RESEARCH CENTRE

Inria Centre at Université Côte d'Azur

2024 ACTIVITY REPORT

Project-Team STAMP

Safety Techniques based on Formalized Mathematical Proofs

DOMAIN

Algorithmics, Programming, Software and Architecture

THEME

Proofs and Verification

Inria

Contents

Project-Team STAMP

Creation of the Project-Team: 2019 November 01

Keywords

Computer sciences and digital sciences

A2.1.11. – Proof languages

A2.4.3. – Proofs

A4.5. – Formal methods for security

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

1 Team members, visitors, external collaborators

Research Scientists

- Yves Bertot [Team leader, INRIA, Senior Researcher, HDR]
- Cyril Cohen [INRIA, Researcher, until Jun 2024]
- Laurence Rideau [INRIA, Researcher, leaving for retirement on Dec 21. 2024]
- Enrico Tassi [INRIA, Researcher]
- Laurent Théry [INRIA, Researcher]

Post-Doctoral Fellow

- Paolo Torrini [INRIA, Post-Doctoral Fellow, until May 2024]

PhD Students

- Davide Fissore [UNIV COTE D'AZUR]
- Thomas Portet [INRIA, from Feb 2024]
- Swarn Priya [INRIA, until Jan 2024]
- Quentin Vermande [UNIV COTE D'AZUR]

Technical Staff

- Thomas Portet [INRIA, Engineer, until Jan 2024]
- Romain Tetley [INRIA, Engineer]

Administrative Assistant

- Christine Foggia [INRIA]

External Collaborator

- Julien Puydt [Ministère Armées, from Jun 2024]

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 Coq 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 New software, platforms, open data

5.1 New software

5.1.1 Coq

Name: The Coq Proof Assistant

Keyword: Proof assistant

Scientific Description: Coq 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. Coq 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: Coq 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. Coq also provides a large and extensible set of automatic or semi-automatic proof methods. Coq'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://coq.inria.fr/refman/changes.html#version-8-20 .

News of the Year: Coq version 8.20 adds a new rewrite rule mechanism along with a few new features, a host of improvements to the virtual machine, the notation system, Ltac2 and the standard library.

URL: http://coq.inria.fr/

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 coq-elpi

Keywords: Metaprogramming, Extension

Scientific Description: Coq-elpi provides a Coq plugin that embeds ELPI. It also provides a way to embed Coq terms into lambdaProlog using the Higher-Order Abstract Syntax approach (HOAS) and a way to read terms back. In addition to that it exports to ELPI a set of Coq primitives, e.g. printing a message, accessing the environment of theorems and data types, defining a new constant and so on. For convenience it also provides a quotation and anti-quotation for Coq's syntax in lambdaProlog. E.g. {{nat}} is expanded to the type name of natural numbers, or {{A -> B}} to the representation of a product by unfolding the -> notation. Finally it provides a way to define new vernacular commands and new tactics.

Functional Description: Coq plugin embedding ELPI

Release Contributions: - parsing/execution separation

News of the Year: Coq-Elpi commands to assemble programs and tactics were revised so to better support the separation of parsing from execution introduced in Coq 8.18 and also to take advantage of the new incremental compilation offered by Elpi.

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 lambdaProlog enriched with Constraint Handling Rules.

Being a descendant of Prolog, Elpi is a rule-based language, meaning that its programs are structured as sets of rules that dictate how computations proceed. These rules can be introduced in two ways: dynamically and statically. Dynamic rules are introduced at runtime, particularly when processing binders, to attach data to bound variables in a scoped and context-sensitive manner. Static rules are extended or updated as the logical environment or Rocq evolves, ensuring compatibility with new definitions and proofs as they are added.

Constraints and their handling rules are used to enrich unification variables (holes) with meta data, for example their type. Handling rules are instrumental to maintain the consistency of the constraint store, for example 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, relieving programmers of the complexities associated with De Bruijn indices and unification variables.

Release Contributions: - Faster separate compilation/linking

News of the Year: Elpi's backend was completely rewritten in order to obtain faster compilation and perform type checking early in the compilation pipeline, paving the way to more static analysis. In particular D. Fissore is working on mode and determinacy analysis on top of the new type checking phase.

URL: https://github.com/lpcic/elpi/

Publications: hal-03800154, hal-01176856, hal-01410567, hal-01897468

Contact: Enrico Tassi

Participants: Enrico Tassi, Claudio Sacerdoti Coen

5.1.4 Hierarchy Builder

Keywords: Coq, Metaprogramming

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 Coq requires a lot of manual labor and often a deep expertise in the internals of the prover. Moreover, according to our experience, making a hierarchy evolve without causing breakage in 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 make these hierarchies evolve without breaking user code. The key concepts are the ones of factory, builder and abbreviation that let the hierarchy developer describe an actual interface for their library. Behind that interface the developer can provide appropriate code to ensure retro compatibility. We implement the Hierarchy Builder language in the hierarchy-builder addon for the Coq system using the Elpi extension language.

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

Release Contributions: Compatible with Coq 8.18 to Coq 8.20. Added support for instance saturation

News of the Year: - Major performance improvements in handling large hierarchies (e.g. MathComp 2.0)

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

Publication: hal-02478907

Contact: Enrico Tassi

Participants: Enrico Tassi, Cyril Cohen

Partners: University of Tsukuba, Onera

5.1.5 Trocq

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

Functional Description: Trocq is a prototype of a modular parametricity plugin for Coq, aiming to perform proof transfer by translating the goal into an associated goal featuring the target data structures as well as a rich parametricity witness from which a function justifying the goal substitution can be extracted.

The plugin features a hierarchy of parametricity witness types, ranging from structure-less relations to a new formulation of type equivalence, gathering several pre-existing parametricity translations, including univalent parametricity and CoqEAL, in the same framework.

This modular translation performs a fine-grained analysis and generates witnesses that are rich enough to preprocess the goal yet are not always a full-blown type equivalence, allowing to perform proof transfer with the power of univalent parametricity, but trying not to pull in the univalence axiom in cases where it is not required.

The translation is implemented in Coq-Elpi and features transparent and readable code with respect to a sequent-style theoretical presentation.

Release Contributions: Support for the Prop sort

News of the Year: We released the first version of Trocq, for demo purposes and to support the claims made in the associated paper. Trocq is able to translate non trivial goals between isomorphic or partially isomorphic representations.

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

5.1.6 VsCoq

Name: VsCoq

Keyword: IDE

Functional Description: VsCoq is an extension for Visual Studio Code (VS Code) and VSCodium which provides support for the Coq Proof Assistant.

VsCoq is distributed in two flavours:

- VsCoq 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.

- VsCoq (recommended for Coq >= 8.18) is a full reimplementation around a language server which natively speaks the LSP protocol.

Release Contributions: We have mainly been working on stability and bug fixes, in this release you'll find :

- Some improvements to performance on large files. - Fixing document state invalidation bugs. - Goal view improvements.

News of the Year: We have mainly been working on stability and bug fixes, reaching parity with the legacy branch VSCoq 1. Moreover the Goal view features an efficient rendering algorithm that runs client side and the overall performance of the language server was greatly improved on large files. We also wrote onboarding documentation.

URL: https://github.com/coq/vscoq

Contact: Romain Tetley

6 New results

6.1 Handle the higher-order unification in the Elpi based type-class solver for Coq

Participants: Davide Fissore, Enrico Tassi.

We devise a new technique to take advantage of the higher-order unification of Elpi to solve higher-order unification problems for Coq. The work has been published in [7].

6.2 Trocq

Participants: Cyril Cohen, Enzo Crance *(Gallinette, Inria, Nantes et MERCE, Rennes)*, Assia Mahboubi *(Gallinette, Inria, Nantes)*.

This axis deals with the contribution to the Coq prover libraries to either automate the structuring (using Hierarchy Builder) and transfer (using Trocq) of properties, or provide general mathematical results (using the mathematical components library).

Trocq (Section 5.1.5) is both a new calculus performing a variant of a parametricity translation to achieve transfer of theorems for the calculus of constructions [6], and a prototype plugin for the Coq prover.

6.3 Hierarchy Builder

Participants: Cyril Cohen, Quentin Vermande, Enrico Tassi, Paolo Torrini, Reynald Affeldt *(AIST, Japan)*, Xavier Allamigeon *(Ecole polytechnique, Paris)*, Samuel Arsac *(PLUME, LIP, ENS de Lyon)*, Russel Harmer *(PLUME, LIP, ENS de Lyon)*, Marie Kerjean *(LIPN, Paris)*, Damien Pous *(PLUME, LIP, ENS de Lyon)*, Pierre Roux *(Onera, Toulouse)*, Kazuhiko Sakaguchi *(Galinnette, Inria then LIP, ENS de Lyon)*, Zachary Stone *(Independent contributor)*.

Hierarchy Builder (HB see Section 5.1.4) is a domain specific language integrated to Coq and designed to formally describe hierarchies of mathematical structures (e.g. Groups, Rings, Vector Spaces, etc.), their various equivalent axiomatizations, their generic theory and their instances. As the impact of this project grows, several axes have emerged. In the context of the CoREACT ANR, we apply and extend HB to provide concise formalization of categories, adhesive categories, internal categories and double categories. We also extend the Math-Components and math-comp-analysis libraries with several structures (including preorders, semi-modules, and topological vector spaces). This is a stress test for HB and many contributors – including Quentin Vermande as part of his PhD thesis – improved and optimized HB or the Coq prover.

6.4 Computing safe trajectories between straight line obstacles

Participants: Yves Bertot, Enrico Tassi, Laurent Théry.

We are developing a formally proved sequence of algorithms to compute trajectories of points between obstacles. This year, we completed two proofs of the correctness for the first difficult algorithm, which is concerned with decomposing the working space into a collection of cells. The first proof concerns the easier case where the segments describing the obstacles are allowed to touch at their extremities but

are otherwise not allowed to be vertically aligned. The second proof removes the constraint on vertical alignment. Future work on this topic will concern removing more constraints on the segments, proving the correctness of the other algorithms in the sequence, and proving a concrete implementation in a conventional programming language.

An article describing the current progress of this project has been published in [9].

6.5 A single type of numbers for formalized mathematics

Participants: Yves Bertot, Thomas Portet.

The Inria challenge "LiberAbaci" aims at studying and removing the obstacles to make Type Theory based proof assistants like Coq tools to teach mathematics at introductory levels.

One of the obstacles that we identified is that formalized mathematics in proof assistants typically rely on several different types of numbers to represent natural numbers, integers, rational numbers and real numbers (thus following a tradition inherited from programming languages). This habit implies that there are several type-theoretic functions to represent the same basic number operations, like addition, multiplication, and subtraction, sometimes with subtle differences in behavior, imposed by the need to respect type discipline.

We propose an alternative approach where a single type of numbers describes all the numeric aspects in a given introductory mathematical course. We choose to use the type of real numbers, aiming to provide a corpus of formalized mathematics that makes it possible to describe recursive sequences. In this approach, natural numbers are viewed as a subset of the real numbers, so that only one function is used for each of the basic operations, like addition, multiplication, subtraction, and division. Properties that were guaranteed by the typing discipline when using different types must now be shown explicitly by stability theorems (the sum of two natural numbers is a natural number, for instance). However, the behavior of subtraction is different, and we contend that the behavior we propose is better suited for teaching mathematics.

One of the advantages of using different types for different categories of numbers is to make computation easy when the numbers can be described by an inductive type (which is the case for natural numbers, integers, or rational numbers). We show how this computation capability can be recovered thanks to the meta-programming capabilities of coq-elpi, especially for one-argument functions, which is important to study recursive sequences. An article describing this experiment has been accepted for publication in 2025 [4]. Future work will expand this to multiple-argument functions, for instance making it possible to consider binomial numbers.

Other features of the mathematics language that rely on natural numbers are the description of iterated operations, by applying a function a certain number of times, as in f^n, and finite indexed sums as in $\sum_{i=0}^{n} f(i)$. We are also studying how these features can be made easily available in formalized mathematics while still using the type of real numbers. In particular, this requires that we make it easier to replace sub-formulas containing bound variables (like i in $\sum_{i=0}^{n} f(i)$) by equal formulas. Such a replacement, known as *rewriting under* λ is typically more verbose in formalized mathematics than in educational mathematics. We are working on solutions to reduce this discrepancy, again taking advantage of the meta-programming capabilities of coq-elpi.

6.6 Handling subsets and subtypes in Hierarchy Builder

Participants: Cyril Cohen, Quentin Vermande.

We are experimenting with new design patterns to automate the conversion between sets and types, to automatically prove set membership and to automatically cast between types even when an external proof is required. The result of these experiments will be integrated in Hierarchy Builder in order to

extend its expressiveness, in particular in the formalization of topology, number theory and category theory.

This is ongoing work without any publication yet. Early experiments were presented during meetings of the LiberAbaci and CoREACT projects.

6.7 Automatic functoriality

Participants: Cyril Cohen, Noah Loutchmia *(Gallinette, Inria, Nantes)*, Assia Mahboubi *(Gallinette, Inria, Nantes)*, Vojtěch Štěpančík *(Gallinette, Inria, Nantes)*.

We also work – starting with Noah Loutchmia's internship and continuing with Vojtěch Štěpančík PhD thesis – towards the automated reconstruction of structure from partial descriptions, e.g. reconstructing the definition of morphisms from the description of objects, the morphism part of a functor from the object part, and naturality properties.

6.8 Finite maps

Participants: Cyril Cohen, Quentin Vermande, Enrico Tassi, Reynald Affeldt *(AIST, Japan)*, Georges Gonthier *(Inria Saclay)*, Pierre Roux *(Onera, Toulouse)*, Kazuhiko Sakaguchi *(Galinnette, Inria then LIP, ENS de Lyon)*.

In the current state of the Mathematical Components library, finite sets (represented as finite predicates over some type) are only allowed on finite types (types for which there exists an enumeration function of their elements). This working group is attempting to lift this limitation without breaking compatibility with projects depending on it (more than 500000 lines of publicly known code).

6.9 Formalization of the CAD in the Coq prover

Participants: Quentin Vermande.

As part of his PhD thesis, Quentin Vermande wrote and formally certified Collins' Cylindrical Algebraic Decomposition (CAD) algorithm in Coq. This is one more step towards the efficient and certified implementation of quantifier elimination for the theory of real closed fields. This work will be presented at JFLA 2025 [8] and then submitted for publication at a high-impact international venue.

6.10 Formal study of the Fast Fourier Transform

Participants: Nicolas Brisebarre *(CNRS)*, Laurence Rideau, Laurent Théry.

We have completed our work on formalizing Fast Fourier Transforms by deriving a tactic that proves equality between two integer polynomials. This tactic uses our certified version of polynomial multiplication based on Fourier. The code is available on github/fft. This work was supported by the ANR `Nuscap`.

6.11 Continuous fraction and Ostrowski Expansion

Participants: Laurent Théry.

We have formalized some basic properties of Ostrowski numeration. This lets us to a nice alternative proof of best approximations as given in [13]. For example, such theorems are useful while formalizing algorithms about computer arithmetics as in [12]. The code is available on github/CFRAC.

7 Partnerships and cooperations

7.1 International initiatives

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

FLAVOR

Title: Formal Library of Analysis for the Verification of Robots

Duration: 2020 → 2024

Coordinator: Reynald Affeldt

Partners:

- National Institute of Advanced Industrial Science and Technology Tokyo (Japon)

Inria contact: Yves Bertot

Summary: The objective is to apply formal methods based on Coq to software and designs that are concerned with robots. Covered topics concern mathematical formalization for real analysis, control theory, kinematic chains, and motion planning.

7.2 International research visitors

7.2.1 Visits of international scientists

Inria International Chair

Participants: Robbert Krebbers *(Junior Section)*.

status Professor,

Institution Radboud University, Nijmegen, the Netherlands.

comments Inria International Chair awarded on February 28, 2024. After reviewing the financial conditions, Robbert Krebbers decided to not use the benefits of this International Chair.

7.3 National initiatives

7.3.1 ANR

- NuSCAP "Numerical Safety for Computer-Aided Proofs", started on February 1st, 2021 for 48 months (extended till end 2025), with a grant covering travel costs. Other partners are CNRS-LIP, Sorbonne University LIP6, and CNRS-LAAS. The corresponding researcher for this contract is Laurence Rideau.
- 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 Cyril Cohen.

7.3.2 Inria Challenges

- Liber Abaci. Yves Bertot coordinates the Inria challenge LiberAbaci on the use of a Type-theory based proof assistant to improve mathematics education for the first years of higher education (undergraduate mathematics).

8 Dissemination

Participants: Yves Bertot, Cyril Cohen, Davide Fissore, Enrico Tassi, Laurent Théry, Paolo Torrini.

8.1 Promoting scientific activities

8.1.1 Scientific events: selection

Chair of conference program committees

- Yves Bertot was co-chair of the program committee for the conference "Interactive Theorem Proving" that took place in Tbilisi, Georgia, in September.

Member of the conference program committees

- Cyril Cohen is member of the steering committee of ITP.
- Enrico Tassi is member of the program committee of Coq Workshop 2024, OCaml workshop 2024, CoqPL workshop 2024, and CPP 2024. He is also in the steering committee of LFMTP.

Reviewer

- Laurent Théry did reviews for FUN2024 and ITP2024.

8.1.2 Journal

Reviewer - reviewing activities

- Laurent Théry did reviews for Journal of Automated Reasoning and Logical Methods in Computer Science.

8.1.3 Leadership within the scientific community

- Yves Bertot is the chairman of the Coq Consortium.

8.1.4 Scientific expertise

- Laurent Théry did a review for a Vici research proposal (Netherlands).

8.2 Teaching - Supervision - Juries

8.2.1 Supervision

- Yves Bertot and Cyril Cohen supervise the thesis of Quentin Vermande (Université Côte d'Azur).
- Yves Bertot and Enrico Tassi supervise the thesis of Davide Fissore (Université Côte d'Azur).
- Yves Bertot supervises the thesis of Thomas Portet (Université Côte d'Azur)

8.2.2 Juries

- Laurent Théry was member of the jury for Pablo Donato at École Polytechnique in April.

8.3 Popularization

8.3.1 Specific official responsibilities in science outreach structures

- Laurent Théry is member of the editorial board of Interstices.

9 Scientific production

9.1 Major publications

[1] C. Cohen, K. Sakaguchi and E. Tassi. 'Hierarchy Builder: algebraic hierarchies made easy in Coq with Elpi'. In: FSCD 2020 - 5th International Conference on Formal Structures for Computation and Deduction. 167. Paris, France, 2020, 34:1–34:21. DOI: 10.4230/LIPIcs.FSCD.2020.34. URL: https://inria.hal.science/hal-02478907.

[2] J.-M. Muller and L. Rideau. 'Formalization of double-word arithmetic, and comments on "Tight and rigorous error bounds for basic building blocks of double-word arithmetic"'. In: *ACM Transactions on Mathematical Software* 48.1 (Mar. 2022), pp. 1–24. DOI: 10.1145/3484514. URL: https://hal.science/hal-02972245.

[3] E. Tassi. 'Elpi: an extension language for Coq (Metaprogramming Coq in the Elpi λProlog dialect)'. In: The Fourth International Workshop on Coq for Programming Languages. Los Angeles (CA), United States, 13th Jan. 2018. URL: https://inria.hal.science/hal-01637063.

9.2 Publications of the year

International peer-reviewed conferences

[4] 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. 9).

[5] C. Cohen, E. Crance and A. Mahboubi. 'Artifact Report: Trocq: Proof Transfer for Free, With or Without Univalence'. In: *Lecture Notes in Computer Science*. ESOP 2024 - 33rd European Symposium on Programming. Vol. 14576. Luxembourg, Luxembourg: Springer Nature Switzerland, 5th Apr. 2024, pp. 269–274. DOI: 10.1007/978-3-031-57262-3_11. URL: https://inria.hal.science/hal-04623207.

[6] C. Cohen, E. Crance and A. Mahboubi. 'Trocq: Proof Transfer for Free, With or Without Univalence'. In: *Lecture Notes in Computer Science*. ESOP 2024 - 33rd European Symposium on Programming. Vol. 14576. Lecture Notes in Computer Science. Luxembourg, Luxembourg: Springer Nature Switzerland, 2024, pp. 239–268. DOI: 10.1007/978-3-031-57262-3_10. URL: https://hal.science/hal-04177913 (cit. on p. 8).

[7] D. Fissore and E. Tassi. 'Higher-Order unification for free!: Reusing the meta-language unification for the object language'. In: PPDP 2024: 26th International Symposium on Principles and Practice of Declarative Programming. Milan, Italy: ACM; ACM, 2024, pp. 1–13. DOI: 10.1145/3678232.3678233. URL: https://inria.hal.science/hal-04547069 (cit. on p. 8).

[8] Q. Vermande. 'Décomposition Algébrique Cylindrique en Coq/Rocq'. In: 36es Journées Francophones des Langages Applicatifs (JFLA 2025). Roiffé, France, 28th Jan. 2025. URL: https://inria.hal.science/hal-04859512 (cit. on p. 10).

Scientific book chapters

[9] Y. Bertot. 'Safe smooth paths between straight line obstacles'. In: *Logics and Type Systems in Theory and Practice*. Vol. LNCS-14560. Lecture Notes in Computer Science. Springer, May 2024, pp. 36–53. DOI: 10.1007/978-3-031-61716-4_3. URL: https://inria.hal.science/hal-04312815 (cit. on p. 9).

Reports & preprints

[10] T. Hubrecht, C.-P. Jeannerod, P. Zimmermann, L. Rideau and L. Théry. *Towards a correctly-rounded and fast power function in binary64 arithmetic*. 8th Feb. 2024. URL: https://inria.hal.science/hal-04159652.

[11] Q. Vermande. *Décomposition Algébrique Cylindrique en Coq/Rocq*. 30th Oct. 2024. URL: https://hal.science/hal-04760627.

9.3 Cited publications

[12] A. Blot, J. Muller and L. Théry. 'Formal Correctness of Comparison Algorithms Between Binary64 and Decimal64 Floating-Point Numbers'. In: *NSV*. Ed. by A. Abate and S. Boldo. Vol. 10381. LNCS. Springer, 2017, pp. 25–37 (cit. on p. 11).

[13] A. M. Rockett and P. Szusz. *Continued fractions*. World Scientific, 1992 (cit. on p. 11).