RESEARCH CENTRE

Saclay - Île-de-France

IN PARTNERSHIP WITH:

CNRS, Université Paris-Saclay

2021

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

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]
- Guillaume Melquiond [Inria, Senior Researcher, HDR]

Faculty Members

- Sylvain Conchon [Univ Paris-Saclay, Professor, HDR]
- Andrei Paskevich [Univ Paris-Saclay, Associate Professor]

PhD Students

- Louise Ben Salem-Knapp [CEA, Until Sept 2021]
- Xavier Denis [Univ Paris-Saclay]
- Diane Gallois-Wong [Univ Paris-Saclay, until Feb 2021]
- Quentin Garchery [Université de Paris]
- Antoine Lanco [Inria]
- Houda Mouhcine [Inria, from Oct 2021]
- Clément Pascutto [Tarides, CIFRE]

Technical Staff

- Benedikt Becker [Inria, Engineer, until Oct 2021]
- Guillaume Cluzel [Inria, Engineer, from Jun 2021 until Nov 2021]
- Yacine El Haddad [Inria, Engineer, from Sep 2021]

Interns and Apprentices

- Louise Leclerc [Inria, from Jun 2021 until Jul 2021]
- Houda Mouhcine [Inria, from Apr 2021 until Jul 2021]
- Paul Patault [Inria, from May 2021 until Jul 2021]
- Gaetan Serre [Inria, from May 2021 until Jul 2021]

Administrative Assistant

- Alexandra Merlin [Inria]

External Collaborators

- Thibaut Balabonski [Univ Paris-Saclay]
- Jacques-Henri Jourdan [CNRS, from Mar 2021]
- Chantal Keller [Univ Paris-Saclay]

2 Overall objectives

2.1 Presentation

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, and more and more in cars), communication (e.g., internet, smartphones), health devices, 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 (i.e., 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 they cannot ensure that all the bugs are caught, and remaining bugs may have catastrophic causes (e.g., the Heartbleed bug in OpenSSL library discovered in 2014).

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 consider the other ones in some cases.

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 [40] and other railway-related systems; a formally proved C compiler was developed using the Coq proof assistant [62]; 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 [61]. A bug in the JDK implementation of TimSort was discovered using the KeY environment [57] and a fixed version was proved sound. Another sign of recent progress is the emergence of deductive verification competitions (e.g., VerifyThis [2]). Finally, recent trends in the industrial practice for development of critical software is to require more and more guarantees of safety, e.g., the new 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 [36], JML [46]) and automatically producing a set of mathematical formulas (the *verification conditions*) which are typically proved using automatic provers (e.g., Z3 [64], Alt-Ergo [49], CVC4 [35]).

The first family of approaches usually offers a higher level of assurance 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. Indeed, an axis of research of Toccata is the development of languages, methods and tools that are themselves formally proved correct. Recent advances in the foundations of deductive verification include various aspects such as reasoning efficiently on bitvector programs [54] or providing counterexamples when a proof does not succeed [50].

A specifically challenging aspect of deductive verification methods is how does one deal with memory mutation in general, an issue that appear under various similar forms such the reasoning on mutable data structures or on concurrent programs, with the common denominator of the tracking of memory change on shared data. The ability to track aliasing is also a key for the ability of specifying programs and conduct proofs using the advanced notion of *ghost code* [7], notion that can be push forward very far as demonstrated by our work on ghost monitors [48].

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 [34], various examples of applications of that approach [44], the use of the Gappa solver for proving numerical algorithms [52], an approach to take architectures and compilers into account when dealing with floating-point programs [45, 66]. We contributed to the CompCert verified compiler, regarding the support for floating-point operations [3]. We also contributed to the Handbook of Floating-Point Arithmetic [65]. 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 [42] [41]. We published a reference book on the verification of floating-point algorithms with Coq [4]. Our experience led us to a conclusion that verification of numerical programs can benefit a lot from combining automatic and interactive theorem proving [43, 44, 55]. Verification of numerical programs is another main axis of Toccata.

Deductive program verification methods are built upon theorem provers to decide whether a expected proof obligation on a program is a valid mathematical proposition, hence working on deductive verification requires a certain amount of work on the aspect of design of theorem provers. We are involved in particular in the Alt-Ergo SMT solver, for which we designed an original approach for reasoning on arithmetic facts [6] [10] ; the Gappa tool dedicated to reasoning on rounding errors in floating-point computations [63]; and the interval tactic to reason about real approximations [8]. Proof by reflection is also a powerful approach for advanced reasoning about programs [9].

In the past, we have been more and more involved in the development of significantly large case studies and applications, such as for example the verification of matrix multiplication algorithms [5], the design of verified OCaml librairies [47], the realization of a platform for verification of shell scripts [38] [1], or the correct-by-construction design of an efficient library for arbitrary-precision arithmetic [9].

Our scientific programme detailed below is structured into 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.

Let us conclude with more general considerations about our agenda of the next four years: we want to keep on

- with general audience actions;
- industrial transfer, in particular through an extension of the perimeter of the ProofInUse joint lab.

3.1 Foundations and spreading of deductive program verification

Permanent researchers: S. Conchon, J.-C. Filliâtre, C. Marché, G. Melquiond, A. Paskevich

This axis covers the central theme of the team: deductive verification, from the point of view of its foundations but also our will to spread its use in software development. The general motto we want to defend is "deductive verification for the masses". A non-exhaustive list of subjects we want to address is as follows.

- The verification of general-purpose algorithms and data structures: the challenge is to discover adequate invariants to obtain a proof, in the most automatic way as possible, in the continuation of the current VOCaL project and the various case studies presented in Axis 4 below.
- 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.
- Automated reasoning dedicated to deductive verification, so as to improve proof automation; improved combination of interactive provers and fully automated ones, proof by reflection.
- Improved feedback in case of proof failures: based on generation of counterexamples, or symbolic execution, or possibly randomized techniques à la quickcheck.
- Reduction of the trusted computing base in our toolchains: production of certificates from automatic proofs, for goal transformations (like those done by Why3), and from the generation of VCs

A significant part of the work achieved in this axis is related to the Why3 toolbox and its ecosystem, displayed on Figure 1. The boxes in red background correspond to the tools we develop in the Toccata team.

3.2 Reasoning on mutable memory in program verification

Permanent researchers: J.-C. Filliâtre, C. Marché, G. Melquiond, A. Paskevich

This axis concerns specifically the techniques for reasoning on programs where 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 planned 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 system in the style of Rust, involving ownership and borrowing, will be considered.
- 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.
- Concurrent programming on weak memory models, on one hand as an extension of parameterized systems above, but also in the specific context of OCaml multicore.
- In particular in the context of the ProofInUse joint lab, design methods for Ada, C, C++ or Java using memory models involving fine-grain analysis of pointers. Rust programs could be considered as well.

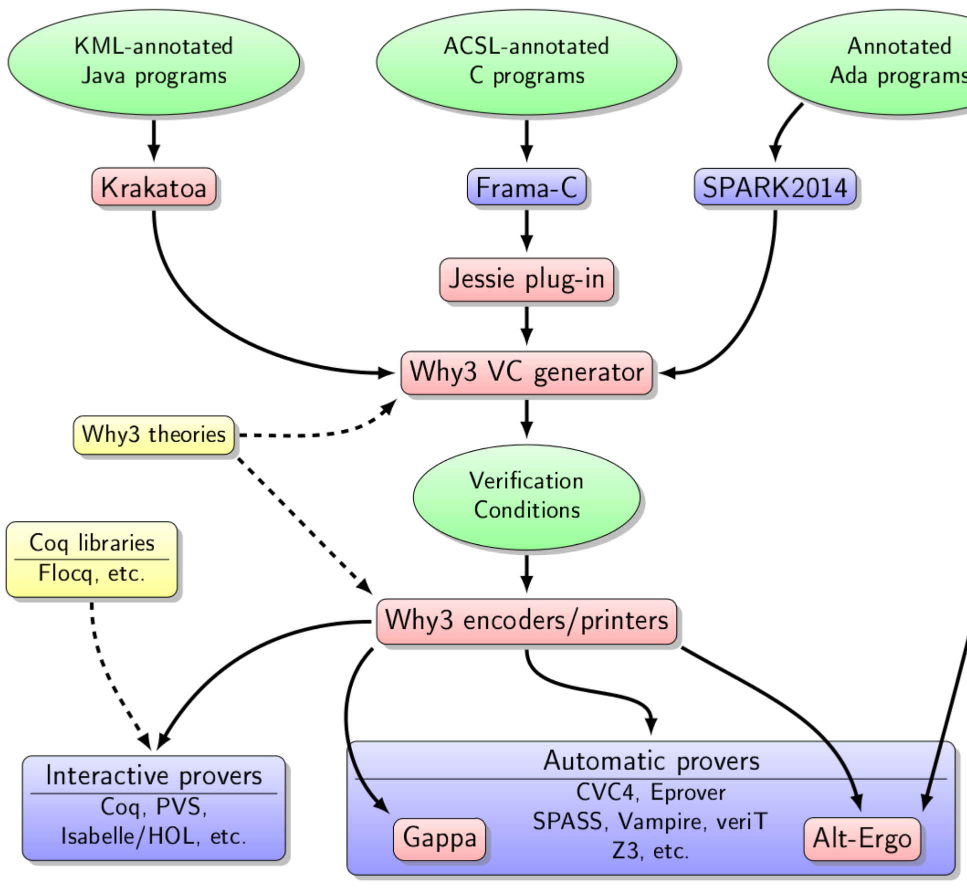

Figure 1: The Why3 ecosystem

3.3 Verification of Computer Arithmetic

Permanent researchers: S. Boldo, C. Marché, G. Melquiond

We of course want to keep this axis which is a major originality of Toccata. The main topics of the next 4 years will be:

- Fundamental studies concerning formalization of floating-point computations, algorithms, and error analysis. Related to numerical integration, we will develop the relationships between mathematical stability and floating-point stability of numerical schemes.
- A significant effort dedicated to verification of numerical programs written in C, Ada, C++. This involves combining specifications in real numbers and computation in floating-point, and underlying automated reasoning techniques with floating-point numbers and real numbers. A new approach we have in mind concerns some variant of symbolic execution of both code and specifications involving real numbers.
- We have not yet studied embedded systems. Our approach is first to tackle numerical filters. This requires more results on fixed-point arithmetic and a careful study of overflows.
- Also a specific focus on arbitrary precision integer arithmetic, in the continuation of the ongoing PhD thesis of R. Rieu-Helft.

3.4 Spreading Formal Proofs

Permanent researchers: S. Boldo, S. Conchon, J.-C. Filliâtre, C. Marché, G. Melquiond, A. Paskevich

This axis covers applications in general. The applications we currently have in mind are:

- Hybrid Systems, i.e., systems mixing discrete and continuous transitions. This theme covers many aspects such as general techniques for formally reasoning of differential equations, and extending SMT-based reasoning. The challenge is to support both abstract mathematical reasoning and concrete program execution (e.g., using floating-point representation). Hybrid systems will be a common effort with other members of the future laboratory joint with LSV of ENS Cachan.
- Applied mathematics, in the continuation of the current efforts towards verification of Finite Element Method. It has only been studied in the mathematical point of view during this period. We plan to also consider their floating-point behavior and a demanding application is that of molecular simulation exhibited in the new EMC2 project. The challenge here is both in the mathematics to be formalized, in the numerical errors that have never been studied (and that may be huge in specific cases), and in the size of the programs, which requires that our tools scale.
- Continuation of our work on analysis of shell scripts. The challenge is to be able to analyze a large number of scripts (more than 30,000 in the corpus of Debian packages installation scripts) in an automatic manner. An approach that will be considered is some form of symbolic execution.
- Explore proof tools for mathematics, in particular automated reasoning for real analysis (application: formalization of the weak Goldbach conjecture), and in number theory.
- Obtain and distribute verified OCaml libraries, as expected outcome of the VOCaL project.
- Formalization of abstract interpretation and WP calculi: in the continuation of the former project Verasco, and an ongoing project proposal joint with CEA List. The difficulty of achieving full verification of such tools will be mitigated by use of certificate techniques.

4 Application domains

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.

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 future.

4.1 Safety-Critical Software in Transportation

Transfer to aeronautics: Airbus France Development of the control software of Airbus planes historically includes advanced usage of formal methods. A first aspect is the usage of the CompCert verified compiler for compiling C source code. Our work in cooperation with Gallium team for the safe compilation of floating-point arithmetic operations [3] is directly in application in this context. A second aspect is the usage of the Frama-C environment for static analysis to verify the C source code. In this context, both our tools Why3 and Alt-Ergo are indirectly used to verify C code.

Transfer to the community of Atelier B In the former ANR project BWare, 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. The transfer effort continues nowadays through the FUI project LCHIP.

ProofInUse joint lab: transfer to the community of Ada development Through 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 the 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.

The impact of ProofInUse can also be measured in term of job creation: the first two ProofInUse engineers, D. Hauzar and C. Fumex, employed initially on Inria temporary positions, have now been hired on permanent positions in AdaCore company. It is also interesting to notice that this effort allowed AdaCore company to get new customers, in particular the domains of application of deductive formal verification went beyond the historical domain of aerospace: application in automotive, cyber-security, health (artificial heart).

4.2 Formal Analysis of Debian packages

Impactful results were produced in the context of the CoLiS project for the formal analysis of Debian packages. A first important step was the version 2 of the design of the CoLiS language done by B. Becker, C. Marché and other co-authors [39], that includes a modified formal syntax, a extended formal semantics, together with the design of concrete and symbolic interpreters. Those interpreters are specified and implemented in Why3, proved correct (following the initial approach for the concrete interpreter published in 2018 [58] and an approach for symbolic interpretation [38]), and finally extracted to OCaml code.

To make the extracted code effective, it must be linked together with a library that implements a solver for feature constraints [60], and also a library that formally specifies the behavior of basic UNIX utilities. The latter library is documented in details in a research report [59].

A third result is a large verification campaign running the CoLiS toolbox on all the packages of the current Debian distribution. The results of this campaign were reported in another article [37] that was submitted to TACAS conference in 2020, and finally presented in the 2021 edition. The most visible side effect of this experiment is the discovery of bugs: more than 150 bug reports have been filled against various Debian packages.

4.3 Extensions of ProofInUse joint lab

The current plans for continuation of the ProofInUse joint lab is to form a ProofInUse Consortium with an extension at a larger perimeter than Ada applications. We started to collaborate with the TrustInSoft company for the verification of C and C++ codes, including the use of Why3 to design verified and reusable C libraries (ongoing CIFRE PhD thesis). We also started to collaborate with Mitsubishi Electric R&D Centre Europe in Rennes for a specific usage of Why3 for verifying embedded devices (logic controllers). The recent best paper award at the FMICS conference is a result of this last collaboration.

5 Social and environmental responsibility

5.1 Footprint of research activities

Our research activities make use of computers for developing software and developing formal proofs. A continuous integration methodology for mature software like Why3 is 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 2021 remained very low with respect to previous years, of course because of the Covid pandemic. The impact on research was mitigated thanks to the possibility of participating to conferences using remote communication systems. 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 more safe 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*).

6 Highlights of the year

6.1 Agrégation d'Informatique

In the past years, increasingly more Computer Science topics have been introduced in the high school curricula. This evolution resulted in an increasing need for teachers with high skills in that domain. The French ministry of education has decided in 2021 to create a new discipline in the *concours de l'agrégation*, which is the most selective competition for recruiting high school teachers. To prepare the first round of this recruiting competition taking place in 2022, Sylvie Boldo has been nominated as president of the competition committee. Notice that she is the only full-time researcher to chair an *agrégation* committee this year.

6.2 Awards

VerifyThis Competition 2021 The team of Jean-Christophe Filliâtre and Andrei Paskevich got the *Best overall team* award. The team of Quentin Garchery and Xavier Denis won the first place for the *Best student team* award.

VerifyThis is a series of program verification competitions, which takes place annually since 2011. The competition offers a number of challenges presented in natural language and pseudo-code. Participants have to formalize the requirements, implement a solution, and formally verify the implementation for adherence to the specification.

FMICS 2021 best paper Cláudio Belo Lourenço and Claude Marché, with co-authors from Mitsubishi Electric R&D (Rennes, France) received the *Best-Paper Award* at the 26th International Conference on Formal Methods for Industrial Critical Systems.

Their contribution *Automated Verification of Temporal Properties of Ladder Programs* was valued by the jury as a "good example for how formal methods can be used in industrial applications" with "industrial interest for both legacy Ladder programs and programs to be developed".

7 New software and platforms

The following lists all the software distributed publicly and for which at least one author is member of the team.

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: `http://alt-ergo.lri.fr`

Contact: Sylvain Conchon

Participants: Alain Mebsout, Évelyne Contejean, Mohamed Iguernelala, Stéphane 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://gitlab.inria.fr/coqinterval/interval`

Publications: hal-00180138, hal-00797913, hal-01086460, hal-01289616, hal-01630143

Contact: Guillaume Melquiond

Participants: Assia Mahboubi, Érik 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/`

Contact: Sylvie Boldo

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: `http://cubicle.lri.fr/`

Contact: Sylvain Conchon

Participants: Alain Mebsout, Sylvain Conchon

7.1.5 Flocq

Name: The Flocq library for formalizing floating-point arithmetic in Coq

Keywords: Floating-point, Arithmetic code, Coq

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: `http://flocq.gforge.inria.fr/`

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

Contact: Sylvie Boldo

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: 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

Participant: Guillaume Melquiond

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: http://why3.lri.fr/

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

Keywords: Proof, Certification, Formalisation

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: Coq version 8.14 integrates many usability improvements, as well as an important change in the core language. The main changes include:

- The internal representation of match has changed to a more space-efficient and cleaner structure, allowing the fix of a completeness issue with cumulative inductive types in the type-checker. The internal representation is now closer to the user-level view of match, where the argument context of branches and the inductive binders "in" and "as" do not carry type annotations.

- A new "coqnative" binary performs separate native compilation of libraries, starting from a .vo file. It is supported by coq_makefile.

- Improvements to typeclasses and canonical structure resolution, allowing more terms to be considered as classes or keys.

- More control over notation declarations and support for primitive types in string and number notations.

- Removal of deprecated tactics, notably omega, which has been replaced by a greatly improved lia, along with many bug fixes.

- New Ltac2 APIs for interaction with Ltac1, manipulation of inductive types and printing.

Many changes and additions to the standard library in the numbers, vectors and lists libraries. A new signed primitive integers library Sint63 is available in addition to the unsigned Uint63 library.

News of the Year: Coq version 8.14 integrates many usability improvements, as well as an important change in the core language. See the changelog at https://coq.inria.fr/refman/changes.html#version-8-14 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, Frederic Besson, Tej Chajed, Cyril Cohen, Pierre Corbineau, Pierre Courtieu, Maxime Denes, Jim Fehrle, Julien Forest, Emilio Jesús Gallego Arias, Gaetan Gilbert, Georges Gonthier, Benjamin Grégoire, Jason Gross, Hugo Herbelin, Vincent Laporte, Olivier Laurent, Assia Mahboubi, Kenji Maillard, Érik 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

Partners: CNRS, Université Paris-Sud, ENS Lyon, Université Paris-Diderot

8 New results

8.1 Foundations and Spreading of Deductive Program Verification

Participants: Andrei Paskevich, Antoine Lanco, Benedikt Becker, Claude Marché, Cláudio Belo Lourenço, Clément Pascutto, Guillaume Cluzel, Guillaume Melquiond, Jean-Christophe Filliâtre, Léo Andrès, Quentin Garchery, Sylvain Conchon, Sylvie Boldo, Xavier Denis, Yacine El Haddad.

A strong call-by-need calculus T. Balabonski, A. Lanco, and G. Melquiond have devised a call-by-need lambda-calculus that enables strong reduction (that is, reduction inside the body of abstractions) and guarantees that arguments are only evaluated if needed and at most once [15]. This calculus uses explicit substitutions and subsumes the existing strong-call-by-need strategy, but allows for more reduction sequences, and often shorter ones, while preserving the neededness. The calculus is normalizing in a strong sense: Whenever a lambda-term t admits a normal form n in the lambda-calculus, then any reduction sequence from t in the calculus eventually reaches a representative of the normal form n. Moreover, by adding some restrictions to it, the calculus gains the diamond property and only performs reduction sequences of minimal length, which makes it systematically better than the existing strategy. The Abella proof assistant has been used to formalize part of this calculus.

Certificates for Logic Transformations In a context of formal program verification, using automatic provers, the trusted code base of verification environments is typically very broad. An environment such as Why3 implements many complex procedures: generation of verification conditions, logical transformations of proof tasks, and interactions with external provers. Considering only the logical transformations of Why3, their implementation already amounts to more than 17,000 lines of OCaml code. In order to increase our confidence in the correction of such a verification tool, Q. Garchery proposed a mechanism of certifying transformations, producing certificates that can be validated by an external tool, according to the *skeptical* approach. He explored two methods to validate certificates: one based on a dedicated verifier developed in OCaml, the other based on the universal proof checker Dedukti. A specificity of these certificates is to be "small grains" and composable, which makes the approach incremental, allowing to gradually add new certifying transformations [56]. Among the new results in that topics, the approach is now supporting built-in theories such as equality and integer arithmetic [20]. It is even possible to check certificates for proofs by strong induction on integers.

Simpler Proofs with Decentralized Invariants When verifying programs where the data have some recursive structure, it is natural to make use of global invariants that are themselves recursively defined. Though this is mathematically elegant, this makes the proofs more complex, as the preservation of these invariants now requires induction. In particular, this makes the proofs less amenable to automation. An alternative is to use local invariants attached to individual components of the structure and which only involve a bounded number of elements. These are called *decentralized invariants*. When the structure is updated, the footprint of the modification only impacts a bounded number of invariants and reestablishing them does not require induction. In this paper [13], Filliâtre illustrates this idea on three non-trivial programs, for which fully automated proofs are achieved.

This paper appears in a special issue "E pur si muove" of the Journal of Logical and Algebraic Methods in Programming, that is tribute to José Manuel Esgalhado Valença on the occasion of his jubilation.

Abstraction and Genericity in Why3 The benefits of modularity in programming — abstraction barriers, which allow hiding implementation details behind an opaque interface, and genericity, which allows specializing a single implementation to a variety of underlying data types — apply just as well to deductive program verification, with the additional advantage of helping the automated proof search procedures by reducing the size and complexity of the premises and by instantiating and reusing once-proved properties in a variety of contexts. Filliâtre and Paskevich wrote a paper [53] demonstrating the modularity features of WhyML, the language of the program verification tool Why3. Instead of separating abstract interfaces and fully elaborated implementations, WhyML uses a single concept of *module*, a collection of abstract and concrete declarations, and a basic operation of *cloning* which instantiates a module with respect to a given partial substitution, while verifying its soundness. This mechanism brings into WhyML both abstraction and genericity, which is illustrated on a small verified Bloom filter implementation, translated into executable idiomatic C code.

Explaining Counterexamples with Giant-Step Assertion Checking Identifying the cause of a proof failure during deductive verification of programs is hard: it may be due to an incorrectness in the program, an incompleteness in the program annotations, or an incompleteness of the prover. The changes needed to resolve a proof failure depend on its category, but the prover cannot provide any help on the categorization. When using an SMT solver to discharge a proof obligation, that solver can propose a model from a failed attempt, from which a possible counterexample can be derived. But the counterexample may be invalid, in which case it may add more confusion than help. To check the validity of a counterexample and to categorise the proof failure, B. Becker, C. Lourenço and C. Marché propose the comparison between the run-time assertion-checking (RAC) executions under two different semantics, using the counterexample as an oracle. The first RAC execution follows the normal program semantics, and a violation of a program annotation indicates an incorrectness in the program. The second RAC execution follows a novel "giant-step" semantics that does not execute loops nor function calls but instead retrieves return values and values of modified variables from the oracle. A violation of the program annotations only observed under giant-step execution characterizes an incompleteness of the program annotations. They implemented this approach in Why3 and evaluated it using examples from prior literature [27, 16]

8.2 Reasoning on mutable memory in program verification

Participants: Andrei Paskevich, Benedikt Becker, Claude Marché, Cláudio Belo Lourenço, Clément Pascutto, Guillaume Cluzel, Guillaume Melquiond, Jean-Christophe Filliâtre, Léo Andrès, Sylvain Conchon, Xavier Denis, Yacine El Haddad.

Deductive program verification for a language with a Rust-like typing discipline 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 X. Denis, supervised by J.-H. Jourdan and C. 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 [51]. This method is implemented in a new standalone tool called Creusot. 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. Prophecies also permit efficient automated reasoning for verifying about such programs. Moreover, Rust provides advanced abstraction features based on a notion of *traits*, extensively used in the standard library and in user code. The support for traits is another main feature of Creusot, because it is at the heart of its approach, in particular for providing complex abstraction of the functional behavior of programs [30].

8.3 Verification of Computer Arithmetic

Participants: Claude Marché, Diane Gallois-Wong, Guillaume Melquiond, Houda Mouhcine, Louise Ben Salem-Knapp, Sylvie Boldo.

Plotting in a formally verified way An invaluable feature of computer algebra systems is their ability to plot the graph of functions. Unfortunately, when one is trying to design a library of mathematical functions, this feature often falls short, producing incorrect and potentially misleading plots, due to accuracy issues inherent to this use case. G. Melquiond has extended the CoqInterval library so as to turn the Coq proof assistant into a tool for plotting function graphs that are guaranteed to be correct, by using reliable polynomial approximations [22].

Some Formal Tools for Computer Arithmetic: Flocq and Gappa S. Boldo and G. Melquiond have developed several tools to help the user in writing proofs regarding computer arithmetic, e.g., certifying a bound on a round-off error, while aiming at a high level of guarantee. Flocq is a library of mathematical definitions and theorems for the Coq proof assistant; Gappa is meant to compute bounds of values and errors, while producing the corresponding formal proof. Despite their age, they are still relevant nowadays [17].

Round-Off Errors and Hydrodynamics The growth of the computing capacities makes it possible to obtain more and more precise simulation results, often calculated in *binary64*. However, exascale is pushing back the known limits and the problems of accumulating round-off errors could come back and require to increase further the precision. But working with extended precision, regardless of the method used, has a significant cost in memory, computation time and energy. It is therefore important to measure the robustness of the *binary64* format by anticipating the future computing resources in order to ensure its durability in numerical simulations. For this purpose, W. Weens,

T. Vazquez-Gonzalez and L. Ben Salem-Knapp performed a set of numerical experiments [25]. Those were performed with *weak floats* which were specifically designed to conduct an empirical study of round-off errors in hydrodynamic simulations and to build an error model that extracts the part due to round-off error in the results. This model confirms that errors remain dominated by the scheme errors in the performed numerical experiments. Other numerical experiments have been done in order to check whether *binary64* provides enough accuracy in the context of hydrodynamics exascale computations [23].

Numerical simulations are carefully-written programs, and their correctness is based on mathematical results. Nevertheless, those programs rely on floating-point arithmetic and the corresponding round-off errors are often ignored. L. Ben Salem-Knapp, S. Boldo and W. Weens studied a specific simple scheme applied to advection, that is a particular equation from hydrodynamics dedicated to the transport of a substance. Their work shows a tight bound on the round-off error of the 1-dimensional and 2-dimensional upwind schemes, with an error roughly proportional to the number of steps. The error bounds are generic with respect to the floating-point format and exceptional behaviors are taken into account. Some experiments give an insight of the quality of the bounds [28].

Emulating round-to-nearest-ties-to-zero "augmented" floating- point operations The 2019 version of the IEEE 754 Standard for Floating-Point Arithmetic recommends that new “augmented” operations should be provided for the binary formats. These operations use a new “rounding direction”: round to nearest *ties-to-zero*. S. Boldo, C. Lauter, and J.-M. Muller show how they can be implemented using the currently available operations, using round-to-nearest *ties-to-even* with a partial formal proof of correctness [12].

Computable analysis and notions of continuity in Coq We give a number of formal proofs of theorems from the field of computable analysis. Many of our results specify executable algorithms that work on infinite inputs by means of operating on finite approximations and are proven correct in Coq in the sense of computable analysis. We also formalize proofs of non-computational results that support the correctness of our definitions. These include that the information theoretic notion of continuity used in the library is equivalent to the metric notion of continuity on Baire space, a complete comparison of the different concepts of continuity that arise from metric and represented-space structures and the discontinuity of the unrestricted limit operator on the real numbers and the task of selecting an element of a closed subset of the natural numbers [14].

A Coq Formalization of Lebesgue Integration of Nonnegative Functions Integration, just as much as differentiation, is a fundamental calculus tool that is widely used in many scientific domains. Formalizing the mathematical concept of integration and the associated results in a formal proof assistant helps in providing the highest confidence on the correctness of numerical programs involving the use of integration, directly or indirectly. By its capability to extend the (Riemann) integral to a wide class of irregular functions, and to functions defined on more general spaces than the real line, the Lebesgue integral is perfectly suited for use in mathematical fields such as probability theory, numerical mathematics, and real analysis. In this article, we present the Coq formalization of σ-algebras, measures, simple functions, and integration of non-negative measurable functions, up to the full formal proofs of the Beppo Levi Theorem (monotone convergence) and Fatou’s Lemma. More than a plain formalization of the known literature, we present several design choices made to balance the harmony between mathematical readability and usability of Coq theorems [11].

8.4 Spreading Formal Proofs

Participants: Andrei Paskevich, Antoine Lanco, Benedikt Becker, Claude Marché, Cláudio Belo Lourenço, Clément Pascutto, Diane Gallois-Wong, Guillaume Cluzel, Guillaume Melquiond, Houda Mouhcine, Jean-Christophe Filliâtre, Louise Ben Salem-Knapp, Léo Andrès, Quentin Garchery, Sylvain Conchon, Sylvie Boldo, Xavier Denis, Yacine El Haddad.

Verification of Ladder programs Programmable Logic Controllers (PLCs) are industrial digital computers used as automation controllers in manufacturing processes. The Ladder language is a programming language used to develop PLC software. The aim of this work is to prove that a given Ladder program conforms to an expected temporal behavior given as a timing chart, describing scenarios of execution. C. Lourenço and C. Marché, in collaboration with Mitsubishi Electric R&D Centre Europe, developed an approach to translate the Ladder code and the timing chart into a program for the Why3 environment. The ultimate goal is two-fold: first, by obtaining a complete proof, one can verify the conformance of the Ladder code with respect to the timing chart with a high degree of confidence. Second, when the proof is not fully completed, one can obtain a counterexample, illustrating a possible execution scenario of the Ladder code which does not conform to the timing chart [32, 21].

Runtime Assertion Checking for OCaml Runtime assertion checking (RAC) is a convenient set of techniques that lets developers abstract away the process of verifying the correctness of their programs by writing formal specifications and automating their verification at runtime. In this work [18], Filliâtre and Pascutto present Ortac, a runtime assertion checking tool for OCaml libraries and programs. OCaml is a functional programming language in which idioms rely on an expressive type system, modules, and interface abstractions. Ortac consumes interfaces annotated with type invariants and function contracts and produces code wrappers with the same signature that check these specifications at runtime. It provides a flexible framework for traditional assertion checking, monitoring misbehaviors without interruptions, and automated fuzz testing for OCaml programs. This work presents an overview of Ortac features and highlights its main design choices.

Leveraging Formal Specifications to Generate Fuzzing Suites When testing a library, developers typically first have to capture the semantics they want to check. They then write the code implementing these tests and find relevant test cases that expose possible misbehaviours. In this work, N. Osborne and C. Pascutto present a tool that automatically takes care of these last two steps by automatically generating fuzz testing suites from OCaml interfaces annotated with formal behavioral specifications. They also show some ongoing experiments on the capabilities and limitations of fuzzing applied to real-world libraries [24]

Formal Analysis of Debian packages Several new results were produced in the context of the CoLiS project for the formal analysis of Debian packages. A first important step is the version 2 of the design of the CoLiS language done by B. Becker, C. Marché and other co-authors [39], that includes a modified formal syntax, a extended formal semantics, together with the design of concrete and symbolic interpreters. Those interpreters are specified and implemented in Why3, proved correct (following the initial approach for the concrete interpreter published in 2018 [58] and the recent approach for symbolic interpretation mentioned above [38]), and finally extracted to OCaml code.

To make the extracted code effective, it must be linked together with a library that implements a solver for feature constraints [60], and also a library that formally specifies the behavior of basic UNIX utilities. The latter library is documented in details in a research report [59].

A third result is a large verification campaign running the CoLiS toolbox on all the packages of the current Debian distribution. The results of this campaign were reported in another article [37] that was presented at TACAS conference in 2021. The most visible side effect of this experiment is the discovery of bugs: more than 150 bugs report have been filled against various Debian packages. A journal paper reporting updated experimental results using an improved implementation of the platform, and on the new Debian stable distribution, is under submission.

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-AdaCore Collaboration

Participants: Claude Marché *(contact)*, Jean-Christophe Filliâtre, Andrei Paskevich, Guillaume Melquiond, Benedikt Becker.

This collaboration is a joint effort of the Inria project-team Toccata and the AdaCore company which provides development tools for the Ada programming language. It is funded by a 5-year bilateral contract from Jan 2019 to Dec 2023.

The SME AdaCore is a software publisher specializing in providing software development tools for critical systems. A previous successful collaboration between Toccata and AdaCore enabled Why3 technology to be put into the heart of the AdaCore-developed SPARK technology.

The objective of ProofInUse-AdaCore is to significantly increase the capabilities and performances of the Spark/Ada verification environment proposed by AdaCore. It aims at integration of verification techniques at the state-of-the-art of academic research, via the generic environment Why3 for deductive program verification developed by Toccata.

9.2 ProofInUse-MERCE Collaboration

Participants: Claude Marché *(contact)*, Guillaume Melquiond, Cláudio Belo Lourenço, Yacine El Haddad.

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 3 years and 6 months from Nov 2019 to April 2023.

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.

9.3 ProofInUse-TrustInSoft Collaboration

Participants: Claude Marché *(contact)*, Guillaume Melquiond, Guillaume Cluzel.

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 is funded by a bilateral contract of 24 months from Dec 2020 to Nov 2022.

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, with a special interest in the generation of counterexample to help the user in case of proof failure.

9.4 CEA-DAM Collaboration

Participants: Sylvie Boldo *(contact)*, Louise Ben Salem-Knapp.

A contract has been signed in 2021 between the CEA-DAM (*"Direction des applications militaires"*) and Toccata about the management of the PhD thesis of Louise Ben Salem-Knapp with William Weens (CEA-DAM) and Guillaume Perrin (CEA-DAM). The PhD has stopped in October 2021, also ending the contract.

This topic of the PhD is between computer science and applied mathematics. We consider algorithms from numerical analysis and verify their good behavior on computers. This behavior, proven by supposing that the computations are perfect, could be put in fault by the problems of round-off errors and of overflows due to computations in floating-point arithmetic. We plan to study the impact of round-off errors in a hydrodynamic code. Hydrodynamics is the skeleton model of many physical models used in industry. It contains numerous technical, mathematical and numerical difficulties, which does not prevent its massive use in the simulation industry on increasingly complex problems. Today, the resolution of such problems requires the use of super-calculators, as well as the implementation of algorithms adapted to massively parallel calculation. The very large number of calculations required to produce results raises the question of their numerical quality.

9.5 CIFRE contract with Tarides company

Participants: Jean-Christophe Filliâtre *(contact)*, Clément Pascutto.

Clément Pascutto started a CIFRE PhD in June 2020, under then supervision of Jean-Christophe Filliâtre (at Toccata) and Thomas Gazagnaire (at Tarides). The subject of the PhD is the dynamic and deductive verification of OCaml programs and its application to distributed data structures.

9.6 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.

10 Partnerships and cooperations

10.1 European initiatives

10.1.1 FP7 & H2020 projects

EMC2, ERC Synergy project

Title: Extreme-scale Mathematically-based Computational Chemistry

Duration: November 2019 – April 2026

Coordinators: E. Cances, L. Grigori, Y. Maday, and J. P. Piquemal

Partners:

- *LJLL and LCT*, Sorbonne Université (France)
- *Cermics*, École Nationale des Ponts et Chaussées (France)

Inria contact: Laura Grigori

Website

EMC2 is an ERC Synergy project that aims to overcome some of the current limitations in the field of molecular simulation and aims to provide academic communities and industrial companies with new generation, dramatically faster and quantitatively reliable molecular simulation software. This will enable those communities to address major technological and societal challenges of the 21st century in health, energy and the environment for instance.

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. However, 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).

11 Dissemination

Participants: Andrei Paskevich, Antoine Lanco, Claude Marché, Diane Gallois-Wong, Guillaume Melquiond, Jean-Christophe Filliâtre, Quentin Garchery, Sylvain Conchon, Sylvie Boldo, Xavier Denis.

11.1 Administration, Collective Responsibilities

- S. Boldo has been nominated president of the (first) *concours de l'agrégation d'informatique* that will recruit French computer science teachers for high school in 2022.
- S. Boldo, deputy scientific director (DSA) of Inria Saclay research center, until June 21.
- S. Boldo, member of the council of the Computer Science Graduate School of Université Paris-Saclay (and co-head for open science).
- S. Boldo, member of the (national) Inria IES commission (*"commission pour l'information et l'édition scientifique"*) about scientific edition and publication models.
- S Boldo, member of the CLFP (*"commission locale de formation permanente"*).
- S. Boldo, member of the CoDiReV Paris-Saclay (committee of the research heads of the Paris-Saclay components and partners) until June 21.
- S. Boldo, deputy member of the CFVU Paris-Saclay (*"commission de la formation et de la vie universitaire"*) until June 21.
- S. Boldo, member of the partners commission for the Digicosme Labex (*"comité des tutelles du labex Digicosme"*) until June 21.
- S. Boldo, member of the crisis unit of Inria Saclay until June 21.
- G. Melquiond, member of the scientific commission of Inria Saclay, in charge of selecting candidates for PhD grants, Post-doc grants, temporary leaves from universities ("délégations").
- G. Melquiond, elected member of the ED STIC doctoral school from Université Paris-Saclay, in charge of selecting candidates for PhD grants and monitoring PhD students.
- G. Melquiond, member of Inria's MissionJC (*Jeunes Chercheurs*), in charge of funding thematic research schools.
- G. Melquiond, responsible of Inria Saclay's FpR (*Formation par la Recherche*), in charge of funding participation of PhD students to thematic research schools.
- A. Paskevich, member of the CCSU ("*commission consultative de spécialistes de l'université*"), section 27, of Université Paris-Saclay.

11.2 Promoting scientific activities

Chair of conference program committees

- A. Paskevich, 6th Workshop on Formal Integrated Development Environment, F-IDE'2021

Member of the conference program committees

- S. Boldo, 2021 IEEE Symposium on Computer Arithmetic, ARITH'2021 (and will be in 2022)
- S. Boldo, 13th NASA Formal Methods Symposium, NFM'2021 (and will be in 2022)
- S. Boldo, 2022 Certified Programs and Proofs, CPP'2022
- S. Boldo, 13th International Conference on Interactive Theorem Proving, ITP'2022
- J.-C. Filliâtre, Verification, Model Checking, and Abstract Interpretation, VMCAI 2021
- J.-C. Filliâtre, 13th NASA Formal Methods Symposium, NFM'2021
- J.-C. Filliâtre, XXV Brazilian Symposium on Programming Languages, SBLP 2021
- J.-C. Filliâtre, Symposium on Languages, Applications and Technologies, SLATE 2021
- G. Melquiond, 28th IEEE Symposium on Computer Arithmetic, ARITH'2021

11.2.1 Journal

Member of the editorial boards

- J.-C. Filliâtre, member of the editorial board of *Journal of Functional Programming*.
- G. Melquiond, member of the editorial board of *Reliable Computing*.
- A. Paskevich, member of the editorial board of *Formal Methods in System Design*.

11.2.2 Invited talks

- X. Denis: "Deductive verification of Rust programs", Workshop Langages, Verification et Preuves, Mar. 12, 2021
- X. Denis: "Creusot: A prototype tool for verification of Rust software", Workshop RustVerify, Apr 12, 2021
- J.-C. Filliâtre: "Gospel, un langage de spécification pour OCaml", Workshop Langages, Verification et Preuves, Mar. 12, 2021
- Q. Garchery: "Génération et vérification de certificats pour les transformations logiques", Workshop Langages, Verification et Preuves, Nov. 23, 2021

11.2.3 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, chair of IFIP WG 1.9/2.15 verified Software.

11.2.4 Scientific expertise

- S. Boldo, member of the HCERES evaluation committee for the LaBRI laboratory in Bordeaux, January 18–21.
- S. Boldo, vice-president of the Inria CRCN-ISFP recruitment committee for Inria Saclay – Île-de-France research center.
- S. Boldo, member of the Inria DR recruitment committee (written part only)
- S. Boldo, member of the Inria PEDR committees
- S. Boldo, member of the Scientific Council of CentraleSupélec (until June 21)
- J.-C. Filliâtre, grading the entrance examination at X/ENS (*"option informatique"*).
- C. Marché, president of the Inria CRCN-ISFP recruitment committee for Inria-Paris-Rocquencourt research center.
- G. Melquiond, grading the entrance examination at X/ENS (*"option informatique"*).

11.3 Teaching - Supervision - Juries

11.3.1 Teaching

- S. Boldo, *Floating-point Arithmetic and Beyond*, 6h, M2, École Normale Supérieure de Lyon, France.
- S. Conchon and J.-C. Filliâtre, *DIU Enseigner l'Informatique au Lycée*, 2 weeks, rectorat de Versailles (together with T. Balabonski and K. Nguyen).
- 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.
- Q. Garchery, *Compilation*, 24h, L3, Université Paris-Saclay, France.
- Q. Garchery, *Programmation Fonctionnelle Avancée*, 24h, L3, Université Paris-Saclay, France.
- Q. Garchery, *Algorithmique*, 10h, 2nd year, Polytech, Université Paris-Saclay, France.
- A. Lanco, *Projet*, 20h, L3, Université Paris-Saclay, France.
- A. Lanco, *Compilation*, 24h, L3, Université 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.
- G. Melquiond, *Floating-point Arithmetic and Beyond*, 12h, M2, École Normale Supérieure de Lyon, France.
- A. Paskevich, *Vérification Déductive*, 12h, M1, MPRI, Université Paris-Saclay, France.
- A. Paskevich, *Principes de systèmes d'exploitation*, 40h+54h, DUT2, IUT d'Orsay, Université Paris-Saclay, France.

11.3.2 Supervision

- PhD: D. Gallois-Wong, “Vérification formelle et filtres numériques”, since Oct. 2017, supervised by S. Boldo and T. Hilaire, defended in March 2021 [26].
- PhD in progress: Q. Garchery, “Certification de la génération et de la transformation d’obligations de preuve”, since Oct. 2018, supervised by C. Keller, C. Marché and A. Paskevich.
- PhD in progress: A. Lanco, “Stratégies pour la réduction forte”, since Oct. 2019, supervised by T. Balabonski and G. Melquiond.
- PhD in progress: C. Pascutto, “Runtime and Deductive Verification of OCaml programs and applications to Mergeable Data Structures”, since June 2020, supervised by J.-C. Filliâtre.
- PhD stopped: L. Ben Salem-Knapp, “Erreurs d’arrondi sur un code d’hydrodynamique”, from Oct. 2020 to Oct. 2021, supervised by S. Boldo, W. Weens and G. Perrin.
- PhD in progress: X. Denis, “Deductive program verification for Rust”, since Oct. 2020, supervised by J.-H. Jourdan and C. Marché.
- PhD in progress: L. Andrès, “Formalization of a garbage collector for WebAssembly”, since Oct. 2021, supervised by J.-C. Filliâtre.

11.3.3 Juries

- S. Boldo, member and reviewer of the habilitation (HDR) committee of Assia Mahboubi (Jan 5)
- S. Boldo, member and president of the PhD defense of Maxime Jacquemin (U. Paris-Saclay, July 15)

11.4 Popularization

11.4.1 Interventions

- S. Boldo, participant of a speed-dating with high school female students during the RJMI (*Rencontres des Jeunes Mathématiciennes et Informaticiennes*) organized online by Inria Saclay (Feb 26)
- S. Boldo, animation of a stand for the Inria Saclay - Île-de-France *Fête de la science* (Oct 8)

12 Scientific production

12.1 Major publications

[1] B. Becker, N. Jeannerod, C. Marché, Y. Régis-Gianas, M. Sighireanu and R. Treinen. ‘Analysing installation scenarios of Debian packages’. In: *Tools and Algorithms for the Construction and Analysis of Systems*. TACAS 2020 - 26th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Vol. 12079. Lecture Notes in Computer Science. The conference took place on-line, because it couldn’t be held in Dublin, Ireland, 17th Apr. 2020, pp. 235–253. DOI: `10.1007/978-3-030-45237-7_14`. URL: `https://hal.archives-ouvertes.fr/hal-02355602`.

[2] 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`.

[3] S. Boldo, J.-H. Jourdan, X. Leroy and G. Melquiond. ‘Verified Compilation of Floating-Point Computations’. In: *Journal of Automated Reasoning* 54.2 (Feb. 2015), pp. 135–163. URL: `https://hal.inria.fr/hal-00862689`.

[4] 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`.

[5] M. Clochard, L. Gondelman and M. Pereira. 'The Matrix Reproved'. In: *Journal of Automated Reasoning* 60.3 (2018), pp. 365–383. URL: https://hal.inria.fr/hal-01617437.

[6] 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*. 2017. URL: https://hal.inria.fr/hal-01522770.

[7] J.-C. Filliâtre, L. Gondelman and A. Paskevich. 'The Spirit of Ghost Code'. In: *Formal Methods in System Design* 48.3 (2016), pp. 152–174. URL: https://hal.archives-ouvertes.fr/hal-01396864v1.

[8] 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. DOI: 10.1007/s10817-018-9463-7. URL: https://hal.inria.fr/hal-01630143.

[9] G. Melquiond and R. Rieu-Helft. 'A Why3 Framework for Reflection Proofs and its Application to GMP's Algorithms'. In: *9th International Joint Conference on Automated Reasoning*. Ed. by D. Galmiche, S. Schulz and R. Sebastiani. Lecture Notes in Computer Science. Oxford, United Kingdom, July 2018. URL: https://hal.inria.fr/hal-01699754.

[10] P. Roux, M. Iguernlala and S. Conchon. 'A Non-linear Arithmetic Procedure for Control-Command Software Verification'. In: *24th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Thessalonique, Greece, Apr. 2018. URL: https://hal.archives-ouvertes.fr/hal-01737737.

12.2 Publications of the year

International journals

[11] 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* (2021). DOI: 10.1007/s10817-021-09612-0. URL: https://hal.inria.fr/hal-03471095.

[12] S. Boldo, C. Q. Lauter and J.-M. Muller. 'Emulating round-to-nearest ties-to-zero "augmented" floating-point operations using round-to-nearest ties-to-even arithmetic'. In: *IEEE Transactions on Computers* 70.7 (July 2021), pp. 1046–1058. DOI: 10.1109/TC.2020.3002702. URL: https://hal.archives-ouvertes.fr/hal-02137968.

[13] J.-C. Filliâtre. 'Simpler Proofs with Decentralized Invariants'. In: *Journal of Logical and Algebraic Methods in Programming* 121 (June 2021). URL: https://hal.inria.fr/hal-02518570.

[14] F. Steinberg, L. Théry and H. Thies. 'Computable analysis and notions of continuity in Coq'. In: *Logical Methods in Computer Science* (12th May 2021). URL: https://hal.inria.fr/hal-03324295.

International peer-reviewed conferences

[15] T. Balabonski, A. Lanco and G. Melquiond. 'A strong call-by-need calculus'. In: *Proceedings of the 6th International Conference on Formal Structures for Computation and Deduction*. FSCD 2021 - 6th International Conference on Formal Structures for Computation and Deduction. Vol. 195. Leibniz International Proceedings in Informatics 9. Buenos Aires, Argentina, July 2021, pp. 1–22. DOI: 10.4230/LIPIcs.FSCD.2021.9. URL: https://hal.inria.fr/hal-03149692.

[16] B. Becker, C. B. Lourenço and C. Marché. 'Explaining Counterexamples with Giant-Step Assertion Checking'. In: F-IDE 2021 - 6th Workshop on Formal Integrated Development Environments. Virtual, United States: Electronic Proceedings in Theoretical Computer Science, May 2021. DOI: 10.4204/EPTCS.338.10. URL: https://hal.inria.fr/hal-03217393.

[17] S. Boldo and G. Melquiond. 'Some Formal Tools for Computer Arithmetic: Flocq and Gappa'. In: ARITH 2021 - 28th IEEE International Symposium on Computer Arithmetic. Online, Italy, June 2021. URL: https://hal.inria.fr/hal-03233227.

[18] J.-C. Filliâtre and C. Pascutto. 'Ortac: Runtime Assertion Checking for OCaml'. In: RV'21 - 21st International Conference on Runtime Verification. Los Angeles, CA, United States, 11th Oct. 2021. URL: https://hal.inria.fr/hal-03252901.

[19] 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.

[20] Q. Garchery. 'A Framework for Proof-carrying Logical Transformations'. In: Proof eXchange for Theorem Proving. Vol. 336. Electronic Proceedings in Theoretical Computer Science. Virtual, United States: EPTCS, 7th July 2021, pp. 5–23. DOI: 10.4204/EPTCS.336.2. URL: https://hal.archives-ouvertes.fr/hal-03349223.

[21] C. Lourenço, D. Cousineau, F. Faissole, C. Marché, D. Mentré and H. Inoue. 'Automated Verification of Temporal Properties of Ladder Programs'. In: FMICS 2021 - Formal Methods for Industrial Critical Systems. Vol. 12863. Lecture Notes in Computer Science. Paris, France, 2021. DOI: 10.1007/978-3-030-85248-1_2. URL: https://hal.inria.fr/hal-03281580.

[22] G. Melquiond. 'Plotting in a Formally Verified Way'. In: Proceedings of the 6th Workshop on Formal Integrated Development Environment. Vol. 338. Electronic Proceedings in Theoretical Computer Science. Online, United States, May 2021, pp. 39–45. DOI: 10.4204/EPTCS.338.6. URL: https://hal.inria.fr/hal-03168208.

Conferences without proceedings

[23] L. Ben Salem-Knapp, T. Vazquez-Gonzalez and W. Weens. 'La double précision suffit-elle à l'exascale ?' In: AFADL 2021 - 20èmes journées Approches Formelles dans l'Assistance au Développement de Logiciels. Vannes, France, 16th June 2021. URL: https://hal.inria.fr/hal-03351615.

[24] N. Osborne and C. Pascutto. 'Leveraging Formal Specifications to Generate Fuzzing Suites'. In: OCaml Users and Developers Workshop, co-located with the 26th ACM SIGPLAN International Conference on Functional Programming. Virtual, United States, 22nd Aug. 2021. URL: https://hal.inria.fr/hal-03328646.

[25] W. Weens, T. Vazquez-Gonzalez and L. B. Salem-Knapp. 'Modeling round-off errors in hydrodynamic simulations'. In: NSV 2021 - 14th International Workshop on Numerical Software Verification. Los Angeles, United States, 18th July 2021. URL: https://hal.inria.fr/hal-03351754.

Doctoral dissertations and habilitation theses

[26] D. Gallois-Wong. 'Coq formalization of digital filter algorithms computed using finite precision arithmetic'. Université Paris-Saclay, 4th Mar. 2021. URL: https://tel.archives-ouvertes.fr/tel-03202580.

Reports & preprints

[27] B. Becker, C. Belo Lourenço and C. Marché. *Giant-step Semantics for the Categorisation of Counterexamples*. RR-9407. Inria, Apr. 2021, p. 43. URL: https://hal.inria.fr/hal-03213438.

[28] L. Ben Salem-Knapp, S. Boldo and W. Weens. *Bounding the Round-Off Error of the Upwind Scheme for Advection*. 31st Aug. 2021. URL: https://hal.inria.fr/hal-03329933.

[29] S. Boldo, F. Clément and L. Leclerc. *A Coq Formalization of the Bochner integral*. 7th Jan. 2022. URL: https://hal.inria.fr/hal-03516749.

[30] X. Denis, J.-H. Jourdan and C. Marché. *The Creusot Environment for the Deductive Verification of Rust Programs*. RR-9448. Inria Saclay - Île de France, 2021. URL: https://hal.inria.fr/hal-03526634.

[31] J.-C. Filliâtre. *Compte-rendu de fin de projet ANR-15-CE25-0008 "VOCaL": Programme CE25 2015*. LMF - Laboratoire Méthodes Formelles, June 2021. URL: https://hal.inria.fr/hal-03326775.

[32] C. Lourenço, D. Cousineau, F. Faissole, C. Marché, D. Mentré and H. Inoue. *Formal Analysis of Ladder Programs using Deductive Verification*. RR-9402. Inria, Apr. 2021, p. 25. URL: https://hal.inria.fr/hal-03199464.

[33] A. Paskevich. *Continuation Passing as an Abstract Syntax for Deductive Verification*. 19th Jan. 2021. URL: https://hal.inria.fr/hal-03115120.

12.3 Cited publications

[34] 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.

[35] C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds and C. Tinelli. 'CVC4'. In: *Computer Aided Verification*. Vol. 6806. Lecture Notes in Computer Science. Springer, 2011, pp. 171–177.

[36] P. Baudin, J.-C. Filliâtre, C. Marché, B. Monate, Y. Moy and V. Prevosto. *ACSL: ANSI/ISO C Specification Language, version 1.4*. 2009.

[37] B. Becker, N. Jeannerod, C. Marché, Y. Régis-Gianas, M. Sighireanu and R. Treinen. 'Analysing installation scenarios of Debian packages'. In: *Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 12079. Lecture Notes in Computer Science. 2020, pp. 235–253. URL: https://hal.archives-ouvertes.fr/hal-02355602.

[38] B. Becker and C. Marché. 'Ghost Code in Action: Automated Verification of a Symbolic Interpreter'. In: *Verified Software: Tools, Techniques and Experiments*. Ed. by S. Chakraborty and J. A.Navas. Vol. 12031. Lecture Notes in Computer Science. New York, United States, July 2019. URL: https://hal.inria.fr/hal-02276257.

[39] B. Becker, C. Marché, N. Jeannerod and R. Treinen. *Revision 2 of CoLiS language: formal syntax, semantics, concrete and symbolic interpreters*. Technical Report. HAL Archives Ouvertes, Oct. 2019. URL: https://hal.inria.fr/hal-02321743.

[40] 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.

[41] 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: *Proceedings of the First Interactive Theorem Proving Conference*. Ed. by M. Kaufmann and L. C. Paulson. Vol. 6172. LNCS. Edinburgh, Scotland: Springer, July 2010, pp. 147–162. URL: http://hal.inria.fr/inria-00450789/.

[42] 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/.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] A. Charguéraud, J.-C. Filliâtre, C. B. Lourenço and M. Pereira. 'GOSPEL — Providing OCaml with a Formal Specification Language'. In: *FM 2019 23rd International Symposium on Formal Methods*. Ed. by A. McIver and M. ter Beek. Porto, Portugal, Oct. 2019. URL: https://hal.inria.fr/hal-02157484.

[48] 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.

[49] 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.

[50] S. Dailler, D. Hauzar, C. Marché and Y. Moy. 'Instrumenting a Weakest Precondition Calculus for Counterexample Generation'. In: *Journal of Logical and Algebraic Methods in Programming* 99 (2018), pp. 97–113. URL: https://hal.inria.fr/hal-01802488.

[51] 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.

[52] 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/.

[53] J.-C. Filliâtre and A. Paskevich. 'Abstraction and Genericity in Why3'. In: *9th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISoLA)*. Ed. by T. Margaria and B. Steffen. Vol. 12476. Lecture Notes in Computer Science. Rhodes, Greece: Springer, Oct. 2020, pp. 122–142. URL: https://hal.inria.fr/hal-02696246.

[54] C. Fumex, C. Dross, J. Gerlach and C. Marché. 'Specification and Proof of High-Level Functional Properties of Bit-Level Programs'. In: *8th NASA Formal Methods Symposium*. Ed. by S. Rayadurgam and O. Tkachuk. Vol. 9690. Lecture Notes in Computer Science. Minneapolis, MN, USA: Springer, June 2016, pp. 291–306. URL: https://hal.inria.fr/hal-01314876.

[55] 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/.

[56] Q. Garchery, C. Keller, C. Marché and A. Paskevich. 'Des transformations logiques passent leur certicat'. In: *Trente-et-unièmes Journées Francophones des Langages Applicatifs*. Ed. by Z. Dargaye and Y. Régis-Gianas. Gruissan, France, Jan. 2020. URL: https://hal.inria.fr/hal-02384946.

[57] 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.

[58] N. Jeannerod, C. Marché and R. Treinen. 'A Formally Verified Interpreter for a Shell-like Programming Language'. 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-01534747.

[59] N. Jeannerod, Y. Régis-Gianas, C. Marché, M. Sighireanu and R. Treinen. *Specification of UNIX Utilities*. Technical Report. HAL Archives Ouvertes, Oct. 2019. URL: https://hal.inria.fr/hal-02321691.

[60] N. Jeannerod and R. Treinen. ‘Deciding the First-Order Theory of an Algebra of Feature Trees with Updates’. In: *9th International Joint Conference on Automated Reasoning*. Oxford, United Kingdom, July 2018. URL: https://hal.archives-ouvertes.fr/hal-01807474.

[61] 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.

[62] 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/.

[63] É. Martin-Dorel and G. Melquiond. ‘Proving Tight Bounds on Univariate Expressions with Elementary Functions in Coq’. In: *Journal of Automated Reasoning* (2016). URL: https://hal.inria.fr/hal-01086460.

[64] 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.

[65] 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.

[66] 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.