RESEARCH CENTRE

Inria Saclay Centre at Université Paris-Saclay

IN PARTNERSHIP WITH:

CNRS, Université Paris-Saclay

2024

ACTIVITY REPORT

Project-Team TOCCATA

Certified Programs, Certified Tools, Certified Floating-Point Computations

IN COLLABORATION WITH: Laboratoire de Méthodes Formelles

DOMAIN

Algorithmics, Programming, Software and Architecture

THEME

Proofs and Verification

Inria

Contents

Project-Team TOCCATA

Creation of the Project-Team: 2014 July 01

Keywords

Computer sciences and digital sciences

A2.1.1. – Semantics of programming languages

A2.1.4. – Functional programming

A2.1.6. – Concurrent programming

A2.1.10. – Domain-specific languages

A2.1.11. – Proof languages

A2.4.2. – Model-checking

A2.4.3. – Proofs

A6.2.1. – Numerical analysis of PDE and ODE

A7.2. – Logic in Computer Science

A7.2.1. – Decision procedures

A7.2.2. – Automated Theorem Proving

A7.2.3. – Interactive Theorem Proving

A7.2.4. – Mechanized Formalization of Mathematics

A8.10. – Computer arithmetic

Other research topics and application domains

B5.2.2. – Railway

B5.2.3. – Aviation

B5.2.4. – Aerospace

B6.1. – Software industry

B9.5.1. – Computer science

B9.5.2. – Mathematics

1 Team members, visitors, external collaborators

Research Scientists

- Claude Marché [Team leader, INRIA, Senior Researcher, HDR]
- Sylvie Boldo [INRIA, Senior Researcher, HDR]
- Jean-Christophe Filliâtre [CNRS, Senior Researcher, HDR]
- Armaël Guéneau [INRIA, Researcher]
- Guillaume Melquiond [INRIA, Senior Researcher, HDR]

Faculty Members

- Sylvain Conchon [UNIV PARIS SACLAY, Professor, HDR]
- Andrei Paskevich [UNIV PARIS SACLAY, Associate Professor]

PhD Students

- Léo Andrès [OCamlPro, CIFRE]
- Paul Geneau De Lamarlière [Mitsubishi Electric R&D Centre Europe, CIFRE]
- Arnaud Golfouse [INRIA, from May 2024]
- Arnaud Golfouse [MESR, until Apr 2024]
- David Hamelin [EDF, CIFRE, from Dec 2024]
- Josue Moreau [INRIA]
- Houda Mouhcine [INRIA]
- Paul Patault [UNIV PARIS-SACLAY]

Technical Staff

- Paul Bonnot [INRIA, Engineer, until Jul 2024]
- David Hamelin [INRIA, Engineer, until Nov 2024]
- Matteo Manighetti [INRIA, Engineer]
- Li-Yao Xia [INRIA, Engineer, from Sep 2024]

Interns and Apprentices

- Gurvan Debaussart [INRIA, Intern, from Mar 2024 until Aug 2024]
- Valeran Maytie [INRIA, Intern, from Mar 2024 until Aug 2024]

Administrative Assistant

- Joyce Soares Brito [INRIA]

Visiting Scientists

- Florent Hivert [UNIV PARIS-SACLAY, from Sep 2024, Délégation Inria, HDR]
- Micaela Mayero [UNIV PARIS-NORD, Délégation Inria, HDR]

External Collaborators

- Thibaut Balabonski [UNIV PARIS-SACLAY]
- Jacques-Henri Jourdan [CNRS]
- Chantal Keller [UNIV PARIS-SACLAY]

2 Overall objectives

The general objective of the Toccata project is to promote formal specification and computer-assisted proof in the development of software that requires high assurance in terms of safety and correctness with respect to its intended behavior. Such safety-critical software appears in many application domains like transportation (*e.g.* aviation, aerospace, railway, automotive), communication (*e.g.* internet, smartphones), health devices, data management on clouds (confidentialty issues), etc. The number of tasks performed by software is quickly increasing, together with the number of lines of code involved. Given the need of high assurance of safety in the functional behavior of such applications, the need for automated (in the sense computer-assisted) methods and techniques to bring guarantee of safety became a major challenge. In the past and at present, the most widely used approach to check safety of software is to apply heavy test campaigns, which take a large part of the costs of software development. Yet these campaigns cannot ensure that all the bugs are caught, and remaining bugs may have catastrophic consequences.

Generally speaking, software verification approaches pursue three goals: (1) verification should be sound, in the sense that no bugs should be missed, (2) verification should not produce false alarms, or as few as possible, (3) it should be as automatic as possible. Reaching all three goals at the same time is a challenge. A large class of approaches emphasizes goals (2) and (3): testing, run-time verification, symbolic execution, model checking, etc. Static analysis, such as abstract interpretation, emphasizes goals (1) and (3). Deductive verification emphasizes (1) and (2). The Toccata project is mainly interested in exploring the deductive verification approach, although we also combine with the other techniques occasionally.

In the past decade, significant progress has been made in the domain of deductive program verification. This is emphasized by some success stories of application of these techniques on industrial-scale software. For example, the *Atelier B* system was used to develop part of the embedded software of the Paris metro line 14 [44] and other railway-related systems; a formally proved C compiler was developed using the Coq proof assistant [71]; the L4-verified project developed a formally verified micro-kernel with high security guarantees, using analysis tools on top of the Isabelle/HOL proof assistant [70]. A bug in the JDK implementation of TimSort was discovered using the KeY environment [67] and a fixed version was proved sound. Another sign of recent progress is the emergence of deductive verification competitions (*e.g.* VerifyThis [45]). Finally, recent trends in the industrial practice for development of critical software is to require more and more guarantees of safety, *e.g.* the DO-178C standard for developing avionics software adds to the former DO-178B the use of formal models and formal methods. It also emphasizes the need for certification of the analysis tools involved in the process.

3 Research program

Panorama of Deductive Verification There are two main families of approaches for deductive verification. Methods in the first family build on top of mathematical proof assistants (*e.g.* Coq, Isabelle) in which both the model and the program are encoded; the proof that the program meets its specification is typically conducted in an interactive way using the underlying proof construction engine. Methods from the second family proceed by the design of standalone tools taking as input a program in a particular programming language (*e.g.* C, Java) specified with a dedicated annotation language (*e.g.* ACSL [29], JML [56]) and automatically producing a set of mathematical formulas (the *verification conditions*) which are typically proved using automatic provers (*e.g.* Z3 [74], Alt-Ergo [58], CVC5 [43]).

The first family of approaches usually offers a smaller *Trusted Code Base* (TCB) than the second, but also demands more work to perform the proofs (because of their interactive nature) and makes them less easy to adopt by industry. Moreover, they generally do not allow to directly analyze a program written

in a mainstream programming language like Java or C. The second kind of approaches has benefited in the past years from the tremendous progress made in SAT and SMT solving techniques, allowing more impact on industrial practices, but suffers from a lower level of trust: in all parts of the proof chain (the model of the input programming language, the VC generator, the back-end automatic prover), potential errors may appear, compromising the guarantee offered. Moreover, while these approaches are applied to mainstream languages, they usually support only a subset of their features.

Overall Goals of the Toccata Project One of our original skills is the ability to conduct proofs by using automatic provers and proof assistants at the same time, depending on the difficulty of the program, and specifically the difficulty of each particular verification condition. We thus believe that we are in a good position to propose a bridge between the two families of approaches of deductive verification presented above. Establishing this bridge is one of the goals of the Toccata project: we want to provide methods and tools for deductive program verification that can offer both a high amount of proof automation and a high guarantee of validity.

In industrial applications, numerical calculations are very common (e.g. control software in transportation). Typically they involve floating-point numbers. Some of the members of Toccata have an internationally recognized expertise on deductive program verification involving floating-point computations. Our past work includes a new approach for proving behavioral properties of numerical C programs using Frama-C/Jessie [39], various examples of applications of that approach [52], the use of the Gappa solver for proving numerical algorithms [63], an approach to take architectures and compilers into account when dealing with floating-point programs [54, 76]. We also contributed to the Handbook of Floating-Point Arithmetic [75] and co-published a survey on floating-point arithmetic [4]. A representative case study is the analysis and the proof of both the method error and the rounding error of a numerical analysis program solving the one-dimension acoustic wave equation [48] [47]. Our experience led us to a conclusion that verification of numerical programs can benefit a lot from combining automatic and interactive theorem proving [51, 52, 65, 66]. Verification of numerical programs is another main axis of Toccata.

Let us conclude with more general considerations: we want to keep on with general audience actions and industrial transfer through sustained long-term collaboration with industrial partners (Section 4). Our scientific programme detailed below is structured into the following four axes.

1. Foundations and spreading of deductive program verification;
2. Reasoning on mutable memory in program verification;
3. Verification of Computer Arithmetic;
4. Spreading Formal Proofs.

3.1 Foundations and spreading of deductive program verification

This axis covers the fundational studies we pursue regarding deductive verification. A non-exhaustive list of subjects we want to address is as follows.

- The search for improved methods to generate verification conditions, relying for example on new calculi, on better notion of abstraction, or on automatic discovery of invariants.
- Uniform approaches to obtain correct-by-construction programs and libraries, in particular by automatic extraction of executable code (in OCaml, C, CakeML, etc.) from verified programs, and including innovative general methods like advanced ghost code, ghost monitoring, etc. A representative publication is the presentation of a new notion called ghost monitors [57].
- Improvement of automated reasoning techniques: methods dedicated to deductive verification, so as to improve proof automation; improved combination of interactive provers and fully automated ones, proof by reflection.
- Providing feedback in case of proof failures, *e.g.* based on generation of counterexamples, or symbolic execution.

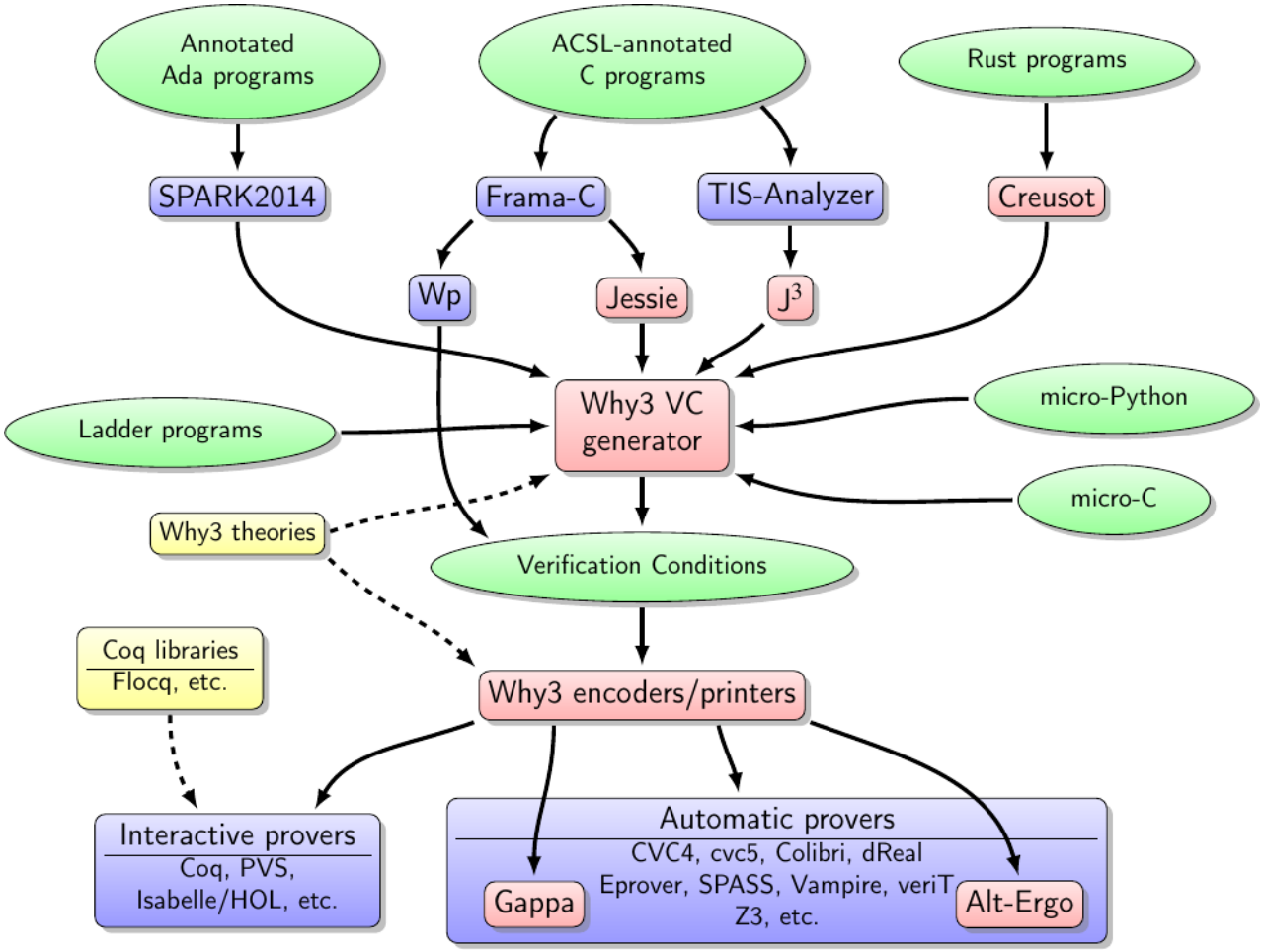

Figure 1: The Why3 ecosystem in 2024.

A significant part of the work achieved in this axis is related to the Why3 toolbox and its ecosystem, displayed on Figure 1. The red background boxes represent tools that we develop ourselves, whereas blue background ones are developed by others. SPARK2014 is developed by AdaCore. Frama-C and Wp are developed by CEA-list and directly produce logical formulas to be passed to provers. TIS-Analyzer is developed by TrustInSoft and J^3 is a collaboration between TrustInSoft and us. We develop the frontends micro-C and micro-Python mainly for teaching purpose. The front-end for Ladder programs is a software developed internally by MERCE. Yellow background boxes represent libraries of specifications of logic datatypes with their logical properties. A representative publication is an article on abstraction and genericity features of Why3 [7].

3.2 Reasoning on mutable memory in program verification

This axis concerns specifically the techniques for reasoning on programs where memory aliasing is the central issue. It covers the methods based on type-based alias analysis and related memory models, on specific program logics such as separation logics, and extended model-checking. It concerns the application on analysis of C or C++ codes, on Ada codes involving pointers, but also concurrent programs in general. The main topics are:

- The study of advanced type systems dedicated to verification, for controlling aliasing, and their use for obtaining easier-to-prove verification conditions. Modern typing systems in the style of Rust, involving ownership and borrowing, are considered. A representation publication is a paper [9] on the semantic foundation of the verification of Rust programs.

- The design of front-ends of Why3 for the proofs of programs where aliasing cannot be fully controlled statically, via adequate memory models, aiming in particular at extraction to C; and also for concurrent programs.

- The continuation of fruitful work on concurrent parameterized systems, and its corresponding specific SMT-based model-checking. Reference publication are [5] and [6].

There are also other various topics considered in this axis, for example around capability machines, which denote a type of CPU allowing fine-grained privilege separation using *capabilities*, with machine words that represent certain kinds of authority. Guéneau *et al.* [8] present a mathematical model and accompanying proof methods that can be used for formal verification of functional correctness of programs running on a capability machine, even when they invoke and are invoked by unknown (and possibly malicious) code. Other topics concern reasoning on ressources [17], and cross-language verification [68].

3.3 Verification of Computer Arithmetic

This axis, which bridges the domains of computer arithmetic and of formal verification, is a major originality of Toccata. The main topics are as follows.

- We are studying the fundamental blocks of formalizing floating-point computations, algorithms, and error analysis.

- A significant effort is dedicated to verification of numerical programs written in mainstream languages such as C or Ada. This involves combining specifications in real numbers and computation in floating-point, and underlying automated reasoning techniques with floating-point numbers and real numbers. We also contributed to the automation of reasoning on floating-point numbers [59].

- Related to the formalization of mathematics, we aim at verifying numerical analysis programs, in particular numerical schemes for solving partial differential equations. A representative publication is a paper on the formalization of Lebesgue integration [3] and a paper on certified approximations of integrals [72].

Boldo and Melquiond are authors of a reference book [53] on the formal verification of numerical programs.

3.4 Spreading Formal Proofs

The general goal of this axis, which was a new one proposed in 2019, was to encourage spreading of deductive verification through actions showing how our methods and tools can be used on programs that we develop ourselves. Since this axis is dedicated to applications in a general manner, positioning barely makes sense since a vast majority of research groups in computer science in the world would claim to conduct case studies and large-scale applications.

Representative of these significant case studies are the automated analysis of Debian packages installation [1], a certified library for arbitrary-precision arithmetic [10], and the automated analysis of Ladder programs [2].

4 Application domains

4.1 Industrial Transfer Actions

The application domains we target involve safety-critical software, that is where a high-level guarantee of soundness of functional execution of the software is wanted. Currently our industrial collaborations or impact mainly belong to the domain of transportation: aerospace, aviation, railway, automotive.

Transfer to the community of Atelier B in the context of the FUI project LCHIP, we investigated the use of Why3 and Alt-Ergo as an alternative back-end for checking proof obligations generated by Atelier B, whose main applications are railroad-related.

ProofInUse-AdaCore collaboration: transfer to the community of Ada development Since the creation of the ProofInUse joint lab in 2014, with AdaCore company, we have a growing impact on the community of industrial development of safety-critical applications written in Ada. See that web page for a an overview of AdaCore's customer projects, in particular those involving the use of the SPARK Pro tool set. This impact involves both the use of Why3 for generating VCs on Ada source codes, and the use of Alt-Ergo for performing proofs of those VCs. This action allowed AdaCore company to get new customers, in particular the domains of application of deductive formal verification are from the historical domain of aerospace (e.g. this link or this link) but went beyond: application in automotive (e.g. Denso, Toyata), medical and security (e.g. Nvidia). A joint publication of Nvidia and AdaCore [36] in 2023 exposes, with their own words, the benefit of using high level verification for securing Nvidia chips.

ProofInUse-TrustInSoft collaboration In 2017 we started to collaborate with the TrustInSoft company for the verification of C and C++ codes. We started with a CIFRE thesis funding, which explored the use of Why3 to design verified and reusable C libraries [77], and then with a bilateral contract towards the design of the J^3 plugin in TIS-Analyzer, bringing deductive verification techniques in this platfrom, including counterexamples when proofs fail. The impact on TrustInSoft customers is not yet easily identifiable; it will hopefully increase in particular in the context of the new project Décysif led by TrustInSoft.

ProofInUse-MERCE collaboration In 2019 we started to collaborate with Mitsubishi Electric R&D Centre Europe in Rennes, France. The R&D programme is two-fold: first the verification of Ladder programs for PLCs, second the verification of numerical C codes. MERCE has now a mature platform for Ladder verification, which has yet to be made really usable by development teams. This work received the FMICS best paper award in 2021. The work of numerical programs is increasing in importance. We have results on log-sum-exp functions [55, 12] and a CIFRE thesis started in 2023, aiming at designing better proof environments for verifying programs with complex numeric computations. A patent entitled *Automatic implementation of formally-verified numerical programs* has been filled in 2022 at EPO. A new axis of collaboration with MERCE started in 2024, regarding the validation of invariants on data representing railway networks.

CIFRE thesis with Tarides The CIFRE thesis of Clément Pascutto with Tarides, in 2020–2023, brought mature tooling for verifying function contracts and invariants on OCaml at runtime. The resulting

tool, `ortac`, efficiently addresses the problem of capturing prestates in order to evaluation function postconditions [64]. Tarides continues the development of `ortac` and uses it on its own code base.

CIFRE thesis with OCamlPro The CIFRE thesis of Léo Andrès with OCamlPro, in 2021–2024, targets the compilation of OCaml to WebAssembly (Wasm for short), as an alternative to its compilation to JavaScript. It requires some extensions to Wasm, such as Wasm-GC, and the thesis already confirmed the adequacy of such extensions [38]. A by-product of the thesis is the implementation of a new, efficient interpreter for Wasm, `owi`.

Generally speaking, we believe that our increasing industrial impact is a representative success for our general goal of spreading deductive verification methods to a larger audience, and we are firmly engaged into continuing such kind of actions in the next years.

4.2 Other socio-economic impact

We believe our impact is not limited to industrial actions per se.

A first point is that during the years, the young students that we train, either as a PhD position or a temporary engineer positions, easily got positions in private companies. Indeed we believe we can say that we contributed to the creation of jobs in several companies.

Another important part of our social impact is our work with high school students. With new curricula including more computer science than ever before, it was important to provide good reference books. With this in mind, we have contributed three books aimed at high school and preparatory school students [41, 40, 42].

The impact is not limited to books: we also helped a teacher to design a lesson to learn the basic notions of program verification (say: loop invariants) using the Why3 tool (article IREMI). We are also part each year of stands at "Fête de la science" in November or special events towards girls. We also often go to (high) schools for presenting either our job or our research (except during the Covid pandemic).

The social impact in national education is finally made highly evident by our implication in the organization of the new *agrégation d'informatique* which is in charge to select and recruit the best high-level teachers for the new programmes.

5 Social and environmental responsibility

5.1 Footprint of research activities

Our research activities make use of standard computers for developing software and developing formal proofs. We have no use of specific large size computing resources. Though, we are making use of external services for *continuous integration*. A continuous integration methodology for mature software like Why3 is indeed mandatory for ensuring a safe software engineering process for maintenance and evolution. We make the necessary efforts to keep the energy consumption of such a continuous integration process as low as possible.

Ensuring the reproducibility of proofs in formal verification is essential. It is thus mandatory to replay such proofs regularly to make sure that our changes in our software do not loose existing proofs. For example, we need to make sure that the case studies in formal verification that we present in our gallery are reproducible. We also make the necessary efforts to keep the energy consumption for replaying proofs low, by doing it only when necessary.

As widely accepted nowadays, the major sources of environmental impact of research is travel to international conferences by plane, and renewal of electronic devices. The number of travels we made in 2022 remained very low with respect to previous years, of course because of the Covid pandemic, and the fact that many conferences were now proposed online participation. We intend to continue limiting the environmental impact of our travels. Concerning renewal of electronic devices, that is mainly laptops and monitors, we have always been careful on keeping them usable for as long time as possible.

5.2 Impact of research results

Our research results aims at improving the quality of software, in particular in mission-critical contexts. As such, making software safer is likely to reduce the necessity for maintenance operations and thus reducing energy costs.

Our efforts are mostly towards ensuring the safety of functional behavior of software, but we also increasingly consider the verification of their time or memory consumption. Reducing those would naturally induce a reduction in energy consumption.

Our research never involve any processing of personal data, and consequently we have no concern about preserving individual privacy, and no concern with respect to the RGPD (*Règlement Général sur la Protection des Données*).

In 2024, S. Boldo was in the program committee of the first PROPL workshop (Programming for the Planet) to see how we may help topics such as climate analysis, modelling, forecasting, policy, and diplomacy.

6 Highlights of the year

6.1 Awards

Xavier Denis was awarded the "prix de thèse 2023 du GdR Génie de la Programmation et Logiciel", during the national days of that GdR in June 2024. Xavier is a former PhD student in our team, supervised by Jacques-Henri Jourdan and Claude Marché. His thesis is about Deductive Verification for Rust Programs [61].

6.2 Institutional Life

Note : Readers are advised that the Institute does not endorse the text in the "Highlights of the year" section, which is the sole responsibility of the team leader.

At the end of 2024, Inria's top management enacted a new "contrat d'objectifs, de moyens et de performance" (COMP), which defines Inria's objectives for the period 2024–2028. **We are very unhappy and concerned** about the content of this document and the way it was imposed.

- Neither the staff nor their representative bodies were given the opportunity to participate in (or influence) the drafting of this document.
- The document defines Inria's main mission as "contributing to the digital sovereignty of the Nation through research and innovation" and proposes to amend Inria's founding decree to reflect this new definition. We strongly believe that our primary mission is (and should remain) the advancement of human knowledge through research. Research is not a means to achieve "digital sovereignty", whatever that may mean. Research should not be associated with any particular nation, whatever that nation may be.
- The document announces the creation of a funding agency within Inria. France already has an independent funding agency, the ANR. The creation of a new funding agency within a research institute is unnecessary and a waste of resources. It is also likely to create confusion, opacity, and conflicts of interest.
- Many aspects of the document reflect a desire to drive research in a top-down manner, for example through the selection of "strategic partner institutions" and "strategic themes". This threatens the fundamental freedom of researchers to choose their research topics and collaborations.
- The document indicates that all of Inria's research should have "dual nature", that is, both civilian and military applications. While some of the institute's research may have military applications, the vast majority of it is independent of the military, and should remain so.

- The document announces a desire to place all of Inria in a “restricted regime area” (ZRR), which means that the hiring of researchers and interns will be reviewed and possibly vetoed by the Fonctionnaire Sécurité Défense. This creates administrative delays, subjects hiring to opaque criteria, and discourages the hiring of foreign nationals, thus harming research and collaboration.
- Staff opposition to these policies, which has been expressed in several votes and petitions, has been largely ignored.

7 New software, platforms, open data

7.1 New software

7.1.1 Alt-Ergo

Name: Automated theorem prover for software verification

Keywords: Software Verification, Automated theorem proving

Functional Description: Alt-Ergo is an automatic solver of formulas based on SMT technology. It is especially designed to prove mathematical formulas generated by program verification tools, such as Frama-C for C programs, or SPARK for Ada code. Initially developed in Toccata research team, Alt-Ergo's distribution and support are provided by OCamlPro since September 2013.

Release Contributions: the "SAT solving" part can now be delegated to an external plugin, new experimental SAT solver based on mini-SAT, provided as a plugin. This solver is, in general, more efficient on ground problems, heuristics simplification in the default SAT solver and in the matching (instantiation) module, re-implementation of internal literals representation, improvement of theories combination architecture, rewriting some parts of the formulas module, bugfixes in records and numbers modules, new option "-no-Ematching" to perform matching without equality reasoning (i.e. without considering "equivalence classes"). This option is very useful for benchmarks coming from Atelier-B, two new experimental options: "-save-used-context" and "-replay-used-context". When the goal is proved valid, the first option allows to save the names of useful axioms into a ".used" file. The second one is used to replay the proof using only the axioms listed in the corresponding ".used" file. Note that the replay may fail because of the absence of necessary ground terms generated by useless axioms (that are not included in .used file) during the initial run.

URL: https://alt-ergo.ocamlpro.com/

Contact: Sylvain Conchon

Participants: Alain Mebsout, Évelyne Contejean, Mohamed Iguernelala, Stephane Lescuyer, Sylvain Conchon

Partner: OCamlPro

7.1.2 CoqInterval

Name: Interval package for Coq

Keywords: Interval arithmetic, Coq

Functional Description: CoqInterval is a library for the proof assistant Coq.

It provides several tactics for proving theorems on enclosures of real-valued expressions. The proofs are performed by an interval kernel which relies on a computable formalization of floating-point arithmetic in Coq.

The Marelle team developed a formalization of rigorous polynomial approximation using Taylor models in Coq. In 2014, this library has been included in CoqInterval.

URL: https://coqinterval.gitlabpages.inria.fr/

Publications: hal-04114233, hal-04515714, hal-04702129, hal-03168208, tel-02194683, hal-04859533, hal-00180138, hal-00797913, hal-01086460, hal-01289616, hal-01630143

Contact: Guillaume Melquiond

Participants: Pierre Roux, Paul Geneau De Lamarliere, Assia Mahboubi, Erik Martin Dorel, Guillaume Melquiond, Jean-Michel Muller, Laurence Rideau, Laurent Théry, Micaela Mayero, Mioara Joldes, Nicolas Brisebarre, Thomas Sibut-Pinote

7.1.3 Coquelicot

Name: The Coquelicot library for real analysis in Coq

Keywords: Coq, Real analysis

Functional Description: Coquelicot is library aimed for supporting real analysis in the Coq proof assistant. It is designed with three principles in mind. The first is the user-friendliness, achieved by implementing methods of automation, but also by avoiding dependent types in order to ease the stating and readability of theorems. This latter part was achieved by defining total function for basic operators, such as limits or integrals. The second principle is the comprehensiveness of the library. By experimenting on several applications, we ensured that the available theorems are enough to cover most cases. We also wanted to be able to extend our library towards more generic settings, such as complex analysis or Euclidean spaces. The third principle is for the Coquelicot library to be a conservative extension of the Coq standard library, so that it can be easily combined with existing developments based on the standard library.

URL: http://coquelicot.saclay.inria.fr/

Publications: tel-01228517, hal-00860648, hal-00712938, hal-00880212, hal-01169321, hal-00642206

Contact: Guillaume Melquiond

Participants: Catherine Lelay, Guillaume Melquiond, Sylvie Boldo

7.1.4 Cubicle

Name: The Cubicle model checker modulo theories

Keywords: Model Checking, Software Verification

Functional Description: Cubicle is an open source model checker for verifying safety properties of array-based systems, which corresponds to a syntactically restricted class of parametrized transition systems with states represented as arrays indexed by an arbitrary number of processes. Cache coherence protocols and mutual exclusion algorithms are typical examples of such systems.

URL: https://github.com/cubicle-model-checker/cubicle

Contact: Sylvain Conchon

Participants: Alain Mebsout, Sylvain Conchon

7.1.5 Flocq

Name: The Flocq formalization of floating-point arithmetic for the Coq proof assistant

Keyword: Floating-point

Functional Description: The Flocq library for the Coq proof assistant is a comprehensive formalization of floating-point arithmetic: core definitions, axiomatic and computational rounding operations, high-level properties. It provides a framework for developers to formally verify numerical applications.

Flocq is currently used by the CompCert verified compiler to support floating-point computations.

URL: https://flocq.gitlabpages.inria.fr/

Publications: hal-03233227, inria-00534854, hal-00743090, hal-00862689, hal-01091186, hal-01091189, hal-01632617

Contact: Guillaume Melquiond

Participants: Guillaume Melquiond, Pierre Roux, Sylvie Boldo

7.1.6 Gappa

Name: The Gappa tool for automated proofs of arithmetic properties

Keywords: Floating-point, Arithmetic code, Software Verification, Constraint solving

Functional Description: Gappa is a tool intended to help formally verifying numerical programs dealing with floating-point or fixed-point arithmetic. It has been used to write robust floating-point filters for CGAL and it is used to verify elementary functions in CRlibm. While Gappa is intended to be used directly, it can also act as a backend prover for the Why3 software verification plateform or as an automatic tactic for the Coq proof assistant.

URL: https://gappa.gitlabpages.inria.fr/

Publications: hal-03233227, tel-02194683, inria-00070739, inria-00344518, inria-00070330, tel-01094485, inria-00071232, inria-00432726, ensl-00379167, ensl-00200830, hal-01110666, hal-01110669, hal-01632617

Contact: Guillaume Melquiond

Participants: Guillaume Melquiond, Tom Hubrecht

7.1.7 Why3

Name: The Why3 environment for deductive verification

Keywords: Formal methods, Trusted software, Software Verification, Deductive program verification

Functional Description: Why3 is an environment for deductive program verification. It provides a rich language for specification and programming, called WhyML, and relies on external theorem provers, both automated and interactive, to discharge verification conditions. Why3 comes with a standard library of logical theories (integer and real arithmetic, Boolean operations, sets and maps, etc.) and basic programming data structures (arrays, queues, hash tables, etc.). A user can write WhyML programs directly and get correct-by-construction OCaml programs through an automated extraction mechanism. WhyML is also used as an intermediate language for the verification of C, Java, or Ada programs.

URL: https://www.why3.org/

Contact: Claude Marche

Participants: Andriy Paskevych, Claude Marche, François Bobot, Guillaume Melquiond, Jean-Christophe Filliâtre, Levs Gondelmans, Martin Clochard

Partners: CNRS, Université Paris-Sud

7.1.8 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 8.16 integrates changes to the Coq kernel and performance improvements along with a few new features. See the detailed changes at https://coq.inria.fr/refman/changes.html#version-8-16 for an overview of the new features and changes, along with the full list of contributors.

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

7.1.9 creusot

Name: Creusot

Keywords: Rust, Specification language, Deductive program verification

Functional Description: Creusot is a tool for deductive verification of Rust code. It allows you to annotate your code with specifications, invariants and assertions and then verify them formally and automatically, proving, mathematically, that your code satisfies your specifications.

Creusot works by translating Rust code to WhyML, the verification and specification language of Why3. Users can then leverage the full power of Why3 to (semi)-automatically discharge the verification conditions.

Release Contributions: This is the first version, providing the main process to go from a Rust program annotated with Pearlite specifications to a set of verifications conditions to be discharged by external SMT solvers.

URL: https://github.com/xldenis/creusot/

Publications: hal-03737878, hal-03526634, hal-02962804

Contact: Xavier Denis

Participants: Xavier Denis, Jacques-Henri Jourdan, Claude Marche

Partners: Université Paris-Saclay, CNRS

7.1.10 coq-num-analysis

Name: Numerical analysis Coq library

Keywords: Formal methods, Coq, Numerical analysis, Finite element modelling

Scientific Description: These Coq developments are based on the Coquelicot library for real analysis. Version 1.0 includes the formalization and proof of: (1) the Lax-Milgram theorem, including results from linear algebra, geometry, functional analysis and Hilbert spaces, (2) the Lebesgue integral, including large parts of the measure theory,the building of the Lebesgue measure on real numbers, integration of nonnegative measurable functions with the Beppo Levi (monotone convergence) theorem, Fatou's lemma, the Tonelli theorem, and the Bochner integral with the dominated convergence theorem.

Functional Description: Formal developments and proofs in Coq of numerical analysis problems. The current long-term goal is to formally prove parts of a C++ library implementing the Finite Element Method.

News of the Year: The formalization in Coq of simplicial Lagrange finite elements is complete. This include the formalizations of the definitions and main properties of monomials, their representation using multi-indices, Lagrange polynomials, the vector space of polynomials of given maximum degree (about 6 kloc). This also includes algebraic complements on the formalization of the definitions and main properties of operators on finite families of any type, the specific cases of abelian monoids (sum), vector spaces (linear combination), and affine spaces (affine combination, barycenter, affine mapping), sub-algebraic structures, and basics of finite dimension linear algebra (about 31 kloc). A new version (2.0) of the opam package will be available soon, and a paper will follow.

We have also contributed to the Coquelicot library by adding the algebraic structure of abelian monoid, which is now the base of the hierarchy of canonical structures of the library.

URL: https://lipn.univ-paris13.fr/coq-num-analysis/

Publications: hal-01344090, hal-01391578, hal-03105815, hal-03471095, hal-03516749, hal-03889276, hal-04713897, tel-04884651

Contact: Sylvie Boldo

Participants: Sylvie Boldo, François Clement, Micaela Mayero, Vincent Martin, Stéphane Aubry, Florian Faissole, Houda Mouhcine, Louise Leclerc

7.2 Open data

Toccata's Gallery of Verified Programs

Contributors: Ali Ayad, Andrei Paskevich, Asma Tafat, Benedikt Becker, Christine Paulin-Mohring, Claire Dross, Claude Marché, Cláudio Belo Lourenço, Clément Fumex, François Bobot, Guillaume Melquiond, Jean-Christophe Filliâtre, Josué Moreau, Leon Gondelman, Léo Andrès, Martin Clochard, Mário Pereira, Nicolas Jeannerod, Paul Bonnot, Paul Patault, Quentin Garchery, Ran Chen, Raphaël Rieu-Helft, Romain Bardou, Sylvain Dailler, Sylvie Boldo, Thi Minh Tuyen Nguyen, Xavier Denis, Yannick Moy, Yuto Takei

Description: This data set is a collection of programs formally verified, performed by tools developed in our team. The programs range from simple representative code, as good examples for teaching, to large case studies. These also include solutions to the successive VerifyThis challenges that appeared during the recent years.

Project link: Toccata Gallery

Publications: [45], see also all references given inside the data set itself

Contact: Claude Marché, Jean-Christophe Filliâtre

Release contributions: The current version on the web is synchronised with Why3 version 1.8.0 released in December 2024

8 New results

8.1 Foundations and Spreading of Deductive Program Verification

Participants: Andrei Paskevich, Claude Marché, Guillaume Melquiond, Jean-Christophe Filliâtre, Léo Andrès, Sylvain Conchon, Paul Patault, Henri Saudubray, Matteo Manighetti.

Improving Verification Condition Generation Filliâtre, Paskevich and Patault introduce Coma, a formally defined intermediate verification language. Specification annotations in Coma take the form of assertions mixed with the executable program code. A special programming construct representing the abstraction barrier is used to separate, inside a subroutine, the "interface" part of the code, which is verified at every call site, from the "implementatio" part, which is verified only once, at the definition site. In comparison with traditional contract-based specification, this offers the user an additional degree of freedom, as they can provide separate specification (or none at all) for different execution paths. They define a verification condition generator for Coma and prove its correctness. For programs where specification is given in a traditional way, with abstraction barriers at the function entries and exits, the verification conditions are similar to the ones produced by a classical weakest-precondition calculus. For programs where abstraction barriers are placed in the middle of a function definition, the user-written specification is seamlessly completed with the verification conditions generated for the exposed part of the code. In addition, their procedure can factorize selected subgoals on the fly, which leads to more compact verification conditions. They illustrate the use of Coma on two non-trivial examples, which have been formalized and verified using their implementation: a second-order regular expression engine and a sorting algorithm written in unstructured assembly code. An article presenting this work is accepted for presenting at the ESOP Conference in 2025 [34].

Inference of Invariants The automatic discovery of invariants is an important topic to increase the automation of deductive verification. A fully automatic generation of invariants was studied in collaboration with an industrial partner: Cousineau and Marché [20] devised an original abstract interpretation based approach using a domain of parametrized binary decision diagrams. This includes an application to the verification of Ladder programs, discussed below in axis 4.

Formal Specification Language for C code ACSL, short for ANSI/ISO C Specification Language [29], is meant to express precisely and unambiguously the expected behavior of a piece of C code. It plays a central role in Frama-C, as nearly all plug-ins eventually manipulate ACSL specifications, either to generate properties that are to be verified, or to assess that the code is conforming to these specifications. It is thus very important to have a clear view of ACSL's semantics in order to be sure that what you check with Frama-C is really what you mean. Marché contributed to a chapter [27] of the Frama-C book, describing the language in an agnostic way, independently of the various verification plug-ins that are implemented in the Frama-C platform. It contains many examples and exercises that introduce the main

features of the language and insists on the most common pitfalls that users, even experienced ones, may encounter.

Automated Reasoning on Inductive Predicates Filliâtre, Paskevich and Saudubray [24, 22] worked on an extension of the Why3 tool to allow proofs by induction on instances of inductive predicates, i.e. on finitely constructed derivations. It consists of a new matching construction to analyze the form of such a derivation, on the one hand, and a new notion of variant to justify the termination of a recursive function that proceeds according to the size of a derivation, on the other hand. They show how this extension can be implemented conservatively, with almost nothing changed in the verification condition generator of Why3. They illustrate the robustness of this contribution by translating from Coq to Why3 a non-trivial proof containing a large number of reasonings by induction on inductive predicates.

Cross-Language Symbolic Execution One key problem in the field of software security is to expose software vulnerabilities effectively. Traditional testing methods have shown inadequacy in terms of path coverage and bug detection, due to their reliance on concrete input values. Hui and Andrès [14] proposed the idea of combining symbolic execution with runtime annotation checking. By using symbolic input values instead of concrete ones, the specified program properties can be checked on all the corner cases. They introduce two implementations of symbolic runtime annotation checkers: one for C, using the ANSI/ISO C Specification Language, and one for WebAssembly, using the Weasel specification language designed by them. Their approach combines the ease of use and the expressiveness of runtime annotation checking, as well as the power of program analysis.

Reasoning on the Amortized Time Complexity Ownership can also be used to reason about resources other than program memory. Guéneau, Jourdan et al. [17] present formal reasoning rules for verifying *amortized complexity bounds* in a language with thunks. Thunks can be used to construct persistent data structures with good amortized complexity, by suspending expensive computations and memoizing their result. Based on the notion of *time credits and debits*, this work presents a complete machine-checked reconstruction of Okasaki's reasoning rules on thunks in a rich separation logic with time credits, and demonstrates their applicability by verifying several of Okasaki's data structures.

Peano Arithmetic and μMALL Formal theories of arithmetic have traditionally been based on either classical or intuitionistic logic, leading to the development of Peano and Heyting arithmetic, respectively. In a collaboration with D. Miller, Manighetti [33] propose to use μMALL as a formal theory of arithmetic based on linear logic. This formal system is presented as a sequent calculus proof system that extends the standard proof system for multiplicative-additive linear logic (MALL) with the addition of the logical connectives universal and existential quantifiers (first-order quantifiers), term equality and non-equality, and the least and greatest fixed point operators. They demonstrate how functions defined using μMALL relational specifications can be computed using a simple proof search algorithm. By incorporating weakening and contraction into μMALL, they obtain μLK+, a natural candidate for a classical sequent calculus for arithmetic. While important proof theory results are still lacking for μLK+ (including cut-elimination and the completeness of focusing), they prove that μLK+ is consistent and that it contains Peano arithmetic. They also prove some conservativity results regarding μLK+ over μMALL.

8.2 Reasoning on mutable memory in program verification

Participants: Andrei Paskevich, Armaël Guéneau, Arnaud Golfouse, Claude Marché, Guillaume Melquiond, Jean-Christophe Filliâtre, Josué Moreau, Léo Andrès, Sylvain Conchon.

Verification of Rust programs One of the major success of Toccata during the last years is represented by the results obtained concerning the verification of Rust programs. Rust is a fairly recent programming

language for system programming, bringing static guarantees of memory safety through a strong *ownership* policy. This feature opens promising advances for deductive verification of Rust code. The project underlying the PhD thesis of Denis [61], supervised by Jourdan and Marché, is to propose techniques for the verification of Rust program, using a translation to a purely-functional language. The challenge of this translation is the handling of mutable borrows: pointers which control of aliasing in a region of memory. To overcome this, we used a technique inspired by prophecy variables to predict the final values of borrows [60]. This method is implemented in a standalone tool called Creusot [62]. The specification language of Creusot features the notion of prophecy mentioned above, which is central for the specification of behavior of programs performing memory mutation. However, so far, this prophecy-based encoding has only been described in the idealized setting of a core calculus.

Recently, Golfouse, Jourdan and Guéneau, in collaboration with Xavier Denis and Dominic Stolz, show how one might "scale" this technique up to the setting of a realistic verification tool. After describing why doing so is non-trivial, we show how to integrate this encoding with two common features of deductive verification systems: ghost code and type invariants. Additionally, we provide concrete implementation strategies for key aspects of the encoding that were unspecified but turn out to be crucial when considering realistic programs. We implemented this work as an extension of Creusot and present it in a draft paper, which we plan to submit for publication in 2025.

Ownership and Well-Bracketedness Ability to reason about ownership is also fertile ground for designing reasoning principles that capture powerful semantic properties of programs. In particular, Guéneau et al. [18] show that it is possible to capture *well-bracketedness* in a Hoare-style program logic based on separation logic, providing proof rules to show correctness of well-bracketed programs both directly and also through defining unary and binary logical relations models based on this program logic.

Multi-language Verification Guéneau continued work on the Melocoton program multi-language verification framework [68], together with master interns Gurvan Debaussart and Valeran Maytie. They extended Melocoton and its Coq mechanization with new semantics and reasoning rules for the interactions between OCaml exceptions and the OCaml FFI, and for so-called "GC roots" provided by the OCaml FFI.

Inference of Specifications for Closures In many programs, closures allow to express data transformations in a concise way. But when a verification tool is used, they must be accompanied by specifications that are often longer than their body. This is a particularly unpleasant problem, since these closures are often simple and their specification is redundant. Patault, Golfouse and Denis [16] presented a mechanism for inferring the specification of closures for the formal verification of Rust programs. They propose the use of the intermediate verification language Coma as a backend for the deductive verification tool Creusot. Their design is able to manage the internal mutable state of a closure and to infer its specification. They use this mechanism to verify in an ergonomic and modular way a series of Rust programs using higher-order functions.

Safe Libraries for Computer Algebra Low-level libraries used in computer algebra systems, such as GMP, BLAS/LAPACK, etc., are usually written in C, Fortran, and Assembly, and make heavy use of arrays and pointers. Melquiond and Moreau [15, 21] have designed Capla, a programming language dedicated to writing such libraries. This language, halfway between C and Rust, is designed to be safe and to ease the deductive verification of programs, while being low-level enough to be suitable for this kind of computationally intensive applications. They have also written a compiler for this language, based on CompCert. The safety of the language has been formally proved using the Coq proof assistant, and so has the property of semantics preservation for the compiler.

8.3 Verification of Computer Arithmetic

Participants: Claude Marché, David Hamelin, Guillaume Melquiond, Houda Mouhcine, Paul Bonnot, Paul Geneau de Lamarlière, Sylvie Boldo.

Reasoning about Floating-Point Programs Numerical programs make use of the floating-point representation of numbers to perform computations that ideally should be done on mathematical real numbers. The floating-point representation induces approximations on the computations ultimately performed. We have a long tradition of study of subtle algorithms involving such numerical computations. A set of numerical programs that we studied this year is related to the combination of exponential and logarithm functions, that is, the log-sum-exp function known in the context of Machine Learning [73], for which Bonnot et al. [12] provide certified bounds on its accuracy.

Boyer, Faissole, and Melquiond [35] have patented a method and system to transform a program, which includes mathematical functions applied to floating-point variables (and thus suffers from numerical approximations and rounding errors), in order to achieve a specified global accuracy.

Bonnot, Boyer, Faissole, and Marché [31] propose to bound the error between the computed result and the ideal computation by a formula involving the assumed precision of the input of the programs, together with the accuracy properties of the auxiliary mathematical functions that are called as basic blocks. Obtaining such an accuracy property needs a high expertise in floating-point computer arithmetic. The proposed methodology is able to automatically generate such form of accuracy formulas from a given input program. Moreover the generated formulas are certified correct with a high level of confidence, thanks to the automated construction of formal proofs of their validity. The methodology is implemented and experimented on several examples involving approximations of elementary functions such as sine, cosine, exponential and logarithm.

Writing a formal proof offers the highest possible confidence in the correctness of a mathematical library. This comes at a large cost though, since formal proofs require taking into account all the details, even the seemingly insignificant ones, which makes them tedious to write. Faissole, Geneau and Melquiond [13, 19] propose a methodology and some dedicated automation, and apply them to the use case of a faithful binary64 approximation of exponential. The peculiarity of this use case is that the target of the formal verification is not a simple modeling of an external code, it is an actual floating-point function defined in the logic of the Coq proof assistant, which is thus usable inside proofs once its correctness has been fully verified. This function presents all the attributes of a state-of-the-art implementation: bit-level manipulations, large tables of constants, obscure floating-point transformations, exceptional values, etc. This function has been integrated into the proof strategies of the CoqInterval library, bringing a 20x speedup with respect to the previous implementation.

Formalization of Mathematics for Numerical Analysis The correctness of programs solving partial differential equations may rely on mathematics yet unformalized, such as Sobolev spaces. Boldo et al. [46] therefore formalized the mathematical concept of Lebesgue integration and the associated results in Coq (σ-algebras, measures, simple functions, and integration of non-negative measurable functions, up to the full formal proofs of the Beppo Levi Theorem and Fatou's Lemma). Boldo et al. [50, 49] extended this formalization with Tonelli's theorem, stating that the (double) integral of a nonnegative measurable function of two variables can be computed by iterated integrals, and allowing to switch the order of integration.

In her PhD thesis [28] defended in 2024, Houda Mouhcine presented results about Lagrange polynomials on simplicial finite elements, and their fundamental property called *unisolvence*. All the developments are distributed as part of the Coq-Num-Analysis library.

8.4 Spreading Formal Proofs

Participants: Andrei Paskevich, Claude Marché, David Hamelin, Guillaume Melquiond, Jean-Christophe Filliâtre, Josué Moreau, Léo Andrès, Sylvie Boldo.

Compiling OCaml to WebAssembly As part of his CIFRE PhD with OCamlPro, Andrès [37] formalizes a compilation scheme from OCaml to WebAssembly. This on-going work already validated several Wasm extensions [38]. A by-product of the thesis is the implementation of a new, efficient interpreter for Wasm, `owi`. Filliâtre and Andrès collaborate with José Fragoso Santos and Filipe Marques (Universidade de

Lisboa, Portugal), who are using `owi` for concolic execution of WebAssembly programs [11]. Owi is built around a modular, monadic interpreter capable of both normal and symbolic execution of Wasm programs. Monads have been identified as a way to write modular interpreters since 1995 and this strategy has allowed us to build a robust and performant symbolic execution tool which our evaluation shows to be the best currently available for Wasm. Moreover, because WebAssembly is a compilation target for multiple languages (such as Rust and C), Owi can be used to find bugs in C and Rust code, as well as in codebases mixing the two. They demonstrate this flexibility through illustrative examples and evaluate its scalability via comprehensive experiments using the 2024 Test-Comp benchmarks. Results show that Owi achieves comparable performance to state-of-the-art tools like KLEE and Symbiotic, and exhibits advantages in specific scenarios where KLEE's approximations could lead to false negatives.

Programmable Logic Controllers Programmable Logic Controllers are industrial digital computers used as automation controllers in manufacturing processes. The Ladder language is a programming language used to develop software for such controllers. A long-term collaboration with MERCE as for goal the verification that a given Ladder program conforms to an expected behaviour expressed by a timing chart, describing a scenario of execution. This method relies on a modelling of Ladder programs in WhyML, the language of the Why3 environment for deductive program verification. In this approach, the WhyML modelling of individual Ladder instructions has to be trusted. Recently, Cousineau et al. [32] propose a methodology to increase the trust in the WhyML modelling of Ladder instructions. The approach relies on a comparison of the execution of Ladder programs with an execution by Why3 of a simulation of the translated program. With this technique, they have been able to validate their modelling of Ladder instructions, and also discover and fix a subtle bug in the modelling of one particular instruction.

Purely Functional Catenable Deques, Formally Verified Twenty-five years ago, Kaplan and Tarjan [69] established a striking result: there exist "purely functional, real-time deques with catenation". In this ongoing work, Guéneau, in collaboration with Jules Viennot, Arthur Wendling and François Pottier, present the first implementation of Kaplan and Tarjan's catenable deques. This implementation is expressed in the purely functional subset of the OCaml language. Furthermore, they present the first verified implementation of Kaplan and Tarjan's catenable deques. This implementation is expressed in Gallina, the programming language of the Coq proof assistant, and its correctness is stated and verified within Coq. Source code and a journal paper for this contribution are being finalized and will be submitted for publication at the beginning of 2025.

Coq as a Computer Algebra System User interfaces for the Coq proof assistant focus on the ability to write and verify proofs, and mostly ignore Coq's ability to compute. Melquiond [26] shows how the tactic-in-term feature and some adhoc vernacular commands can provide a user experience closer to the one found in computer algebra systems. This work has been implemented in the CoqInterval library.

Teaching Mathematics with Coq As part of the LiberAbaci défi, we have worked on how to use Coq for teaching mathematics.

M. Mayero et al. have been teaching an 18 hour introductory course in formal proofs to L1 students for 3 years at "Sorbonne Paris Nord" University. We present the used methodology and some of the encountered pitfalls, providing both technical and pedagogical aspects [25].

We also worked on worksheets in Coq for students to learn about divisibility and binomials. These basic topics are a good case study as they are widely taught in the early academic years (or before in France). Boldo et al. [23, 30] present technical and pedagogical choices, together with the numerous exercises they developed. As expected, it requires additional Coq material such as other lemmas and dedicated tactics. The worksheets are freely available and flexible in several ways.

9 Bilateral contracts and grants with industry

We have bilateral contracts which are closely related to a joint effort called the ProofInUse consortium. The objective of ProofInUse is to provide verification tools, based on mathematical proof, to industry

users. These tools are aimed at replacing or complementing the existing test activities, whilst reducing costs.

This consortium is a follow-up of the former LabCom ProofInUse between Toccata and the SME AdaCore, funded by the ANR programme “Laboratoires communs”, from April 2014 to March 2017.

9.1 ProofInUse-MERCE Collaboration

Participants: Claude Marché *(contact)*, Guillaume Melquiond, Paul Bonnot, Paul Geneau de Lamarlière.

This bilateral contract is part of the ProofInUse effort. This collaboration joins efforts of the Inria project-team Toccata and the company Mitsubishi Electric R&D (MERCE) in Rennes. It is funded by a bilateral contract of 6 years and 6 months from Nov 2019 to April 2026.

MERCE has strong and recognized skills in the field of formal methods. In the industrial context of the Mitsubishi Electric Group, MERCE has acquired knowledge of the specific needs of the development processes and meets the needs of the group in different areas of application by providing automatic verification and demonstration tools adapted to the problems encountered.

The objective of ProofInUse-MERCE is to significantly improve on-going MERCE tools regarding the verification of Programmable Logic Controllers and also regarding the verification of numerical C codes.

9.2 ProofInUse-TrustInSoft Collaboration

Participants: Claude Marché *(contact)*, Guillaume Melquiond, Raphaël Rieu-Helft, Paul Bonnot.

This bilateral contract is part of the ProofInUse effort. This collaboration joins efforts of the Inria project-team Toccata and the company TrustInSoft in Paris. It was funded by a bilateral contract of 48 months from Dec 2020 to Nov 2024.

TrustInSoft is an SME that offers the TIS-Analyzer environment for analysis of safety and security properties of source codes written in C and C++ languages. A version of TIS-Analyzer is available online, under the name TaaS (TrustInSoft as a Service).

The objective of ProofInUse-TrustInSoft is to integrate Deductive Verification in the platform TIS-Analyzer, under the form of a new plug-in called J-cube. One specific interest resides in the generation of counterexample to help the user in case of proof failure.

This contract ended in Nov 2024, the research efforts are continued within the Décysif project presented below.

9.3 CIFRE contract with OCamlPro company

Participants: Jean-Christophe Filliâtre *(contact)*, Léo Andrès.

Léo Andrès started a CIFRE PhD in October 2021, under the supervision of Jean-Christophe Filliâtre (at Toccata) and Pierre Chambart and Vincent Laviron (at OCamlPro). The subject of the PhD is the design, formalization, and implementation of a garbage collector for WebAssembly. The thesis also proposes a symbolic execution engine for WebAssembly, which can serve as a cross-language assertion checker. The thesis was defended in December 2024 [37].

9.4 CIFRE contract with MERCE

Participants: Guillaume Melquiond *(contact)*, Paul Geneau de Lamarlière.

Paul Geneau de Lamarlière started a CIFRE PhD in March 2023, under the supervision of Guillaume Melquiond and Florian Faissole at MERCE. It aims at the design of better proof environments for verifying programs with complex floating-point computations [13, 19].

10 Partnerships and cooperations

10.1 European initiatives

10.1.1 H2020 projects

EMC2, ERC Synergy Project

Participants: Sylvie Boldo, Houda Mouhcine.

EMC2 project on cordis.europa.eu

Title: Extreme-scale Mathematically-based Computational Chemistry

Duration: From September 1, 2019 to February 28, 2026

Partners:

- INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET AUTOMATIQUE (INRIA), France
- ECOLE POLYTECHNIQUE FEDERALE DE LAUSANNE (EPFL), Switzerland
- ECOLE NATIONALE DES PONTS ET CHAUSSEES (ENPC), France
- CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE CNRS (CNRS), France
- SORBONNE UNIVERSITE, France

Inria contact: Laura GRIGORI (Alpines)

Summary: Molecular simulation has become an instrumental tool in chemistry, condensed matter physics, molecular biology, materials science, and nanosciences. It will allow to propose de novo design of e.g. new drugs or materials provided that the efficiency of underlying software is accelerated by several orders of magnitude.

The ambition of the EMC2 project is to achieve scientific breakthroughs in this field by gathering the expertise of a multidisciplinary community at the interfaces of four disciplines: mathematics, chemistry, physics, and computer science. It is motivated by the twofold observation that, i) building upon our collaborative work, we have recently been able to gain efficiency factors of up to 3 orders of magnitude for polarizable molecular dynamics in solution of multi-million atom systems, but this is not enough since ii) even larger or more complex systems of major practical interest (such as solvated biosystems or molecules with strongly-correlated electrons) are currently mostly intractable in reasonable clock time. The only way to further improve the efficiency of the solvers, while preserving accuracy, is to develop physically and chemically sound models, mathematically certified and numerically efficient algorithms, and implement them in a robust and scalable way on various architectures (from standard academic or industrial clusters to emerging heterogeneous and exascale architectures).

EMC2 has no equivalent in the world: there is nowhere such a critical number of interdisciplinary researchers already collaborating with the required track records to address this challenge. Under the leadership of the 4 PIs, supported by highly recognized teams from three major institutions in

the Paris area, EMC2 will develop disruptive methodological approaches and publicly available simulation tools, and apply them to challenging molecular systems. The project will strongly strengthen the local teams and their synergy enabling decisive progress in the field.

FRESCO, ERC Consolidator project

Title: Fast and Reliable Symbolic Computation

Duration: November 2021 – October 2026

Coordinator: Assia Mahboubi

Website

Using computers to formulate conjectures and consolidate proof steps pervades all mathematics fields, even the most abstract. Most computer proofs are produced by symbolic computations, using computer algebra systems. However, these systems suffer from severe, intrinsic flaws, rendering computational correction and verification challenging. The FRESCO project aims to shed light on whether computer algebra could be both reliable and fast. Researchers will disrupt the architecture of proof assistants, which serve as the best tools for representing mathematics in silico, enriching their programming features while preserving their compatibility with their logical foundations. They will also design novel mathematical software that should feature a high-level, performance-oriented programming environment for writing efficient code to boost computational mathematics.

10.2 National initiatives

10.2.1 ANR NuSCAP

Participants: Guillaume Melquiond *(contact)*, Sylvie Boldo.

The last twenty years have seen the advent of computer-aided proofs in mathematics and this trend is getting more and more important. They request various levels of numerical safety, from fast and stable computations to formal proofs of the computations. Hovewer, the necessary tools and routines are usually ad hoc, sometimes unavailable, or inexistent. On a complementary perspective, numerical safety is also critical for complex guidance and control algorithms, in the context of increased satellite autonomy. We plan to design a whole set of theorems, algorithms and software developments, that will allow one to study a computational problem on all (or any) of the desired levels of numerical rigor. Key developments include fast and certified spectral methods and polynomial arithmetic, with subsequent formal verifications. There will be a strong feedback between the development of our tools and the applications that motivate it.

The project led by École Normale Supérieure de Lyon (LIP) has started in February 2021 and lasts for 4 years. Partners: Inria (teams Aric, Galinette, Lfant, Marelle, Toccata), École Polytechnique (LIX), Sorbonne Université (LIP6), Université Sorbonne Paris Nord (LIPN), CNRS (LAAS).

10.2.2 ANR GOSPEL

Participants: Jean-Christophe Filliâtre *(contact)*, Andrei Paskevich, Armaël Guéneau.

A specification language extends a programming language by allowing code and specifications to be written in a single document. Examples include SparkAda, JML, and ACSL, which extend Ada, Java, and C with syntax for specifications.

By offering **a specification language** to programmers, one encourages them to document, test, and verify their code as they write it, not as a separate step that is too easily postponed. From a technical

point of view, the presence of specifications makes it possible to test or verify each module independently and is the key to **scalability**. From a pragmatic point of view, embedding specifications in the code allows them to be automatically distributed (via a package management system) to every programmer; this is the key to **practical adoption**.

The GOSPEL project proposes to develop **Gospel**, a specification language that extends the programming language OCaml; to develop an ecosystem of tools based on Gospel; and to demonstrate and validate these tools via several case studies.

The project led by Inria Paris has started in October 2022 and lasts for 4 years. Partners: Inria Paris (team Cambium), Université Paris-Saclay (LMF), Tarides, Nomadic Labs.

10.2.3 Project "SecurEval" of PEPR Cybersécurité

Participants: Sylvain Conchon *(contact)*.

The SecureVal project aims to design new tools, benefiting from new digital technologies, to verify the absence of hardware and software vulnerabilities, and carry out the proof of compliance required.

In order to deal effectively with modern digital systems, code analysis techniques, which originated in the world of critical systems, must be overhauled to adapt to the objectives of security assessments and to scale up to complex systems, combining dedicated functionalities and third-party libraries. For example, the design of new fault models, the support of emerging languages, the visualization of formal guarantees, the use of learning techniques to automate repetitive actions or optimize the extraction of relevant information, or the development of approaches combining static and dynamic analyses.

The project is led by CEA-List, it started in 2022 and lasts for 6 years.

10.2.4 Project I-Demo "Décysif"

The Décysif project is a project started in december 2023, for 4 years. Its general goal is the promotion of formal verification for critical systems regarding cybersecurity. This project will fund our future research on Rust program verification, and it contains a workpackage dedicated towards industrialization of the Creusot tool.

The project is led by TrustInSoft company, with AdaCore and OCamlPro as other partners.

10.2.5 Inria Project LiberAbaci

Participants: Sylvie Boldo *(contact)*.

The *Défi* Inria LiberAbaci is a collaborative project aimed at improving the accessibility of the Coq interactive proof system for an audience of mathematics students in the early academic years.

The head is Yves Bertot and the involved teams are: Cambium (Paris), Camus (Strasbourg), Gallinette (Nantes) PiCube (Paris), Spades (Grenoble), Stamp (Sophia Antipolis), Toccata (Saclay), LIPN (Villetaneuse).

11 Dissemination

Participants: Andrei Paskevich, Armaël Guéneau, Claude Marché, Guillaume Melquiond, Jean-Christophe Filliâtre, Josué Moreau, Paul Geneau de Lamarlière, Sylvain Conchon, Sylvie Boldo.

11.1 Promoting scientific activities

11.1.1 Scientific events: organisation

General chair, scientific chair

- A. Guéneau, ACM SIGPLAN OCaml Users and Developers Workshop 2024, OCaml'2024

11.1.2 Scientific events: selection

Chair of conference program committees

- G. Melquiond, 32nd IEEE Symposium on Computer Arithmetic, ARITH'2025.

Member of the conference program committees

- S. Boldo, 31st IEEE Symposium on Computer Arithmetic, ARITH'2024
- S. Boldo, 1st workshop on Programming for the Planet, PROPL'2024
- S. Boldo, 35th Journées Francophones des Langages Applicatifs, JFLA'2024
- S. Boldo, 32nd IEEE Symposium on Computer Arithmetic, ARITH'2025
- S. Boldo, 17th NASA Formal Methods Symposium, NFM'2025
- S. Boldo, Certified Programs and Proofs Symposium, CPP'2025
- J.-C. Filliâtre, 35th Journées Francophones des Langages Applicatifs, JFLA'2024
- J.-C. Filliâtre, NASA Formal Methods Symposium, NFM'2025
- A. Guéneau, 12th ACM SIGPLAN Workshop on Higher-Order Programming with Effects, HOPE'2024

11.1.3 Journal

Member of the editorial boards

- S. Boldo: member of the editorial board of *IEEE Transactions on Emerging Topics in Computing* (TETC), since 2023.
- J.-C. Filliâtre: member of the editorial board of the *Journal of Functional Programming*, since 2011.
- J.-C. Filliâtre: member of the editorial board of the journal *Formal Aspects of Computing*, since 2021.
- A. Paskevich, member of the editorial board of *Formal Methods in System Design*, since 2021.
- G. Melquiond: member of the editorial board of *Reliable Computing*, since 2019.

11.1.4 Invited talks

- S. Boldo, March 27th 2024, university-industry day organized around the Coq proof assistant by Hugo Herbelin
- J.-C. Filliâtre, October 2024, workshop "Big Specification: Specification, Proof, and Testing at Scale", Cambridge, UK.
- A. Guéneau, March 27th 2024, university-industry day organized around the Coq proof assistant by Hugo Herbelin
- A. Guéneau, January 20th 2024, 10th International Workshop on Coq for Programming Languages

11.1.5 Leadership within the scientific community

- S. Boldo, elected chair of the ARITH working group of the GDR-IM (a CNRS subgroup of computer science) with L.-S. Didier (Univ. Toulon).
- S. Boldo, steering committee member of the IEEE International Symposium on Computer Arithmetic.
- J.-C. Filliâtre, steering committee member of JFLA (Journées Francophones des Langages Applicatifs).
- J.-C. Filliâtre, member of IFIP WG 1.9/2.15 Verified Software.

11.1.6 Scientific expertise

- C. Marché, member of the recruitment committee for CRCN/ISFP positions at Inria Saclay, 2024.
- C. Marché, external reviewer for the examination of a promotion to a Full Professor position, University of British Columbia, Canada, 2024.
- S. Boldo and J.-C. Filliâtre, (local) members of the LMF scientific council.
- S. Boldo, member of the ENSIIE scientific council.

11.1.7 Research administration

- S. Boldo, steering committee member of the IEEE International Symposium on Computer Arithmetic from 2019 to 2024.
- S. Boldo, president of the *concours de l'agrégation d'informatique*, 2022-2025.
- J.-C. Filliâtre, leader of the *Proof and Languages* department of LMF.
- A. Guéneau, member of the *Commission des Utilisateurs des Moyens Informatiques* of Inria Saclay.
- A. Guéneau, member of the *Commission de Développement Technologique* of Inria Saclay.
- G. Melquiond, member of the *Bureau du Comité des Projets* of Inria Saclay.
- G. Melquiond, member of the *Commission Consultative de l'Université Paris-Saclay* (section 27).
- G. Melquiond, member of the *Conseil de Politique Doctorale* of Université Paris Saclay.
- G. Melquiond, member of the doctoral school *STIC* of Université Paris Saclay.
- G. Melquiond, member of the *Mission Jeunes Chercheurs* of Inria.
- G. Melquiond, member of the laboratory council of LMF.
- A. Paskevich, leader of the team *Program Verification* of LMF.

11.2 Teaching - Supervision - Juries

11.2.1 Teaching

- J.-C. Filliâtre, *Langages de programmation et compilation*, 25h, L3, École Normale Supérieure, France.
- J.-C. Filliâtre, *Les bases de l'algorithmique et de la programmation*, 15h, L3, École Polytechnique, France.
- J.-C. Filliâtre, *Compilation*, 18h, M1, École Polytechnique, France.

- P. Geneau de Lamarlière, *Qualité de développement*, 32h, BUT1, IUT d'Orsay, Université Paris-Saclay, France.
- P. Geneau de Lamarlière, *Qualité de développement*, 24h, BUT2, IUT d'Orsay, Université Paris-Saclay, France.
- P. Geneau de Lamarlière, *Projet transverse*, 9h, BUT1, IUT d'Orsay, Université Paris-Saclay, France.
- P. Geneau de Lamarlière, *Projet transverse*, 8h, BUT3, IUT d'Orsay, Université Paris-Saclay, France.
- P. Geneau de Lamarlière, *Développement efficace*, 12h, BUT3, IUT d'Orsay, Université Paris-Saclay, France.
- P. Geneau de Lamarlière, *Qualité algorithmique*, 12h, BUT5, IUT d'Orsay, Université Paris-Saclay, France.
- A. Guéneau, *Programmation avancée*, 12h, L3, ENS Paris-Saclay, France.
- C. Marché, Proofs of Programs, 12h, M2, Master Parisien de Recherche en Informatique (MPRI).
- G. Melquiond, *Initiation à la recherche*, 12h, M1, MPRI, École Normale Supérieure Paris-Saclay, France.
- J. Moreau, *Architecture des ordinateurs*, 24h, L2, Université Paris-Saclay, France.
- J. Moreau, *Compilation*, 24h, L3, Université Paris-Saclay, France.
- A. Paskevich, *Vérification Déductive*, 12h, M1, MPRI, Université Paris-Saclay, France.
- A. Paskevich, *Programmation système*, 56h, BUT2, IUT d'Orsay, Université Paris-Saclay, France.

11.2.2 Supervision

- PhD: L. Andrès, "Exécution symbolique pour tous ou Compilation d'OCaml vers WebAssembly", since Oct. 2021, supervised by J.-C. Filliâtre. Defended in December 2024 [37].
- PhD: H. Mouhcine, "Formal Proofs in Applied Mathematics: A Coq Formalization of Simplicial Lagrange Finite Elements", since Oct 2021, supervised by S. Boldo, F. Clément, and M. Mayero. Defended in December 2024 [28].
- PhD in progress: P. Geneau de Lamarlière, "Design of formally verified floating-point components", since Sep. 2022, supervised by G. Melquiond.
- PhD in progress: J. Moreau, "A low-level programming language for formally verified computer algebra", since Oct. 2022, supervised by G. Melquiond.
- PhD in progress: A. Golfouse, "Vérification de programme Rust avancée: invariants de types, code fantôme, possession fantôme et algèbre de ressources, concurrence et aliasing", since Oct. 2023, supervised by J.-H. Jourdan and A. Guéneau.
- PhD in progress: P. Patault, "Conception et étude d'un langage de programmation adapté à la vérification déductive", since Oct. 2023, supervised by J.-C. Filliâtre and A. Paskevich.
- PhD in progress: D. Hamelin, "Implémentation de calculs numériques en virgule fixe avec maîtrise de l'erreur d'arrondi.", since Dec. 2024, supervised by S. Boldo, T. Hilaire (Sorbonne University) and P.-Y. Piriou (EDF).

11.2.3 Juries

- S. Boldo, president of the PhD jury of Louise Dubois De Prisque, “Prétraitement compositionnel en Coq”, July 10th, 2024, University Paris-Saclay.
- S. Boldo, president of the PhD jury of Matthieu Robeyns, “Algorithmes en précision mixte pour des approximations de rang faible de matrices et tenseurs”, December 10th, 2024, University Paris-Saclay.
- J.-C. Filliâtre, reviewer of the PhD thesis of J. Giet, September 26, 2024, Univ. PSL.
- J.-C. Filliâtre, reviewer of the habilitation thesis of F. Dabrowski, Nov 5, 2024, Univ. Orléans.
- J.-C. Filliâtre, reviewer of the PhD thesis of G. Parthasarathy, Nov 6, 2024, ETH Zurich.
- J.-C. Filliâtre, examiner of the PhD thesis of O. Martinot, Dec 2, 2024, Univ. Paris Cité.
- C. Marché, reviewer of the PhD thesis of Vytautas Astraukas “Leveraging Uniqueness for Modular Verification of Heap-Manipulating Programs”, March 27th, 2024, ETH Zürich, Switzerland.
- G. Melquiond, reviewer of the PhD thesis of Nathanaëlle Courant, “Vers un vérificateur de preuves Coq efficace et formellement prouvé”, September 19th, 2024, Université Paris Cité.

12 Scientific production

12.1 Major publications

[1] B. Becker, N. Jeannerod, C. Marché, Y. Régis-Gianas, M. Sighireanu and R. Treinen. ‘The CoLiS Platform for the Analysis of Maintainer Scripts in Debian Software Packages’. In: *International Journal on Software Tools for Technology Transfer* (2022). URL: https://inria.hal.science/hal-03737886 (cit. on p. 7).

[2] C. Belo Lourenço, D. Cousineau, F. Faissole, C. Marché, D. Mentré and H. Inoue. ‘Automated Formal Analysis of Temporal Properties of Ladder Programs’. In: *International Journal on Software Tools for Technology Transfer* 24.6 (2022), pp. 977–997. DOI: 10.1007/s10009-022-00680-0. URL: https://inria.hal.science/hal-03737869 (cit. on p. 7).

[3] S. Boldo, F. Clément, F. Faissole, V. Martin and M. Mayero. ‘A Coq Formalization of Lebesgue Integration of Nonnegative Functions’. In: *Journal of Automated Reasoning* 66 (2022), pp. 175–213. DOI: 10.1007/s10817-021-09612-0. URL: https://inria.hal.science/hal-03471095 (cit. on p. 6).

[4] S. Boldo, C.-P. Jeannerod, G. Melquiond and J.-M. Muller. ‘Floating-point arithmetic’. In: *Acta Numerica* 32 (May 2023), pp. 203–290. DOI: 10.1017/S0962492922000101. URL: https://hal.science/hal-04095151 (cit. on p. 4).

[5] S. Conchon, G. Delzanno and A. Ferrando. ‘Declarative Parameterized Verification of Distributed Protocols via the Cubicle Model Checker’. In: *Fundamenta Informaticae* 178.4 (9th Feb. 2021), pp. 347–378. DOI: 10.3233/FI-2021-2010. URL: https://inria.hal.science/hal-03476675 (cit. on p. 6).

[6] S. Conchon and A. Korneva. ‘The Cubicle Fuzzy Loop: A Fuzzing-Based Extension for the Cubicle Model Checker’. In: *Lecture Notes in Computer Science*. SEFM 2023 - Software Engineering and Formal Methods. Vol. LNCS-14323. Software Engineering and Formal Methods. Eindhoven, Netherlands: Springer Nature Switzerland, 31st Oct. 2023, pp. 30–46. DOI: 10.1007/978-3-031-47115-5_3. URL: https://inria.hal.science/hal-04394062 (cit. on p. 6).

[7] J.-C. Filliâtre and A. Paskevich. ‘Abstraction and Genericity in Why3’. In: ISoLA 2021 - 9th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation. Vol. 12476. Rhodes, Greece, 2020. DOI: 10.1007/978-3-030-61362-4_7. URL: https://hal.inria.fr/hal-02696246 (cit. on p. 6).

[8] A. L. Georges, A. Guéneau, T. Van Strydonck, A. Timany, A. Trieu, D. Devriese and L. Birkedal. 'Cerise: Program Verification on a Capability Machine in the Presence of Untrusted Code'. In: *Journal of the ACM (JACM)* (14th Sept. 2023). DOI: 10.1145/3623510. URL: https://hal.science/hal-03826854 (cit. on p. 6).

[9] Y. Matsushita, X. Denis, J.-H. Jourdan and D. Dreyer. 'RustHornBelt: a semantic foundation for functional verification of Rust programs with unsafe code'. In: PLDI 2022 - 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation. San Diego CA USA, United States: ACM, 9th June 2022, pp. 841–856. DOI: 10.1145/3519939.3523704. URL: https://inria.hal.science/hal-03777103 (cit. on p. 6).

[10] G. Melquiond and R. Rieu-Helft. 'WhyMP, a Formally Verified Arbitrary-Precision Integer Library'. In: *Journal of Symbolic Computation* 115 (Mar. 2023), pp. 74–95. DOI: 10.1016/j.jsc.2022.07.007. URL: https://inria.hal.science/hal-03233220 (cit. on p. 7).

12.2 Publications of the year

International journals

[11] L. Andrès, F. Marques, A. Carcano, P. Chambart, J. Fragoso Femenin dos Santos and J.-C. Filliâtre. 'Owi: Performant Parallel Symbolic Execution Made Easy, an Application to WebAssembly'. In: *The Art, Science, and Engineering of Programming* 9.2 (15th Oct. 2024). URL: https://hal.science/hal-04627413 (cit. on p. 19).

International peer-reviewed conferences

[12] P. Bonnot, B. Boyer, F. Faissole, C. Marché and R. Rieu-Helft. 'Formally Verified Rounding Errors of the Logarithm-Sum-Exponential Function'. In: Formal Methods in Computer-Aided Design - FMCAD 2024. Prague, Czech Republic: IEEE, 14th Oct. 2024. URL: https://inria.hal.science/hal-04674600 (cit. on pp. 7, 18).

[13] F. Faissole, P. Geneau de Lamarlière and G. Melquiond. 'End-to-End Formal Verification of a Fast and Accurate Floating-Point Approximation'. In: *Leibniz International Proceedings in Informatics*. 15th International Conference on Interactive Theorem Proving. Vol. 309. Tbilisi, Georgia, 9th Sept. 2024, 14:1–14:18. DOI: 10.4230/LIPIcs.ITP.2024.14. URL: https://hal.science/hal-04515714 (cit. on pp. 18, 21).

[14] Z. Hui and L. Andrès. 'Cross-Language Symbolic Runtime Annotation Checking'. In: 36es Journées Francophones des Langages Applicatifs (JFLA 2025). Roiffé, France, 28th Jan. 2025. URL: https://inria.hal.science/hal-04798756 (cit. on p. 16).

[15] G. Melquiond and J. Moreau. 'A Safe Low-level Language for Computer Algebra and its Formally Verified Compiler'. In: *Proceedings of the ACM on Programming Languages*. 29th ACM SIGPLAN International Conference on Functional Programming. Vol. 8. ICFP. Milan, Italy, 15th Aug. 2024, pp. 121–146. DOI: 10.1145/3674629. URL: https://inria.hal.science/hal-04485670 (cit. on p. 17).

[16] P. Patault, A. Golfouse and X. Denis. 'Remonter les barrières pour ouvrir une clôture: Inférence de spécification des clôtures pour la preuve de programmes Rust avec COMA'. In: JFLA 2025 - 36es Journées Francophones des Langages Applicatifs. Roiffé, France, 28th Jan. 2025. URL: https://inria.hal.science/hal-04859517 (cit. on p. 17).

[17] F. Pottier, A. Guéneau, J.-H. Jourdan and G. Mével. 'Thunks and Debits in Separation Logic with Time Credits'. In: *Proceedings of the ACM*. POPL 2024 - 51st ACM SIGPLAN Symposium on Principles of Programming Languages. Vol. 8. POPL. Londres, United Kingdom: ACM, Jan. 2024. URL: https://hal.science/hal-04238691 (cit. on pp. 6, 16).

[18] A. Timany, A. Guéneau and L. Birkedal. 'The Logical Essence of Well-Bracketed Control Flow'. In: *Proceedings of the ACM*. POPL 2024 - 51st ACM SIGPLAN Symposium on Principles of Programming Languages. Londres, United Kingdom: ACM, 14th Jan. 2024. URL: https://hal.science/hal-04271457 (cit. on p. 17).

National peer-reviewed Conferences

[19] P. Geneau de Lamarlière. 'Vérification de bout en bout d'une fonction de bibliothèque mathématique'. In: JFLA 2025 - 36es Journées Francophones des Langages Applicatifs. Roiffé, France, 28th Jan. 2025. URL: https://inria.hal.science/hal-04859533 (cit. on pp. 18, 21).

[20] C. Marché and D. Cousineau. 'De l'avantage de nuancer les décisions binaires'. In: 35es Journées Francophones des Langages Applicatifs (JFLA 2024). Saint-Jacut-de-la-Mer, France, Jan. 2024. URL: https://inria.hal.science/hal-04342273 (cit. on p. 15).

[21] J. Moreau. 'Des briques de calcul formel plus solides avec Capla'. In: JFLA 2025 - 36es Journées Francophones des Langages Applicatifs. Roiffé, France, 28th Jan. 2025. URL: https://inria.hal.science/hal-04859452 (cit. on p. 17).

[22] H. Saudubray, J.-C. Filliâtre and A. Paskevich. 'Faire chauffer Why3 avec de l'induction'. In: JFLA 2025 - 36es Journées Francophones des Langages Applicatifs. Roiffé, France, 28th Jan. 2025. URL: https://inria.hal.science/hal-04859412 (cit. on p. 16).

Conferences without proceedings

[23] S. Boldo, F. Clément, D. Hamelin, M. Mayero and P. Rousselin. 'Teaching Divisibility and Binomials with Coq'. In: 13th International Workshop on Theorem proving components for Educational software - ThEdu 2024. Nancy, France, 1st July 2024. URL: https://hal.science/hal-04725586 (cit. on p. 19).

[24] J.-C. Filliâtre, A. Paskevich and H. Saudubray. 'Proofs on Inductive Predicates in Why3'. In: Big Specification: Specification, Proof, and Testing at Scale. Cambridge, United Kingdom, 8th Oct. 2024. URL: https://hal.science/hal-04734466 (cit. on p. 16).

[25] M. Kerjean, M. Mayero and P. Rousselin. 'Maths with Coq in L1, a pedagogical experiment'. In: ThEdu 2024 - 13th International Workshop on Theorem proving components for Educational software. Nancy, France, 1st July 2024. URL: https://hal.science/hal-04823220 (cit. on p. 19).

[26] G. Melquiond. 'Turning the Coq Proof Assistant into a Pocket Calculator'. In: Coq 2024 - 15th Coq Workshop. Tbilisi, Georgia, 14th Sept. 2024. URL: https://inria.hal.science/hal-04702129 (cit. on p. 19).

Scientific book chapters

[27] A. Blanchard, C. Marché and V. Prévosto. 'Formally Expressing what a Program Should Do: the ACSL Language'. In: *Guide to Software Verification with Frama-C - Core Components, Usages, and Applications*. Springer, 2024, pp. 3–80. DOI: 10.1007/978-3-031-55608-1_1. URL: https://inria.hal.science/hal-04265707 (cit. on p. 15).

Doctoral dissertations and habilitation theses

[28] H. Mouhcine. 'Preuves formelles en mathématiques appliquées : formalisation en Coq des éléments finis de Lagrange simpliciaux'. Université Paris-Saclay, 9th Dec. 2024. URL: https://theses.hal.science/tel-04884651 (cit. on pp. 18, 26).

Reports & preprints

[29] P. Baudin, P. Cuoq, J.-C. Filliâtre, C. Marché, B. Monate, Y. Moy and V. Prévosto. *ANSI/ISO C Specification Language Version 1.20*. CEA List, 2024. URL: https://hal.science/hal-04523865 (cit. on pp. 3, 15).

[30] S. Boldo, F. Clément, D. Hamelin, M. Mayero and P. Rousselin. *Teaching Divisibility and Binomials with Coq*. RR-9547. Inria, Apr. 2024, p. 13. URL: https://inria.hal.science/hal-04550762 (cit. on p. 19).

[31] P. Bonnot, B. Boyer, F. Faissole and C. Marché. *Generating and Certifying Accuracy Properties of Floating-Point Programs*. RR-9564. inria, Dec. 2024. URL: https://inria.hal.science/hal-04820735 (cit. on p. 18).

[32] D. Cousineau, H. Inoue, C. Marché and D. Mentré. *A Methodological Guide for the Validation of Logic Modelling of Ladder Instructions*. RT-0522. Inria, Mar. 2024. URL: https://inria.hal.science/hal-04487766 (cit. on p. 19).

[33] M. Manighetti and D. Miller. *Peano Arithmetic and μMALL*. 4th Nov. 2024. DOI: 10.48550/arXiv.2312.13634. URL: https://hal.science/hal-04824175 (cit. on p. 16).

[34] A. Paskevich, P. Patault and J.-C. Filliâtre. *Coma, an Intermediate Verification Language with Explicit Abstraction Barriers*. 20th Dec. 2024. URL: https://hal.science/hal-04839768 (cit. on p. 15).

Patents

[35] B. Boyer, F. Faissole and G. Melquiond. 'Method and system for converting an input computer program into an output computer program achieving a target global accuracy'. EP4235397 (France). 19th June 2024. URL: https://inria.hal.science/hal-04872869 (cit. on p. 18).

12.3 Cited publications

[36] AdaCore. *NVIDIA: Adoption of SPARK Ushers in a New Era in Security-Critical Software Development*. web publication https://www.adacore.com/papers/nvidia-adoption-of-spark-new-era-in-security-critical-software-development. 2023 (cit. on p. 7).

[37] L. Andrès. 'Exécution symbolique pour tous ou Compilation d'OCaml vers WebAssembly'. PhD thesis. Université Paris-Saclay, 2024 (cit. on pp. 18, 20, 26).

[38] L. Andrès, P. Chambart and J.-C. Filliâtre. 'Wasocaml: compiling OCaml to WebAssembly'. In: *35th Symposium on Implementation and Application of Functional Languages*. Ed. by J. Saraiva and J. Fernandes. 2023. URL: https://inria.hal.science/hal-04311345 (cit. on pp. 8, 18).

[39] A. Ayad and C. Marché. 'Multi-Prover Verification of Floating-Point Programs'. In: *Fifth International Joint Conference on Automated Reasoning*. Ed. by J. Giesl and R. Hähnle. Vol. 6173. Lecture Notes in Artificial Intelligence. Edinburgh, Scotland: Springer, July 2010, pp. 127–141. URL: http://hal.inria.fr/inria-00534333 (cit. on p. 4).

[40] T. Balabonski, S. Conchon, J.-C. Filliâtre and K. Nguyen. *Numérique et Sciences Informatiques, 24 leçons avec exercices corrigés. Terminale*. Ellipses, 2020. URL: https://hal.inria.fr/hal-03023099 (cit. on p. 8).

[41] T. Balabonski, S. Conchon, J.-C. Filliâtre and K. Nguyen. *Numérique et Sciences Informatiques, 30 leçons avec exercices corrigés. Première*. Ellipses, 2019. URL: https://inria.hal.science/hal-02379073 (cit. on p. 8).

[42] T. Balabonski, S. Conchon, J.-C. Filliâtre, K. Nguyen and L. Sartre. *Informatique - MP2I/MPI - CPGE 1re et 2e années - Cours et exercices corrigés*. Ellipses, 2022. URL: https://hal.inria.fr/hal-03886751 (cit. on p. 8).

[43] H. Barbosa, C. W. Barrett, M. Brain, G. Kremer, H. Lachnitt, M. Mann, A. Mohamed, M. Mohamed, A. Niemetz, A. Nötzli, A. Ozdemir, M. Preiner, A. Reynolds, Y. Sheng, C. Tinelli and Y. Zohar. 'cvc5: A Versatile and Industrial-Strength SMT Solver'. In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by D. Fisman and G. Rosu. Vol. 13243. Lecture Notes in Computer Science. Springer, 2022, pp. 415–442 (cit. on p. 3).

[44] P. Behm, P. Benoit, A. Faivre and J.-M. Meynadier. 'METEOR : A successful application of B in a large project'. In: *Proceedings of FM'99: World Congress on Formal Methods*. Ed. by J. M. Wing, J. Woodcock and J. Davies. Lecture Notes in Computer Science (Springer-Verlag). Springer Verlag, Sept. 1999, pp. 369–387 (cit. on p. 3).

[45] F. Bobot, J.-C. Filliâtre, C. Marché and A. Paskevich. 'Let's Verify This with Why3'. In: *International Journal on Software Tools for Technology Transfer (STTT)* 17.6 (2015), pp. 709–727. URL: http://hal.inria.fr/hal-00967132/en (cit. on pp. 3, 15).

[46] S. Boldo, F. Clément, F. Faissole, V. Martin and M. Mayero. 'A Coq Formalization of Lebesgue Integration of Nonnegative Functions'. In: *Journal of Automated Reasoning* 66 (2022), pp. 175–213. URL: https://hal.inria.fr/hal-03471095 (cit. on p. 18).

[47] S. Boldo, F. Clément, J.-C. Filliâtre, M. Mayero, G. Melquiond and P. Weis. 'Formal Proof of a Wave Equation Resolution Scheme: the Method Error'. In: *Interactive Theorem Proving*. Vol. 6172. Lecture Notes in Computer Science. Springer, 2010, pp. 147–162. URL: http://hal.inria.fr/inria-00450789/en (cit. on p. 4).

[48] S. Boldo, F. Clément, J.-C. Filliâtre, M. Mayero, G. Melquiond and P. Weis. 'Wave Equation Numerical Resolution: a Comprehensive Mechanized Proof of a C Program'. In: *Journal of Automated Reasoning* 50.4 (Apr. 2013), pp. 423–456. URL: http://hal.inria.fr/hal-00649240/en/ (cit. on p. 4).

[49] S. Boldo, F. Clément, V. Martin, M. Mayero and H. Mouhcine. 'A Coq Formalization of Lebesgue Induction Principle and Tonelli's Theorem'. In: *25th International Symposium on Formal Methods (FM 2023)*. Vol. 14000. Lecture Notes in Computer Science. 2023, pp. 39–55. URL: https://hal.inria.fr/hal-03889276 (cit. on p. 18).

[50] S. Boldo, F. Clément, V. Martin, M. Mayero and H. Mouhcine. *Lebesgue Induction and Tonelli's Theorem in Coq*. Research Report 9457. Inria, 2023. URL: https://inria.hal.science/hal-03564379 (cit. on p. 18).

[51] S. Boldo, J.-C. Filliâtre and G. Melquiond. 'Combining Coq and Gappa for Certifying Floating-Point Programs'. In: *16th Symposium on the Integration of Symbolic Computation and Mechanised Reasoning*. Vol. 5625. Lecture Notes in Artificial Intelligence. Grand Bend, Canada: Springer, July 2009, pp. 59–74 (cit. on p. 4).

[52] S. Boldo and C. Marché. 'Formal verification of numerical programs: from C annotated programs to mechanical proofs'. In: *Mathematics in Computer Science* 5 (4 2011), pp. 377–393. URL: http://hal.inria.fr/hal-00777605 (cit. on p. 4).

[53] S. Boldo and G. Melquiond. *Computer Arithmetic and Formal Proofs: Verifying Floating-point Algorithms with the Coq System*. ISTE Press - Elsevier, Dec. 2017. URL: https://hal.inria.fr/hal-01632617 (cit. on p. 6).

[54] S. Boldo and T. M. T. Nguyen. 'Proofs of numerical programs when the compiler optimizes'. In: *Innovations in Systems and Software Engineering* 7 (2 2011), pp. 151–160. URL: http://hal.inria.fr/hal-00777639 (cit. on p. 4).

[55] P. Bonnot, B. Boyer, F. Faissole, C. Marché and R. Rieu-Helft. *Formally Verified Bounds on Rounding Errors in Concrete Implementations of Logarithm-Sum-Exponential Functions*. Research Report 9531. Inria, 2023. URL: https://inria.hal.science/hal-04343157 (cit. on p. 7).

[56] L. Burdy, Y. Cheon, D. R. Cok, M. D. Ernst, J. R. Kiniry, G. T. Leavens, K. R. M. Leino and E. Poll. 'An overview of JML tools and applications'. In: *International Journal on Software Tools for Technology Transfer (STTT)* 7.3 (June 2005), pp. 212–232 (cit. on p. 3).

[57] M. Clochard, C. Marché and A. Paskevich. 'Deductive Verification with Ghost Monitors'. In: *Principles of Programming Languages*. New Orleans, United States, 2020. URL: https://hal.inria.fr/hal-02368284 (cit. on p. 4).

[58] S. Conchon, A. Coquereau, M. Iguernlala and A. Mebsout. 'Alt-Ergo 2.2'. In: *SMT Workshop: International Workshop on Satisfiability Modulo Theories*. Oxford, United Kingdom, July 2018. URL: https://hal.inria.fr/hal-01960203 (cit. on p. 3).

[59] S. Conchon, M. Iguernlala, K. Ji, G. Melquiond and C. Fumex. 'A Three-tier Strategy for Reasoning about Floating-Point Numbers in SMT'. In: *Computer Aided Verification*. Vol. 10427. Lecture Notes in Computer Science. 2017, pp. 419–435. URL: https://hal.inria.fr/hal-01522770 (cit. on p. 6).

[60] X. Denis. *Deductive program verification for a language with a Rust-like typing discipline*. Internship report. Université de Paris, Sept. 2020. URL: https://hal.archives-ouvertes.fr/hal-02962804 (cit. on p. 17).

[61] X. Denis. 'Deductive Verification of Rust Programs'. PhD thesis. Université Paris-Saclay, 2023. URL: https://hal.science/tel-04517581 (cit. on pp. 9, 17).

[62] X. Denis, J.-H. Jourdan and C. Marché. 'Creusot: a Foundry for the Deductive Verication of Rust Programs'. In: *International Conference on Formal Engineering Methods - ICFEM*. Lecture Notes in Computer Science. Madrid, Spain: Springer, 2022. URL: https://hal.inria.fr/hal-03737878 (cit. on p. 17).

[63] F. de Dinechin, C. Lauter and G. Melquiond. 'Certifying the floating-point implementation of an elementary function using Gappa'. In: *IEEE Transactions on Computers* 60.2 (2011), pp. 242–253. URL: http://hal.inria.fr/inria-00533968/en/ (cit. on p. 4).

[64] J.-C. Filliâtre and C. Pascutto. 'Optimizing Prestate Copies in Runtime Verification of Function Postconditions'. In: *22nd International Conference on Runtime Verification*. 2022. URL: https://hal.inria.fr/hal-03690675v1 (cit. on p. 8).

[65] C. Fumex, C. Marché and Y. Moy. *Automated Verification of Floating-Point Computations in Ada Programs*. Research Report RR-9060. Inria, Apr. 2017, p. 53. URL: https://hal.inria.fr/hal-01511183 (cit. on p. 4).

[66] C. Fumex, C. Marché and Y. Moy. 'Automating the Verification of Floating-Point Programs'. In: *Verified Software: Theories, Tools, and Experiments. Revised Selected Papers Presented at the 9th International Conference VSTTE*. Ed. by A. Paskevich and T. Wies. Lecture Notes in Computer Science 10712. Heidelberg, Germany: Springer, Dec. 2017. URL: https://hal.inria.fr/hal-01534533/ (cit. on p. 4).

[67] S. de Gouw, J. Rot, F. S. de Boer, R. Bubel and R. Hähnle. 'OpenJDK's Java.utils.Collection.sort() Is Broken: The Good, the Bad and the Worst Case'. In: *Computer Aided Verification: 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*. Ed. by D. Kroening and C. S. Păsăreanu. Cham: Springer International Publishing, 2015, pp. 273–289 (cit. on p. 3).

[68] A. Guéneau, J. Hostert, S. Spies, M. Sammler, L. Birkedal and D. Dreyer. 'Melocoton: A Program Logic for Verified Interoperability Between OCaml and C'. In: *Object-Oriented Programming, Systems, Languages & Applications*. ACM, 2023. URL: https://inria.hal.science/hal-04203298 (cit. on pp. 6, 17).

[69] H. Kaplan and R. E. Tarjan. 'Purely functional, real-time deques with catenation'. In: 46.5 (1999), pp. 577–603 (cit. on p. 19).

[70] G. Klein, J. Andronick, K. Elphinstone, G. Heiser, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch and S. Winwood. 'seL4: Formal verification of an OS kernel'. In: *Communications of the ACM* 53.6 (June 2010), pp. 107–115 (cit. on p. 3).

[71] X. Leroy. 'A formally verified compiler back-end'. In: *Journal of Automated Reasoning* 43.4 (2009), pp. 363–446. URL: http://hal.inria.fr/inria-00360768/en/ (cit. on p. 3).

[72] A. Mahboubi, G. Melquiond and T. Sibut-Pinote. 'Formally Verified Approximations of Definite Integrals'. In: *Journal of Automated Reasoning* 62.2 (Feb. 2019), pp. 281–300. URL: https://hal.inria.fr/hal-01630143 (cit. on p. 6).

[73] T. Miyagawa and A. F. Ebihara. 'The Power of Log-Sum-Exp: Sequential Density Ratio Matrix Estimation for Speed-Accuracy Optimization'. In: *International Conference on Machine Learning*. Vol. 139. Proceedings of Machine Learning Research. 2021, pp. 7792–7804. URL: https://proceedings.mlr.press/v139/miyagawa21a.html (cit. on p. 18).

[74] L. de Moura and N. Bjørner. 'Z3, An Efficient SMT Solver'. In: *TACAS*. Vol. 4963. Lecture Notes in Computer Science. Springer, 2008, pp. 337–340 (cit. on p. 3).

[75] J.-M. Muller, N. Brunie, F. de Dinechin, C.-P. Jeannerod, M. Joldes, V. Lefèvre, G. Melquiond, N. Revol and S. Torres. *Handbook of Floating-point Arithmetic (2nd edition)*. Birkhäuser Basel, July 2018. URL: https://hal.inria.fr/hal-01766584 (cit. on p. 4).

[76] T. M. T. Nguyen and C. Marché. 'Hardware-Dependent Proofs of Numerical Programs'. In: *Certified Programs and Proofs*. Ed. by J.-P. Jouannaud and Z. Shao. Lecture Notes in Computer Science. Springer, Dec. 2011, pp. 314–329. URL: http://hal.inria.fr/hal-00772508 (cit. on p. 4).

[77] R. Rieu-Helft. 'A Why3 proof of GMP algorithms'. In: *Journal of Formalized Reasoning* (2019). URL: https://inria.hal.science/hal-02477578 (cit. on p. 7).