IN PARTNERSHIP WITH:
Université Paris-Sud (Paris 11)

Activity Report 2017

Project-Team TOCCATA

Certified Programs, Certified Tools, Certified Floating-Point Computations

IN COLLABORATION WITH: Laboratoire de recherche en informatique (LRI)

RESEARCH CENTER
Saclay - Île-de-France

THEME
Proofs and Verification

Table of contents

Project-Team TOCCATA

Creation of the Team: 2012 September 01, updated into Project-Team: 2014 July 01

Keywords:

Computer Science and Digital Science:

A2.1.1. - Semantics of programming languages
A2.1.3. - 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
B9.4.1. - Computer science
B9.4.2. - Mathematics

1. Personnel

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, Researcher]

Faculty Members

Sylvain Conchon [Univ Paris-Sud, Professor, HDR]
Andrei Paskevich [Univ Paris-Sud, Associate Professor]
Thibault Hilaire [Associate Professor, Délégation de l'Univ Pierre et Marie Curie, from Sep 2017]

PhD Students

Ran Chen [Institute of Software, Chinese Academy of Sciences, Beijing, China, visiting until Feb. 2017]
Martin Clochard [Univ Paris-Sud until Aug, Inria since Sep]
Albin Coquereau [École Nationale Supérieure de Techniques Avancées]
David Declerck [Univ Paris-Sud]
Florian Faissole [Inria]

Diane Gallois-Wong [Univ Paris-Sud, from Oct 2017]
Mário Pereira [Grant Portuguese government]
Raphaël Rieu-Helft [CIFRE TrustInSoft, from Oct 2017]
Mattias Roux [Univ Paris-Sud]

Technical staff

Sylvain Dailler [Inria]
Clément Fumex [Inria, until Mar 2017]

Interns

Diane Gallois-Wong [Ecole Normale Supérieure Paris, from Apr 2017 until Jul 2017]
Raphaël Rieu-Helft [Ecole Normale Supérieure Paris, until Mar 2017]
Lucas Baudin [Ecole Normale Supérieure Paris, until Jan 2017]
Ilham Dami [Inria, from May 2017 until Jul 2017]
Vincent Tourneur [Inria, from Sep 2017]

Administrative Assistant

Katia Evrat [Inria]

2. Overall Objectives

2.1. Overall Objectives

The general objective of the Toccata project is to promote formal specification and computer-assisted proof in the development of software that requires high assurance in terms of safety and correctness with respect to the intended behavior of the software.

2.1.1. Context

The importance of software in critical systems increased a lot in the last decade. Critical software appears in various application domains like transportation (e.g., aviation, railway), communication (e.g., smartphones), banking, 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. These campaigns take a large part of the costs of software development, yet they cannot ensure that all the bugs are caught.

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 automated 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 others in some cases.

In the past decade, there has been significant progress made in the domain of deductive program verification. They are 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 [50] and other railroad-related systems; a formally proved C compiler was developed using the Coq proof assistant [110]; Microsoft's hypervisor for highly secure virtualization was verified using VCC [89] and the Z3 prover [129]; 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 [106]. Another sign of recent progress is the emergence of deductive verification competitions (e.g., VerifyThis [2], VScomp [100]).

Finally, recent trends in the industrial practice for development of critical software is to require more and more guarantees of safety, e.g., the upcoming 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.

2.1.2. 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 [49], JML [71]) and automatically producing a set of mathematical formulas (the *verification conditions*) which are typically proved using automatic provers (e.g., Z3, Alt-Ergo [52], CVC3 [48], CVC4).

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 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.

3. Research Program

3.1. Introduction

In the former ProVal project, we have been working on the design of methods and tools for deductive verification of programs. One of our original skills was 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. Toward this objective, a new axis of research was proposed: the development of *certified* analysis tools that are themselves formally proved correct.

The reader should be aware that the word “certified” in this scientific programme means “verified by a formal specification and a formal proof that the program meets this specification”. This differs from the standard meaning of “certified” in an industrial context where it means a conformance to some rigorous process and/or norm. We believe this is the right term to use, as it was used for the *Certified Compiler* project [110], the new conference series *Certified Programs and Proofs*, and more generally the important topics of *proof certificates*.

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 [46], various examples of applications of that approach [68], the use of the Gappa solver for proving numerical algorithms [128], an approach to take architectures and compilers into account when dealing with floating-point programs [69], [121]. We also contributed to the Handbook of Floating-Point Arithmetic [120]. 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 [3] [60]. Our experience led us to a conclusion that verification of numerical programs can benefit a lot from combining automatic and interactive theorem proving [62], [68]. Certification of numerical programs is the other main axis of Toccata.

Our scientific programme in structured into four objectives:

1. deductive program verification;
2. automated reasoning;
3. formalization and certification of languages, tools and systems;
4. proof of numerical programs.

We detail these objectives below.

3.2. Deductive Program Verification

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

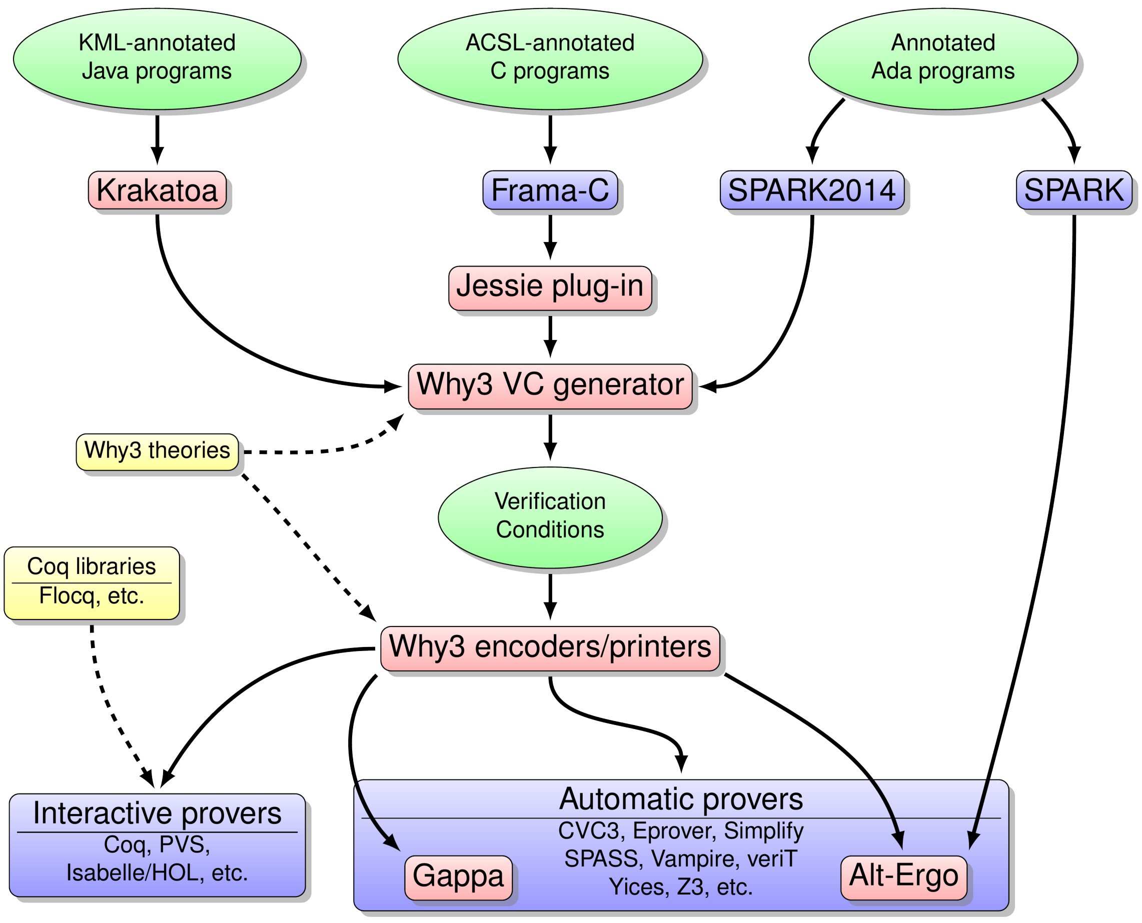

Figure 1. The Why3 ecosystem

3.2.1. The Why3 Ecosystem

This ecosystem is central in our work; it is displayed on Figure 1. The boxes in red background correspond to the tools we develop in the Toccata team.

- The initial design of Why3 was presented in 2012 [55], [99]. In the past years, the main improvements concern the specification language (such as support for higher-order logic functions [76]) and the support for provers. Several new interactive provers are now supported: PVS 6 (used at NASA), Isabelle2014 (planned to be used in the context of Ada program via Spark), and Mathematica. We also added support for new automated provers: CVC4, Metitarski, Metis, Beagle, Princess, and Yices2. More technical improvements are the design of a Coq tactic to call provers via Why3 from Coq, and the design of a proof session mechanism [54]. Why3 was presented during several invited talks [98], [97], [94], [95].
- At the level of the C front-end of Why3 (via Frama-C), we have proposed an approach to add a notion of refinement on C programs [127], and an approach to reason about pointer programs with a standard logic, via *separation predicates* [53]
- The Ada front-end of Why3 has mainly been developed during the past three years, leading to the release of SPARK2014 [105] (http://www.spark-2014.org/)
- In collaboration with J. Almeida, M. Barbosa, J. Pinto, and B. Vieira (University do Minho, Braga, Portugal), J.-C. Filliâtre has developed a method for certifying programs involving cryptographic methods. It uses Why as an intermediate language [45].
- With M. Pereira and S. Melo de Sousa (Universidade da Beira Interior, Covilhã, Portugal), J.-C. Filliâtre has developed an environment for proving ARM assembly code. It uses Why3 as an intermediate VC generator. It was presented at the Inforum conference [124] (best student paper).

3.2.2. Concurrent Programming

- S. Conchon and A. Mebsout, in collaboration with F. Zaïdi (VALS team, LRI), A. Goel and S. Krstić (Strategic Cad Labs, INTEL) have proposed a new model-checking approach for verifying safety properties of array-based systems. This is 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. It was first presented at CAV 2012 [5] and detailed further [87]. It was applied to the verification of programs with fences [83]. The core algorithm has been extended with a mechanism for inferring invariants. This new algorithm, called BRAB, is able to automatically infer invariants strong enough to prove industrial cache coherence protocols. BRAB computes over-approximations of backward reachable states that are checked to be unreachable in a finite instance of the system. These approximations (candidate invariants) are then model-checked together with the original safety properties. Completeness of the approach is ensured by a mechanism for backtracking on spurious traces introduced by too coarse approximations [84], [116].
- In the context of the ERC DeepSea project [1], A. Charguéraud and his co-authors have developed a unifying semantics for various different paradigms of parallel computing (fork-join, async-finish, and futures), and published a conference paper describing this work [44]. Besides, A. Charguéraud and his co-authors have polished their previous work on granularity control for parallel algorithms using user-provided complexity functions, and produced a journal article [43].

3.2.3. Case Studies

- To provide an easy access to the case studies that we develop using Why3 and its front-ends, we have published a *gallery of verified programs* on our web page http://toccata.lri.fr/gallery/. Part of these examples are the solutions to the competitions VerifyThis 2011 [70], VerifyThis 2012 [2], and the competition VScomp 2011 [100].

[1] Arthur Charguéraud is involved 40% of his time in the ERC DeepSea project, which is hosted at Inria Paris Rocquencourt (team Gallium).

- Other case studies that led to publications are the design of a library of data-structures based on AVLs [75], and the verification a two-lines C program (solving the N-queens puzzle) using Why3 [96].
- A. Charguéraud, with F. Pottier (Inria Paris), extended their formalization of the correctness and asympotic complexity of the classic Union Find data structure, which features the bound expressed in terms of the inverse Ackermann function [42]. The proof, conducted using CFML extended with time credits, was refined using a slightly more complex potential function, allowing to derive a simpler and richer interface for the data structure [73].

For other case studies, see also sections of numerical programs and formalization of languages and tools.

3.2.4. Project-team Positioning

Several research groups in the world develop their own approaches, techniques, and tools for deductive verification. With respect to all these related approaches and tools, our originality is our will to use more sophisticated specification languages (with inductive definitions, higher-order features and such) and the ability to use a large set of various theorem provers, including the use of interactive theorem proving to deal with complex functional properties.

- The RiSE team [2] at Microsoft Research Redmond, USA, partly in collaboration with team "programming methodology" team [3] at ETH Zurich develop tools that are closely related to ours: Boogie and Dafny are direct competitors of Why3, VCC is a direct competitor of Frama-C/Jessie.
- The KeY project [4] (several teams, mainly at Karlsruhe and Darmstadt, Germany, and Göteborg, Sweden) develops the KeY tool for Java program verification [41], based on dynamic logic, and has several industrial users. They use a specific modal logic (dynamic logic) for modeling programs, whereas we use standard logic, so as to be able to use off-the-shelf automated provers.
- The "software engineering" group at Augsburg, Germany, develops the KIV system [5], which was created more than 20 years ago (1992) and is still well maintained and efficient. It provides a semi-interactive proof environment based on algebraic-style specifications, and is able to deal with several kinds of imperative style programs. They have a significant industrial impact.
- The VeriFast system [6] aims at verifying C programs specified in Separation Logic. It is developed at the Katholic University at Leuven, Belgium. We do not usually use separation logic (so as to use off-the-shelf provers) but alternative approaches (e.g. static memory separation analysis).
- The Mobius Program Verification Environment [7] is a joint effort for the verification of Java source annotated with JML, combining static analysis and runtime checking. The tool ESC/Java2 [8] is a VC generator similar to Krakatoa, that builds on top of Boogie. It is developed by a community leaded by University of Copenhagen, Denmark. Again, our specificity with respect to them is the consideration of more complex specification languages and interactive theorem proving.
- The Lab for Automated Reasoning and Analysis [9] at EPFL, develop methods and tools for verification of Java (Jahob) and Scala (Leon) programs. They share with us the will and the ability to use several provers at the same time.
- The TLA environment [10], developed by Microsoft Research and the Inria team Veridis, aims at the verification of concurrent programs using mathematical specifications, model checking, and interactive or automated theorem proving.
- The F* project [11], developed by Microsoft Research and the Inria Prosecco team, aims at providing a rich environment for developing programs and proving them.

[2] http://research.microsoft.com/en-us/groups/rise/default.aspx
[3] http://www.pm.inf.ethz.ch/
[4] http://www.key-project.org/
[5] http://www.isse.uni-augsburg.de/en/software/kiv/
[6] http://people.cs.kuleuven.be/~bart.jacobs/verifast/
[7] http://kindsoftware.com/products/opensource/Mobius/
[8] http://kindsoftware.com/products/opensource/ESCJava2/
[9] http://lara.epfl.ch/w/
[10] http://research.microsoft.com/en-us/um/people/lamport/tla/tla.html
[11] http://research.microsoft.com/en-us/projects/fstar/

The KeY and KIV environments mentioned above are partly based on interactive theorem provers. There are other approaches on top of general-purpose proof assistants for proving programs that are not purely functional:

- The Ynot project [12] is a Coq library for writing imperative programs specified in separation logic. It was developed at Harvard University, until the end of the project in 2010. Ynot had similar goals as CFML, although Ynot requires programs to be written in monadic style inside Coq, whereas CFML applies directly on programs written in OCaml syntax, translating them into logical formulae.
- Front-ends to Isabelle were developed to deal with simple sequential imperative programs [126] or C programs [123]. The L4-verified project [106] is built on top of Isabelle.

3.3. Automated Reasoning

Permanent researchers: S. Conchon, G. Melquiond, A. Paskevich

3.3.1. Generalities on Automated Reasoning

- J. C. Blanchette and A. Paskevich have designed an extension to the TPTP TFF (Typed First-order Form) format of theorem proving problems to support rank-1 polymorphic types (also known as ML-style parametric polymorphism) [51]. This extension, named TFF1, has been incorporated in the TPTP standard.
- S. Conchon defended his *habilitation à diriger des recherches* in December 2012. The memoir [80] provides a useful survey of the scientific work of the past 10 years, around the SMT solving techniques, that led to the tools Alt-Ergo and Cubicle as they are nowadays.

3.3.2. Quantifiers and Triggers

- C. Dross, J. Kanig, S. Conchon, and A. Paskevich have proposed a generic framework for adding a decision procedure for a theory or a combination of theories to an SMT prover. This mechanism is based on the notion of instantiation patterns, or *triggers*, which restrict instantiation of universal premises and can effectively prevent a combinatorial explosion. A user provides an axiomatization with triggers, along with a proof of completeness and termination in the proposed framework, and obtains in return a sound, complete and terminating solver for his theory. A prototype implementation was realized on top of Alt-Ergo. As a case study, a feature-rich axiomatization of doubly-linked lists was proved complete and terminating [92]. C. Dross defended her PhD thesis in April 2014 [93]. The main results of the thesis are: (1) a formal semantics of the notion of *triggers* typically used to control quantifier instantiation in SMT solvers, (2) a general setting to show how a first-order axiomatization with triggers can be proved correct, complete, and terminating, and (3) an extended DPLL(T) algorithm to integrate a first-order axiomatization with triggers as a decision procedure for the theory it defines. Significant case studies were conducted on examples coming from SPARK programs, and on the benchmarks on B set theory constructed within the BWare project.

3.3.3. Reasoning Modulo Theories

- S. Conchon, É. Contejean and M. Iguernelala have presented a modular extension of ground AC-completion for deciding formulas in the combination of the theory of equality with user-defined AC symbols, uninterpreted symbols and an arbitrary signature-disjoint Shostak theory X [82]. This work extends the results presented in [81] by showing that a simple preprocessing step allows to get rid of a full AC-compatible reduction ordering, and to simply use a partial multiset extension of a *non-necessarily AC-compatible* ordering.
- S. Conchon, M. Iguernelala, and A. Mebsout have designed a collaborative framework for reasoning modulo simple properties of non-linear arithmetic [86]. This framework has been implemented in the Alt-Ergo SMT solver.

[12] http://ynot.cs.harvard.edu/

- S. Conchon, G. Melquiond and C. Roux have described a dedicated procedure for a theory of floating-point numbers which allows reasoning on approximation errors. This procedure is based on the approach of the Gappa tool: it performs saturation of consequences of the axioms, in order to refine bounds on expressions. In addition to the original approach, bounds are further refined by a constraint solver for linear arithmetic [88]. This procedure has been implemented in Alt-Ergo.
- In collaboration with A. Mahboubi (Inria project-team Typical), and G. Melquiond, the group involved in the development of Alt-Ergo have implemented and proved the correctness of a novel decision procedure for quantifier-free linear integer arithmetic [1]. This algorithm tries to bridge the gap between projection and branching/cutting methods: it interleaves an exhaustive search for a model with bounds inference. These bounds are computed provided an oracle capable of finding constant positive linear combinations of affine forms. An efficient oracle based on the Simplex procedure has been designed. This algorithm is proved sound, complete, and terminating and is implemented in Alt-Ergo.
- Most of the results above are detailed in M. Iguernelala's PhD thesis [103].

3.3.4. Applications

- We have been quite successful in the application of Alt-Ergo to industrial development: qualification by Airbus France, integration of Alt-Ergo into the Spark Pro toolset.
- In the context of the BWare project, aiming at using Why3 and Alt-Ergo for discharging proof obligations generated by Atelier B, we made progress into several directions. The method of translation of B proof obligations into Why3 goals was first presented at ABZ'2012 [119]. Then, new drivers have been designed for Why3, in order to use new back-end provers Zenon modulo and iProver modulo. A notion of rewrite rule was introduced into Why3, and a transformation for simplifying goals before sending them to back-end provers was designed. Intermediate results obtained so far in the project were presented both at the French conference AFADL [91] and at ABZ'2014 [90].

  On the side of Alt-Ergo, recent developments have been made to efficiently discharge proof obligations generated by Atelier B. This includes a new plugin architecture to facilitate experiments with different SAT engines, new heuristics to handle quantified formulas, and important modifications in its internal data structures to boost performances of core decision procedures. Benchmarks realized on more than 10,000 proof obligations generated from industrial B projects show significant improvements [85].
- Hybrid automatons interleave continuous behaviors (described by differential equations) with discrete transitions. D. Ishii and G. Melquiond have worked on an automated procedure for verifying safety properties (that is, global invariants) of such systems [104].

3.3.5. Project-team Positioning

Automated Theorem Proving is a large community, but several sub-groups can be identified:

- The SMT-LIB community gathers people interested in reasoning modulo theories. In this community, only a minority of participants are interested in supporting first-order quantifiers at the same time as theories. SMT solvers that support quantifiers are Z3 (Microsoft Research Redmond, USA), CVC3 and its successor CVC4 [13].
- The TPTP community gathers people interested in first-order theorem proving.
- Other Inria teams develop provers: veriT by team Veridis, and Psyche by team Parsifal.
- Other groups develop provers dedicated to very specific cases, such as Metitarski [14] at Cambridge, UK, which aims at proving formulas on real numbers, in particular involving special functions such as log or exp. The goal is somewhat similar to our CoqInterval library, *cf* objective 4.

[13] http://cvc4.cs.nyu.edu/web/
[14] http://www.cl.cam.ac.uk/~lp15/papers/Arith/

It should be noticed that a large number of provers mentioned above are connected to Why3 as back-ends.

3.4. Formalization and Certification of Languages, Tools and Systems

Permanent researchers: S. Boldo, A. Charguéraud, C. Marché, G. Melquiond, C. Paulin

3.4.1. Real Numbers, Real Analysis, Probabilities

- S. Boldo, C. Lelay, and G. Melquiond have worked on the Coquelicot library, designed to be a user-friendly Coq library about real analysis [65], [66]. An easier way of writing formulas and theorem statements is achieved by relying on total functions in place of dependent types for limits, derivatives, integrals, power series, and so on. To help with the proof process, the library comes with a comprehensive set of theorems and some automation. We have exercised the library on several use cases: on an exam at university entry level [108], for the definitions and properties of Bessel functions [107], and for the solution of the one-dimensional wave equation [109]. We have also conducted a survey on the formalization of real arithmetic and real analysis in various proof systems [67].
- Watermarking techniques are used to help identify copies of publicly released information. They consist in applying a slight and secret modification to the data before its release, in a way that should remain recognizable even in (reasonably) modified copies of the data. Using the Coq ALEA library, which formalizes probability theory and probabilistic programs, D. Baelde together with P. Courtieu, D. Gross-Amblard from Rennes and C. Paulin have established new results about the robustness of watermarking schemes against arbitrary attackers [47]. The technique for proving robustness is adapted from methods commonly used for cryptographic protocols and our work illustrates the strengths and particularities of the ALEA style of reasoning about probabilistic programs.

3.4.2. Formalization of Languages, Semantics

- P. Herms, together with C. Marché and B. Monate (CEA List), has developed a certified VC generator, using Coq. The program for VC calculus and its specifications are both written in Coq, but the code is crafted so that it can be extracted automatically into a stand-alone executable. It is also designed in a way that allows the use of arbitrary first-order theorem provers to discharge the generated obligations [102]. On top of this generic VC generator, P. Herms developed a certified VC generator for C source code annotated using ACSL. This work is the main result of his PhD thesis [101].
- A. Tafat and C. Marché have developed a certified VC generator using Why3 [112], [113]. The challenge was to formalize the operational semantics of an imperative language, and a corresponding weakest precondition calculus, without the possibility to use Coq advanced features such as dependent types or higher-order functions. The classical issues with local bindings, names and substitutions were solved by identifying appropriate lemmas. It was shown that Why3 can offer a significantly higher amount of proof automation compared to Coq.
- A. Charguéraud, together with Alan Schmitt (Inria Rennes) and Thomas Wood (Imperial College), has developed an interactive debugger for JavaScript. The interface, accessible as a webpage in a browser, allows to execute a given JavaScript program, following step by step the formal specification of JavaScript developped in prior work on *JsCert* [56]. Concretely, the tool acts as a double-debugger: one can visualize both the state of the interpreted program and the state of the interpreter program. This tool is intended for the JavaScript committee, VM developpers, and other experts in JavaScript semantics.
- M. Clochard, C. Marché, and A. Paskevich have developed a general setting for developing programs involving binders, using Why3. This approach was successfully validated on two case studies: a verified implementation of untyped lambda-calculus and a verified tableaux-based theorem prover [79].

- M. Clochard, J.-C. Filliâtre, C. Marché, and A. Paskevich have developed a case study on the formalization of semantics of programming languages using Why3 [76]. This case study aims at illustrating recent improvements of Why3 regarding the support for higher-order logic features in the input logic of Why3, and how these are encoded into first-order logic, so that goals can be discharged by automated provers. This case study also illustrates how reasoning by induction can be done without need for interactive proofs, via the use of *lemma functions*.
- M. Clochard and L. Gondelman have developed a formalization of a simple compiler in Why3 [77]. It compiles a simple imperative language into assembler instructions for a stack machine. This case study was inspired by a similar example developed using Coq and interactive theorem proving. The aim is to improve significantly the degree of automation in the proofs. This is achieved by the formalization of a Hoare logic and a Weakest Precondition Calculus on assembly programs, so that the correctness of compilation is seen as a formal specification of the assembly instructions generated.

3.4.3. Project-team Positioning

The objective of formalizing languages and algorithms is very general, and it is pursued by several Inria teams. One common trait is the use of the Coq proof assistant for this purpose: Pi.r2 (development of Coq itself and its meta-theory), Gallium (semantics and compilers of programming languages), Marelle (formalization of mathematics), SpecFun (real arithmetic), Celtique (formalization of static analyzers).

Other environments for the formalization of languages include

- ACL2 system [15]: an environment for writing programs with formal specifications in first-order logic based on a Lisp engine. The proofs are conducted using a prover based on the Boyer-Moore approach. It is a rather old system but still actively maintained and powerful, developed at University of Texas at Austin. It has a strong industrial impact.
- Isabelle environment [16]: both a proof assistant and an environment for developing pure applicative programs. It is developed jointly at University of Cambridge, UK, Technische Universität München, Germany, and to some extent by the VALS team at LRI, Université Paris-Sud. It features highly automated tactics based on ATP systems (the Sledgehammer tool).
- The team "Trustworthy Systems" at NICTA in Australia [17] aims at developing highly trustable software applications. They developed a formally verified micro-kernel called seL4 [106], using a home-made layer to deal with C programs on top of the Isabelle prover.
- The PVS system [18] is an environment for both programming and proving (purely applicative) programs. It is developed at the Computer Science Laboratory of SRI international, California, USA. A major user of PVS is the team LFM [19] at NASA Langley, USA, for the certification of programs related to air traffic control.

In the Toccata team, we do not see these alternative environments as competitors, even though, for historical reasons, we are mainly using Coq. Indeed both Isabelle and PVS are available as back-ends of Why3.

3.5. Proof of Numerical Programs

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

- Linked with objective 1 (Deductive Program Verification), the methodology for proving numerical C programs has been presented by S. Boldo in her habilitation [58] and as invited speaker [59]. An application is the formal verification of a numerical analysis program. S. Boldo, J.-C. Filliâtre, and G. Melquiond, with F. Clément and P. Weis (POMDAPI team, Inria Paris - Rocquencourt), and M. Mayero (LIPN), completed the formal proof of the second-order centered finite-difference scheme for the one-dimensional acoustic wave [61][3].

[15]http://www.cs.utexas.edu/~moore/acl2/
[16]http://isabelle.in.tum.de/
[17]http://ssrg.nicta.com.au/projects/TS/
[18]http://pvs.csl.sri.com/
[19]http://shemesh.larc.nasa.gov/fm/fm-main-team.html

- Several challenging floating-point algorithms have been studied and proved. This includes an algorithm by Kahan for computing the area of a triangle: S. Boldo proved an improvement of its error bound and new investigations in case of underflow [57]. This includes investigations about quaternions. They should be of norm 1, but due to the round-off errors, a drift of this norm is observed over time. C. Marché determined a bound on this drift and formally proved it correct [8]. P. Roux formally verified an algorithm for checking that a matrix is semi-definite positive [125]. The challenge here is that testing semi-definiteness involves algebraic number computations, yet it needs to be implemented using only approximate floating-point operations.
- Because of compiler optimizations (or bugs), the floating-point semantics of a program might change once compiled, thus invalidating any property proved on the source code. We have investigated two ways to circumvent this issue, depending on whether the compiler is a black box. When it is, T. Nguyen has proposed to analyze the assembly code it generates and to verify it is correct [122]. On the contrary, S. Boldo and G. Melquiond (in collaboration with J.-H. Jourdan and X. Leroy) have added support for floating-point arithmetic to the CompCert compiler and formally proved that none of the transformations the compiler applies modify the floating-point semantics of the program [64], [63].
- Linked with objectives 2 (Automated Reasoning) and 3 (Formalization and Certification of Languages, Tools and Systems), G. Melquiond has implemented an efficient Coq library for floating-point arithmetic and proved its correctness in terms of operations on real numbers [117]. It serves as a basis for an interval arithmetic on which Taylor models have been formalized. É. Martin-Dorel and G. Melquiond have integrated these models into CoqInterval [9]. This Coq library is dedicated to automatically proving the approximation properties that occur when formally verifying the implementation of mathematical libraries (libm).
- Double rounding occurs when the target precision of a floating-point computation is narrower than the working precision. In some situations, this phenomenon incurs a loss of accuracy. P. Roux has formally studied when it is innocuous for basic arithmetic operations [125]. É. Martin-Dorel and G. Melquiond (in collaboration with J.-M. Muller) have formally studied how it impacts algorithms used for error-free transformations [115]. These works were based on the Flocq formalization of floating-point arithmetic for Coq.
- By combining multi-precision arithmetic, interval arithmetic, and massively-parallel computations, G. Melquiond (in collaboration with G. Nowak and P. Zimmermann) has computed enough digits of the Masser-Gramain constant to invalidate a 30-year old conjecture about its closed form [118].

3.5.1. Project-team Positioning

This objective deals both with formal verification and floating-point arithmetic, which is quite uncommon. Therefore our competitors/peers are few. We may only cite the works by J. Duracz and M. Konečný, Aston University in Birmingham, UK.

The Inria team AriC (Grenoble - Rhône-Alpes) is closer to our research interests, but they are lacking manpower on the formal proof side; we have numerous collaborations with them. The Inria team Caramel (Nancy - Grand Est) also shares some research interests with us, though fewer; again, they do not work on the formal aspect of the verification; we have some occasional collaborations with them.

There are many formalization efforts from chip manufacturers, such as AMD (using the ACL2 proof assistant) and Intel (using the Forte proof assistants) but the algorithms they consider are quite different from the ones we study. The works on the topic of floating-point arithmetic from J. Harrison at Intel using HOL Light are really close to our research interests, but they seem to be discontinued.

A few deductive program verification teams are willing to extend their tools toward floating-point programs. This includes the KeY project and SPARK. We have an ongoing collaboration with the latter, in the context of the ProofInUSe project.

Deductive verification is not the only way to prove programs. Abstract interpretation is widely used, and several teams are interested in floating-point arithmetic. This includes the Inria team Antique (Paris - Rocquencourt) and a CEA List team, who have respectively developed the Astrée and Fluctuat tools. This approach targets a different class of numerical algorithms than the ones we are interested in.

Other people, especially from the SMT community (*cf* objective 2), are also interested in automatically proving formulas about floating-point numbers, notably at Oxford University. They are mainly focusing on pure floating-point arithmetic though and do not consider them as approximation of real numbers.

Finally, it can be noted that numerous teams are working on the verification of numerical programs, but assuming the computations are real rather than floating-point ones. This is out of the scope of this objective.

4. Application Domains

4.1. Domain

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 mainly belong to the domain of transportation, including aeronautics, railroad, space flight, automotive.

Verification of C programs, Alt-Ergo at Airbus Transportation is the domain considered in the context of the ANR U3CAT project, led by CEA, in partnership with Airbus France, Dassault Aviation, Sagem Défense et Sécurité. It included proof of C programs via Frama-C/Jessie/Why, proof of floating-point programs [114], the use of the Alt-Ergo prover via CAVEAT tool (CEA) or Frama-C/WP. Within this context, we contributed to a qualification process of Alt-Ergo with Airbus industry: the technical documents (functional specifications and benchmark suite) have been accepted by Airbus, and these documents were submitted by Airbus to the certification authorities (DO-178B standard) in 2012. This action is continued in the new project Soprano.

Certified compilation, certified static analyzers Aeronautics is the main target of the Verasco project, led by Verimag, on the development of certified static analyzers, in partnership with Airbus. This is a follow-up of the transfer of the CompCert certified compiler (Inria team Gallium) to which we contributed to the support of floating-point computations [64].

Transfer to the community of Ada development The former FUI project Hi-Lite, led by Adacore company, introduced the use of Why3 and Alt-Ergo as back-end to SPARK2014, an environment for verification of Ada programs. This is applied to the domain of aerospace (Thales, EADS Astrium). At the very beginning of that project, Alt-Ergo was added in the Spark Pro toolset (predecessor of SPARK2014), developed by Altran-Praxis: Alt-Ergo can be used by customers as an alternate prover for automatically proving verification conditions. Its usage is described in the new edition of the Spark book [20] (Chapter "Advanced proof tools"). This action is continued in the new joint laboratory ProofInUse. A recent paper [72] provides an extensive list of applications of SPARK, a major one being the British air control management *iFacts*.

Transfer to the community of Atelier B In the current ANR project BWare, we investigate 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 software,

a collaboration with Mitsubishi Electric R&D Centre Europe (Rennes) (joint publication [119]) and ClearSy (Aix-en-Provence).

SMT-based Model-Checking: Cubicle S. Conchon (with A. Mebsout and F. Zaidi from VALS team at LRI) has a long-term collaboration with S. Krstic and A. Goel (Intel Strategic Cad Labs in Hillsboro, OR, USA) that aims in the development of the SMT-based model checker Cubicle (http://cubicle.lri.fr/) based on Alt-Ergo [116][5]. It is particularly targeted to the verification of concurrent programs and protocols.

[20] http://www.altran-praxis.com/book/

5. Highlights of the Year

5.1. Highlights of the Year

S. Conchon has co-organized POPL'2017 (January, Paris, http://conf.researchr.org/home/POPL-2017).

C. Marché has co-organized the first joint Frama-C/SPARK day (May, Paris, http://frama-c.com/FCSD17.html), in the context of the Open Source Innovation Spring (http://www.open-source-innovation-spring.org/).

S. Boldo and G. Melquiond have published a book: Computer Arithmetic and Formal Proofs, Verifying Floating-point Algorithms with the Coq System [32].

5.1.1. Awards

M. Pereira and R. Rieu-Helft received the "Best student team" award, and J.-C. Filliâtre the "Best overall team" award, at the *VerifyThis@ETAPS2017 verification competition*.

6. New Software and Platforms

6.1. Alt-Ergo

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 FUNCTIONAL DESCRIPTION: 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.

- Participants: Alain Mebsout, Évelyne Contejean, Mohamed Iguernelala, Stéphane Lescuyer and Sylvain Conchon
- Partner: OCamlPro
- Contact: Sylvain Conchon
- URL: http://alt-ergo.lri.fr

6.2. CFML

Interactive program verification using characteristic formulae
KEYWORDS: Coq - Software Verification - Deductive program verification - Separation Logic

FUNCTIONAL DESCRIPTION: The CFML tool supports the verification of OCaml programs through interactive Coq proofs. CFML proofs establish the full functional correctness of the code with respect to a specification. They may also be used to formally establish bounds on the asymptotic complexity of the code. The tool is made of two parts: on the one hand, a characteristic formula generator implemented as an OCaml program that parses OCaml code and produces Coq formulae, and, on the other hand, a Coq library that provides notations and tactics for manipulating characteristic formulae interactively in Coq.

- Participants: Arthur Charguéraud, Armaël Guéneau and François Pottier
- Contact: Arthur Charguéraud
- URL: http://www.chargueraud.org/softs/cfml/

6.3. Coq

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 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 FUNCTIONAL DESCRIPTION: Version 8.7 features a large amount of work on cleaning and speeding up the code base, notably the work of Pierre-Marie Pédrot on making the tactic-level system insensitive to existential variable expansion, providing a safer API to plugin writers and making the code more robust.

New tactics: Variants of tactics supporting existential variables "eassert", "eenough", etc. by Hugo Herbelin. Tactics "extensionality in H" and "inversion_sigma" by Jason Gross, "specialize with" accepting partial bindings by Pierre Courtieu.

Cumulative Polymorphic Inductive Types, allowing cumulativity of universes to go through applied inductive types, by Amin Timany and Matthieu Sozeau.

The SSReflect plugin by Georges Gonthier, Assia Mahboubi and Enrico Tassi was integrated (with its documentation in the reference manual) by Maxime Dénès, Assia Mahboubi and Enrico Tassi.

The "coq_makefile" tool was completely redesigned to improve its maintainability and the extensibility of generated Makefiles, and to make "_CoqProject" files more palatable to IDEs by Enrico Tassi.

A lot of other changes are described in the CHANGES file.
NEWS OF THE YEAR: Version 8.7 was released in October 2017 and version 8.7.1 in December 2017, development started in January 2017. This is the second release of Coq developed on a time-based development cycle. Its development spanned 9 months from the release of Coq 8.6 and was based on a public road-map. It attracted many external contributions. Code reviews and continuous integration testing were systematically used before integration of new features, with an important focus given to compatibility and performance issues.

The main scientific advance in this version is the integration of cumulative inductive types in the system. More practical advances in stability, performance, usability and expressivity of tactics were also implemented, resulting in a mostly backwards-compatible but appreciably faster and more robust release. Much work on plugin extensions to Coq by the same development team has also been going on in parallel, including work on JSCoq by Emilio JG Arias, Ltac 2 by P.M-Pédrot, which required synchronised changes of the main codebase. In 2017, the construction of the Coq Consortium by Yves Bertot and Maxime Dénès has greatly advanced and is now nearing its completion.

- Participants: Abhishek Anand, C. J. Bell, Yves Bertot, Frédéric Besson, Tej Chajed, Pierre Courtieu, Maxime Denes, Julien Forest, Emilio Jesús Gallego Arias, Gaëtan Gilbert, Benjamin Grégoire, Jason Gross, Hugo Herbelin, Ralf Jung, Matej Kosik, Sam Pablo Kuper, Xavier Leroy, Pierre Letouzey, Assia Mahboubi, Cyprien Mangin, Érik Martin-Dorel, Olivier Marty, Guillaume Melquiond, Pierre-Marie Pédrot, Benjamin C. Pierce, Lars Rasmusson, Yann Régis-Gianas, Lionel Rieg, Valentin Robert, Thomas Sibut-Pinote, Michael Soegtrop, Matthieu Sozeau, Arnaud Spiwack, Paul Steckler, George Stelle, Pierre-Yves Strub, Enrico Tassi, Hendrik Tews, Laurent Théry, Amin Timany, Vadim Zaliva and Théo Zimmermann
- Partners: CNRS - Université Paris-Sud - ENS Lyon - Université Paris-Diderot
- Contact: Matthieu Sozeau
- Publication: The Coq Proof Assistant, version 8.7.1
- URL: http://coq.inria.fr/

6.4. CoqInterval

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.

- Participants: Assia Mahboubi, Érik Martin-Dorel, Guillaume Melquiond, Jean-Michel Muller, Laurence Rideau, Laurent Théry, Micaela Mayero, Mioara Joldes, Nicolas Brisebarre and Thomas Sibut-Pinote
- Contact: Guillaume Melquiond
- Publications: Proving bounds on real-valued functions with computations - Floating-point arithmetic in the Coq system - Proving Tight Bounds on Univariate Expressions with Elementary Functions in Coq - Formally Verified Approximations of Definite Integrals - Formally Verified Approximations of Definite Integrals
- URL: http://coq-interval.gforge.inria.fr/

6.5. Coquelicot

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.

- Participants: Catherine Lelay, Guillaume Melquiond and Sylvie Boldo
- Contact: Sylvie Boldo
- URL: http://coquelicot.saclay.inria.fr/

6.6. Cubicle

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.

- Participants: Alain Mebsout and Sylvain Conchon
- Contact: Sylvain Conchon
- URL: http://cubicle.lri.fr/

6.7. Flocq

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.

- Participants: Guillaume Melquiond, Pierre Roux and Sylvie Boldo
- Contact: Sylvie Boldo
- Publications: Flocq: A Unified Library for Proving Floating-point Algorithms in Coq - A Formally-Verified C Compiler Supporting Floating-Point Arithmetic - Verified Compilation of Floating-Point Computations - Innocuous Double Rounding of Basic Arithmetic Operations - Formal Proofs of Rounding Error Bounds - Computer Arithmetic and Formal Proofs
- URL: http://flocq.gforge.inria.fr/

6.8. Gappa

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.

- Participant: Guillaume Melquiond
- Contact: Guillaume Melquiond
- Publications: Generating formally certified bounds on values and round-off errors - Formal certification of arithmetic filters for geometric predicates - Assisted verification of elementary functions - From interval arithmetic to program verification - Formally Certified Floating-Point Filters For Homogeneous Geometric Predicates - Combining Coq and Gappa for Certifying Floating-Point Programs - Handbook of Floating-Point Arithmetic - Certifying the floating-point implementation of an elementary function using Gappa - Automations for verifying floating-point algorithms - Automating the verification of floating-point algorithms - Computer Arithmetic and Formal Proofs
- URL: http://gappa.gforge.inria.fr/

6.9. Why3

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.

- Participants: Andriy Paskevych, Claude Marché, François Bobot, Guillaume Melquiond, Jean-Christophe Filliâtre, Levs Gondelmans and Martin Clochard
- Partners: CNRS - Université Paris-Sud
- Contact: Claude Marché
- URL: http://why3.lri.fr/

7. New Results

7.1. Deductive Verification

Synthetic topology in HoTT for probabilistic programming. F. Faissole and B. Spitters have developed a mathematical formalism based on synthetic topology and homotopy type theory to interpret probabilistic algorithms. They suggest to use proof assistants to prove such programs [39] [31]. They also have formalized synthetic topology in the Coq proof assistant using the HoTT library. It consists of a theory of lower reals, valuations and lower integrals. All the results are constructive. They apply their results to interpret probabilistic programs using a monadic approach [28].

Defunctionalization for proving higher-order programs. J.-C. Filliâtre and M. Pereira proposed a new approach to the verification of higher-order programs, using the technique of defunctionalization, that is, the translation of first-class functions into first-order values. This is an early experimental work, conducted on examples only within the Why3 system. This work was published at JFLA 2017 [29].

Extracting Why3 programs to C programs. R. Rieu-Helft, C. Marché, and G. Melquiond devised a simple memory model for representing C-like pointers in the Why3 system. This makes it possible to translate a small fragment of Why3 verified programs into idiomatic C code [30]. This extraction mechanism was used to turn a verified Why3 library of arbitrary-precision integer arithmetic into a C library that can be substituted to part of the GNU Multi-Precision (GMP) library [23].

Verification of highly imperative OCaml programs with Why3 J.-C. Filliâtre, M. Pereira and S. Melo de Sousa proposed a new methodology for proving highly imperative OCaml programs with Why3. For a given OCaml program, a specific memory model is built and one checks a Why3 program that operates on it. Once the proof is complete, they use Why3's extraction mechanism to translate its programs to OCaml, while replacing the operations on the memory model with the corresponding operations on mutable types of OCaml. This method is evaluated on several examples that manipulate linked lists and mutable graphs [20].

7.2. Automated Reasoning

A Three-tier Strategy for Reasoning about Floating-Point Numbers in SMT. The SMT-LIB standard defines a formal semantics for a theory of floating-point (FP) arithmetic (FPA). This formalization reduces FP operations to reals by means of a rounding operator, as done in the IEEE-754 standard. Closely following this description, S. Conchon, M. Iguernlala, K. Ji, G. Melquiond and C. Fumex propose a three-tier strategy to reason about FPA in SMT solvers. The first layer is a purely axiomatic implementation of the automatable semantics of the SMT-LIB standard. It reasons with exceptional cases (e.g. overflows, division by zero, undefined operations) and reduces finite representable FP expressions to reals using the rounding operator. At the core of the strategy, a second layer handles a set of lemmas about the properties of rounding. For these lemmas to be used effectively, the instantiation mechanism of SMT solvers is extended to tightly cooperate with the third layer, the NRA engine of SMT solvers, which provides interval information. The strategy is implemented in the Alt-Ergo SMT solver and validated on a set of benchmarks coming from the SMT-LIB competition, and also from the deductive verification of C and Ada programs. The results show that the approach is promising and compete with existing techniques implemented in state-of-the-art SMT solvers. This work was presented at the CAV conference [18].

Lightweight Approach for Declarative Proofs. M. Clochard designed an extension of first-order logic, for describing reasoning steps needed to discharge a proof obligation. The extension is under the form of two new connectives, called proof indications, that allow the user to encode reasoning steps inside a logic formula. This extension makes possible to use the syntax of formulas as a proof language. The approach was presented at the JFLA conference [26] and implemented in Why3. It brings a lightweight mechanism for declarative proofs in an environment like Why3 where provers are used as black boxes. Moreover, this mechanism restricts the scope of auxiliary lemmas, reducing the size of proof obligations sent to external provers.

7.3. Certification of Algorithms, Languages, Tools and Systems

Formalization and closedness of finite dimensional subspaces. F. Faissole formalized a theory of finite dimensional subspaces of Hilbert spaces in order to apply the Lax-Milgram Theorem on such subspaces. He had to prove, in the Coq proof assistant, that finite dimensional subspaces of Hilbert spaces are closed in the context of general topology using filters [19]. He also formalized both finite dimensional modules and finite dimensional subspaces of modules. He compared the two formalizations and showed a complementarity between them. He proved that the product of two finite dimensional modules is a finite dimensional module [27].

Verified numerical approximations of improper definite integrals. The CoqInterval library provides some tactics for computing and formally verifying numerical approximations of real-valued expressions inside the Coq system. In particular, it is able to compute reliable bounds on proper definite integrals [111]. A. Mahboubi, G. Melquiond, and T. Sibut-Pinote extended these algorithms to also cover some improper integrals, e.g., those with an unbounded integration domain [40]. This makes CoqInterval one of the very few tools able to produce reliable results for improper integrals, be they formally verified or not.

A Coq Formal Proof of the Lax–Milgram theorem. S. Boldo, F. Clément, F. Faissole, V. Martin, and M. Mayero worked on a Coq formal proof of the Lax–Milgram theorem. It is one of the theoretical cornerstone for the correctness of the Finite Element Method. It required many results from linear algebra, geometry, functional analysis, and Hilbert spaces [13] [24].

Formalization of numerical filters S. Boldo, D. Gallois-Wong, and T. Hilaire developped a formalization in the Coq proof assistant of numerical filters. It includes equivalences between several expressions and the formal proof of the Worst-Case Peak Gain Theorem to bound the magnitude of the outputs (and every intern variable) of stable filters.

A Verified OCaml Library. Abstract Libraries are the basic building blocks of any realistic programming project. It is thus of utmost interest for a programmer to build her software on top of bug-free libraries. At the ML family workshop [38], A. Charguéraud, J.-C. Filliâtre, M. Pereira and F. Pottier presented the ongoing VOCAL project, which aims at building a mechanically verified library of general-purpose data structures and algorithms, written in the OCaml language. A key ingredient of VOCAL is the design of a specification language for OCaml, independently of any verification tool.

Formal Analysis of shell scripts. The shell language is widely used for various system administration tasks on UNIX machines. The CoLiS project aims at applying formal methods for verifying scripts used for installation of packages of software distributions. The syntax and semantics of shell are particularly treacherous. They proposed a new language called CoLiS which, on the one hand, has well-defined static semantics and avoids some of the pitfalls of the shell, and, on the other hand, is close enough to the shell to be the target of an automated translation of the scripts in our corpus. In collaboration with N. Jeannerod and R. Treinen, C. Marché formalized the syntax and semantics of CoLiS in Why3, defined an interpreter for the language in the WhyML programming language, and present an automated proof in the Why3 proof environment of soundness and completeness of this interpreter with respect to the formal semantics [22]. The development is available in Toccata's gallery http://toccata.lri.fr/gallery/colis_interpreter.en.html. This formalized interpreter is extracted to OCaml and the verified code is integrated into a prototype software toolset developed by I. Dami and C. Marché [36].

A verified yet efficient arbitrary-precision integer library. R. Rieu-Helft used the Why3 system to implement, specify, and verify a library of arbitrary-precision integer arithmetic: comparison, addition, multiplication, shifts, division. A lot of efforts were put into replicating and verifying the numerous implementation tricks the GMP library uses to achieve state-of-the-art performances, especially for the division algorithm. While the resulting library is nowhere near as fast as the hand-written assembly code GMP uses, it is competitive with the generic C code of GMP for small integers (i.e., mini-GMP) [23]. The development is available in Toccata's gallery http://toccata.lri.fr/gallery/multiprecision.en.html.

Case study: algorithms for matrix multiplication. M. Clochard, L. Gondelman and M. Pereira worked on a case study about matrix multiplication. Two variants for the multiplication of matrices are proved: a naive version using three nested loops and Strassen's algorithm. To formally specify the two multiplication algorithms, they developed a new Why3 theory of matrices, and they applied a reflection methodology to conduct some of the proofs. A first version of this work was presented at the VSTTE Conference in 2016 [78]. An extended version that considers arbitrary rectangular matrices instead of square ones is published in the Journal of Automated Reasoning [12]. The development is available in Toccata's gallery http://toccata.lri.fr/gallery/verifythis_2016_matrix_multiplication.en.html.

Case studies: Strongly Connected Components in Directed Graphs As part of a larger set of case studies on algorithms on graphs http://pauillac.inria.fr/~levy/why3/, R. Chen and J.-J. Lévy work on formal verification of algorithms for computing strongly connected components of directed graphs.

The formal proofs are conducted using Why3. The formal proof of Tarjan's algorithm was presented at the French-speaking symposium JFLA 2017 [25] and then at the VSTTE 2017 international conference [17]

A Formally Proved, Complete Algorithm for Path Resolution with Symbolic Links In the context of file systems like those of Unix, path resolution is the operation that given a character string denoting an access path, determines the target object (a file, a directory, etc.) designated by this path. This operation is not trivial because of the presence of symbolic links. Indeed, the presence of such links may induce infinite loops in the resolution process. R. Chen, M. Clochard and C. Marché consider a path resolution algorithm that always terminates, detecting if it enters an infinite loop and reports a resolution failure in such a case. They propose a formal specification of path resolution and they formally prove that their algorithm terminates on any input, and is correct and complete with respect to this formal specification. [11]. The development is available in Toccata's gallery http://toccata.lri.fr/gallery/path_resolution.en.html.

7.4. Floating-Point and Numerical Programs

Computer Arithmetic and Formal Proofs: Verifying Floating-point Algorithms with the Coq System S. Boldo and G. Melquiond published a book that provides a comprehensive view of how to formally specify and verify tricky floating-point algorithms with the Coq proof assistant. It describes the Flocq formalization of floating-point arithmetic and some methods to automate theorem proofs. It then presents the specification and verification of various algorithms, from error-free transformations to a numerical scheme for a partial differential equation. The examples cover not only mathematical algorithms but also C programs as well as issues related to compilation [32].

Automating the Verification of Floating-Point Programs. The level of proof success and proof automation highly depends on the way the floating-point operations are interpreted in the logic supported by back-end provers. C. Fumex, C. Marché and Y. Moy addressed this challenge by combining multiple techniques to separately prove different parts of the desired properties. They use abstract interpretation to compute numerical bounds of expressions, and use multiple automated provers, relying on different strategies for representing floating-point computations. One of these strategies is based on the native support for floating-point arithmetic recently added in the SMT-LIB standard. The approach is implemented in the Why3 environment and its front-end SPARK 2014. It is validated experimentally on several examples originating from industrial use of SPARK 2014 [37], [21].

Round-off Error Analysis of Explicit One-Step Numerical Integration Methods. S. Boldo, A. Chapoutot, and F. Faissole provided bounds on the round-off errors of explicit one-step numerical integration methods, such as Runge-Kutta methods. They developed a fine-grained analysis that takes advantage of the linear stability of the scheme, a mathematical property that vouches the scheme is well-behaved [14].

Robustness of 2Sum and Fast2Sum. S. Boldo, S.Graillat, and J.-M. Muller worked on the 2Sum and Fast2Sum algorithms, that are important building blocks in numerical computing. They are used (implicitly or explicitly) in many compensated algorithms or for manipulating floating-point expansions. They showed that these algorithms are much more robust than it is usually believed: the returned result makes sense even when the rounding function is not round-to-nearest, and they are almost immune to overflow [10].

Formal Verification of a Floating-Point Expansion Renormalization Algorithm. Many numerical problems require a higher computing precision than the one offered by standard floating-point formats. A common way of extending the precision is to use floating-point expansions. S. Boldo, M. Joldes, J.-M. Muller, and V. Popescu proved one of the algorithms used as a basic brick when computing with floating-point expansions: renormalization that “compresses” an expansion [15].

8. Bilateral Contracts and Grants with Industry

8.1. Bilateral Contracts with Industry

8.1.1. ProofInUse Joint Laboratory

Participants: Claude Marché [contact], Jean-Christophe Filliâtre, Andrei Paskevich.

ProofInUse is a joint project between the Toccata team and the SME AdaCore. It was selected and funded by the ANR programme "Laboratoires communs", starting from April 2014, for 3 years http://www.spark-2014.org/proofinuse.

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 goal is now to promote and transfer the use of deduction-based verification tools to industry users, who develop critical software using the programming language Ada. The proof tools are aimed at replacing or complementing the existing test activities, whilst reducing costs.

9. Partnerships and Cooperations

9.1. Regional Initiatives

9.1.1. ELEFFAN

Participant: Sylvie Boldo [contact].

ELEFFAN is a Digicosme project funding the PhD of F. Faissole. S. Boldo is the principal investigator. It began in 2016 for three years. https://project.inria.fr/eleffan/

The ELEFFAN project aims at formally proving rounding error bounds of numerical schemes.

Partners: ENSTA Paristech (A. Chapoutot)

9.2. National Initiatives

9.2.1. ANR CoLiS

Participants: Claude Marché [contact], Andrei Paskevich.

The CoLiS research project is funded by the programme "Société de l'information et de la communication" of the ANR, for a period of 60 months, starting on October 1st, 2015. http://colis.irif.univ-paris-diderot.fr/

The project aims at developing formal analysis and verification techniques and tools for scripts. These scripts are written in the POSIX or bash shell language. Our objective is to produce, at the end of the project, formal methods and tools allowing to analyze, test, and validate scripts. For this, the project will develop techniques and tools based on deductive verification and tree transducers stemming from the domain of XML documents.

Partners: Université Paris-Diderot, IRIF laboratory (formerly PPS & LIAFA), coordinator; Inria Lille, team LINKS

9.2.2. ANR Vocal

Participants: Jean-Christophe Filliâtre [contact], Andrei Paskevich.

The Vocal research project is funded by the programme "Société de l'information et de la communication" of the ANR, for a period of 60 months, starting on October 1st, 2015. https://vocal.lri.fr/

The goal of the Vocal project is to develop the first formally verified library of efficient general-purpose data structures and algorithms. It targets the OCaml programming language, which allows for fairly efficient code and offers a simple programming model that eases reasoning about programs. The library will be readily available to implementers of safety-critical OCaml programs, such as Coq, Astrée, or Frama-C. It will provide the essential building blocks needed to significantly decrease the cost of developing safe software. The project intends to combine the strengths of three verification tools, namely Coq, Why3, and CFML. It will use Coq to obtain a common mathematical foundation for program specifications, as well as to verify purely functional components. It will use Why3 to verify a broad range of imperative programs with a high degree of proof automation. Finally, it will use CFML for formal reasoning about effectful higher-order functions and data structures making use of pointers and sharing.

Partners: team Gallium (Inria Paris-Rocquencourt), team DCS (Verimag), TrustInSoft, and OCamlPro.

9.2.3. ANR FastRelax

Participants: Sylvie Boldo [contact], Guillaume Melquiond.

This is a research project funded by the programme “Ingénierie Numérique & Sécurité” of the ANR. It is funded for a period of 48 months and it has started on October 1st, 2014. http://fastrelax.gforge.inria.fr/

Our aim is to develop computer-aided proofs of numerical values, with certified and reasonably tight error bounds, without sacrificing efficiency. Applications to zero-finding, numerical quadrature or global optimization can all benefit from using our results as building blocks. We expect our work to initiate a "fast and reliable" trend in the symbolic-numeric community. This will be achieved by developing interactions between our fields, designing and implementing prototype libraries and applying our results to concrete problems originating in optimal control theory.

Partners: team ARIC (Inria Grenoble Rhône-Alpes), team MARELLE (Inria Sophia Antipolis - Méditerranée), team SPECFUN (Inria Saclay - Île-de-France), Université Paris 6, and LAAS (Toulouse).

9.2.4. ANR Soprano

Participants: Sylvain Conchon [contact], Guillaume Melquiond.

The Soprano research project is funded by the programme “Sciences et technologies logicielles” of the ANR, for a period of 42 months, starting on October 1st, 2014. http://soprano-project.fr/

The SOPRANO project aims at preparing the next generation of verification-oriented solvers by gathering experts from academia and industry. We will design a new framework for the cooperation of solvers, focused on model generation and borrowing principles from SMT (current standard) and CP (well-known in optimization). Our main scientific and technical objectives are the following. The first objective is to design a new collaboration framework for solvers, centered around synthesis rather than satisfiability and allowing cooperation beyond that of Nelson-Oppen while still providing minimal interfaces with theoretical guarantees. The second objective is to design new decision procedures for industry-relevant and hard-to-solve theories. The third objective is to implement these results in a new open-source platform. The fourth objective is to ensure industrial-adequacy of the techniques and tools developed through periodical evaluations from the industrial partners.

Partners: team DIVERSE (Inria Rennes - Bretagne Atlantique), Adacore, CEA List, Université Paris-Sud, and OCamlPro.

9.2.5. FUI LCHIP

Participant: Sylvain Conchon [contact].

LCHIP (Low Cost High Integrity Platform) is aimed at easing the development of safety critical applications (up to SIL4) by providing: (i) a complete IDE able to automatically generate and prove bounded complexity software (ii) a low cost, safe execution platform. The full support of DSLs and third party code generators will enable a seamless deployment into existing development cycles. LCHIP gathers scientific results obtained during the last 20 years in formal methods, proof, refinement, code generation, etc. as well as a unique return of experience on safety critical systems design. http://www.clearsy.com/en/2016/10/4260/

Partners: 2 technology providers (ClearSy, OcamlPro), in charge of building the architecture of the platform; 3 labs (IFSTTAR, LIP6, LRI), to improve LCHIP IDE features; 2 large companies (SNCF, RATP), representing public ordering parties, to check compliance with standard and industrial railway use-case.

The project lead by ClearSy has started in April 2016 and lasts 3 years. It is funded by BpiFrance as well as French regions.

9.2.6. ANR PARDI

Participant: Sylvain Conchon [contact].

Verification of PARameterized DIstributed systems. A parameterized system specification is a specification for a whole class of systems, parameterized by the number of entities and the properties of the interaction, such as the communication model (synchronous/asynchronous, order of delivery of message, application ordering) or the fault model (crash failure, message loss). To assist and automate verification without parameter instantiation, PARDI uses two complementary approaches. First, a fully automatic model checker modulo theories is considered. Then, to go beyond the intrinsic limits of parameterized model checking, the project advocates a collaborative approach between proof assistant and model checker. http://pardi.enseeiht.fr/

The proof lead by Toulouse INP/IRIT started in 2016 and lasts for 4 years. Partners: Université Pierre et Marie Curie (LIP6), Université Paris-Sud (LRI), Inria Nancy (team VERIDIS)

9.3. European Initiatives

9.3.1. Collaborations in European Programs, Except FP7 & H2020

Program: COST (European Cooperation in Science and Technology).

Project acronym: EUTypes https://eutypes.cs.ru.nl/

Project title: The European research network on types for programming and verification

Duration: 2015-2019

Coordinator: Herman Geuvers, Radboud University Nijmegen, The Netherlands

Other partners: 36 members countries, see http://www.cost.eu/COST_Actions/ca/CA15123?parties

Abstract: Types are pervasive in programming and information technology. A type defines a formal interface between software components, allowing the automatic verification of their connections, and greatly enhancing the robustness and reliability of computations and communications. In rich dependent type theories, the full functional specification of a program can be expressed as a type. Type systems have rapidly evolved over the past years, becoming more sophisticated, capturing new aspects of the behaviour of programs and the dynamics of their execution.

This COST Action will give a strong impetus to research on type theory and its many applications in computer science, by promoting (1) the synergy between theoretical computer scientists, logicians and mathematicians to develop new foundations for type theory, for example as based on the recent development of "homotopy type theory", (2) the joint development of type theoretic tools as proof assistants and integrated programming environments, (3) the study of dependent types for programming and its deployment in software development, (4) the study of dependent types for verification and its deployment in software analysis and verification. The action will also tie together these different areas and promote cross-fertilisation.

9.4. International Research Visitors

9.4.1. Visits of International Scientists

Ran Chen is a PhD student from Institute of Software (Chinese Academy of Sciences, Beijing, China) visiting the team for 10 months under the supervision of C. Marché and J.-J. Lévy (PiR2 team, Inria Paris). She worked on the formal verification of graphs algorithms [25], [17], and also in the context of the CoLiS project on verification of some aspects of the Unix file system and shell scripts [74] [11]

10. Dissemination

10.1. Promoting Scientific Activities

10.1.1. Scientific Events Organisation

10.1.1.1. General Chair, Scientific Chair

S. Boldo, vice-president of the 28th “Journées Francophones des Langages Applicatifs” (JFLA 2017)

S. Boldo, president of the 29th “Journées Francophones des Langages Applicatifs” (JFLA 2018)

J.-C. Filliâtre, scientific chair and co-organizer of EJCP (École Jeunes Chercheurs en Programmation du GDR GPL) at Toulouse on June 26–30, 2017. http://ejcp2017.enseeiht.fr/

10.1.1.2. Member of the Organizing Committees

S. Conchon, local chair for the 44th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2017), held in Paris, France in January 2017. http://conf.researchr.org/home/POPL-2017).

C. Marché, co-organizer of the first joint Frama-C/SPARK day (May, Paris, http://frama-c.com/FCSD17.html), in the context of the Open Source Innovation Spring (http://www.open-source-innovation-spring.org/).

10.1.2. Scientific Events Selection

10.1.2.1. Chair of Conference Program Committees

A. Paskevich, program chair of the 9th Working Conference on Verified Software: Theories, Tools, and Experiments (VSTTE 2017), in collaboration with Thomas Wies (NYU) [35].

S. Boldo, program chair of the 10th International Workshop on Numerical Software Verification (NSV 2017) in collaboration with Alessandro Abate (Oxford) [33].

S. Boldo, program vice-chair of the 28th “Journées Francophones des Langages Applicatifs” (JFLA 2017) [34].

S. Boldo, program chair of the 29th “Journées Francophones des Langages Applicatifs” (JFLA 2018).

10.1.2.2. Member of the Conference Program Committees

S. Boldo, PC of the 24th IEEE Symposium on Computer Arithmetic (ARITH 2017)

S. Boldo, PC of the 25th IEEE Symposium on Computer Arithmetic (ARITH 2018)

S. Boldo, PC of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs (CPP 2017)

S. Boldo, PC of the 7th ACM SIGPLAN Conference on Certified Programs and Proofs (CPP 2018)

S. Boldo, PC of the 8th International Conference on Interactive Theorem Proving (ITP 2017)

S. Boldo, PC of the Tenth NASA Formal Methods Symposium (NFM 2018)

G. Melquiond, PC of the 3rd International Workshop on Coq for Programming Languages (CoqPL 2017).

G. Melquiond, PC of the 1st ACM SIGPLAN Workshop on Machine Learning and Programming Languages (MAPL 2017).

G. Melquiond, PC of the 10th International Workshop on Numerical Software Verification (NSV 2017).

10.1.2.3. Reviewer

The members of the Toccata team have reviewed papers for numerous international conferences.

10.1.3. Journal

10.1.3.1. Member of the Editorial Boards

G. Melquiond, member of the editorial board of *Reliable Computing*.

S. Boldo, member of the editorial board of Binaire http://binaire.blog.lemonde.fr, the blog of the French Computer Science Society.

10.1.3.2. Reviewer - Reviewing Activities

The members of the Toccata team have reviewed numerous papers for numerous international journals.

10.1.4. Invited Talks

S. Boldo gave a talk at EDF in Palaiseau on April 20th

S. Boldo gave a talk at the ModeliScale IPL in Paris on July 4th

S. Boldo gave a talk to teachers in Luminy on May 4th

S. Boldo gave a talk at the université of Saint-Denis de la Réunion on December 8th

10.1.5. Leadership within the Scientific Community

S. Boldo, elected chair of the ARITH working group of the GDR-IM (a CNRS subgroup of computer science) with J. Detrey (Inria Nancy).

10.1.6. Scientific Expertise

C. Marché, member of the scientific commission of Inria-Saclay, in charge of selecting candidates for PhD grants, Post-doc grants, temporary leaves from universities ("délégations").

C. Marché, member of the "Bureau du Comité des Projets" of Inria-Saclay, in charge of examining proposals for creation of new Inria project-teams.

S. Boldo, member of the program committee for selecting postdocs of the maths/computer science program of the Labex mathématique Hadamard.

S. Boldo, member of a hiring committee for an associate professor position in computer science at Université Joseph Fourier, Grenoble, France.

S. Boldo, member of the 2017 committee for the Gilles Kahn PhD award of the French Computer Science Society.

10.1.7. Research Administration

G. Melquiond, member of the committee for the monitoring of PhD students (*"commission de suivi doctoral"*).

10.2. Teaching - Supervision - Juries

10.2.1. Teaching

Master Parisien de Recherche en Informatique (MPRI) https://wikimpri.dptinfo.ens-cachan.fr/doku.php: "Proofs of Programs" http://www.lri.fr/~marche/MPRI-2-36-1/ (M2), C. Marché (12h), A. Charguéraud (12h), Université Paris-Diderot, France.

Master: Fondements de l'informatique et ingénierie du logiciel (FIIL) https://www.lri.fr/~conchon/parcours_fiil/: "Software Model Checking" (M2), S. Conchon (9h), "Programmation C++11 avancée" (M2), G. Melquiond (16h), "Vérification déductive de programmes" (M2), A. Paskevich (10.5h), Université Paris-Sud, France.

DUT (Diplôme Universitaire de Technologie): M1101 "Introduction aux systèmes informatiques", A. Paskevich (36h), M3101 "Principes des systèmes d'exploitation", A. Paskevich (58.5h), IUT d'Orsay, Université Paris-Sud, France.

Licence: “Langages de programmation et compilation” (L3), J.-C. Filliâtre (26h), École Normale Supérieure, France.

Licence: “INF411: Les bases de l’algorithmique et de la programmation” (L3), J.-C. Filliâtre (16h), École Polytechnique, France.

Master: “INF564: Compilation” (M1), J.-C. Filliâtre (18h), École Polytechnique, France.

Licence: “Programmation fonctionnelle avancée” (L3), S. Conchon (45h), Université Paris-Sud, France.

Licence: “Introduction à la programmation fonctionnelle” (L2), S. Conchon (25h), Université Paris-Sud, France.

10.2.2. Internships

R. Rieu-Helft (ENS, Paris) was a pre-PhD student doing an internship under supervision of C. Marché and G. Melquiond. He worked on the design and the formal verification of a library for unbounded integer arithmetic [23]. He implemented in Why3 a mechanism for extracting code to the C language, in order to obtain a certified code that runs very efficiently [30].

D. Gallois-Wong was a Master-2 intern for 4 months under the supervision of S. Boldo. She began the formalization in Coq of numerical filters.

V. Tourneur was a Master-1 intern for 4 months under the supervision of S. Boldo. He developed and proved a new algorithm for computing the average of two floating-point numbers when the radix is 10.

10.2.3. Supervision

PhD in progress: M. Clochard, “Méthodes et outils pour la spécification et la preuve de propriétés difficiles de programmes séquentiels”, since Oct. 2013, supervised by C. Marché and A. Paskevich.

PhD in progress: D. Declerck, “Vérification par des techniques de test et model checking de programmes C11”, since Sep. 2014, supervised by F. Zaïdi (LRI) and S. Conchon.

PhD in progress: M. Roux, “Model Checking de systèmes paramétrés et temporisés”, since Sep. 2015, supervised by Sylvain Conchon.

PhD in progress: M. Pereira, “A Verified Graph Library. Tools and techniques for the verification of modular higher-order programs, with extraction”, since May 2015, supervised by J.-C. Filliâtre.

PhD in progress: A. Coquereau, “[ErgoFast] Amélioration de performances pour le solveur SMT Alt-Ergo : conception d’outils d’analyse, optimisations et structures de données efficaces pour OCaml”, since Sep. 2015, supervised by S. Conchon, F. Le Fessant et M. Mauny.

PhD in progress: F. Faissole, “Stabilité(s): liens entre l’arithmétique flottante et l’analyse numérique”, since Oct. 2016, supervised by S. Boldo and A. Chapoutot.

PhD in progress: R. Rieu-Helft, “Développement et vérification de bibliothèques d’arithmétique entière en précision arbitraire”, since Oct. 2017, supervised by G. Melquiond and P. Cuoq (TrustIn-Soft).

PhD in progress: D. Gallois-Wong, “Vérification formelle et filtres numériques”, since Oct. 2017, supervised by S. Boldo and T. Hilaire.

10.2.4. Juries

C. Marché: reviewer of the habilitation thesis of R. Bubel, “Deductive Verification: From Theory to Practice”, Technische Universität Darmstadt, Germany, November 2017.

S. Boldo: reviewer and member of the PhD defense of A. Plet, École Normale Supérieure de Lyon, Lyon, France, July 2017.

S. Boldo: reviewer and member of the PhD defense of F. Maurica, Université de la Réunion, Saint-Denis, France, December 2017.

S. Boldo: president of the PhD defense of T. Sibut-Pinote, Université Paris-Saclay, Palaiseau, France, December 2017.

10.3. Popularization

S. Boldo, scientific head for Saclay for the MECSI group for networking about computer science popularization inside Inria.

S. Boldo gave a talk at the Inria Saclay about how to popularize programming.

During the “Fête de la science” on October 13th, S. Boldo demonstrated unplugged computer science to teenagers and F. Faissole run a stand about an introduction to programming with robots. S. Boldo also did this activity to kids from 7 to 17 at the Massy opera on November, 17th.

S. Boldo gave a talk during at a *Girls can code* weekend on August 23rd in Paris.

S. Boldo went to the Arpajon high-school for presenting Women in Science on December 19th.

S. Boldo gave a popularization talk to the administrative staff of Inria at Rocquencourt for the Inria birthday on November 16th.

11. Bibliography

Major publications by the team in recent years

[1] F. Bobot, S. Conchon, É. Contejean, M. Iguernelala, A. Mahboubi, A. Mebsout, G. Melquiond. *A Simplex-Based Extension of Fourier-Motzkin for Solving Linear Integer Arithmetic*, in "IJCAR 2012: Proceedings of the 6th International Joint Conference on Automated Reasoning", Manchester, UK, B. Gramlich, D. Miller, U. Sattler (editors), Lecture Notes in Computer Science, Springer, June 2012, vol. 7364, pp. 67–81, http://hal.inria.fr/hal-00687640

[2] F. Bobot, J.-C. Filliâtre, C. Marché, A. Paskevich. *Let's Verify This with Why3*, in "International Journal on Software Tools for Technology Transfer (STTT)", 2015, vol. 17, n^o 6, pp. 709–727, http://hal.inria.fr/hal-00967132/en

[3] S. Boldo, F. Clément, J.-C. Filliâtre, M. Mayero, G. Melquiond, P. Weis. *Wave Equation Numerical Resolution: a Comprehensive Mechanized Proof of a C Program*, in "Journal of Automated Reasoning", April 2013, vol. 50, n^o 4, pp. 423–456, http://hal.inria.fr/hal-00649240/en/

[4] S. Boldo, G. Melquiond. *Flocq: A Unified Library for Proving Floating-point Algorithms in Coq*, in "Proceedings of the 20th IEEE Symposium on Computer Arithmetic", Tübingen, Germany, E. Antelo, D. Hough, P. Ienne (editors), 2011, pp. 243–252, http://hal.archives-ouvertes.fr/inria-00534854/

[5] S. Conchon, A. Goel, S. Krstić, A. Mebsout, F. Zaïdi. *Cubicle: A Parallel SMT-based Model Checker for Parameterized Systems*, in "CAV 2012: Proceedings of the 24th International Conference on Computer Aided Verification", Berkeley, California, USA, M. Parthasarathy, S. A. Seshia (editors), Lecture Notes in Computer Science, Springer, July 2012, vol. 7358, http://hal.archives-ouvertes.fr/hal-00799272

[6] J.-C. Filliâtre, L. Gondelman, A. Paskevich. *The Spirit of Ghost Code*, in "Formal Methods in System Design", 2016, vol. 48, n^o 3, pp. 152–174, https://hal.archives-ouvertes.fr/hal-01396864v1

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

[8] C. MARCHÉ. *Verification of the Functional Behavior of a Floating-Point Program: an Industrial Case Study*, in "Science of Computer Programming", March 2014, vol. 96, n^{o} 3, pp. 279–296, http://hal.inria.fr/hal-00967124/en

[9] É. MARTIN-DOREL, G. MELQUIOND. *Proving Tight Bounds on Univariate Expressions with Elementary Functions in Coq*, in "Journal of Automated Reasoning", 2016, https://hal.inria.fr/hal-01086460

Publications of the year

Articles in International Peer-Reviewed Journals

[10] S. BOLDO, S. GRAILLAT, J.-M. MULLER. *On the robustness of the 2Sum and Fast2Sum algorithms*, in "ACM Transactions on Mathematical Software", July 2017, vol. 44, n^{o} 1, https://hal-ens-lyon.archives-ouvertes.fr/ensl-01310023

[11] R. CHEN, M. CLOCHARD, C. MARCHÉ. *A Formally Proved, Complete Algorithm for Path Resolution with Symbolic Links*, in "Journal of Formalized Reasoning", November 2017, vol. 10, n^{o} 1, https://hal.inria.fr/hal-01652148

[12] M. CLOCHARD, L. GONDELMAN, M. PEREIRA. *The Matrix Reproved: Verification Pearl*, in "Journal of Automated Reasoning", October 2017, pp. 1–19 [*DOI :* 10.1007/s10817-017-9436-2], https://hal.inria.fr/hal-01617437

International Conferences with Proceedings

[13] S. BOLDO, F. CLÉMENT, F. FAISSOLE, V. MARTIN, M. MAYERO. *A Coq Formal Proof of the Lax–Milgram theorem*, in "6th ACM SIGPLAN Conference on Certified Programs and Proofs", Paris, France, January 2017 [*DOI :* 10.1145/3018610.3018625], https://hal.inria.fr/hal-01391578

[14] S. BOLDO, F. FAISSOLE, A. CHAPOUTOT. *Round-off Error Analysis of Explicit One-Step Numerical Integration Methods*, in "24th IEEE Symposium on Computer Arithmetic", London, United Kingdom, July 2017 [*DOI :* 10.1109/ARITH.2017.22], https://hal.archives-ouvertes.fr/hal-01581794

[15] S. BOLDO, M. JOLDES, J.-M. MULLER, V. POPESCU. *Formal Verification of a Floating-Point Expansion Renormalization Algorithm*, in "8th International Conference on Interactive Theorem Proving (ITP'2017)", Brasilia, Brazil, Lecture Notes in Computer Science, September 2017, vol. 10499, https://hal.archives-ouvertes.fr/hal-01512417

[16] A. CHARGUÉRAUD, F. POTTIER. *Temporary Read-Only Permissions for Separation Logic*, in "Proceedings of the 26th European Symposium on Programming (ESOP 2017)", Uppsala, Sweden, April 2017, https://hal.inria.fr/hal-01408657

[17] R. CHEN, J.-J. LÉVY. *A Semi-automatic Proof of Strong connectivity*, in "9th Working Conference on Verified Software: Theories, Tools and Experiments (VSTTE)", Heidelberg, Germany, July 2017, https://hal.inria.fr/hal-01632947

[18] S. CONCHON, M. IGUERNELALA, K. JI, G. MELQUIOND, C. FUMEX. *A Three-tier Strategy for Reasoning about Floating-Point Numbers in SMT*, in "29th International Conference on Computer Aided Verification", Heidelberg, Germany, V. KUNCAK, R. MAJUMDAR (editors), Lecture Notes in Computer Science,

Springer, July 2017, vol. 10427, pp. 419-435 [*DOI :* 10.1007/978-3-319-63390-9_22], https://hal.inria.fr/hal-01522770

[19] F. FAISSOLE. *Formalization and closedness of finite dimensional subspaces*, in "SYNASC 2017 - 19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing", Timioara, Romania, September 2017, pp. 1-7, https://hal.inria.fr/hal-01630411

[20] J.-C. FILLIÂTRE, M. PEREIRA, S. MELO DE SOUSA. *Vérification de programmes OCaml fortement impératifs avec Why3*, in "JFLA 2018 - Journées Francophones des Langages Applicatifs", Banyuls-sur-Mer, France, January 2018, pp. 1-14, https://hal.inria.fr/hal-01649989

[21] C. FUMEX, C. MARCHÉ, Y. MOY. *Automating the Verification of Floating-Point Programs*, in "9th Working Conference on Verified Software: Theories, Tools and Experiments", Heidelberg, Germany, A. PASKEVICH, T. WIES (editors), Lecture Notes in Computer Science, Springer, July 2017, vol. 10712, https://hal.inria.fr/hal-01534533

[22] N. JEANNEROD, C. MARCHÉ, R. TREINEN. *A Formally Verified Interpreter for a Shell-like Programming Language*, in "VSTTE 2017 - 9th Working Conference on Verified Software: Theories, Tools, and Experiments", Heidelberg, Germany, Lecture Notes in Computer Science, July 2017, vol. 10712, https://hal.archives-ouvertes.fr/hal-01534747

[23] R. RIEU-HELFT, C. MARCHÉ, G. MELQUIOND. *How to Get an Efficient yet Verified Arbitrary-Precision Integer Library*, in "9th Working Conference on Verified Software: Theories, Tools, and Experiments", Heidelberg, Germany, A. PASKEVICH, T. WIES (editors), Lecture Notes in Computer Science, July 2017, vol. 10712, pp. 84-101 [*DOI :* 10.1007/978-3-319-72308-2_6], https://hal.inria.fr/hal-01519732

National Conferences with Proceedings

[24] S. BOLDO, F. CLÉMENT, F. FAISSOLE, V. MARTIN, M. MAYERO. *Preuve formelle du théorème de Lax–Milgram*, in "16èmes journées Approches Formelles dans l'Assistance au Développement de Logiciels", Montpellier, France, June 2017, https://hal.archives-ouvertes.fr/hal-01581807

[25] R. CHEN, J.-J. LÉVY. *Une preuve formelle de l'algorithme de Tarjan-1972 pour trouver les composantes fortement connexes dans un graphe*, in "JFLA 2017 - Vingt-huitièmes Journées Francophones des Langages Applicatifs", Gourette, France, Vingt-huitièmes Journées Francophones des Langages Applicatifs, January 2017, https://hal.inria.fr/hal-01422215

[26] M. CLOCHARD. *Preuves taillées en biseau*, in "vingt-huitièmes Journées Francophones des Langages Applicatifs (JFLA)", Gourette, France, January 2017, https://hal.inria.fr/hal-01404935

[27] F. FAISSOLE. *Définir le fini : deux formalisations d'espaces de dimension finie*, in "JLFA 2017 - Journées Francophones des Langages Applicatifs", Banyuls-sur-mer, France, 29èmes Journées Francophones des Langages Applicatifs, January 2018, pp. 1-6, https://hal.inria.fr/hal-01654457

[28] F. FAISSOLE, B. SPITTERS. *Preuves constructives de programmes probabilistes*, in "JFLA 2018 - Journées Francophones des Langages Applicatifs", Banyuls-sur-Mer, France, 29èmes Journées Francophones des Langages Applicatifs, January 2018, https://hal.inria.fr/hal-01654459

[29] M. PEREIRA. *Défonctionnaliser pour prouver*, in "JFLA 2017 - Vingt-huitième Journées Francophones des Langages Applicatifs", Gourette, France, January 2017, https://hal.inria.fr/hal-01378068

[30] R. RIEU-HELFT. *Un mécanisme d'extraction vers C pour Why3*, in "29èmes Journées Francophones des Langages Applicatifs", Banyuls-sur-Mer, France, January 2018, https://hal.inria.fr/hal-01653153

Conferences without Proceedings

[31] F. FAISSOLE, B. SPITTERS. *Synthetic topology in HoTT for probabilistic programming*, in "The Third International Workshop on Coq for Programming Languages (CoqPL 2017)", Paris, France, January 2017, https://hal.inria.fr/hal-01405762

Scientific Books (or Scientific Book chapters)

[32] S. BOLDO, G. MELQUIOND. *Computer Arithmetic and Formal Proofs: Verifying Floating-point Algorithms with the Coq System*, ISTE Press - Elsevier, December 2017, 326 p. , https://hal.inria.fr/hal-01632617

Books or Proceedings Editing

[33] A. ABATE, S. BOLDO (editors). *10th International Workshop on Numerical Software Verification*, Springer, July 2017, https://hal.inria.fr/hal-01662076

[34] S. BOLDO, J. SIGNOLES (editors). *Vingt-huitièmes Journées Francophones des Langages Applicatifs*, Published by the authors, January 2017, https://hal.inria.fr/hal-01662072

[35] A. PASKEVICH, T. WIES (editors). *Verified Software: Theories, Tools, and Experiments, Revised Selected Papers Presented at the 9th International Conference VSTTE*, springer, Heidelberg, Germany, December 2017, vol. Lecture Notes in Computer Science, n^{O} 10712 [*DOI :* 10.1007/978-3-319-72308-2], https://hal.inria.fr/hal-01670145

Research Reports

[36] I. DAMI, C. MARCHÉ. *The CoLiS language: syntax, semantics and associated tools*, Inria Saclay Ile de France, October 2017, n^{O} RT-0491, pp. 1-22, https://hal.inria.fr/hal-01614488

[37] C. FUMEX, C. MARCHÉ, Y. MOY. *Automated Verification of Floating-Point Computations in Ada Programs*, Inria Saclay Ile de France, April 2017, n^{O} RR-9060, 53 p. , https://hal.inria.fr/hal-01511183

Other Publications

[38] A. CHARGUÉRAUD, J.-C. FILLIÂTRE, M. PEREIRA, F. POTTIER. *VOCAL – A Verified OCAml Library*, September 2017, ML Family Workshop 2017, https://hal.inria.fr/hal-01561094

[39] F. FAISSOLE, B. SPITTERS. *Synthetic topology in homotopy type theory for probabilistic programming*, January 2017, 3 p. , PPS 2017 - Workshop on probabilistic programming semantics , Poster, https://hal.inria.fr/hal-01485397

[40] A. MAHBOUBI, G. MELQUIOND, T. SIBUT-PINOTE. *Formally Verified Approximations of Definite Integrals*, February 2017, working paper or preprint, https://hal.inria.fr/hal-01630143

References in notes

[41] B. Beckert, R. Hähnle, P. H. Schmitt (editors). *Verification of Object-Oriented Software: The KeY Approach*, Lecture Notes in Computer Science, Springer, 2007, vol. 4334

[42] U. A. Acar, A. Charguéraud, M. Rainey. *Theory and Practice of Chunked Sequences*, in "European Symposium on Algorithms", Wroclaw, Poland, A. Schulz, D. Wagner (editors), Springer, September 2014, vol. Lecture Notes in Computer Science, n^{o} 8737, pp. 25–36, https://hal.inria.fr/hal-01087245

[43] U. A. Acar, A. Charguéraud, M. Rainey. *Oracle-Guided Scheduling for Controlling Granularity in Implicitly Parallel Languages*, in "Journal of Functional Programming", November 2016, vol. 26, https://hal.inria.fr/hal-01409069

[44] U. A. Acar, A. Charguéraud, M. Rainey, F. Sieczkowski. *Dag-calculus: a calculus for parallel computation*, in "Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming (ICFP)", Nara, Japan, September 2016, pp. 18–32, https://hal.inria.fr/hal-01409022

[45] J. B. Almeida, M. Barbosa, J.-C. Filliâtre, J. S. Pinto, B. Vieira. *CAOVerif: An Open-Source Deductive Verification Platform for Cryptographic Software Implementations*, in "Science of Computer Programming", October 2012

[46] A. Ayad, C. Marché. *Multi-Prover Verification of Floating-Point Programs*, in "Fifth International Joint Conference on Automated Reasoning", Edinburgh, Scotland, J. Giesl, R. Hähnle (editors), Lecture Notes in Artificial Intelligence, Springer, July 2010, vol. 6173, pp. 127–141, http://hal.inria.fr/inria-00534333

[47] D. Baelde, P. Courtieu, D. Gross-Amblard, C. Paulin-Mohring. *Towards Provably Robust Watermarking*, in "ITP 2012", Lecture Notes in Computer Science, August 2012, vol. 7406, http://hal.inria.fr/hal-00682398

[48] C. Barrett, C. Tinelli. *CVC3*, in "19th International Conference on Computer Aided Verification", Berlin, Germany, W. Damm, H. Hermanns (editors), Lecture Notes in Computer Science, Springer, July 2007, vol. 4590, pp. 298–302

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

[50] P. Behm, P. Benoit, A. Faivre, J.-M. Meynadier. *METEOR : A successful application of B in a large project*, in "Proceedings of FM'99: World Congress on Formal Methods", J. M. Wing, J. Woodcock, J. Davies (editors), Lecture Notes in Computer Science (Springer-Verlag), Springer Verlag, September 1999, pp. 369–387

[51] J. C. Blanchette, A. Paskevich. *TFF1: The TPTP typed first-order form with rank-1 polymorphism*, in "24th International Conference on Automated Deduction (CADE-24)", Lake Placid, USA, Lecture Notes in Artificial Intelligence, Springer, June 2013, vol. 7898, http://hal.inria.fr/hal-00825086

[52] F. Bobot, S. Conchon, É. Contejean, M. Iguernelala, S. Lescuyer, A. Mebsout. *The Alt-Ergo Automated Theorem Prover*, 2008

[53] F. BOBOT, J.-C. FILLIÂTRE. *Separation Predicates: a Taste of Separation Logic in First-Order Logic*, in "14th International Conference on Formal Ingineering Methods (ICFEM)", Kyoto, Japan, Lecture Notes in Computer Science, Springer, November 2012, vol. 7635, http://hal.inria.fr/hal-00825088

[54] F. BOBOT, J.-C. FILLIÂTRE, C. MARCHÉ, G. MELQUIOND, A. PASKEVICH. *Preserving User Proofs Across Specification Changes*, in "Verified Software: Theories, Tools, Experiments (5th International Conference VSTTE)", Atherton, USA, E. COHEN, A. RYBALCHENKO (editors), Lecture Notes in Computer Science, Springer, May 2013, vol. 8164, pp. 191–201, http://hal.inria.fr/hal-00875395

[55] F. BOBOT, J.-C. FILLIÂTRE, C. MARCHÉ, G. MELQUIOND, A. PASKEVICH. *The Why3 platform, version 0.81*, version 0.81, LRI, CNRS & Univ. Paris-Sud & Inria Saclay, March 2013, http://hal.inria.fr/hal-00822856/

[56] M. BODIN, A. CHARGUÉRAUD, D. FILARETTI, P. GARDNER, S. MAFFEIS, D. NAUDZIUNIENE, A. SCHMITT, G. SMITH. *A Trusted Mechanised JavaScript Specification*, in "Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages", San Diego, USA, ACM Press, January 2014, http://hal.inria.fr/hal-00910135

[57] S. BOLDO. *How to Compute the Area of a Triangle: a Formal Revisit*, in "Proceedings of the 21th IEEE Symposium on Computer Arithmetic", Austin, Texas, USA, 2013, http://hal.inria.fr/hal-00790071

[58] S. BOLDO. *Deductive Formal Verification: How To Make Your Floating-Point Programs Behave*, Université Paris-Sud, October 2014, Thèse d'habilitation, https://hal.inria.fr/tel-01089643

[59] S. BOLDO. *Formal verification of tricky numerical computations*, in "16th GAMM-IMACS International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics", Würzburg, Germany, September 2014, https://hal.inria.fr/hal-01088692

[60] S. BOLDO, F. CLÉMENT, J.-C. FILLIÂTRE, M. MAYERO, G. MELQUIOND, P. WEIS. *Formal Proof of a Wave Equation Resolution Scheme: the Method Error*, in "Proceedings of the First Interactive Theorem Proving Conference", Edinburgh, Scotland, M. KAUFMANN, L. C. PAULSON (editors), LNCS, Springer, July 2010, vol. 6172, pp. 147–162, http://hal.inria.fr/inria-00450789/

[61] S. BOLDO, F. CLÉMENT, J.-C. FILLIÂTRE, M. MAYERO, G. MELQUIOND, P. WEIS. *Trusting Computations: a Mechanized Proof from Partial Differential Equations to Actual Program*, in "Computers and Mathematics with Applications", 2014, vol. 68, n^{o} 3, pp. 325–352, http://hal.inria.fr/hal-00769201

[62] S. BOLDO, J.-C. FILLIÂTRE, G. MELQUIOND. *Combining Coq and Gappa for Certifying Floating-Point Programs*, in "16th Symposium on the Integration of Symbolic Computation and Mechanised Reasoning", Grand Bend, Canada, Lecture Notes in Artificial Intelligence, Springer, July 2009, vol. 5625, pp. 59–74

[63] S. BOLDO, J.-H. JOURDAN, X. LEROY, G. MELQUIOND. *A Formally-Verified C Compiler Supporting Floating-Point Arithmetic*, in "Proceedings of the 21th IEEE Symposium on Computer Arithmetic", Austin, Texas, USA, 2013, http://hal.inria.fr/hal-00743090

[64] S. BOLDO, J.-H. JOURDAN, X. LEROY, G. MELQUIOND. *Verified Compilation of Floating-Point Computations*, in "Journal of Automated Reasoning", February 2015, vol. 54, n^{o} 2, pp. 135-163, https://hal.inria.fr/hal-00862689

[65] S. Boldo, C. Lelay, G. Melquiond. *Improving Real Analysis in Coq: a User-Friendly Approach to Integrals and Derivatives*, in "Proceedings of the Second International Conference on Certified Programs and Proofs", Kyoto, Japan, C. Hawblitzel, D. Miller (editors), Lecture Notes in Computer Science, December 2012, vol. 7679, pp. 289–304, http://hal.inria.fr/hal-00712938

[66] S. Boldo, C. Lelay, G. Melquiond. *Coquelicot: A User-Friendly Library of Real Analysis for Coq*, in "Mathematics in Computer Science", June 2015, vol. 9, n^{o} 1, pp. 41-62, http://hal.inria.fr/hal-00860648

[67] S. Boldo, C. Lelay, G. Melquiond. *Formalization of Real Analysis: A Survey of Proof Assistants and Libraries*, in "Mathematical Structures in Computer Science", 2016, http://hal.inria.fr/hal-00806920

[68] S. Boldo, C. Marché. *Formal verification of numerical programs: from C annotated programs to mechanical proofs*, in "Mathematics in Computer Science", 2011, vol. 5, pp. 377–393, http://hal.inria.fr/hal-00777605

[69] S. Boldo, T. M. T. Nguyen. *Proofs of numerical programs when the compiler optimizes*, in "Innovations in Systems and Software Engineering", 2011, vol. 7, pp. 151–160, http://hal.inria.fr/hal-00777639

[70] T. Bormer, M. Brockschmidt, D. Distefano, G. Ernst, J.-C. Filliâtre, R. Grigore, M. Huisman, V. Klebanov, C. Marché, R. Monahan, W. Mostowski, N. Polikarpova, C. Scheben, G. Schellhorn, B. Tofan, J. Tschannen, M. Ulbrich. *The COST IC0701 Verification Competition 2011*, in "Formal Verification of Object-Oriented Software, Revised Selected Papers Presented at the International Conference, FoVeOOS 2011", B. Beckert, F. Damiani, D. Gurov (editors), Lecture Notes in Computer Science, Springer, 2012, vol. 7421, http://hal.inria.fr/hal-00789525

[71] L. Burdy, Y. Cheon, D. R. Cok, M. D. Ernst, J. R. Kiniry, G. T. Leavens, K. R. M. Leino, E. Poll. *An overview of JML tools and applications*, in "International Journal on Software Tools for Technology Transfer (STTT)", June 2005, vol. 7, n^{o} 3, pp. 212–232

[72] R. Chapman, F. Schanda. *Are We There Yet? 20 Years of Industrial Theorem Proving with SPARK*, in "Interactive Theorem Proving - 5th International Conference, ITP 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings", G. Klein, R. Gamboa (editors), Lecture Notes in Computer Science, Springer, 2014, vol. 8558, pp. 17–26

[73] A. Charguéraud, F. Pottier. *Verifying the Correctness and Amortized Complexity of a Union-Find Implementation in Separation Logic with Time Credits*, in "Journal of Automated Reasoning", September 2017

[74] R. Chen, M. Clochard, C. Marché. *A Formal Proof of a Unix Path Resolution Algorithm*, Inria, December 2016, n^{o} RR-8987, https://hal.inria.fr/hal-01406848

[75] M. Clochard. *Automatically verified implementation of data structures based on AVL trees*, in "6th Working Conference on Verified Software: Theories, Tools and Experiments (VSTTE)", Vienna, Austria, D. Giannakopoulou, D. Kroening (editors), Lecture Notes in Computer Science, Springer, July 2014, vol. 8471, pp. 167–180, http://hal.inria.fr/hal-01067217

[76] M. Clochard, J.-C. Filliâtre, C. Marché, A. Paskevich. *Formalizing Semantics with an Automatic Program Verifier*, in "6th Working Conference on Verified Software: Theories, Tools and Experiments

(VSTTE)", Vienna, Austria, D. GIANNAKOPOULOU, D. KROENING (editors), Lecture Notes in Computer Science, Springer, July 2014, vol. 8471, pp. 37–51, http://hal.inria.fr/hal-01067197

[77] M. CLOCHARD, L. GONDELMAN. *Double WP: vers une preuve automatique d'un compilateur*, in "Vingt-sixièmes Journées Francophones des Langages Applicatifs", Val d'Ajol, France, January 2015, https://hal.inria.fr/hal-01094488

[78] M. CLOCHARD, L. GONDELMAN, M. PEREIRA. *The Matrix Reproved*, in "8th Working Conference on Verified Software: Theories, Tools and Experiments (VSTTE)", Toronto, Canada, S. BLAZY, M. CHECHIK (editors), Lecture Notes in Computer Science, Springer, July 2016, https://hal.inria.fr/hal-01316902

[79] M. CLOCHARD, C. MARCHÉ, A. PASKEVICH. *Verified Programs with Binders*, in "Programming Languages meets Program Verification (PLPV)", ACM Press, 2014, http://hal.inria.fr/hal-00913431

[80] S. CONCHON. *SMT Techniques and their Applications: from Alt-Ergo to Cubicle*, Université Paris-Sud, December 2012, In English, http://www.lri.fr/~conchon/publis/conchonHDR.pdf, Thèse d'habilitation

[81] S. CONCHON, É. CONTEJEAN, M. IGUERNELALA. *Canonized Rewriting and Ground AC Completion Modulo Shostak Theories*, in "Tools and Algorithms for the Construction and Analysis of Systems", Saarbrücken, Germany, P. A. ABDULLA, K. R. M. LEINO (editors), Lecture Notes in Computer Science, Springer, April 2011, vol. 6605, pp. 45-59, http://hal.inria.fr/hal-00777663

[82] S. CONCHON, É. CONTEJEAN, M. IGUERNELALA. *Canonized Rewriting and Ground AC Completion Modulo Shostak Theories : Design and Implementation*, in "Logical Methods in Computer Science", September 2012, vol. 8, n^{o} 3, pp. 1–29, http://hal.inria.fr/hal-00798082

[83] S. CONCHON, D. DECLERCK, L. MARANGET, A. MEBSOUT. *Vérification de programmes C concurrents avec Cubicle : Enfoncer les barrières*, in "Vingt-cinquièmes Journées Francophones des Langages Applicatifs", Fréjus, France, January 2014, https://hal.inria.fr/hal-01088655

[84] S. CONCHON, A. GOEL, S. KRSTIĆ, A. MEBSOUT, F. ZAÏDI. *Invariants for Finite Instances and Beyond*, in "FMCAD", Portland, Oregon, États-Unis, October 2013, pp. 61–68, http://hal.archives-ouvertes.fr/hal-00924640

[85] S. CONCHON, M. IGUERNELALA. *Tuning the Alt-Ergo SMT Solver for B Proof Obligations*, in "Abstract State Machines, Alloy, B, VDM, and Z (ABZ)", Toulouse, France, Lecture Notes in Computer Science, Springer, June 2014, vol. 8477, pp. 294–297, https://hal.inria.fr/hal-01093000

[86] S. CONCHON, M. IGUERNELALA, A. MEBSOUT. *A Collaborative Framework for Non-Linear Integer Arithmetic Reasoning in Alt-Ergo*, 2013, https://hal.archives-ouvertes.fr/hal-00924646

[87] S. CONCHON, A. MEBSOUT, F. ZAÏDI. *Vérification de systèmes paramétrés avec Cubicle*, in "Vingt-quatrièmes Journées Francophones des Langages Applicatifs", Aussois, France, February 2013, http://hal.inria.fr/hal-00778832

[88] S. CONCHON, G. MELQUIOND, C. ROUX, M. IGUERNELALA. *Built-in Treatment of an Axiomatic Floating-Point Theory for SMT Solvers*, in "SMT workshop", Manchester, UK, P. FONTAINE, A. GOEL (editors), LORIA, 2012, pp. 12–21

[89] M. DAHLWEID, M. MOSKAL, T. SANTEN, S. TOBIES, W. SCHULTE. *VCC: Contract-based modular verification of concurrent C*, in "31st International Conference on Software Engineering, ICSE 2009, May 16-24, 2009, Vancouver, Canada, Companion Volume", IEEE Comp. Soc. Press, 2009, pp. 429-430

[90] D. DELAHAYE, C. DUBOIS, C. MARCHÉ, D. MENTRÉ. *The BWare Project: Building a Proof Platform for the Automated Verification of B Proof Obligations*, in "Abstract State Machines, Alloy, B, VDM, and Z (ABZ)", Toulouse, France, Lecture Notes in Computer Science, Springer, June 2014, vol. 8477, pp. 290–293, http://hal.inria.fr/hal-00998092/en/

[91] D. DELAHAYE, C. MARCHÉ, D. MENTRÉ. *Le projet BWare : une plate-forme pour la vérification automatique d'obligations de preuve B*, in "Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL)", Paris, France, EasyChair, June 2014, http://hal.inria.fr/hal-00998094/en/

[92] C. DROSS, S. CONCHON, J. KANIG, A. PASKEVICH. *Reasoning with Triggers*, in "SMT workshop", Manchester, UK, P. FONTAINE, A. GOEL (editors), LORIA, 2012

[93] C. DROSS. *Generic Decision Procedures for Axiomatic First-Order Theories*, Université Paris-Sud, April 2014, http://tel.archives-ouvertes.fr/tel-01002190

[94] J.-C. FILLIÂTRE. *Combining Interactive and Automated Theorem Proving in Why3 (invited talk)*, in "Automation in Proof Assistants 2012", Tallinn, Estonia, K. HELJANKO, H. HERBELIN (editors), April 2012

[95] J.-C. FILLIÂTRE. *Combining Interactive and Automated Theorem Proving using Why3 (invited tutorial)*, in "Second International Workshop on Intermediate Verification Languages (BOOGIE 2012)", Berkeley, California, USA, Z. RAKAMARIĆ (editor), July 2012

[96] J.-C. FILLIÂTRE. *Verifying Two Lines of C with Why3: an Exercise in Program Verification*, in "Verified Software: Theories, Tools, Experiments (4th International Conference VSTTE)", Philadelphia, USA, R. JOSHI, P. MÜLLER, A. PODELSKI (editors), Lecture Notes in Computer Science, Springer, January 2012, vol. 7152, pp. 83–97

[97] J.-C. FILLIÂTRE. *Deductive Program Verification*, in "Programming Languages Mentoring Workshop (PLMW 2013)", Rome, Italy, N. FOSTER, P. GARDNER, A. SCHMITT, G. SMITH, P. THIEMAN, T. WRIGSTAD (editors), January 2013, http://hal.inria.fr/hal-00799190

[98] J.-C. FILLIÂTRE. *One Logic To Use Them All*, in "24th International Conference on Automated Deduction (CADE-24)", Lake Placid, USA, Lecture Notes in Artificial Intelligence, Springer, June 2013, vol. 7898, pp. 1–20, http://hal.inria.fr/hal-00809651/en/

[99] J.-C. FILLIÂTRE, A. PASKEVICH. *Why3 — Where Programs Meet Provers*, in "Proceedings of the 22nd European Symposium on Programming", M. FELLEISEN, P. GARDNER (editors), Lecture Notes in Computer Science, Springer, March 2013, vol. 7792, pp. 125–128, http://hal.inria.fr/hal-00789533

[100] J.-C. FILLIÂTRE, A. PASKEVICH, A. STUMP. *The 2nd Verified Software Competition: Experience Report*, in "COMPARE2012: 1st International Workshop on Comparative Empirical Evaluation of Reasoning Systems", Manchester, UK, V. KLEBANOV, S. GREBING (editors), EasyChair, June 2012, http://hal.inria.fr/hal-00798777

[101] P. HERMS. *Certification of a Tool Chain for Deductive Program Verification*, Université Paris-Sud, January 2013, http://tel.archives-ouvertes.fr/tel-00789543

[102] P. HERMS, C. MARCHÉ, B. MONATE. *A Certified Multi-prover Verification Condition Generator*, in "Verified Software: Theories, Tools, Experiments (4th International Conference VSTTE)", Philadelphia, USA, R. JOSHI, P. MÜLLER, A. PODELSKI (editors), Lecture Notes in Computer Science, Springer, January 2012, vol. 7152, pp. 2–17, http://hal.inria.fr/hal-00639977

[103] M. IGUERNELALA. *Strengthening the Heart of an SMT-Solver: Design and Implementation of Efficient Decision Procedures*, Université Paris-Sud, June 2013, http://tel.archives-ouvertes.fr/tel-00842555

[104] D. ISHII, G. MELQUIOND, S. NAKAJIMA. *Inductive Verification of Hybrid Automata with Strongest Postcondition Calculus*, in "Proceedings of the 10th Conference on Integrated Formal Methods", Turku, Finland, E. B. JOHNSEN, L. PETRE (editors), Lecture Notes in Computer Science, 2013, vol. 7940, pp. 139–153, http://hal.inria.fr/hal-00806701

[105] J. KANIG, E. SCHONBERG, C. DROSS. *Hi-Lite: the convergence of compiler technology and program verification*, in "Proceedings of the 2012 ACM Conference on High Integrity Language Technology, HILT '12", Boston, USA, B. BROSGOL, J. BOLENG, S. T. TAFT (editors), ACM Press, 2012, pp. 27–34

[106] G. KLEIN, J. ANDRONICK, K. ELPHINSTONE, G. HEISER, D. COCK, P. DERRIN, D. ELKADUWE, K. ENGELHARDT, R. KOLANSKI, M. NORRISH, T. SEWELL, H. TUCH, S. WINWOOD. *seL4: Formal verification of an OS kernel*, in "Communications of the ACM", June 2010, vol. 53, n^o 6, pp. 107–115

[107] C. LELAY. *A New Formalization of Power Series in Coq*, in "5th Coq Workshop", Rennes, France, July 2013, pp. 1–2, http://hal.inria.fr/hal-00880212

[108] C. LELAY. *Coq passe le bac*, in "JFLA - Journées francophones des langages applicatifs", Fréjus, France, January 2014

[109] C. LELAY, G. MELQUIOND. *Différentiabilité et intégrabilité en Coq. Application à la formule de d'Alembert*, in "Vingt-troisièmes Journées Francophones des Langages Applicatifs", Carnac, France, February 2012, http://hal.inria.fr/hal-00642206/fr/

[110] X. LEROY. *A formally verified compiler back-end*, in "Journal of Automated Reasoning", 2009, vol. 43, n^o 4, pp. 363–446, http://hal.inria.fr/inria-00360768/en/

[111] A. MAHBOUBI, G. MELQUIOND, T. SIBUT-PINOTE. *Formally Verified Approximations of Definite Integrals*, in "Proceedings of the 7th International Conference on Interactive Theorem Proving", Nancy, France, J. C. BLANCHETTE, S. MERZ (editors), Lecture Notes in Computer Science, August 2016, vol. 9807, https://hal.inria.fr/hal-01289616

[112] C. MARCHÉ, A. TAFAT. *Weakest Precondition Calculus, revisited using Why3*, Inria, December 2012, n^o RR-8185, http://hal.inria.fr/hal-00766171

[113] C. MARCHÉ, A. TAFAT. *Calcul de plus faible précondition, revisité en Why3*, in "Vingt-quatrièmes Journées Francophones des Langages Applicatifs", Aussois, France, February 2013, http://hal.inria.fr/hal-00778791

[114] C. MARCHÉ. *Verification of the Functional Behavior of a Floating-Point Program: an Industrial Case Study*, in "Science of Computer Programming", March 2014, vol. 96, n^{o} 3, pp. 279–296, http://hal.inria.fr/hal-00967124/en

[115] É. MARTIN-DOREL, G. MELQUIOND, J.-M. MULLER. *Some Issues related to Double Roundings*, in "BIT Numerical Mathematics", 2013, vol. 53, n^{o} 4, pp. 897–924, http://hal-ens-lyon.archives-ouvertes.fr/ensl-00644408

[116] A. MEBSOUT. *Invariants inference for model checking of parameterized systems*, Université Paris-Sud, September 2014, https://tel.archives-ouvertes.fr/tel-01073980

[117] G. MELQUIOND. *Floating-point arithmetic in the Coq system*, in "Information and Computation", 2012, vol. 216, pp. 14–23, http://hal.inria.fr/hal-00797913

[118] G. MELQUIOND, W. G. NOWAK, P. ZIMMERMANN. *Numerical Approximation of the Masser-Gramain Constant to Four Decimal Digits: delta=1.819...*, in "Mathematics of Computation", 2013, vol. 82, pp. 1235–1246, http://hal.inria.fr/hal-00644166/en/

[119] D. MENTRÉ, C. MARCHÉ, J.-C. FILLIÂTRE, M. ASUKA. *Discharging Proof Obligations from Atelier B using Multiple Automated Provers*, in "ABZ'2012 - 3rd International Conference on Abstract State Machines, Alloy, B and Z", Pisa, Italy, S. REEVES, E. RICCOBENE (editors), Lecture Notes in Computer Science, Springer, June 2012, vol. 7316, pp. 238–251, http://hal.inria.fr/hal-00681781/en/

[120] J.-M. MULLER, N. BRISEBARRE, F. DE DINECHIN, C.-P. JEANNEROD, V. LEFÈVRE, G. MELQUIOND, N. REVOL, D. STEHLÉ, S. TORRES. *Handbook of Floating-Point Arithmetic*, Birkhäuser, 2010

[121] T. M. T. NGUYEN, C. MARCHÉ. *Hardware-Dependent Proofs of Numerical Programs*, in "Certified Programs and Proofs", J.-P. JOUANNAUD, Z. SHAO (editors), Lecture Notes in Computer Science, Springer, December 2011, pp. 314–329, http://hal.inria.fr/hal-00772508

[122] T. M. T. NGUYEN. *Taking architecture and compiler into account in formal proofs of numerical programs*, Université Paris-Sud, June 2012, http://tel.archives-ouvertes.fr/tel-00710193

[123] M. NORRISH. *C Formalised in HOL*, University of Cambridge, November 1998

[124] M. PEREIRA, J.-C. FILLIÂTRE, S. M. DE SOUSA. *ARMY: a Deductive Verification Platform for ARM Programs Using Why3*, in "INForum 2012", September 2012

[125] P. ROUX. *Formal Proofs of Rounding Error Bounds*, in "Journal of Automated Reasoning", 2015, https://hal.archives-ouvertes.fr/hal-01091189

[126] N. SCHIRMER. *Verification of Sequential Imperative Programs in Isabelle/HOL*, Technische Universität München, 2006

[127] A. TAFAT. *Preuves par raffinement de programmes avec pointeurs*, Université Paris-Sud, September 2013, http://tel.archives-ouvertes.fr/tel-00874679

[128] F. DE DINECHIN, C. LAUTER, G. MELQUIOND. *Certifying the floating-point implementation of an elementary function using Gappa*, in "IEEE Transactions on Computers", 2011, vol. 60, n^{o} 2, pp. 242–253, http://hal.inria.fr/inria-00533968/en/

[129] L. DE MOURA, N. BJØRNER. *Z3, An Efficient SMT Solver*, in "TACAS", Lecture Notes in Computer Science, Springer, 2008, vol. 4963, pp. 337–340