The foundations of proof assistants rest on the very foundations of mathematics. As a consequence, all aspects of reasoning must be made completely explicit in the process of formally verifying an algorithm. All aspects of the formal verification of an algorithm are expressed in a discourse whose consistency is verified by the computer, so that unclear or intuitive arguments need to be replaced by precise logical inferences.

One of the foundational features on which we rely extensively is *Type Theory*. In this approach a very simple programming language is equipped with a powerful discipline to check the consistency of usage: types represent sets of data with similar behavior, functions represent algorithms mapping types to other types, and the consistency can be verified by a simple computer program, a *type-checker*. Although they can be verified by a simple program, types can express arbitrary complex objects or properties, so that the verification work lives in an interesting realm, where verifying proofs is decidable, but finding the proofs is undecidable.

This process for producing new algorithms and theorems is a novelty in the development of mathematical knowledge or algorithms, and new working methods must be devised for it to become a productive approach to high quality software development. Questions that arise are numerous. How do we avoid requiring human assistance to work on mundane aspects of proofs? How do we take advantage of all the progress made in automatic theorem proving? How do we organize the maintenance of ambitious corpora of formally verified knowledge in the long term?

To acquire hands-on expertise, we concentrate our activity on two aspects. The first one is foundational: we develop and maintain a library of mathematical facts that covers many aspects of algebra and analysis. In the past, we applied this library to proofs in group theory, but it is increasingly used for many different areas of mathematics and by other teams around the world, from combinatorics to elliptic cryptography, for instance. The second aspect is application to robotics, as we believe that the current trend towards more and more autonomous robots and vehicles will raise questions of safety and trustability where formal verification can bring significant added value.

## 4 Application domains

### 4.1 Mathematical Components

The Mathematical Components library is the main by-product of an effort started almost two decades ago to provide a formally verified proof for a major theorem in group theory. Because this major theorem had a proof published in books of several hundreds of pages, with elements coming from character theory, other coming from algebra, and some coming from real analysis, it was an exercise in building a large library, with results in many domains, and in establishing clear guidelines for further increase and data search.

This library has proved to be a useful repository of mathematical facts for a wide area of applications, so that it has a growing community of users in many countries (Denmark, France, Germany, Japan, Singapore, Spain, Sweden, UK, USA) and for a wide variety of topics (transcendental number theory, elliptic curve cryptography, articulated robot kinematics, recently block chain foundations).

Interesting questions on this library range around the importance of decidability and proof irrelevance, the way to structure knowledge to automatically inherit theorems from one topic to another, the way to generate infrastructure to make this automation efficient and predictable. In particular, we want to concentrate on adding a new mathematical topic to this library: real analysis and then complex analysis (Mathematical Components Analysis).

On the front of automation, we are convinced that a higher level language is required to describe similarities between theories, to generate theorems that are immediate consequences of structures, etc, and for

this reason, we invest in the development of a new language on top of the proof assistant (ELPI, Embeddable Lambda Prolog Interpreter).

## 4.2 Proofs for robotics

Robots are man-made artifacts where numerous design decisions can be argued based on logical or mathematical principles. For this reason, we wish to use this domain of application as a focus for our investigations. The questions for which we are close to providing answers involve precision issues in numeric computation, obstacle avoidance and motion planning (including questions of graph theory), articulated limb kinematics and dynamics, and balance and active control.

From the mathematical perspective, these topics require that we improve our library to cover real algebraic geometry, computational geometry, real analysis, graph theory, and refinement relations between abstract algorithms and executable programs.

In the long run, we hope to exhibit robots where pieces of software and part of the design have been subject to formal verification.

# 5 Latest software developments, platforms, open data

## 5.1 Latest software developments

### 5.1.1 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 organized 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-Claudel, Kazuhiko Sakaguchi, Vincent Semeria, Michael Soegtrop, Arnaud Spiwack, Matthieu Sozeau, Enrico Tassi, Laurent Théry, Anton Trunov, Li-Yao Xia, Theo Zimmermann

### 5.1.2 rocq-elpi

**Keywords:** Metaprogramming, Extension

**Scientific Description:** Rocq-Elpi provides a Rocq plugin that embeds Elpi. It offers a way to embed Rocq terms into  $\lambda$ Prolog using the Higher-Order Abstract Syntax (HOAS) approach, along with mechanisms to read terms back. In addition, it exports a comprehensive set of Rocq primitives to Elpi, such as printing messages, accessing the environment of theorems and data types, defining new constants, and more. For convenience, it includes quotations and anti-quotations for Rocq syntax within  $\lambda$ Prolog code: e.g., `{{nat}}` expands to the type name of natural numbers, and `{{A -> B}}` to the representation of a product (unfolding the `->` notation). Finally, it enables the definition of new vernacular commands and tactics.

**Functional Description:** Rocq plugin embedding ELPI

**Release Contributions:** - parsing/execution separation

**News of the Year:** Commands implemented in Rocq-Elpi can now start interactive proofs. APIs dealing with Rocq universes have been revised, paving the way for upcoming changes in Rocq's universe system, namely the introduction of algebraic polymorphic universes.

**Publications:** [hal-01897468](#), [hal-01637063](#), [hal-04547069](#), [hal-03800154](#)

**Contact:** Enrico Tassi

**Participants:** Enrico Tassi, Davide Fissore

### 5.1.3 ELPI

**Name:** Embeddable Lambda Prolog Interpreter

**Keywords:** Constraint Programming, Programming language, Higher-order logic

**Scientific Description:** Elpi implements a variant of  $\lambda$ Prolog enriched with Constraint Handling Rules (CHR).

As a descendant of Prolog, Elpi is a rule-based language: programs consist of sets of rules that govern how computations proceed.

These rules can be introduced in two ways: dynamically or statically.

Dynamic rules are added at runtime—particularly when processing binders—to attach data to bound variables in a scoped, context-sensitive manner.

Static rules are extended or updated as Rocq's logical environment evolves, ensuring compatibility with new definitions and proofs.

Constraints and their handling rules enrich unification variables (holes) with metadata (e.g., their type) and maintain consistency in the constraint store (e.g., by detecting incompatible type assignments for the same hole).

**Functional Description:** Elpi is a high-level programming language designed to implement new commands and tactics for the Rocq prover. It provides native support for syntax trees with binders and holes, sparing programmers the complexities of De Bruijn indices and unification variables.

**Release Contributions:** - Determinacy checker (static analysis)

**News of the Year:** Davide Fissore developed a static determinacy checker covering many language features, including the cut operator, higher-order predicates, and dynamic predicates.

**URL:** <https://github.com/lpcic/elpi/>

**Publications:** [hal-03800154](#), [hal-01176856](#), [hal-01410567](#), [hal-01897468](#)

**Contact:** Enrico Tassi

**Participants:** Davide Fissore, Enrico Tassi, Claudio Sacerdoti Coen

#### 5.1.4 Hierarchy Builder

**Keywords:** Metaprogramming, Rocq

**Scientific Description:** It is nowadays customary to organize libraries of machine-checked proofs around hierarchies of algebraic structures. One influential example is the Mathematical Components library, on top of which the long and intricate proof of the Odd Order Theorem could be fully formalized. Still, building algebraic hierarchies in a proof assistant such as Rocq requires a lot of manual labor and often deep expertise in the prover's internals. Moreover, according to our experience, making a hierarchy evolve without breaking client code is equally tricky: even a simple refactoring such as splitting a structure into two simpler ones is hard to get right.

Hierarchy Builder is a high-level language to build hierarchies of algebraic structures and to evolve these hierarchies without breaking user code. The key concepts are factory, builder, and abbreviation, which let the hierarchy developer describe an actual interface for their library. Behind that interface, the developer can provide appropriate code to ensure backward compatibility.

We implement the Hierarchy Builder language in the hierarchy-builder add-on for the Rocq system using the Elpi extension language.

**Functional Description:** Hierarchy Builder is a high-level language for Rocq to build hierarchies of algebraic structures and to evolve these hierarchies without breaking user code. The key concepts are factory, builder, and abbreviation, which let the hierarchy developer describe an actual interface for their library. Behind that interface, the developer can provide appropriate code to ensure backward compatibility.

**Release Contributions:** Compatible with Coq 8.18 to Coq 8.20, Rocq 9.0 and Rocq 9.1.

**News of the Year:** New support for structures with mixins on different subjects. Matteo Calosci used this new feature to unify two hierarchies in Mathematical Components, in particular the one on iterated functions with the base of the algebraic hierarchy (e.g., monoids and rings).

**URL:** <https://github.com/math-comp/hierarchy-builder>

**Publication:** [hal-02478907](#)

**Contact:** Enrico Tassi

**Participants:** Kazuhiko Sakaguchi, Enrico Tassi, Cyril Cohen

**Partners:** University of Tsukuba, Onera

#### 5.1.5 VsRocq

**Name:** VsRocq

**Keyword:** IDE

**Functional Description:** VsRocq is an extension for Visual Studio Code (VS Code) and VSCodium that provides support for the Rocq Interactive Theorem Prover.

VsRocq is distributed in two flavors:

- **VsRocq Legacy** (required for Coq < 8.18, compatible with Coq >= 8.7) is based on the original VsCoq implementation by C.J. Bell. It uses the legacy XML protocol spoken by CoqIDE.
- **VsRocq** (recommended for Rocq and Coq >= 8.18) is a full reimplementaion built around a language server that natively speaks the LSP protocol.

**Release Contributions:** We have mainly focused on stability and bug fixes. In this release you'll find:

- Improvements to performance on large files.
- Fixes for document state invalidation bugs.
- Goal view enhancements.

**News of the Year:** We have mainly worked on stability and bug fixes. We also improved the performance of continuous document parsing by making the process interruptible, allowing early termination of parsing for outdated documents.

**URL:** <https://github.com/coq/vscoq>

**Contact:** Romain Tetley

### 5.1.6 Mastic

**Name:** Mastic

**Keywords:** Parser, Resiliency

**Functional Description:** Mastic is an experimental library for writing error-resilient parsers on top of the Menhir parser generator.

Its intended use is in the (future) language servers for Elpi and Jasmin, to be developed by Inria's SED team. An error-resilient parser always produces a syntax tree containing error nodes, enabling the language server to provide services based on that tree rather than plain text. For example, the syntax tree can be type-checked by ignoring subtrees that represent parse errors, allowing users to query typing information (e.g., via hover) even on incomplete or incorrect code.

**URL:** <https://github.com/gares/mastic>

**Contact:** Enrico Tassi

**Participant:** Enrico Tassi

## 6 New results

### 6.1 Determinacy checker

**Participants:** Davide Fissore, Enrico Tassi.

We have finalized the implementation of the Determinacy Checker in the Elpi programming language (Section 5.1.3). A description of the main idea behind the determinacy checker in ELPI can be found at [tag v3.0.0](#). The changes were released in [ELPI v3.0.0](#). We have ported the [Rocq-ELPI v3.0.0](#) and [HIERARCHY-BUILDER v1.8.1](#) libraries to the new ELPI version.

This work is presented in [5] and will be presented at [PADL 2026](#).

## 6.2 Rocq formalization of the determinacy checker

**Participants:** Davide Fissore, Enrico Tassi.

We have formalized in Rocq:

- a semantics of Prolog-with-cut using a tree-like representation,
- the semantics of the target ELPi language (without the  $\pi$  operator) together with the proof of the equivalence of these two semantics (see [github.com/FissoreD/elpi-formalization/tree/v1.2.0](https://github.com/FissoreD/elpi-formalization/tree/v1.2.0)).

We have proved that the determinacy checker is correct in the first-order case (*i.e.* when variables are first-order). We are currently extending the proof to handle higher-order variables in the language (see [github.com/FissoreD/elpi-formalization/tree/ho-check-ffun](https://github.com/FissoreD/elpi-formalization/tree/ho-check-ffun)).

## 6.3 Formal Semantics for Hierarchy Builder

**Participants:** Matteo Calosci, Enrico Tassi.

During his 3-month internship, Matteo Calosci worked on Hierarchy Builder (HB) in the context of the CoREACT ANR project. He defined a formal semantics for HB and restructured the hierarchy of the Mathematical Components library to eliminate duplication between the sub-hierarchies of algebraic structures and iterated operators.

## 6.4 Subsets and Subtypes in Hierarchy Builder

**Participants:** Cyril Cohen, Quentin Vermande.

Cyril Cohen and Quentin Vermande have developed a first prototype that automates:

- conversion between sets and types,
- proofs of set membership,
- type casts — even when an external proof is required.

This functionality has been integrated into Hierarchy Builder. A port of Mathematical Components Analysis using this prototype has been started.

## 6.5 Improving Rocq unification

**Participants:** Quentin Vermande.

Quentin Vermande has worked on improving the unification algorithm in Rocq. In particular, he has:

- made existential variable instantiation more robust,
- reduced the overhead of handling dependent functions compared to non-dependent ones,
- optimized the Canonical Structures heuristic.

The last contribution was presented at UNIF 2025 [8].

## 6.6 Recursive functions on real numbers with more than one argument

**Participants:** Yves Bertot, Thomas Portet.

Yves Bertot and Thomas Portet developed teaching-oriented libraries for the Rocq proof assistant. The emphasis is on using the type of real numbers instead of natural numbers, thus reducing the cognitive load for students: there is only one number type, operations have regular properties, and many logical reasoning steps can be delegated to automatic tactics with a clear domain of application (`ring` and `lia`). This work requires a mechanism to define recursive functions whose inputs are real numbers but that are only well-defined on the subset of natural numbers. Such a tool was already provided in 2024 for unary functions [3] and has been extended this year to functions with an arbitrary number of arguments.

## 6.7 A tool to recognize differences between terms

**Participants:** Yves Bertot, Thomas Portet.

A generic tactic such as `ring` is not well-suited to algebraic reasoning on expressions that contain functions outside its known vocabulary. For instance, proving that  $\cos(x + y) = \cos(y + x)$  is handled poorly by `ring` because `cos` lies outside its domain. We observe that mathematics students typically view the presence of `cos` as incidental, with the essential claim being the simple commutativity  $x + y = y + x$ , precisely what the `ring` handles well. To address this, we developed a packaging tactic based on anti-unification. It recognizes such situations and allows students to use a natural idiom to invoke this type of proof step.

## 6.8 Describing positions in terms

**Participants:** Yves Bertot, Thomas Portet, Laurent Théry.

Several past experiments have shown that logical reasoning can be guided by directly interpreting positions in goals as instructions to bring those positions into the proof focus. An example dating from the 1990s is the concept of proof-by-pointing [12]. More recently, Kaustuv Chaudhuri’s work has concentrated on this notion of focus, while Pablo Donato’s work describes interaction modes in which one or two positions can be interpreted as proof steps. Laurent Théry has taken advantage of the Rocq ELPI tool to revive research on proof-by-pointing and to incorporate some aspects of handling two positions simultaneously. Thomas Portet has been studying ways to make position descriptions more concise.

## 6.9 Formalized introductory course on trigonometry

**Participants:** Yves Bertot, Julien Puydt.

In the curriculum of “classes préparatoires aux grandes écoles”, mathematics teachers have to follow a schedule that does not follow the order of dependence among concepts. As a result, some relatively advanced notions — such as trigonometric functions — are introduced early enough for students to use them in other subjects (*e.g.*, physics). This leads to a presentation order that differs substantially from that in a formalized mathematical development, where trigonometric functions would only appear after power series, which are themselves introduced after sequences and topology. Julien Puydt, a mathematics professor in a “classe préparatoire”, prepared a trigonometry course that follows the usual order in his classroom, and Yves Bertot formalized it in the Rocq proof assistant. This experiment helps identify suitable frameworks for supporting

late definitions combined with early usage of a variety of concepts — functions, but also theorems (for which several alternative proofs can be provided, depending on the context).

## 6.10 Elementary programming constructs for teaching

**Participants:** Yves Bertot.

In continuity with the work on avoiding natural numbers when teaching mathematics, Yves Bertot developed a collection of elementary functions for teaching basic programming constructs that model loops in the Rocq proof assistant. This relies on a general function that describes repetitive computations from a type to itself, equipped with a halting test — akin to a repeat-until loop in conventional imperative programming languages — except that the number of iterations is bounded, as required for terminating functions in Rocq’s functional programming language. This construct is accompanied by several theorems that facilitate proving properties of derived programs using invariants, in the spirit of Hoare logic. We believe this approach will facilitate entry-level teaching.

## 6.11 Computing safe trajectories between straight line obstacles

**Participants:** Yves Bertot.

We have completed the formal proof of an algorithm to decompose the working space into vertical cells for a point in a 2-dimensional working space, where obstacles are given by straight line segments. This work has been published in [4].

## 6.12 Formalization of the CAD in the Rocq prover

**Participants:** Quentin Vermande.

The formalization of Collins’ Cylindrical Algebraic Decomposition (CAD) algorithm in Rocq has been completed. This work has been accepted for publication at CPP 2026 [16].

## 6.13 Formal study of the Fast Fourier Transform

**Participants:** Laurent Théry.

The paper [13] presents our experiment on the formalization of the Fast Fourier Transform in Rocq.

## 6.14 Formalizing Recreative Mathematics

**Participants:** Laurent Théry.

A factorion is a natural number that equals the sum of the factorials of its decimal digits. The paper [10] presents a formalization carried out in Rocq.

A prime number is truncatable if repeatedly removing one digit from either end always results in a prime number. The paper [11] presents a formalization done in Rocq.

An addition chain is a sequence of additions that, starting from 1, allows one to reach a given number  $n$ . The paper [9] presents a formalization done in Rocq.

The first two contributions illustrate the use of computation in proofs in Rocq. The last one shows a nice relation between addition chains and continued fractions.

## 7 Partnerships and cooperations

**Participants:** Yves Bertot, Thomas Portet, Enrico Tassi, Laurent Théry.

### 7.1 International initiatives

#### 7.1.1 Inria associate team not involved in an IIL or an international program

##### FormaSys

**Duration :** 2025 - 2027

**Summary:** FormaSys is a French-Japanese Inria Associate Team, regrouping researchers from several distinct Inria Project Teams (including the STAMP team) and researchers from various Japanese Institutions. The main goal of this project is to extend MathComp and MathComp-Analysis with more mathematical structures and results useful for mathematics applied to physical systems.

### 7.2 National initiatives

#### 7.2.1 ANR

CoREACT “Coq-based Rewriting: towards Executable Applied Category Theory”, started on March 1st, 2023, for 48 months, with a grant of 67,3 kEuros for STAMP, funding a post-doc, instruments, material costs and travel costs. Other partners are **IRIF** (Université Paris Cité), **LIP** (ENS-Lyon) and **LIX**(École Polytechnique). The corresponding researcher for this contract is Yves Bertot.

#### 7.2.2 Inria Challenges

LiberAbaci. The Inria challenge, **LiberAbaci**, explores the use of a Type-theory based proof assistant to improve mathematics education for the first years of higher education (undergraduate mathematics). Yves Bertot coordinated it until September.

## 8 Dissemination

**Participants:** Yves Bertot, Davide Fissore, Thomas Portet, Enrico Tassi, Laurent Théry.

### 8.1 Promoting scientific activities

#### 8.1.1 Scientific events: organization

Enrico Tassi co-organized (with Assia Mahboubi) a workshop in honor of Georges Gonthier ([workshop website](#)). The event had 45 participants, including Laurent Théry and Quentin Vermande, with Yves Bertot presenting a talk on the Mathematical Components Library.

**Reviewer** Davide Fissore has done a review for the **PPDP 2025** conference and Enrico Tassi for the **CPP 2026** conference.

### 8.1.2 Invited talks

Enrico Tassi gave an invited talk at [CoqPL 2025](#).

### 8.1.3 Research administration

Yves Bertot is Deputy Scientific Director for the domain “Algorithmics, Programming, Software and Architecture” since July 1st.

## 8.2 Teaching - Supervision - Juries - Educational and pedagogical outreach

### 8.2.1 Teaching

- Thomas Portet gave 21 hours of exercise sessions (TD) for the course “Automates et Langages” from March to June 2025 in the BUT 2 Informatique en alternance program at Université Côte d’Azur.
- Thomas Portet gave 27 hours of exercise sessions for the course “Programmation Fonctionnelle” from September to December 2025 in the 2nd year of the Licence in Computer Science at Université Côte d’Azur.
- Davide Fissore gave 30 hours of exercise sessions for the course “Programmation Fonctionnelle” in the 3rd year of the Licence in Computer Science at Université Côte d’Azur.
- Davide Fissore gave 48 hours of exercise sessions for the course “Programmation Fonctionnelle” in the 2nd year of the Licence in Computer Science at Université Côte d’Azur.
- Yves Bertot gave a 24-hour course entitled “Proofs and Reliable Programming Using Coq” from January to April 2025 at the Master Informatique et Interactions, Université Côte d’Azur.
- Davide Fissore participated in the Deep Learning School summer school held at the SophiaTech campus in July.

### 8.2.2 Supervision

- Enrico Tassi advised Luko van der Maas on rewriting portions of the Iris Proof Mode using Elpi, as well as on encoding inductive predicates as fixpoints in the Iris logic. The resulting work was published at ITP [6].
- Alessandro Rustichelli, an undergraduate student from the University of Modena, is finalizing his L3 dissertation on (homotopy) type theory under Enrico Tassi’s supervision, with the defense scheduled for 2026.

### 8.2.3 Juries

- Yves Bertot has served on the thesis monitoring committee for Rishikesh Hirendu Vaishnav’s PhD at the University of Paris-Saclay.
- Enrico Tassi has served on the thesis monitoring committee for Thomas Lamiaux’s PhD at the University of Nantes.

## 9 Scientific production

### 9.1 Publications of the year

#### International journals

- [1] R. Affeldt, C. Cohen and A. Saito. ‘Semantics of Probabilistic Programs Using s-Finite Kernels in Dependent Type Theory’. In: *ACM Transactions on Probabilistic Machine Learning* 1.3 (29th Aug. 2025), pp. 1–34. DOI: [10.1145/3732291](https://doi.org/10.1145/3732291). URL: <https://hal.science/hal-05318682>.

### Invited conferences

- [2] E. Tassi. ‘Elpi: rule-based meta-language for Rocq’. In: CoqPL 2025 - The Eleventh International Workshop on Coq for Programming Languages. Denver, United States, 25th Jan. 2025. URL: <https://inria.hal.science/hal-04990628>.

### International peer-reviewed conferences

- [3] Y. Bertot and T. Portet. ‘Chassez le naturel dans la formalisation des mathématiques’. In: 36es Journées Francophones des Langages Applicatifs (JFLA 2025). Roiffé, France, 28th Jan. 2025. URL: <https://inria.hal.science/hal-04757635> (cit. on p. 12).
- [4] Y. Bertot and T. Portet. ‘Formally Verifying a Vertical Cell Decomposition Algorithm’. In: *Leibniz International Proceedings in Informatics (LIPIcs)*. ITP 2025 - 16th International Conference on Interactive Theorem Proving. Vol. LIPIcs-352. Reykjavik, Iceland: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025, p. 24. DOI: [10.4230/LIPIcs.ITP.2025.24](https://doi.org/10.4230/LIPIcs.ITP.2025.24). URL: <https://inria.hal.science/hal-05444842> (cit. on p. 13).
- [5] D. Fissore and E. Tassi. ‘Determinacy Checking for Elpi: an Higher-Order Logic Programming language with Cut’. In: PADL 2026 - The 28th International Symposium on Practical Aspects of Declarative Languages. Vol. 16401. Lecture Notes in Computer Science. Rennes, France: Springer Nature Switzerland, 12th Jan. 2026, pp. 77–95. DOI: [10.1007/978-3-032-15981-6\\_5](https://doi.org/10.1007/978-3-032-15981-6_5). URL: <https://inria.hal.science/hal-05026472> (cit. on p. 10).
- [6] R. Krebbers, L. van der Maas and E. Tassi. ‘Inductive Predicates via Least Fixpoints in Higher-Order Separation Logic’. In: *Leibniz International Proceedings in Informatics. LIPIcs*. 16th International Conference on Interactive Theorem Proving (ITP25). 16th International Conference on Interactive Theorem Proving (ITP 2025) LNCS-352. Reykjavik, Iceland, 28th Sept. 2025. DOI: [10.4230/LIPIcs.ITP.2025.28](https://doi.org/10.4230/LIPIcs.ITP.2025.28). URL: <https://inria.hal.science/hal-05217546> (cit. on p. 15).
- [7] Q. Vermande. ‘Décomposition Algébrique Cylindrique en Coq/Rocq’. In: HAL. 36es Journées Francophones des Langages Applicatifs (JFLA 2025). Roiffé, France, 28th Jan. 2025. URL: <https://inria.hal.science/hal-04859512>.
- [8] Q. Vermande. ‘Optimizing Canonical Structures’. In: *UNIF 2025 - Informal Proceedings of the 39th International Workshop on Unification*. UNIF 2025 - 39th International Workshop on Unification. Birmingham, United Kingdom, 14th July 2025. URL: <https://inria.hal.science/hal-05148851> (cit. on p. 11).

### Reports & preprints

- [9] L. Théry. *A Formalisation of Addition Chains*. 28th Feb. 2025. URL: <https://hal.science/hal-04971933> (cit. on p. 14).
- [10] L. Théry. *A Formalisation of factorions in Rocq*. 16th Sept. 2025. URL: <https://inria.hal.science/hal-05265618> (cit. on p. 13).
- [11] L. Théry. *A Formalisation of Truncatable Primes with Rocq*. 14th June 2025. URL: <https://hal.science/hal-05099989> (cit. on p. 13).
- [12] L. Théry. *Proof by Pointing Revisited*. Oct. 2025. URL: <https://hal.science/hal-05337440> (cit. on p. 12).
- [13] L. Théry. *Revisiting the Fast Fourier Transform in Rocq*. 15th Aug. 2025. URL: <https://inria.hal.science/hal-03807965> (cit. on p. 13).

### Software

- [14] [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://hal.science/hal-05038721), URL: <https://inria.hal.science/hal-05038721>, SWHID: [⟨swh:1:dir:5fe4b61acc38fea6066881d466ac64524e90a4cd⟩](https://sw.hiclipper.com/1/dir/5fe4b61acc38fea6066881d466ac64524e90a4cd).

- [15] [SW] C. Cohen, P. Roux, K. Sakaguchi and E. Tassi, *Hierarchy Builder* 18th Apr. 2025. LIC: MIT License. HAL: [hal-05038713](https://hal.inria.fr/hal-05038713), URL: <https://inria.hal.science/hal-05038713>, SWHID: [swh:1:dir:50c5087fc42a2f95913dada99ef4ac1cdba66d7b](https://swh.io/dir/50c5087fc42a2f95913dada99ef4ac1cdba66d7b).

## 9.2 Cited publications

- [16] Q. Vermande. ‘Cylindrical Algebraic Decomposition in Coq/Rocq’. In: *Proceedings of the 15th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP ’26)*. Rennes, France: ACM, Jan. 2026 (cit. on p. 13).