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

Activity Report 2014

Project-Team TOCCATA

Formally Verified Programs, Certified Tools, Numerical 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

Keywords: Formal Methods, Verification, Proofs Of Programs, Theorem Proving, Computer Arithmetic

The Toccata team (http://toccata.lri.fr/) is a research team common to Inria Saclay–Île-de-France, CNRS, and Université Paris-Sud. Team members are also members of the larger VALS research group (Verification of Algorithms, Languages and systems; http://vals.lri.fr/) of the LRI (Laboratoire de Recherche en Informatique, UMR 8623).

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

1. Members

Research Scientists

Claude Marché [Team leader, Inria, Senior Researcher, HdR]
Sylvie Boldo [Team co-leader, Inria, Researcher, HdR]
Arthur Charguéraud [Inria, Researcher]
Évelyne Contejean [CNRS, Researcher, HdR]
Jean-Christophe Filliâtre [CNRS, Researcher, HdR]
Guillaume Melquiond [Inria, Researcher]

Faculty Members

Sylvain Conchon [Univ. Paris-Sud, Professor, HdR]
Andrei Paskevich [Univ. Paris-Sud, Associate Professor]
Christine Paulin-Mohring [Univ. Paris-Sud, Professor, HdR]

Engineer

Clément Fumex [Inria, from Nov 2014, ProofInUse grant]

PhD Students

Martin Clochard [Univ. Paris-Sud]
David Declerck [Univ. Paris-Sud, from Oct. 2014 (M2 intern from Mar. to Sep.)]
Claire Dross [Univ. Paris-Sud, until Apr 2014, CIFRE AdaCore grant]
Stefania Dumbrava [Univ. Paris-Sud]
Levs Gondelmans [Univ. Paris-Sud]
Catherine Lelay [Inria]
Jacques-Charles Mbiada Ndjanda [from Nov 2014, CEA grant]

Post-Doctoral Fellows

Michael Marcozzi [Inria, from Nov 2014, ERCIM grant]
Érik Martin-Dorel [Inria, until Aug 2014, VERASCO grant]
Alain Mebsout [Inria, CAFEIN grant (PhD until Sep.)]
Asma Tafat Bouzid [Univ. Paris-Sud, ATER, until Aug 2014]

Visiting Scientists

Pierre Roux [ONERA, Toulouse, until Aug 2014]
Bas Spitters [Nijmegen University, the Netherlands, from Apr 2014 until Jun 2014]
Andrew Tolmach [Portland State University, USA, from Sep 2014, Digiteo Chair]

Administrative Assistant

Régine Bricquet [Inria]

Others

Yohan Chatelain [L3 intern Univ. Paris-Sud, from May 2014 until Jun 2014]
Jacques-Pascal Deplaix [Intern EPITECH, from May 2014 until Aug 2014]

Julien Grangier [M1 intern ENSIEE, from Jun 2014 until Sep 2014]
Xavier Onfroy [L3 intern ENS Lyon, from Jun 2014 until Jul 2014]
Thibaut Balabonski [Associate member, Univ. Paris-Sud, Associate Professor, from Oct 2014]
Xavier Urbain [Associate Member, ENSIEE, Associate Professor]
Mohamed Iguernelala [Associate Member, Research Engineer at OCamlPro]

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 a high assurance of its safety and its correctness with respect to its intended behavior.

2.1.1. State of the Art of Program Verification

The importance of software in critical systems increased a lot in the past decades. Critical software appears in various application domains such as transportation (e.g. airplanes, railway), communication (e.g. smart phones) or banking. The set of available features is quickly increasing, together with the number of lines of codes 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, the main process to check safety of a software was to apply heavy test campaigns, which take a large part of the costs of software development.

This is why *static analysis* techniques were invented to complement tests. The general aim is to analyze a program code without executing it, to get as much guarantees as possible on all possible executions at once. The main classes of static analysis techniques are:

- *Abstract Interpretation*: it approximates program execution by abstracting program states into well-chosen abstraction domains. The reachable abstract states are then analyzed in order to detect possible mistakes, corresponding to abstract states that should not occur. The efficiency of this approach relies on the quality of the approximation: if it is too coarse, false positives will appear, which the user needs to analyze manually to determine if the error is real or not. A major success of this kind of approach is the verification of absence of run-time errors in the control-command software of the Airbus A380 by the tool *Astrée* [80].
- *Model-checking*: it denotes a class of approaches that got a great success in industry, e.g. the quality of device drivers of Microsoft's Windows operating system increased a lot by systematic application of such an approach [48]. A program is abstracted into a finite graph representing an approximation of its execution. Functional properties expected for the execution can be expressed using formal logic (typically temporal logic) that can be checked valid by an exploration of the graph. The major issue of model-checking is that the size of the graph can get very large. Moreover, to get less coarse approximations, one may be interested in abstracting a program into an infinite graph. In that case, extensions of model-checking are proposed: bounded model-checking, symbolic model-checking, etc. *Predicate Abstraction* is also a rather successful kind of model-checking approach because of its ability of getting iteratively refined to suppress false positives [49].
- *Deductive verification*: it differs from the other approaches in that it does not approximate program execution. It originates from the well-known *Hoare logic* approach. Programs are formally specified using expressive logic languages, and mathematical methods are applied to formally prove that a program meets its specification.

The *Toccata* project is mainly interested in exploring the deductive verification approach, although we believe that the above classes of approaches are compatible. We indeed have studied some way to combine these approaches in the past [11], [95], [76].

2.1.2. Deductive Program Verification Nowadays

In the past decade, significant progresses have been made in the domain of deductive program verification. They are emphasized by some successful application of these techniques on industrial-scale software, e.g. the *Atelier B* system was used to develop part of the embedded software of the Paris metro line 14 [57] and other railroad-related systems, a formally proved C compiler was developed using the *Coq* proof assistant [102], Microsoft's hypervisor for highly secure virtualization was verified using VCC [81] and the Z3 prover [124], 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 [97]. Another sign of recent progress is the emergence of deductive verification competitions.

In the deductive verification context, there are two main families of approaches. Methods in the first family build on top of mathematical proof assistants (e.g. Coq, Isabelle) in which both the models and the programs are encoded, and the proofs that a 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 [56], JML [69]) and automatically producing a set of mathematical formulas (the *verification conditions*) which are typically proved using automatic provers (e.g. Z3 [124], *Alt-Ergo* [58], CVC3 [54]). The first family of approaches usually offers a higher level of assurance than the second, but also demands more work to establish 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 progresses 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. They can be applied to mainstream languages, but usually only a subset of them is supported. Finally, recent trends in the industrial practice for development of critical software is to require more and more guarantees of safety, e.g. the DO-178C standard for developing avionics software adds to the former DO-178B the use of formal models and formal methods. It also emphasizes the need for certification of the analysis tools involved in the process.

2.1.3. Overview of our Former Contributions

To illustrate our past contributions in the domain of deductive verification, we provide below a set of reference publications of our former scientific results, published in the past 5 years.

Concerning Automated Deduction, our references publications are: a TACAS'2011 paper [7] on an SMT decision procedure for associativity and commutativity, an IJCAR'2012 paper [59] about an original contribution to decision procedures for arithmetic, a CAV'2012 paper [76] presenting an SMT-based model checker, a FroCos'2011 paper [3] presenting an new approach for encoding polymorphic theories into monomorphic ones.

Regarding deductive program verification in general, a first reference publication is the habilitation thesis of Filliâtre [9] which provides a very good survey of our contributions until 2011. A shorter version appeared in the STTT journal [87]. The ABZ'2012 paper [109] is a representative publication presenting an application of our *Why3* system to solving proof obligations coming from *Atelier B*. The VSTTE'2012 paper [89] is a reference case study publication: proof of a program solving the n-queens problem, that uses bitwise operations for efficiency.

In industrial applications, numerical calculations are very common (e.g. control software in transportation). Typically they involve floating-point numbers. Concerning the analysis of numerical programs, our representative publications are: a paper in the MCS journal in 2011[5] presenting on various examples our approach of proving behavioral properties of numerical C programs using *Frama-C/Jessie*, a paper in the TC journal in 2010 [123] presenting the use of the Gappa solver for proving numerical algorithms, a paper in the ISSE journal in 2011 [68] together with a paper at the CPP'2011 conference [113] presenting how we can take architectures and compilers into account when dealing with floating-point programs. We also contributed to the

Handbook of Floating-Point Arithmetic in 2009 [112]. 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, presented in part at ITP'2010 [62] and in another part at ICALP'2009 [4] and fully in a paper in the JAR journal [63].

Finally, about the theme of certification of analysis tools, the reference papers are: a PEPM'2010 [79] and a RTA'2011 paper [8] on certified proofs of termination and other related properties of rewriting systems, and a VSTTE'2012 paper [94] presenting a certified VC generator.

The diagram of Figure 1 details how our tools interact with each other and with other tools. The tools in pink boxes are designed by us, while those in blue boxes are designed by partners. The central tool is *Why3* [2], [88], which includes both a Verification Condition Generator and a set of encoders and printers. The VC generator reads source programs in a dedicated input language that includes both specifications and code. It produces verification conditions that are encoded and printed into various formats and syntax so that a large set of interactive and automatic provers can be used to discharge the verification conditions.

Among automated provers, our own tool *Alt-Ergo* [58] is an SMT solver that is able to deal with quantifiers, arithmetic in integers or real numbers, and a few other theories. *Gappa* [10] is a prover designed to support arithmetic on real numbers and floating-point rounding. As front-ends, our tool *Krakatoa* [104] reads annotated Java source code and produces *Why3* code. The tool *Jessie* [111], [11] is a plug-in for the *Frama-C* environment (which we designed in collaboration with CEA-List); it does the same for annotated C code. The GnatProve tool is a prototype developed by Adacore company; it reads annotated Ada code and also produces *Why3* code. Spark is an ancestor of GnatProve developed by Altran-Praxis company; it has a dedicated prover but it was recently modified so as to produce verification conditions in a suitable format for *Alt-Ergo*.

Last but not least, the modeling of programs semantics and the specification of their expected behaviors is based on some libraries of mathematical theories that we develop, either in the logic language of *Why3* or in *Coq*. These are the yellow boxes of the diagram. For example, we developed in *Coq* the Flocq library for the formalization of floating-point computations [6].

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 originalities is our 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. This is a new goal of the team: 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, we propose a new axis of research: to develop certified tools, i.e. analysis tools that are themselves formally proved correct.

As mentioned above, some of the members of the team have an internationally-recognized expertise on deductive program verification involving floating-point computation [6], including both interactive proving and automated solving [10]. Indeed we noticed that the verification of numerical programs is a representative case that can benefit a lot from combining automatic and interactive theorem proving [64], [5]. This motivated our research on the formal verification of numerical programs.

Moreover, we continue the fundamental studies we conducted in the past concerning deductive program verification in general. This is why our detailed scientific programme is structured into three themes:

1. Formally Verified Programs,
2. Certified Tools,
3. Numerical Programs.

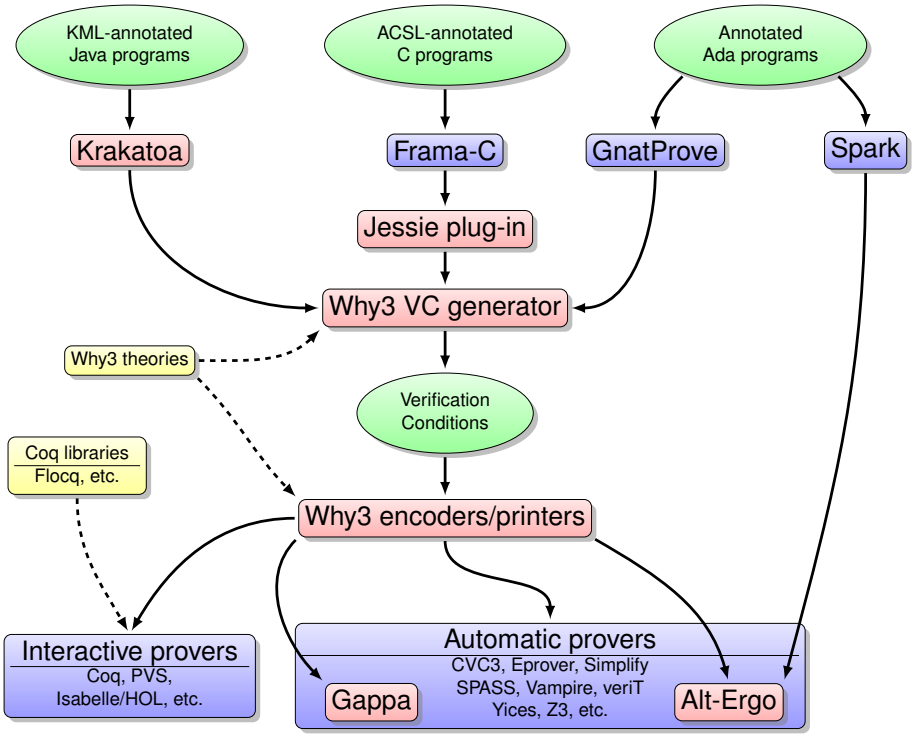

Figure 1. Interactions between our tools and with others

3.2. Formally Verified Programs

Formal program verification is a research theme that builds upon our expertise in the development of methods and tools for proving programs, from source codes annotated with specifications to proofs. In the past, we tackled programs written in mainstream programming languages, with the system *Why3* and the front-ends *Krakatoa* for Java source code, and *Frama-C/Jessie* for C code. However, Java and C programming languages were designed a long time ago, and certainly not with the objective of formal verification in mind. This raises a lot of difficulties when designing specification languages on top of them, and verification condition generators to analyze them. On the other hand, we designed and/or used the *Coq* and *Why3* languages and tools for performing deductive verification, but those were not designed as programming languages that can be compiled into executable programs.

Thus, a new axis of research we propose is the design of an environment that is aimed to both programming and proving, hence that will allow to develop correct-by-construction programs. To achieve this goal, there are two major axes of theoretical research that needs to be conducted, concerning, on the one hand, methods required to support genericity and reusability of verified components, and, on the other hand, the automation of the proof of the verification conditions that will be generated.

3.2.1. Genericity and Reusability of Verified Components

A central ingredient for the success of deductive approaches in program verification is the ability to reuse components that are already proved. This is the only way to scale the deductive approach up to programs of larger size. As for programming languages, a key aspect that allow reusability is *genericity*. In programming languages, genericity typically means parametricity with respect to data types, e.g. *polymorphic types* in functional languages like ML, or *generic classes* in object-oriented languages. Such genericity features are essential for the design of standard libraries of data structures such as search trees, hash tables, etc. or libraries of standard algorithms such as for searching, sorting.

In the context of deductive program verification, designing reusable libraries also requires designing of *generic specifications* which typically involve parametricity not only with respect to data types but also with respect to other program components. For example, a generic component for sorting an array needs to be parametrized by the type of data in the array but also by the comparison function that will be used. This comparison function is thus another program component that is a parameter of the sorting component. For this parametric component, one needs to specify some requirements, at the logical level (such as being a total ordering relation), but also at the program execution level (like being *side-effect free*, i.e. comparing of data should not modify the data). Typically such a specification may require *higher-order* logic.

Another central feature that is needed to design libraries of data structures is the notion of data invariants. For example, for a component providing generic search trees of reasonable efficiency, one would require the trees to remain well-balanced, over all the life time of a program.

This is why the design of reusable verified components requires advanced features, such as *higher-order specifications and programs*, *effect polymorphism* and *specification of data invariants*. Combining such features is considered as an important challenge in the current state of the art (see e.g. [98]). The well-known proposals for solving it include *Separation logic* [121], *implicit dynamic frames* [118], and *considerate reasoning* [120]. Part of our recent research activities were aimed at solving this challenge: first at the level of specifications, e.g. we proposed generic specification constructs upon Java [122] or a system of theory cloning in our system *Why3* [2]; second at the level of programs, which mainly aims at controlling side-effects to avoid unexpected breaking of data invariants, thanks to advanced type checking: approaches based on *memory regions*, *linearity* and *capability-based* type systems [72], [96], [51].

A concrete challenge that should be solved in the future is: what additional constructions should we provide in a specification language like ACSL for C, in order to support modular development of reusable software components? In particular, what would be an adequate notion of module, that would provide a good notion of abstraction, both at the level of program components and at the level of specification components?

3.2.2. Automated Deduction for Program Verification

Verifying that a program meets formal specifications typically amounts to generating *verification conditions* e.g. using a weakest precondition calculus. These verification conditions are purely logical formulas—typically in first-order logic and involving arithmetic in integers or real numbers—that should be checked to be true. This can be done using either automatic provers or interactive proof assistants. Automatic provers do not need user interaction, but may run forever or give no conclusive answer.

There are several important issues to tackle. Of course, the main general objective is to improve automation as much as possible. We continue our efforts around our own automatic prover *Alt-Ergo* towards more expressivity, efficiency, and usability, in the context of program verification. More expressivity means that the prover should better support the various theories that we use for modeling. Toward this direction, we aim at designing specialized proof search strategies in *Alt-Ergo*, directed by rewriting rules, in the spirit of what we did for the theory of associativity and commutativity [7].

A key challenge also lies in the handling of quantifiers. SMT solvers, including *Alt-Ergo*, deal with quantifiers with a somewhat ad-hoc mechanism of heuristic instantiation of quantified hypotheses using the so-called *triggers* that can be given by hand [83], [84]. This is completely different from resolution-based provers of the TPTP category (E-prover, Vampire, etc.) which use unification to apply quantified premises. A challenge is thus to find the best way to combine these two different approaches of quantifiers. Another challenge is to add some support for higher-order functions and predicates in this SMT context, since as said above, reusable verified components will require higher-order specifications. A few solutions have been proposed, essentially based on encoding of higher-order goals into first-order goals [96].

Generally speaking, there are several theories, interesting for program verification, that we would like to add as built-in decision procedures in an SMT context. First, although there already exist decision procedures for variants of bit-vectors, they are not complete enough to support what is needed to reason on programs that manipulate data at the bit-level, in particular if conversions from bit-vectors to integers or floating-point numbers are involved [114]. Regarding floating-point numbers, an important challenge is to integrate in an SMT context a decision procedure like the one implemented in our tool *Gappa*.

Another goal is to improve the feedback given by automatic provers: failed proof attempts should be turned into potential counterexamples, so as to help debugging programs or specifications. A pragmatic goal would be to allow cooperation with other verification techniques. For instance, testing could be performed on unproved goals. Regarding this cooperation objective, an important goal is a deeper integration of automated procedures in interactive proofs, like it already exists in Isabelle [70]. We now have a *Why3* tactic in *Coq* that we plan to improve.

3.2.3. An Environment for Both Programming and Proving

As said before, a new axis of research we follow is the design of a language and an environment for both programming and proving. We believe that this will be a fruitful approach for designing highly trustable software. This is a similar goal as projects Plaid, Trellys, ATS, or Guru, mentioned above.

The basis of this research direction is the *Why3* system, which is in fact a reimplementation from scratch of the former *Why* tool, that we started in January 2011. This new system supports our research at various levels. It is already used as an intermediate language for deductive verification.

The next step for us is to develop its use as a true programming language. Our objective is to propose a language where programs could be both executed (e.g. thanks to a compiler to, say, *OCaml*) and proved correct. The language would basically be purely applicative (i.e. without side-effects, e.g. close to ML) but incorporating specifications in its core. There are, however, some programs (e.g. some clever algorithms) where a bit of imperative programming is desirable. Thus, we want to allow some form of imperative features, but in a very controlled way: it should provide a strict form of imperative programming that is clearly more amenable to proof, in particular dealing with data invariants on complex data structures.

As already said before, reusability is a key issue. Our language should propose some form of modules with interfaces abstracting away implementation details. Our plan is to reuse the known ideas of *data refinement* [110] that was the foundation of the success of the B method. But our language will be less constrained than what is usually the case in such a context, in particular regarding the possibility of sharing data, and the constraints on composition of modules, there will be a need for advanced type systems like those based on regions and permissions.

The development of such a language will be the basis of the new theme regarding the development of certified tools, that is detailed in Section 3.3 below.

3.2.4. Extra Exploratory Axes of Research

Concerning formal verification of programs, there are a few extra exploratory topics that we plan to explore.

Concurrent Programming So far, we only investigated the verification of sequential programs. However, given the spreading of multi-core architectures nowadays, it becomes important to be able to verify concurrent programs. This is known to be a major challenge. We plan to investigate this direction, but in a very careful way. We believe that the verification of concurrent programs should be done only under restrictive conditions on the possible interleaving of processes. In particular, the access and modification of shared data should be constrained by the programming paradigm, to allow reasonable formal specifications. In this matter, the issues are close to the ones about sharing data between components in sequential programs, and there are already some successful approaches like separation logic, dynamic frames, regions, and permissions.

Resource Analysis The deductive verification approaches are not necessarily limited to functional behavior of programs. For example, a formal termination proof typically provides a bound on the time complexity of the execution. Thus, it is potentially possible to verify resources consumption in this way, e.g. we could prove WCET (Worst Case Execution Times) of programs. Nowadays, WCET analysis is typically performed by abstract interpretation, and is applied on programs with particular shape (e.g. no unbounded iteration, no recursion). Applying deductive verification techniques in this context could allow to establish good bounds on WCET for more general cases of programs.

Other Programming Paradigms We are interested in the application of deductive methods in other cases than imperative programming à la C, Java or Ada. Indeed, in the recent years, we applied proof techniques to randomized programs [1], to cryptographic programs [50]. We plan to use proof techniques on applications related to databases. We also have plans to support low-level programs such as assembly code [86], [113] and other unstructured programming paradigm. We are also investigating more and more applications of SMT solving, e.g. in model-checking approach (for example in Cubicle [1] [76]) or abstract interpretation techniques (project Cafein, started in 2013) and also for discharging proof obligations coming from other systems like *Atelier B* [109] (project BWare).

3.3. Certified Tools

One of our goals is to guarantee the soundness of the tools we develop. In fact, it goes beyond that; our goal is to promote our future *Why3* environment so that *others* could develop certified tools. Tools like automated provers or program analyzers are good candidate case studies because they are mainly performing symbolic computations, and as such they are usually programmed in a mostly purely functional style.

We conducted several experiments of development of certified software in the past. First, we have a strong expertise in the development of *libraries* in *Coq*: the Coccinelle library [78] formalizing term rewriting systems, the Alea library [1] for the formalization of randomized algorithms, several libraries formalizing floating-point numbers (Floats [60], Gappalib [107], and now Flocq [6] which unifies the formers). Second we conducted the development of a certified decision procedure [103] that corresponds to a core part of *Alt-Ergo*. Third we developed, still in *Coq*, certified verification condition generators, in a first step [94] for a language similar to *Why*, and in a second step [93] for C annotated in ACSL [56], based on the operational semantics formalized in the CompCert certified compiler project [102].

[1] http://cubicle.lri.fr/

To go further, we have several directions of research in mind.

3.3.1. Formalization of Binders

Using the *Why3* programming language instead of *Coq* allows for more freedom. For example, it should allow one to use a bit of side-effects when the underlying algorithm justifies it (e.g. hash-consing, destructive unification). On the other hand, we will lose some *Coq* features like dependent types that are usually useful when formalizing languages. Among the issues that should be studied, we believe that the question of the formalization of binders is both central and challenging (as exemplified by the POPLmark international challenge [47]).

The support of binders in *Why3* should not be built-in, but should be under the form of a reusable *Why3* library, that should already contain a lot of proved lemmas regarding substitution, alpha-equivalence and such. Of course we plan to build upon the former experiments done for the POPLmark challenge. Although, it is not clear yet that the support of binders only via a library will be satisfactory. We may consider addition of built-in constructs if this shows useful. This could be a form of (restricted) dependent types as in *Coq*, or subset types as in PVS.

3.3.2. Theory Realizations, Certification of Transformations

As an environment for both programming and proving, *Why3* should come with a standard library that includes both verified libraries of programs, but also libraries of specifications (e.g. theories of sets, maps, etc.).

The certification of those *Why3* libraries of specifications should be addressed too. *Why3* libraries for specifying models of programs are commonly expressed using first-order axiomatizations, which have the advantage of being understood by many different provers. However, such style of formalization does not offer strong guarantees of consistency. More generally, the fact that we are calling different kind of provers to discharge our verification conditions raises several challenges for certification: we typically apply various transformations to go from the *Why3* language to those of the provers, and these transformations should be certified too.

A first attempt in considering such an issue was done in earlier work [109]. It was proposed to certify the consistency of a library of specification using a so-called *realization*, which amounts to "implementing" the library in a proof assistant like *Coq*. This is an important topic of the ANR project BWare.

3.3.3. Certified Theorem Proving

The goal is to develop *certified* provers, in the sense that they are proved to give a correct answer. This is an important challenge since there have been a significant amount of soundness bugs discovered in the past, in many tools of this kind.

The former work on the certified core of *Alt-Ergo* [103] should be continued to support more features: more theories (full integer arithmetic, real arithmetic, arrays, etc.), quantifiers. Development of a certified prover that supports quantifiers should build upon the previous topic about binders.

In a similar way, the *Gappa* prover, which is specialized to solving constraints on real numbers and floating-point numbers, should be certified too. However, for very complex decision procedures, developing a certified proof search might be too ambitious. Instead, the idea is to ask *Gappa* to produce Coq proofs on a per-goal basis, so as to check *a posteriori* the soundness of its result on the given instance. More generally, we can have *Gappa* produce traces of its execution that can later be processed by a certified trace checker. This approach was used in the past for certified proofs of termination of rewriting systems [79], and it was also used internally in CompCert for several passes of compilation [102].

3.3.4. Certified VC Generation

The other kind of tools that we would like to certify are the VC generators. This is a continuation of the work on developing in *Coq* a certified VC generator for C code annotated in ACSL [93]. We develop such a generator in *Why3* instead of *Coq* [105]. As before, this builds upon a formalization of binders. There are various kinds of VC generators that are interesting. A generator for a simple language in the style of those of *Why3* is a first

step. Other interesting cases are: a generator implementing the so-called *fast weakest preconditions* [99], and a generator for unstructured programs like assembly, that would operate on an arbitrary control-flow graph.

On a longer term, we wish to be able to certify advanced verification methods like those involving refinement, alias control, regions, permissions, etc.

An interesting question is how one could certify a VC generator that involves a highly expressive logic, like higher-order logic, as it is the case of the *CFML* method [73] which allows one to use the whole *Coq* language to specify the expected behavior. One challenging aspect of such a certification is that a tool that produces *Coq* definitions, including inductive definitions and module definitions, cannot be directly proved correct in *Coq*, because inductive definitions and module definitions cannot be generated through the evaluation of *Coq* definitions. Therefore, it seems necessary to involve, in a way or another, a "deep embedding", that is, a formalization of *Coq* in *Coq*, possibly by reusing the deep embedding developed by B. Barras [53].

3.4. Numerical Programs

In recent years, we demonstrated our capability towards specifying and proving properties of floating-point programs, properties which are both complex and precise about the behavior of those programs: see the publications [67], [123], [62], [117], [66], [61], [108], [106] as well as the web galleries of certified programs at our Web page [2], the Hisseo project [3], S. Boldo's page [4], and industrial case studies in the U3CAT ANR project. The ability to express such complex properties comes from models developed in *Coq* [6]. The ability to combine proof by reasoning and proof by computation is a key aspect when dealing with floating-point programs. Such a modeling provides a safe basis when dealing with C source code [5]. However, the proofs can get difficult even on short programs. To build these proofs, some automation is needed. It can be obtained by combining SMT solvers and *Gappa* [64], [82], [46], [10]. Finally, the precision of the verification is obtained thanks to precise models of floating-point computations, taking into account the peculiarities of the architecture (e.g., x87 80-bit floating-point unit) and also the compiler optimizations [68], [113].

The directions of research concerning floating-point programs that we pursue are the following.

3.4.1. Making Formal Verification of Floating-point Programs Easier

A first goal is to ease the formal verification of floating-point programs: the primary objective is still to improve the scope and efficiency of our methods, so as to ease further the verification of numerical programs. The ongoing development of the Flocq library continues towards the formalization of bit-level manipulations and also of exceptional values (e.g. infinities). We believe that good candidates for applications of our techniques are advanced algorithms to compute efficiently with floats, which operate at the bit-level. The formalization of real numbers is being revamped too: higher-level numerical algorithms are usually built on some mathematical properties (e.g. computable approximations of ideal approximations), which then have to be proved during the formal verification of these algorithms.

Easing the verification of numerical programs also implies more automation. SMT solvers are generic provers well-suited for automatically discharging verification conditions, but they appear to loose their effectiveness when floating-point arithmetic is involved [77]. Our goal is to improve the arithmetic theories of *Alt-Ergo*, so that they support floating-point arithmetic along their other theories, if possible by reusing the heuristics developed for *Gappa*.

3.4.2. Continuous Quantities, Numerical Analysis

Our goal is to handle floating-point programs that are related to continuous quantities. This includes numerical analysis programs we have already worked on [63], [62], [4]. But our work is only a beginning: we were able to solve the difficulties to prove one particular scheme for one particular partial differential equation. We need to be able to easily prove other programs of this kind. This requires new results that handle generic schemes and many partial differential equations. The idea is to design a toolbox to prove these programs with as much

[2] http://toccata.lri.fr/gallery/index.en.html
[3] http://hisseo.saclay.inria.fr/
[4] http://www.lri.fr/~sboldo/research.html

automation as possible. We wish this could be used by numerical analysts that are not or hardly familiar with formal methods, but are nevertheless interested in the formal correctness of their schemes and their programs.

Another very interesting kind of programs (especially for industrial developers) are those based on *hybrid* systems, that is where both discrete and continuous quantities are involved. This is a longer-term goal, but we may try to go towards this direction. A first problem is to be able to specify hybrid systems: what are they exactly expected to do? Correctness usually means not going into a forbidden state but we may want additional behavioral properties. A second problem is the interface with continuous systems, such as sensors. How can we describe their behavior? Can we be sure that the formal specification fits? We may think about Ariane V where one piece of code was shamelessly reused from Ariane IV. Ensuring that such a reuse is allowed requires to correctly specify the input ranges and bandwidths of physical sensors.

Studying hybrid systems is among the goals of the new ANR project Cafein.

3.4.3. Certification of Floating-point Analyses

In coordination with our second theme, another objective is to port the kernel of *Gappa* into either *Coq* or *Why3*, and then extract a certified executable. Rather than verifying the results of the tool *a posteriori* with a proof checker, they would then be certified *a priori*. This would simplify the inner workings of *Gappa*, help to support new features (e.g. linear arithmetic, elementary functions), and make it scale better to larger formulas, since the tool would no longer need to carry certificates along its computations. Overall the tool would then be able to tackle a wider range of verification conditions.

An ultimate goal would be to develop the decision procedure for floating-point computations, for SMT context, that is mentioned in Section 3.2.2, directly as a certified program in *Coq* or *Why3*.

4. Application Domains

4.1. Mission-Critical Software

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. The domains of application include the following. For each of them we refer to our past or current actions, in particular in relations with projects, contracts and industrial partners. Currently our industrial collaborations mainly belong to the first of these domains, transportation.

- **Transportation** including aeronautics, railroad, space flight, automotive.

  These domains were 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, the use of the *Alt-Ergo* prover via CAVEAT tool (CEA) or *Frama-C*/WP. This action is continued in the new project Soprano.

  Aeronautics is the main target of the Verasco project, leaded by Verimag, on the development of certified static analyzers, in partnership with Airbus.

  The former FUI project Hi-Lite, led by Adacore company, uses *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). This action is continued in the new joint laboratory ProofInUse. A recent paper [71] provides an extensive list of applications of SPARK, a major one being the British air control management.

  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 (http://www.methode-b.com/documentation_b/ClearSy-Industrial_Use_of_B.pdf), a collaboration with Mitsubishi Electric R&D Centre Europe (Rennes) and ClearSy (Aix-en-Provence).

- **Energy** is naturally an application in particular with our long-term partner CEA, in the context of U3CAT and Soprano projects.
- **Communications and Data** in particular in contexts with a particular need for security or confidentiality: smart phones, Web applications, health records, electronic voting, etc.

  Part of the applications of SPARK [71] include verification of security-related properties, including cryptographic algorithms.

  Our new AJACS project addresses issues related to security and privacy in web applications written in Javascript, also including correctness properties.

  The Cubicle model checker modulo theories based on the *Alt-Ergo* SMT prover, in collaboration with Intel Strategic Cad Labs (Hillsboro, OR, USA) is particularly targeted to the verification of concurrent programs and protocols (http://cubicle.lri.fr/).
- **Medicine**, including diagnostic devices, computer-assisted surgery

  Such applications involve techniques for control and command close to what is done in transportation. Moreover, in this context, there is a need for modeling using differential equations, finite elements, hybrid systems, which are considered in other projects of us: FastRelax, ELFIC, Cafein.
- **Financial applications, banking**

  We add projects in the past about safety and security of smart cards, in collaboration with Gemalto (European project VerifiCard, two CIFRE theses). Banking is naturally a domain of application of techniques dealing with security and confidentiality already mentioned above.

5. New Software and Platforms

5.1. The Why3 system

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

Criteria for Software Self-Assessment: A-3-up, SO-4, SM-4, EM-4, SDL-5, OC-4.[5]

Why3 is the next generation of *Why*. *Why3* clearly separates the purely logical specification part from generation of verification conditions for programs. It features a rich library of proof task transformations that can be chained to produce a suitable input for a large set of theorem provers, including SMT solvers, TPTP provers, as well as interactive proof assistants.

It is distributed as open source, under GPL license, at http://why3.lri.fr/. It is also distributed as part of major Linux distributions and in the OPAM packaging system http://opam.ocaml.org/packages/why3/why3.0.85/.

Why3 is used as back-end of our own tools *Krakatoa* and *Jessie*, but also as back-end of the GNATprove tool (Adacore company), and of the WP plugin of *Frama-C*. *Why3* has been used to develop and prove a significant part of the programs of our team gallery http://proval.lri.fr/gallery/index.en.html, and used for teaching (e.g., at the Master Parisien de Recherche en Informatique).

Why3 is used by other academic research groups, e.g. within the CertiCrypt/EasyCrypt project (http://easycrypt.gforge.inria.fr/) for certifying cryptographic programs. The *Why3* web site http://why3.lri.fr lists a few other works done by external researchers and relying on the use of *Why3*.

Two versions were released in 2014: 0.83 released in March and 0.84 in September, plus a few days later a bugfix version 0.85.

[5] self-evaluation following the guidelines (http://www.inria.fr/content/download/11783/409665/version/4/file/SoftwareCriteria-V2-CE.pdf) of the Software Working Group of Inria Evaluation Committee (http://www.inria.fr/institut/organisation/instances/commission-d-evaluation)

5.2. The Alt-Ergo theorem prover

Participants: Sylvain Conchon [contact], Évelyne Contejean, Alain Mebsout, Mohamed Iguernelala.

Criteria for Software Self-Assessment: A-3-up, SO-4, SM-4-up, EM-4, SDL-5, OC-4.

Alt-Ergo is an automated proof engine, dedicated to program verification, whose development started in 2006. It is fully integrated in the program verification tool chain developed in our team. It solves goals that are directly written ina *Why*'s annotation language; this means that *Alt-Ergo* fully supports first order polymorphic logic with quantifiers. *Alt-Ergo* also supports the standard [116] defined by the SMT-lib initiative.

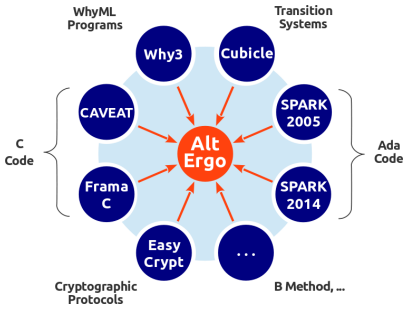

Figure 2.

It is currently used in our team to prove correctness of C and Java programs as part of the *Why* platform and the new *Why3* system. It is used as back-end prover in the environments Frama-C and CAVEAT for static analysis of C developed at CEA. In this context, Alt-Ergo has been qualified by Airbus and is integrated in the next generation of Airbus development process. Alt-Ergo is usable as a back-end prover in the SPARK verifier for ADA programs, since Oct 2010, and is also the main back-end prover of the new SPARK2014.

Alt-Ergo is integrated in several other tools and platforms: the Bware platform for discharging VCs generated by Atelier B, the EasyCrypt environment for verifying cryptographic protocols, the Pangolin programming language http://code.google.com/p/pangolin-programming-language/, etc.

Last but not least, Alt-Ergo is the solver used by the Cubicle model checker described below.

Alt-Ergo is distributed as open source, under the CeCILL-C license, at URL http://alt-ergo.lri.fr/, and in the OPAM packaging system http://opam.ocaml.org/packages/alt-ergo/alt-ergo.0.95.2/. Latest public version is 0.99.1, released in Dec. 2014. Maintenance is done by the OcamlPro company http://alt-ergo.ocamlpro.com/.

5.3. The Cubicle model checker modulo theories

Participants: Sylvain Conchon [contact], Alain Mebsout.

Partners: A. Goel, S. Krstić (Intel Strategic Cad Labs in Hillsboro, OR, USA), F. Zaïdi (LRI, Université Paris-sud)

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.

Cubicle model-checks by a symbolic backward-reachability analysis on infinite sets of states represented by specific simple formulas, called cubes. Cubicle is based on ideas introduced by MCMT (http://users.mat.unimi.it/users/ghilardi/mcmt/) from which, in addition to revealing the implementation details, it differs in a more friendly input language and a concurrent architecture. Cubicle is written in OCaml. Its SMT solver is a tightly integrated, lightweight and enhanced version of *Alt-Ergo*; and its parallel implementation relies on the Functory library.

Cubicle is distributed as open source, under the Apache license, at URL http://cubicle.lri.fr/, and in the OPAM packaging system http://opam.ocaml.org/packages/cubicle/cubicle.1.0.1/. Latest version is 1.0.1, released in Nov. 2014.

5.4. The Flocq library

Participants: Sylvie Boldo [contact], Guillaume Melquiond.

Criteria for Software Self-Assessment: A-2, SO-3, SM-3, EM-3, SDL-5, OC-4.

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 [6]. It provides a framework for developers to formally certify numerical applications.

Flocq is currently used by the CompCert certified compiler for its support of floating-point computations.

It is distributed as open source, under a LGPL license, at http://flocq.gforge.inria.fr/. It was first released in 2010. Current version is 2.4.0 released in Sep. 2014.

5.5. The Gappa tool

Participant: Guillaume Melquiond [contact].

Criteria for Software Self-Assessment: A-3, SO-4, SM-4, EM-3, SDL-5, OC-4.

Given a logical property involving interval enclosures of mathematical expressions, Gappa tries to verify this property and generates a formal proof of its validity. This formal proof can be machine-checked by an independent tool like the *Coq* proof-checker, so as to reach a high level of confidence in the certification [82], [123].

Since these mathematical expressions can contain rounding operators in addition to usual arithmetic operators, Gappa is especially well suited to prove properties that arise when certifying a numerical application, be it floating-point or fixed-point. Gappa makes it easy to compute ranges of variables and bounds on absolute or relative roundoff errors.

Gappa is being used to certify parts of the mathematical libraries of several projects, including CRlibm, FLIP, and CGAL. It is distributed as open source, under a Cecill-B / GPL dual-license, at http://gappa.gforge.inria.fr/. Latest version is 1.1.2 released in October 2014.

Part of the work on this tool was done while in the Arénaire team (Inria Rhône-Alpes), until 2008.

5.6. The Interval package for Coq

Participants: Guillaume Melquiond [contact], Érik Martin-Dorel.

Criteria for Software Self-Assessment: A-3, SO-4, SM-3, EM-3, SDL-4, OC-4.

The Interval package provides several tactics for helping a *Coq* user to prove 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*.

Versions 1.0 and 2.0 were released in 2014. Version 2.0 integrates the CoqApprox library for computing Taylor models, so as to greatly improve performances when bounding univariate expressions [43].

It is distributed as open source, under a CeCILL-C license, at http://coq-interval.gforge.inria.fr/. Latest version is 2.0 released in November 2014.

Part of the work on this library was done while in the Mathematical Components team (Microsoft Research–Inria Joint Research Center).

5.7. The Coquelicot library for real analysis

Participants: Sylvie Boldo [contact], Catherine Lelay, Guillaume Melquiond.

Criteria for Software Self-Assessment: A-3, SO-4, SM-2, EM-3, SDL-4, OC-4.

The Coquelicot library 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. Moreover, we achieved this compatibility without adding any additional axiom.

The result is the Coquelicot library available at http://coquelicot.saclay.inria.fr. Latest version is 2.0.1 released in March 2014. It contains about 1,700 theorems and 37,000 lines of Coq.

5.8. The CFML tool for verifying OCaml code

Participant: Arthur Charguéraud [contact].

Criteria for Software Self-Assessment: A-2, SO-4, SM-2, EM-3, SDL-1, OC-4.

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 notation and tactics for manipulating characteristic formulae interactively in *Coq*.

CFML is distributed under the LGPL license, and is available at http://arthur.chargueraud.org/softs/cfml/. It has been continuously extended since its first release in 2010. In particular, in 2014 support for the verification of asymptotic complexity bounds has been added.

5.9. Other Maintained Tools

5.9.1. The ALEA library for randomized algorithms

Participant: Christine Paulin-Mohring [contact].

Criteria for Software Self-Assessment: A-2, SO-3, SM-2, EM-3, SDL-4, OC-4.

The ALEA library is a *Coq* development for modeling randomized functional programs as distributions using a monadic transformation. It contains an axiomatisation of the real interval $[0, 1]$ and its extension to positive real numbers. It introduces definition of distributions and general rules for approximating the probability that a program satisfies a given property.

ALEA is used as a basis of the Certicrypt environment (MSR-Inria joint research center, Imdea Madrid, Inria Sophia-Antipolis) for formal proofs for computational cryptography [55]. It is also experimented in LABRI as a basis to study formal proofs of probabilistic distributed algorithms.

ALEA is distributed as open source, at http://www.lri.fr/~paulin/ALEA. Latest version is 8 released in May 2013. In particular, it includes a module to reason about random variables with values in positive real numbers.

5.9.2. Bibtex2html

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

Criteria for Software Self-Assessment: A-5, SO-3, SM-3, EM-3, SDL-5, OC-4.

Bibtex2html is a generator of HTML pages of bibliographic references. Distributed as open source since 1997, under the GPL license, at http://www.lri.fr/~filliatr/bibtex2html/. Latest version is 1.98 released in July 2014. Bibtex2html is also distributed as a package in most Linux distributions, and in the OPAM packaging system http://opam.ocaml.org/packages/bibtex2html/bibtex2html.1.98/.

We estimate that between 10000 and 100000 web pages have been generated using Bibtex2html.

5.9.3. The Coccinelle library for term rewriting

Participant: Évelyne Contejean [contact].

Criteria for Software Self-Assessment: A-2, SO-3, SM-2, EM-2, SLD-2, OC-4.

Coccinelle is a *Coq* library for term rewriting. Besides the usual definitions and theorems of term algebras, term rewriting and term ordering, it also models a number of algorithms implemented in the CiME toolbox, such as matching, matching modulo associativity-commutativity, computation of the one-step reducts of a term, recursive path ordering (RPO) comparison between two terms, etc. The RPO algorithm can effectively be run inside *Coq*, and is used in the Color developement (http://color.inria.fr/) as well as for certifying Spike implicit induction theorems in *Coq* (Sorin Stratulat).

Coccinelle is available at http://www.lri.fr/~contejea/Coccinelle, and is distributed under the Cecill-C license.

5.9.4. OCamlgraph

Participants: Jean-Christophe Filliâtre [contact], Sylvain Conchon.

OCamlgraph is a graph library for *OCaml*. It features many graph data structures, together with many graph algorithms. Data structures and algorithms are provided independently of each other, thanks to *OCaml* module system. OCamlgraph is distributed as open source, under the LGPL license, at http://OCamlgraph.lri.fr/. Latest version is 1.8.5, released in March 2014. It is also distributed as a package in several Linux distributions. OCamlgraph is now widely spread among the community of *OCaml* developers, and available as an OPAM package http://opam.ocaml.org/packages/ocamlgraph/ocamlgraph.1.8.5/.

5.9.5. Mlpost

Participant: Jean-Christophe Filliâtre [contact].

Mlpost is a tool to draw scientific figures to be integrated in LaTeX documents. Contrary to other tools such as TikZ or MetaPost, it does not introduce a new programming language; it is instead designed as a library of an existing programming language, namely *OCaml*. Yet it is based on MetaPost internally and thus provides high-quality PostScript figures and powerful features such as intersection points or clipping. Mlpost is distributed as open source, under the LGPL license, at http://mlpost.lri.fr/. Mlpost was presented at JFLA'09 [52].

Mlpost is available as an OPAM package http://opam.ocaml.org/packages/mlpost/mlpost.0.8.1/.

5.9.6. Functory

Participant: Jean-Christophe Filliâtre [contact].

Functory is a distributed computing library for *OCaml*. The main features of this library include (1) a polymorphic API, (2) several implementations to adapt to different deployment scenarios such as sequential, multi-core or network, and (3) a reliable fault-tolerance mechanism. Functory was presented at JFLA 2011 [91] and at TFP 2011 [90].

Functory is distributed as open source, under the LGPL license, at http://functory.lri.fr/, and in the OPAM packaging system http://opam.ocaml.org/packages/functory/functory.0.5/. Latest version is 0.5, release in March 2013.

5.9.7. The Why Environment

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

Criteria for Software Self-Assessment: A-3, SO-4, SM-3, EM-2, SDL-5-down, OC-4.

The *Why* platform is a set of tools for deductive verification of Java and C source code. In both cases, the requirements are specified as annotations in the source, in a special style of comments. For Java (and Java Card), these specifications are given in JML and are interpreted by the *Krakatoa* tool. Analysis of C code must be done using the external *Frama-C* environment, and its *Jessie* plugin which is distributed in *Why*.

The platform is distributed as open source, under GPL license, at http://why.lri.fr/.

It also distributed as part of major Linux distributions and in the OPAM packaging system http://opam.ocaml.org/packages/why/why.2.34/. Version 2.34 was released in August 2014, to provide a version compatible with both *Frama-C*Neon and *Why3* 0.83.

The internal VC generator and the translators to external provers are no longer under active development, as superseded by the *Why3*system described above. The *Krakatoa* and *Jessie* front-ends are still maintained, although using now by default the *Why3* VC generator. These front-ends are described in a specific web page http://krakatoa.lri.fr/. They are used for teaching (University of Evry, École Polytechnique, etc.), used by several research groups in the world, e.g at Fraunhofer Institute in Berlin [92], at Universidade do Minho in Portugal [50], at Moscow State University, Russia (http://journal.ub.tu-berlin.de/eceasst/article/view/255).

6. New Results

6.1. Highlights of the Year

- The ACM Software System Award 2013 was given, during a ceremony in June 2014 in San Francisco, to the Coq proof assistant (http://awards.acm.org/software_system/). The prestigious ACM price was previously awarded to the LLVM compiler infrastructure (2012) and to the Eclipse IDE (2011). Among the 9 recipients of the 2013 award are Christine Paulin and Jean-Christophe Filliâtre, from the Toccata team.
- The *Concours Castor informatique* (http://castor-informatique.fr/) had an even larger success than in the previous years. In November 2014, more than 228,000 teenagers from over 1500 schools participated and solved the interactive tasks of the contest. Arthur Charguéraud and Sylvie Boldo, from the Toccata team, significantly contributed to the prepation of the tasks and to the organization of the contest.

6.2. Deductive Verification

- J.-C. Filliâtre, L. Gondelman, and A. Paskevich have formalized the notion of ghost code implemented in *Why3*, in a paper *The Spirit of Ghost Code* [35] presented at CAV 2014. This is an outcome of L. Gondelman's M2 internship (spring/summer 2013).
- M. Clochard published at the POPL conference a paper presenting a work done during an internship at Rice University (Houston, TX, USA) with S. Chaudhuri and A. Solar-Lezama [29]. It is a new technique for parameter synthesis under boolean and quantitative objectives. The input to the technique is a "sketch"—a program with missing numerical parameters—and a probabilistic assumption about the program's inputs. The goal is to automatically synthesize values for the parameters such that the resulting program satisfies: (1) a boolean specification, which states that the program must meet certain assertions, and (2) a quantitative specification, which assigns a real valued rating to every program and which the synthesizer is expected to optimize.

- J.-C. Filliâtre, C. Marché, and A. Paskevich, together with F. Bobot (CEA LIST), took part in the VerifyThis program verification competition, held at the 18th FM symposium in August 2012. They used *Why3* to solve three challenges (which can be found at http://fm2012.verifythis.org/challenges/), and their solutions have been published in a special issue of the journal *Software Tools for Technology Transfer* [16].
- M. Clochard developed, using *Why3*, verified implementations of several data structures, including random-access lists and ordered maps. These are derived from a common parametric implementation of self-balancing binary trees in the style of Adelson-Velskii and Landis trees (so-called AVLs). This work appeared at the VSTTE conference [30]. Its originality relies on the genericity of the specifications and the code, and the very high level of proof automation. Such a case study is aimed at illustrating the capabilities of *Why3* for designing certified libraries. Development is available from our gallery at http://toccata.lri.fr/gallery/avl.fr.html.
- S. Conchon and A. Mebsout have extended the core algorithm of the Cubicle model checker 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. Details can be found in A. Mebsout's PhD thesis [15].
- A. Charguéraud extended his tool CFML to support, in addition to the verification of the full functional correctness of a piece of code, the verification of the asymptotic complexity of the code. Even though it had been previously established that, in theory, amortized analysis can be explained as the manipulation of *time credits*, and that time credits can be encoded as resources in Separation Logic, CFML is the first practical tool to support the formal verification of amortized analyses for arbitrarily-complex pieces of code. The *time-credit* extension to CFML was put to practice to verify dynamic arrays (Julien Grangier's internship), and to verify a *chunked sequence* data structure [26], particularly challenging due to its use of Tarjan's data structural bootstrapping technique. The latter piece of work was presented in July at the workshop *Semantics of proofs and certified mathematics*, which took place at the Institut Henri Pointcaré. A paper describing the time-credit extension to CFML is under preparation.

6.3. Floating-Point and Numerical Programs

- C. Marché published in the *Science of Computer Programming* journal [22] a detailed description of an industrial research initially conducted in the context of the U3CAT project (ended in 2012) on static analysis of critical C code. The code involves floating-point computations on quaternions that should be of norm 1. Because of the round-off errors, a drift of this norm is observed over time. In this work a bound on this drift is determined and formally proved correct, using *Frama-C*, *Jessie* and *Why3*. Proofs are done using automated provers and in a few complex cases the Coq proof assistant. The published version is up to date with the recent versions of those tools, and the development is available on our gallery at http://toccata.lri.fr/gallery/quat.en.html
- S. Boldo, C. Lelay, and G. Melquiond worked on the Coquelicot library, designed to be a user-friendly Coq library about real analysis. 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: in an exam at university entry level, for the definitions and properties of Bessel functions, and for the solution of the one-dimensional wave equation. These results are published in the journal *Mathematics in Computer Science* [19].

- S. Boldo and G. Melquiond, with J.-H. Jourdan and X. Leroy (Gallium team, Inria Paris - Rocquencourt) extended the CompCert compiler to get the first formally verified C compiler that provably preserves the semantics of floating-point programs This work, published in the *Journal of Automated Reasoning* [18], also covers the formalization of numerous algorithms of conversion between integers and floating-point numbers.
- S. Boldo, C. Lelay, and G. Melquiond, have conducted a survey on the formalization of real arithmetic and real analysis in various proof systems. This work, published in the journal *Mathematical Structures in Computer Science* [20], details the axioms, definitions, theorems, and methods of automation, available in these systems.
- É. Martin-Dorel and G. Melquiond worked on integrating the CoqInterval and CoqApprox libraries into a single package. The CoqApprox library is dedicated to computing verified Taylor models of univariate functions so as to compute approximation errors. The CoqInterval library reuses this work to automatically prove bounds on real-valued expressions. A large formalization effort took place during this work, so as to get rid of all the holes remaining in the formal proofs of CoqInterval. It was also the chance to perform a comparison between numerous decision procedures dedicated to proving nonlinear inequalities involving elementary functions. A report is available [43].
- 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 a numerical analysis program: the second-order centered finite-difference scheme for the one-dimensional acoustic wave. This proof was published with a focus towards numerical analysts, in the journal *Computers and Mathematics with Applications* [17].
- P. Roux formalized the influence of double rounding on the accuracy of floating-point arithmetic operators. In particular, this includes all the corner cases that were ignored from Figueroa's original pen-and-paper proof. Results appeared in the *Journal of Formalized Reasoning* [24].
- P. Roux formalized a theory of numerical analysis for bounding the round-off errors of a floating-point algorithm. This approach was applied to the formal verification of a program for checking that a matrix is semi-definite positive. The challenge here is that testing semi-definiteness involves algebraic number computations, yet it needs to be implemented using only approximate floating-point operations. A report is available [45].

6.4. Automated Reasoning

- 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. 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 [38] and at the international conference on Abstract State Machines, Alloy, B, VDM, and Z [34].

  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 [33].
- C. Dross defended her PhD thesis in April 2014 [14], on the topic of automated reasoning modulo theories, and in particular the handling of quantifiers in the SMT approach. 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.

6.5. Certification of Languages, Tools and Systems

- M. Clochard, C. Marché, and A. Paskevich 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. This work was presented at the PLPV conference in January 2014 [32].
- M. Clochard, J.-C. Filliâtre, C. Marché, and A. Paskevich developed a case study on the formalization of semantics of programming languages using *Why3*. This case study aimed 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*. This works was presented at the VSTTE conference [31].
- M. Clochard and L. Gondelman developed a formalization of a simple compiler in *Why3*. 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. This work conducted in 2014 will be presented at the JFLA conference in January 2015 [75].
- S. Dumbrava and É. Contejean, with V. Benzaken (VALS team, at LRI) proposed a *Coq* formalization of the relational data model which underlies relational database systems. More precisely, they have presented and formalized the data definition part of the model including integrity constraints. They have modelled two different query language formalisms: relational algebra and conjunctive queries. They also present logical query optimization and prove the main "database theorems": algebraic equivalences, the homomorphism theorem and conjunctive query minimization. This work has been published at ESOP 2014 [27].
- A. Charguéraud, together with the other members of the *JsCert* team have developed this year the first complete formalization of the semantics of the JavaScript programming language. This project is joint work with Philippa Gardner, Sergio Maffeis, Gareth Smith, Daniele Filaretti and Daiva Naudziuniene from Imperial College, and Alan Schmitt and Martin Bodin from Inria Rennes (see http://jscert.org). The formalization consists of a set of inductive rules translating the prose from the *ECMAScript Language Specification, version 5*, using the pretty-big-step semantics [74]. These rules can be used to formally reason about program behaviors or to establish the correctness of program transformations. In addition to the inductive rules, a reference interpreter has been proved correct. This interpreter may be used to run actual JavaScript program following the rules of the formal semantics. It has been used in particular to validate the formal semantics against official JavaScript test suites. The formalization of JavaScript has been published at POPL [28].

6.6. Miscellaneous

A. Charguéraud worked together with Umut Acar and Mike Rainey, as part of the ERC project *DeepSea*, on the developement of efficient data structures and algorithms targeting modern, shared memory multicore architectures. Two major results were obtained this year.

The first result is a sequence data structure that provides amortized constant-time access at the two ends, and logarithmic time concatenation and splitting at arbitrary positions. These operations are essential for programming efficient computation in the fork-join model. Compared with prior

work, this novel sequence data structure achieves excellent constant factors, allowing it to be used as a replacement for traditional, non-splittable sequence data structures. This data structure, called *chunked sequence* due to its use of chunks (fixed-capacity arrays), has been implemented both in C++ and in OCaml. It is described in a paper published at ESA [26].

Another result by A. Charguéraud and his co-authors is the development of fast and robust parallel graph traversal algorithms, more precisely for parallel BFS and parallel DFS. The new algorithms leverage the aforementioned sequence data structure for representing the set of edges remaining to be visited. In particular, it uses the split operation for balancing the edges among the several processors involved in the computation. Compared with prior work, these new algorithms are designed to be efficient not just for particular classes of graphs, but for all input graphs. This work has not yet been published, however it is described in details in a technical report [40]. Note that these two graph algorithms, which involve nontrivial use of concurrent data structures, will be very interesting targets for formal verification.

7. Bilateral Contracts and Grants with Industry

7.1. Bilateral Contracts with Industry

7.1.1. 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.

7.1.2. CIFRE contract with Adacore

Participants: Claude Marché [contact], Andrei Paskevich, Claire Dross.

Jointly with the thesis of C. Dross, supervised in collaboration with the AdaCore company, we established a 3-year bilateral collaboration contract, that ended in April 2014.

The aim was to strengthen the usability of the *Alt-Ergo* theorem prover in the context of the GnatProve environment for the verification of safety-critical Ada programs [84]. A focus was made on programs involving Ada containers [85]. C. Dross defended her PhD in April 1st 2014 [14].

7.2. Bilateral Grants with Industry

7.2.1. Intel Grant

Participants: Sylvain Conchon [contact], Alain Mebsout.

S. Conchon has obtained an academic grant by Intel corporation on the development of the Cubicle model checker, for 2 years starting from Dec. 2012 The goal of this project is to develop a new version of Cubicle with significantly improved model-checking power. This required innovative algorithmic enhancements to be implemented and evaluated.

Partner: Intel Strategic Cad Labs in Hillsboro, OR, USA

8. Partnerships and Cooperations

8.1. Regional Initiatives

8.1.1. Coquelicot

Participants: Sylvie Boldo [contact], Catherine Lelay, Guillaume Melquiond.

Coquelicot is a 3-year Digiteo project that started in September 2011. http://coquelicot.saclay.inria.fr/. S. Boldo is the principal investigator of this project.

The Coquelicot project aims at creating a modern formalization of the real numbers in *Coq*, with a focus on practicality [101], [65], [100],[19]. This is sorely needed to ease the verification of numerical applications, especially those involving advanced mathematics.

Partners: team SpecFun from LIX (Palaiseau), University Paris 13

8.1.2. ELFIC

Participants: Sylvie Boldo [contact], Claude Marché, Guillaume Melquiond.

ELFIC is a working group of the Digicosme Labex. S. Boldo is the principal investigator.

Project ELFIC focuses on proving the correctness of the FELiScE (Finite Elements for Life Sciences and Engineering) C++ library which implements the finite element method for approximating solutions to partial differential equations. Finite elements are at the core of numerous simulation programs used in industry. The formal verification of this library will greatly increase confidence in all the programs that rely on it. Verification methods developed in this project will be a breakthrough for the finite element method, but more generally for the reliability of critical software relying on intricate numerical algorithms.

Partners: Inria team Pomdapi; Ecole Polytechnique, LIX; CEA LIST; Université Paris 13, LIPN; UTC, LMAC (Compiègne).

8.2. National Initiatives

8.2.1. ANR Ajacs

Participant: Arthur Charguéraud [contact].

The AJACS research project is funded by the programme "Société de l'information et de la communication" of the ANR, for a period of 42 months, starting on October 1st, 2014.

The goal of the AJACS project is to provide strong security and privacy guarantees on the client side for web application scripts implemented in JavaScript, the most widely used language for the Web. The proposal is to prove correct analyses for JavaScript programs, in particular information flow analyses that guarantee no secret information is leaked to malicious parties. The definition of sub-languages of JavaScript, with certified compilation techniques targeting them, will allow deriving more precise analyses. Another aspect of the proposal is the design and certification of security and privacy enforcement mechanisms for web applications, including the APIs used to program real-world applications. On the Toccata side, the focus will be on the formalization of secure subsets of JavaScript, and on the mechanization of proofs of translations from high-level languages into JavaScript.

Partners: team Celtique (Inria Rennes - Bretagne Atlantique), team Prosecco (Inria Paris - Rocquencourt), team Indes (Inria Sophia Antipolis - Méditerranée), and Imperial College (London).

8.2.2. 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).

8.2.3. ANR Soprano

Participants: Sylvain Conchon [contact], Évelyne Contejean, 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.

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.

8.2.4. ANR CAFEIN

Participant: Sylvain Conchon [contact].

The CAFEIN research project is funded by the programme “Ingénierie Numérique & Sécurité” of the ANR, for a period of 3 years, starting on February 1st, 2013. https://cavale.enseeiht.fr/CAFEIN/.

This project addresses the formal verification of functional properties at specification level, for safety critical reactive systems. In particular, we focus on command and control systems interacting with a physical environment, specified using the synchronous language Lustre.

A first goal of the project is to improve the level of automation of formal verification, by adapting and combining existing verification techniques such as SMT-based temporal induction, and abstract interpretation for invariant discovery. A second goal is to study how knowledge of the mathematical theory of hybrid command and control systems can help the analysis at the controller’s specification level. Third, the project addresses the issue of implementing real valued specifications in Lustre using floating-point arithmetic.

Partners: ONERA, CEA List, ENSTA, teams Maxplus (Inria Saclay - Île-de-France), team Parkas (Inria Paris - Rocquencourt), Perpignan University, Prover Technology, Rockwell Collins.

8.2.5. ANR BWare

Participants: Sylvain Conchon [contact], Évelyne Contejean, Jean-Christophe Filliâtre, Andrei Paskevich, Claude Marché.

The BWare research project is funded by the programme “Ingénierie Numérique & Sécurité” of the ANR, a period of 4 years, starting on September 1st, 2012. http://bware.lri.fr.

BWare is an industrial research project that aims to provide a mechanized framework to support the automated verification of proof obligations coming from the development of industrial applications using the B method and requiring high guarantee of confidence. The methodology used in this project consists in building a generic platform of verification relying on different theorem provers, such as first-order provers and SMT solvers. The variety of these theorem provers aims at allowing a wide panel of proof obligations to be automatically verified by the platform. The major part of the verification tools used in BWare have already been involved in some experiments, which have consisted in verifying proof obligations or proof rules coming from industrial applications [109]. This therefore should be a driving factor to reduce the risks of the project, which can then focus on the design of several extensions of the verification tools to deal with a larger amount of proof obligations.

The partners are: Cedric laboratory at CNAM (CPR Team, project leader); teams Gallium and Deducteam (Inria Paris - Rocquencourt) ; Mitsubishi Electric R&D Centre Europe, ClearSy (the company which develops and maintains *Atelier B*), and the start-up OCamlPro.

8.2.6. ANR Verasco

Participants: Guillaume Melquiond [contact], Sylvie Boldo, Arthur Charguéraud, Claude Marché.

The Versaco research project is funded by the programme “Ingénierie Numérique & Sécurité” of the ANR, for a period of 4 years, starting on January 1st, 2012. Project website: http://verasco.imag.fr.

The main goal of the project is to investigate the formal verification of static analyzers and of compilers, two families of tools that play a crucial role in the development and validation of critical embedded software. More precisely, the project aims at developing a generic static analyzer based on abstract interpretation for the C language, along with a number of advanced abstract domains and domain combination operators, and prove the soundness of this analyzer using the *Coq* proof assistant. Likewise, it will keep working on the CompCert C formally-verified compiler, the first realistic C compiler that has been mechanically proved to be free of miscompilation, and carry it to the point where it could be used in the critical software industry.

Partners: teams Gallium and Abstraction (Inria Paris - Rocquencourt), Airbus avionics and simulation (Toulouse), IRISA (Rennes), Verimag (Grenoble).

8.3. European Initiatives

8.3.1. FP7 & H2020 Projects

Project acronym: ERC Deepsea

Project title: Parallel dynamic computations

Duration: Jun. 2013 - Jun. 2018

Coordinator: Umut A. Acar

Other partners: Carnegie Mellon University

Abstract:

The objective of this project is to develop abstractions, algorithms and languages for parallelism and dynamic parallelism with applications to problems on large data sets. Umut A. Acar (affiliated to Carnegie Mellon University and Inria Paris - Rocquencourt) is the principal investigator of this ERC-funded project. The other main researchers involved are Mike Rainey (Inria, Gallium team), who is full-time on the project, and Arthur Charguéraud (Inria, Toccata team), who works 40% of his time to the project. Project website: http://deepsea.inria.fr/.

8.4. International Initiatives

8.4.1. Inria International Partners

8.4.1.1. Declared Inria International Partners

S. Conchon, A. Mebsout and F. Zaïdi (VALS group, LRI) collaborate with S. Krstic and A. Goel (Intel Strategic Cad Labs in Hillsboro, OR, USA), in particular around the development of the SMT-based model checker Cubicle (see above). This collaboration is partly supported by an academic grant by Intel.

8.5. International Research Visitors

8.5.1. Visits of International Scientists

- P. Roux (ISAE, Onera) visited for 7 months in order to collaborate with S. Boldo and G. Melquiond on the topic of formal verification of numerical algorithms.
- Bas Spitters visited for 3 months from April to June funded by a Digiteo grant. He worked with C. Paulin on the extension of the ALEA library to continuous structures and the use of "lower reals" (monotonic sequences of rationals). He also worked on adapting the Corn and Math-classes libraries to the new Coq release. During that time he published a final version of a paper presented at the Workshop on Quantum Physics and Logic in 2012 [119].
- Andrew Tolmach is a visiting researcher from Portland State University, on a one-year Digiteo Chair. His research project will initiate a new research effort to develop principles, techniques, and tools for large-scale proof engineering. It is focused on the Coq proof assistant and is designed to take advantage of the deep pool of expertise available in the Paris area (at Paris-Sud, LIX, Inria, etc.) concerning both the use and development of Coq. Initial results are expected to include: a precise description of requirements for large proof management; sample prototype tools addressing one or more of these requirements; and a technical survey of relevant proof representation options.

9. Dissemination

9.1. Promoting Scientific Activities

9.1.1. Scientific events organisation

9.1.1.1. General chair, scientific chair

- J.-C. Filliâtre chaired the fifth meeting of the IFIP Working Group 1.9/2.15 (Verified Software) in Vienna, Austria, July 2014.

9.1.1.2. Member of the organizing committee

- S. Boldo and G. Melquiond are members of the organizing committee for the 22nd IEEE Symposium on Computer Arithmetic (ARITH 2015), held in Lyon in June 2015.
- J.-C. Filliâtre organized the sixth meeting of the IFIP Working Group 1.9/2.15 (Verified Software) in Paris, France, December 2014.
- C. Paulin co-organized with Z. Shao (Harvard University) the *Workshop on certification of high-level and low-level programs*, July 7-11, 2014. This workshop was part of the IHP thematic trimester *Semantics of proofs and certified mathematics*, which took place in Paris from April to July.

9.1.2. Scientific events selection

9.1.2.1. Member of the conference program committee

- S. Boldo is a member of the program committee of the 7th International Workshop on Numerical Software Verification (NSV-7). For 2015, she will be in the program committes of the 26th Journées Francophones des Langages Applicatifs (JFLA 2015), of the 8th International Workshop on Numerical Software Verification (NSV-8) and of the 22nd IEEE Symposium on Computer Arithmetic (ARITH 2015).
- A. Charguéraud served on the external review commitee for POPL 2015.

- É. Contejean is a member of the program committee of the ACM SIGPLAN 2014 Workshop on Partial Evaluation and Program Manipulation, (PEPM 2014) affiliated with the POPL 2014 conference.
- J.-C. Filliâtre is a member of the program committees of the Symposium on Languages, Applications and Technologies (SLATE 2014) and the Journées Francophones des Langages Applicatifs (JFLA 2014).
- C. Marché, member of program committee of the first international workshop Formal-IDE (F-IDE), held as a satellite workshop of ETAPS 2014 http://perso.ensta-paristech.fr/~etaps/, and of the next workshop F-IDE 2015.
- C. Paulin is a member of the program committees of the conference Interactive Theorem Proving (ITP 2014) and the Coq Workshop 2014. For 2015, she will be in the program committees of the following conferences: Certified Programs and Proofs (CPP 2015), Mathematics of Program Construction (MPC 2015) and Types for Proofs and Programs (TYPES 2015).
- A. Paskevich is a member of the program committee of the 11th International Workshop on User Interfaces for Theorem Provers (UITP 2014) affiliated with the FLoC 2014 conference.

9.1.2.2. Reviewer

The members of the Toccata team have reviewed papers for numerous international conferences, including: ARITH, CICM, CPP, ESOP, NFM, POPL, SLATE.

9.1.3. Journal

9.1.3.1. Member of the editorial board

- S. Boldo is member of the editorial committee of the popular science web site “)i(”: http://interstices.info/.
- S. Boldo is member of the editorial board of Binaire http://binaire.blog.lemonde.fr, the blog of the French Computer Science Society.
- J.-C. Filliâtre is member of the editorial board of the *Journal of Functional Programming*.
- C. Paulin is member of the editorial board of the *Journal of Formalized Reasoning*.

9.1.3.2. Reviewer

The members of the Toccata team reviewed numerous papers for numerous international journals, including: Journal of Automated Reasoning (JAR), Transactions in Computational Sciences (TCS), Theory of Computing Systems (TCS), IEEE Transactions on Computers (TC), Journal of Scientific Programming, Journal of Field Robotics (JFR), Mathematical Structures in Computer Science (MSCS), Journal of Automated Software Engineering.

9.1.4. Invited Talks

- S. Boldo was invited speaker at the 16th GAMM-IMACS International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics conference (SCAN 2014) [25] about her methodology for the formal verification of floating-point programs.
- S. Boldo and G. Melquiond were invited speakers at an international workshop in Lyon, France called Mathematical Structures of Computation during the week 4: Formal Proof, Symbolic Computation and Computer Arithmetic. http://smc2014.univ-lyon1.fr/doku.php?id=week4.
- C. Lelay was invited speaker at the 25th JFLA (Journées Francophones des Langages Applicatifs), Fréjus, France, Jan 10th, 2014 about “Coq passe le bac”: she tried the 2013 mathematics test of the scientific Baccalaureate in Coq at the same time as the students.
- C. Marché, “Calcul de plus faible précondition, revisité en Why3”, 25th JFLA, special session on selected representative papers of past editions, joint work with A. Tafat, Fréjus, France, Jan 11th, 2014.

- C. Marché, "Beyond SPARK2014: the ProofInUse project", IFIP WG1.9/2.15, Vienna, Austria, Jul 15th, 2014.
- G. Melquiond was invited speaker at the 12th International Workshop on Satisfiability Modulo Theories (SMT 2014).
- C. Paulin was invited speaker at the 25th JFLA (Journées Francophones des Langages Applicatifs), Fréjus, France, Jan 10th, 2014 to give a tutorial presentation of the ALEA library. She also gave an invited talk during the GDR-GPL "Journées Nationales".
- C. Paulin was invited speaker at the workshop "All about Proofs, Proofs for All" (APPA) as part of the Vienna Summer of Logic. She presented an introduction to the Calculus of Inductive Constructions. She wrote a chapter in the post-conference book [115] to appear next year.

9.1.5. Collective Responsibilities

- S. Boldo, elected member of the Inria Evaluation Committee until August 2014. She was in the committee in charge of selecting the Inria permanent researchers (CR2) in Bordeaux and Nancy and in the national committee for first-class Inria permanent researchers (CR1).
- S. Boldo, member of the national Inria admission committee.
- S. Boldo, C. Marché, and G. Melquiond, members of committees in charge of recruiting assistant professors at Université Paris-Sud.
- S. Boldo, member of the CLFP, *comité local de formation permanente*.
- S. Boldo, scientific head for Saclay for the MECSI group for networking about computer science popularization inside Inria.
- S. Boldo, member of the national popularization committee, *comité de médiation scientifique*, of Inria.
- A. Charguéraud is vice-president of *France-ioi*, a non-profit organization in charge of the selection and the training of the French team to the International Olympiads in Informatics (IOI). France-ioi also provides online exercises in programming and algorithmics—in average, over 70,000 such exercises are solved every month on the website.
- A. Charguéraud is a board member of the non-profit organization *Animath*, which aims at developing interest in mathematics among young students.
- A. Charguéraud and G. Melquiond are members of the committee for the monitoring of PhD students (*"commission de suivi des doctorants"*).
- S. Conchon and A. Paskevich, members of the *"commission consultative de spécialistes de l'université"*, Section 27, University Paris-Sud since December 2014.
- É. Contejean, leader of the VALS team since February 2014.
- É. Contejean, nominated member of the *"conseil du laboratoire"* of LRI since April 2010 till June 2014.
- É. Contejean, elected member of the *"section 6 du Comité National de la Recherche Scientifique"* since September 2012.
- É. Contejean, member of the committee of experts for the AERES evaluation of the Verimag laboratory (2014).
- É. Contejean, reviewer for the ANR (2014).
- É. Contejean, reviewer for the START program of the Austrian Science Fund (FWF) (2014).
- S. Dumbrava, elected member of the *"conseil du laboratoire"* of LRI since June 2014.
- J.-C. Filliâtre is *correcteur au concours d'entrée à l'École Polytechnique* (computer science examiner for the entrance exam at École Polytechnique) since 2008.
- C. Lelay, elected member of the *"conseil du laboratoire"* of LRI from November 2011 to June 2014.

- C. Marché and S. Boldo, members of the *"jury de l'agrégation externe de mathématiques"* as experts in computer science, since 2012.
- C. Marché (since April 2007) and C. Paulin (since September 2010), members of the program committee of Digiteo Labs, the world-class research park in *Île-de-France* region dedicated to information and communication science and technology, http://www.digiteo.fr/. C. Marché, president of this committee since July 2013.
- C. Marché, member of the executive team of the LRI (as "chargé de mission"), since July 2014.
- C. Marché, president of the hiring committee of one associate professor position (Maître de Conferences), Université Paris-Sud, April-June 2014. Member of another such committee.
- C. Marché, director of the ProofInUse Joint Laboratory between Inria and AdaCore, http://www.spark-2014.org/proofinuse
- G. Melquiond and C. Paulin, members of the *"commission consultative de spécialistes de l'université"*, Section 27, University Paris-Sud since April 2010. C. Paulin is the president of this committee since December 2014.
- G. Melquiond, elected officer of the IEEE-1788 standardization committee on interval arithmetic since 2008.
- C. Paulin, scientific leader of Labex DigiCosme http://labex-digicosme.fr (Digital Worlds: distributed data, programs and architectures), a project launched by the French Ministry of research and higher education as part of the program "Investissements d'avenir", it involves the 14 research units in computer science and communications from the "Paris-Saclay" cluster.
- C. Paulin, president of the Computer Science Department of the University Paris-Sud https://www.dep-informatique.u-psud.fr/, since February 2012.
- C. Paulin, president of the assembly of directors of graduate schools at the Université Paris-Sud since September 2012.
- C. Paulin, chaired the hiring committee for a professor position in computer science at Université Paris-Sud.
- In 2014, C. Paulin was elected member of the Informatics Section of Academia Europaea http://www.ae-info.org/.
- A. Paskevich was in charge (together with C. Bastoul in 2012–2013 and B. Cautis in 2013–2014) of Licence professionnelle PER (L3) at IUT d'Orsay, Paris-Sud University since September 2012 till August 2014.

9.2. Teaching - Supervision - Juries

9.2.1. 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 7, France.

Licence: "Programmation Fonctionnelle avancée" (L3), M. Clochard (10h), Université Paris-Sud, France.

Master: "Projet de programmation (compilation)" (M1), M. Clochard (50h), Université Paris-Sud, France.

Licence: "Programmation fonctionnelle avancée" (L3), S. Conchon (30h), Université Paris-Sud, France.

Master: "Compilation" (M1), S. Conchon (33h), Université Paris-Sud, France.

DUT (Diplôme Universitaire de Technologie): "Systèmes" (S3), D. Declerck, IUT d'Orsay, Université Paris-Sud, France.

DUT (Diplôme Universitaire de Technologie): “Programmation et administration des bases de données” (S2), S. Dumbrava (62h), IUT d’Orsay, Université Paris-Sud, France.

DUT (Diplôme Universitaire de Technologie): “Bases de données avancées” (S3), S. Dumbrava (36h), IUT d’Orsay, Université Paris-Sud, France.

Licence: “Langages de programmation et compilation” (L3), J.-C. Filliâtre (36h), École Normale Supérieure, France.

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

Licence: “Mathématiques pour l’informatique” (L2), L. Gondelman (30h), Université Paris-Sud, France.

Master: “Compilation” (M1), L. Gondelman (28h), Université Paris-Sud, France.

DUT (Diplôme Universitaire de Technologie): “Programmation système” (S4), A. Paskevich (48h), IUT d’Orsay, Université Paris-Sud, France.

9.2.2. Internships

- Y. Chatelain, a L3 student from Université Paris-Sud, did a 2-month internship under the supervision of J.-C. Filliâtre and A. Paskevich, on the implementation of a new criterion of termination of recursive functions in a pure functional language.
- J.-P. Deplaix, a third-year student of Epitech, did a 4-month internship, supervised by J.-C. Filliâtre and A. Paskevich, on the compilation of *Why3* programs to C.
- J. Grangier, a M1 student from ENSIEE, did a 10-week internship under the supervision of A. Charguéraud, on the verification of the functional correctness and the asymptotic complexity of a dynamic array data structure.
- X. Onfroy, a L3 student form ENS-Lyon, did a 6-week internship, supervised by G. Melquiond, on the formalization of the gauge integral in the setting of the Coquelicot library.

9.2.3. Supervision

HDR: É. Contejean, “Facettes de la preuve, Jeux de reflets entre démonstration automatique et preuve assistée” [13], Université Paris-Sud, June 13th, 2014.

HDR: S. Boldo, “Deductive Formal Verification: How To Make Your Floating-Point Programs Behave” [12], Université Paris-Sud, October 6th, 2014.

PhD: C. Dross, “Theories and Techniques for Automated Proof of programs” [14], Université Paris-Sud, Apr. 1st, 2014, supervised by C. Marché, A. Paskevich, and with industrial supervisors Y. Moy and J. Kanig (AdaCore company).

PhD: A. Mebsout, “SMT-based Model-Checking” [15], Université Paris-Sud, Sep. 29th, 2014, supervised by F. Zaïdi (LRI) and S. Conchon.

PhD in progress: C. Lelay, “Real numbers for the Coq proof assistant”, since Oct. 2011, supervised by S. Boldo and G. Melquiond.

PhD in progress: S. Dumbrava, “Towards data certification”, since Oct. 2012, supervised by V. Benzaken (LRI) and É. Contejean.

PhD in progress: L. Gondelmans, “Obtention de programmes corrects par raffinement dans un langage de haut niveau”, since Oct. 2013, supervised by J.-C. Filliâtre and A. Paskevich.

PhD in progress: M. Clochard, “A unique language for developing programs and prove them at the same time”, since Oct. 2013, supervised by C. Marché and A. Paskevich.

PhD in progress: J. C. Mbiada Djanda, “Augmented Semantics for the Non-interference of critical C Code”, since Nov. 2013, supervised by C. Marché and J. Signoles (CEA LIST).

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.

9.2.4. Juries

S. Boldo: examiner, PhD committee of Mohamed Amine Najahi, "Synthesis of certified programs in fixed-point arithmetic, and its application to linear algebra basic blocks", Perpignan, France, December 2014.

J.-C. Filliâtre: examiner, PhD committee of Jonathan Protzenko, "Mezzo: the language of the future", Université Paris Diderot, September 2014.

C. Marché: reviewer, PhD committee of T. Bormer "Advancing Deductive Program-Level Verification for Real-World Application", Karlsruhe Institute of Technology, Karlsruhe, Germany, October 2014.

C. Marché: examiner, HDR committee of S. Boldo, "Deductive Formal Verification: How To Make Your Floating-Point Programs Behave", Université Paris-Sud, October 2014.

C. Paulin: examiner, PhD committee of Guillaume Cano, "Interaction entre algèbre linéaire et analyse en formalisation des mathématiques", University of Nice-Sophia Antipolis, March 2014.

C. Paulin: examiner, HDR committee of Évelyne Contejean, "Facettes de la preuve, Jeux de reflets entre démonstration automatique et preuve assistée", Université Paris-Sud, June 2014.

C. Paulin: president of the PhD committee of Peva Blanchard "Synchronization and Fault-tolerance in Distributed Algorithms", Université Paris-Sud, September 2014.

C. Paulin: examiner, HDR committee of Stéphane Graham-Lengrand "Polarities & Focussing: a Journey from Realisability to Automated Reasoning", Université Paris-Sud, December 2014.

9.3. Popularization

- S. Boldo presented the *Concours Castor informatique* to computer sciences teachers (ISN) in Nancy on April 17th, 2014. A video of the talk is available at http://videos.univ-lorraine.fr/index.php?act=view&id=1369.
- S. Boldo gave a 2-hour course entitled *Les nombres et l'ordinateur* at the École Normale Supérieure de Cachan, France, on September 9th.
- S. Boldo gave a 2-hour course *Pourquoi mon ordinateur calcule faux?* to a general audience at the Université Inter-Âge in Versailles, France, on March 4th.
- S. Boldo gave a talk for computer sciences teachers (ISN) in Créteil on March 24th
- S. Boldo gave a talk for teenagers at the lycée Maximilien Perretin Alfortville on May 22nd.
- S. Boldo wrote an article with Jean-Michel Muller (ARIC) in the popularization journal La Recherche [42].
- S. Boldo wrote in 2013 an article for the French blog celebrating 2013 as the "Mathematics of Planet Earth" year: http://mpt2013.fr/meme-les-ordinateurs-font-des-erreurs/. This article was selected to be published in a book published in 2014 [41].
- S. Boldo is member of the editorial committee of the popular science Interstices web site, since April 2008. http://interstices.info/.
- S. Boldo is member of the editorial board of Binaire http://binaire.blog.lemonde.fr, the blog of the French Computer Science Society.
- S. Boldo, G. Melquiond, A. Paskevich, and C. Paulin animated two stands at the *Fête de la science*.
- A. Charguéraud and S. Boldo contributed to the preparation of the exercises of the *Concours Castor informatique* http://castor-informatique.fr/. A. Charguéraud is also one of the three organizers of this contest. The purpose of the Concours Castor in to introduce pupils (from *6ème* to *terminale*) to computer sciences. More than 228,000 teenagers played with the interactive exercises in November 2014.
- S. Conchon and J.-C. Filliâtre published a book *"Apprendre à programmer avec OCaml"* for undergraduate students learning computer programming [39] (Eyrolles, September 2014).

10. Bibliography

Major publications by the team in recent years

[1] P. AUDEBAUD, C. PAULIN-MOHRING. *Proofs of Randomized Algorithms in Coq*, in "Science of Computer Programming", 2009, vol. 74, n^{o} 8, pp. 568–589, http://hal.inria.fr/inria-00431771/en/

[2] F. BOBOT, J.-C. FILLIÂTRE, C. MARCHÉ, A. PASKEVICH. *Why3: Shepherd Your Herd of Provers*, in "Boogie 2011: First International Workshop on Intermediate Verification Languages", Wrocław, Poland, August 2011, pp. 53–64, http://hal.inria.fr/hal-00790310

[3] F. BOBOT, A. PASKEVICH. *Expressing Polymorphic Types in a Many-Sorted Language*, in "Frontiers of Combining Systems, 8th International Symposium, Proceedings", Saarbrücken, Germany, C. TINELLI, V. SOFRONIE-STOKKERMANS (editors), Lecture Notes in Computer Science, October 2011, vol. 6989, pp. 87–102

[4] S. BOLDO. *Floats & Ropes: a case study for formal numerical program verification*, in "36th International Colloquium on Automata, Languages and Programming", Rhodos, Greece, Lecture Notes in Computer Science - ARCoSS, Springer, July 2009, vol. 5556, pp. 91–102

[5] 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

[6] 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/

[7] 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

[8] É. CONTEJEAN, P. COURTIEU, J. FOREST, O. PONS, X. URBAIN. *Automated Certified Proofs with CiME3*, in "22nd International Conference on Rewriting Techniques and Applications (RTA 11)", Novi Sad, Serbia, M. SCHMIDT-SCHAUSS (editor), Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2011, vol. 10, pp. 21–30, http://hal.inria.fr/hal-00777669

[9] J.-C. FILLIÂTRE. *Deductive Program Verification*, Université Paris-Sud, December 2011, Thèse d'habilitation

[10] G. MELQUIOND. *Proving bounds on real-valued functions with computations*, in "Proceedings of the 4th International Joint Conference on Automated Reasoning", Sydney, Australia, A. ARMANDO, P. BAUMGARTNER, G. DOWEK (editors), Lecture Notes in Artificial Intelligence, 2008, vol. 5195, pp. 2–17

[11] Y. MOY, C. MARCHÉ. *Modular Inference of Subprogram Contracts for Safety Checking*, in "Journal of Symbolic Computation", 2010, vol. 45, pp. 1184-1211, http://hal.inria.fr/inria-00534331/en/

Publications of the year

Doctoral Dissertations and Habilitation Theses

[12] S. BOLDO. *Deductive Formal Verification: How To Make Your Floating-Point Programs Behave*, Université Paris-Sud, October 2014, Reviewers: Yves Bertot; John Harrison; Philippe Langlois, Habilitation à diriger des recherches, https://hal.inria.fr/tel-01089643

[13] É. CONTEJEAN. *Facettes de la preuve*, Université Paris-Sud, June 2014, Habilitation à diriger des recherches, https://hal.inria.fr/tel-01089490

[14] C. DROSS. *Generic decision procedures for axiomatic first-order theories*, Université Paris Sud - Paris XI, April 2014, https://tel.archives-ouvertes.fr/tel-01002190

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

Articles in International Peer-Reviewed Journals

[16] F. BOBOT, J.-C. FILLIÂTRE, C. MARCHÉ, A. PASKEVICH. *Let's Verify This with Why3*, in "Software Tools for Technology Transfer (STTT)", March 2014, https://hal.inria.fr/hal-00967132

[17] 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", August 2014, vol. 68, n^{o} 3, 28 p. [*DOI :* 10.1016/J.CAMWA.2014.06.004], https://hal.inria.fr/hal-00769201

[18] 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 [*DOI :* 10.1007/S10817-014-9317-X], https://hal.inria.fr/hal-00862689

[19] S. BOLDO, C. LELAY, G. MELQUIOND. *Coquelicot: A User-Friendly Library of Real Analysis for Coq*, in "Mathematics in Computer Science", June 2014, 22 p. [*DOI :* 10.1007/S11786-014-0181-1], https://hal.inria.fr/hal-00860648

[20] S. BOLDO, C. LELAY, G. MELQUIOND. *Formalization of Real Analysis: A Survey of Proof Assistants and Libraries*, in "Mathematical Structures in Computer Science", 2014, 38 p. [*DOI :* 10.1017/S0960129514000437], https://hal.inria.fr/hal-00806920

[21] C. MARCHÉ, J. KANIG. *Bridging the Gap between Testing and Formal Verification in Ada Development*, in "ERCIM News", January 2015, vol. 100, 2 p. , https://hal.inria.fr/hal-01102242

[22] C. MARCHÉ. *Verification of the Functional Behavior of a Floating-Point Program: an Industrial Case Study*, in "Science of Computer Programming", March 2014, vol. 93, n^{o} 3, pp. 279–296 [*DOI :* 10.1016/J.SCICO.2014.04.003], https://hal.inria.fr/hal-00967124

[23] É. MARTIN-DOREL, G. HANROT, M. MAYERO, L. THÉRY. *Formally verified certificate checkers for hardest-to-round computation*, in "Journal of Automated Reasoning", 2015, vol. 54, n^{o} 1, pp. 1-29 [*DOI :* 10.1007/S10817-014-9312-2], https://hal.inria.fr/hal-00919498

[24] P. ROUX. *Innocuous Double Rounding of Basic Arithmetic Operations*, in "Journal of Formalized Reasoning", November 2014, vol. 7, n^{o} 1, pp. 131-142, https://hal.archives-ouvertes.fr/hal-01091186

Invited Conferences

[25] S. BOLDO. *Formal verification of tricky numerical computations*, in "16th GAMM-IMACS International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics", Würzburg, Germany, M. NEHMEIER (editor), September 2014, 39 p. , https://hal.inria.fr/hal-01088692

International Conferences with Proceedings

[26] U. A. ACAR, A. CHARGUÉRAUD, M. RAINEY. *Theory and Practice of Chunked Sequences*, in "European Symposium on Algorithms", Wrocław, Poland, A. SCHULZ, D. WAGNER (editors), Lecture Notes in Computer Science, Springer Berlin Heidelberg, September 2014, n^{o} 8737, pp. 25 - 36 [*DOI :* 10.1007/978-3-662-44777-2_3], https://hal.inria.fr/hal-01087245

[27] V. BENZAKEN, É. CONTEJEAN, S. DUMBRAVA. *A Coq Formalization of the Relational Data Model*, in "ESOP - 23rd European Symposium on Programming", Grenoble, France, Z. SHAO (editor), Lecture Notes in Computer Science, Springer, April 2014, https://hal.inria.fr/hal-00924156

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

[29] S. CHAUDHURI, M. CLOCHARD, A. SOLAR-LEZAMA. *Bridging Boolean and Quantitative Synthesis Using Smoothed Proof Search*, in "POPL - 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages", San Diego, United States, ACM Press, January 2014, https://hal.inria.fr/hal-00920955

[30] 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, https://hal.inria.fr/hal-01067217

[31] 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, https://hal.inria.fr/hal-01067197

[32] M. CLOCHARD, C. MARCHÉ, A. PASKEVICH. *Verified Programs with Binders*, in "Programming Languages meets Program Verification", San Diego, United States, ACM Press, January 2014, https://hal.inria.fr/hal-00913431

[33] S. CONCHON, M. IGUERNELALA. *Tuning the Alt-Ergo SMT Solver for B Proof Obligations*, in "ABZ", Toulouse, France, June 2014, https://hal.inria.fr/hal-01093000

[34] 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", Toulouse, France, Lecture Notes in Computer Science, Springer, June 2014, vol. 8477, pp. 290-293, https://hal.inria.fr/hal-00998092

[35] J.-C. FILLIÂTRE, L. GONDELMAN, A. PASKEVICH. *The Spirit of Ghost Code*, in "CAV - Computer Aided Verification - 26th International Conference", Vienna Summer Logic 2014, Austria, July 2014, https://hal.inria.fr/hal-00873187

National Conferences with Proceedings

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

[37] S. CONCHON, L. MARANGET, A. MEBSOUT, D. DECLERCK. *Vérification de programmes C concurrents avec Cubicle : Enfoncer les barrières*, in "JFLA", Fréjus, France, January 2014, https://hal.inria.fr/hal-01088655

[38] 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", Paris, France, EasyChair, June 2014, https://hal.inria.fr/hal-00998094

Scientific Books (or Scientific Book chapters)

[39] S. CONCHON, J.-C. FILLIÂTRE. *Apprendre à programmer avec OCaml*, Noire, Eyrolles, September 2014, 429 p. , https://hal.inria.fr/hal-01063853

Research Reports

[40] A. ACAR, A. CHARGUÉRAUD, M. RAINEY. *Data Structures and Algorithms for Robust and Fast Parallel Graph Search*, Inria, December 2014, https://hal.inria.fr/hal-01089125

Scientific Popularization

[41] S. BOLDO. *Même les ordinateurs font des erreurs !*, in "Brèves de maths", M. ANDLER, L. BEL, S. BENZONI-GAVAGE, T. GOUDON, C. IMBERT, A. ROUSSEAU (editors), Nouveau Monde Editions, October 2014, pp. 136-137, https://hal.inria.fr/hal-01089095

[42] J.-M. MULLER, S. BOLDO. *Des ordinateurs capables de calculer plus juste*, in "La Recherche", October 2014, pp. 46-53, https://hal-ens-lyon.archives-ouvertes.fr/ensl-01069744

Other Publications

[43] É. MARTIN-DOREL, G. MELQUIOND. *Proving Tight Bounds on Univariate Expressions in Coq*, 2014, https://hal.inria.fr/hal-01086460

[44] C. PAULIN-MOHRING. *Introduction to the Calculus of Inductive Constructions*, November 2014, https://hal.inria.fr/hal-01094195

[45] P. ROUX. *Formal Proofs of Rounding Error Bounds With application to an automatic positive definiteness check*, 2014, https://hal.archives-ouvertes.fr/hal-01091189

References in notes

[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] B. E. AYDEMIR, A. BOHANNON, M. FAIRBAIRN, J. N. FOSTER, B. C. PIERCE, P. SEWELL, D. VYTINIOTIS, G. WASHBURN, S. WEIRICH, S. ZDANCEWIC. *Mechanized metatheory for the masses: The POPLmark Challenge*, in "Proceedings of the Eighteenth International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2005)", Lecture Notes in Computer Science, Springer, 2005, n^{o} 3603, pp. 50–65

[48] T. BALL, B. COOK, V. LEVIN, S. RAJAMANI. *SLAM and Static Driver Verifier: Technology Transfer of Formal Methods inside Microsoft*, in "Integrated Formal Methods", E. BOITEN, J. DERRICK, G. SMITH (editors), Lecture Notes in Computer Science, Springer, 2004, vol. 2999, pp. 1–20

[49] T. BALL, R. MAJUMDAR, T. MILLSTEIN, S. K. RAJAMANI. *Automatic predicate abstraction of C programs*, in "Proceedings of the ACM SIGPLAN 2001 conference on Programming Language Design and Implementation", ACM Press, 2001, pp. 203–213

[50] M. BARBOSA, J.-C. FILLIÂTRE, J. S. PINTO, B. VIEIRA. *A Deductive Verification Platform for Cryptographic Software*, in "4th International Workshop on Foundations and Techniques for Open Source Software Certification (OpenCert 2010)", Pisa, Italy, Electronic Communications of the EASST, September 2010, vol. 33

[51] R. BARDOU. *Verification of Pointer Programs Using Regions and Permissions*, Université Paris-Sud, October 2011, http://tel.archives-ouvertes.fr/tel-00647331

[52] R. BARDOU, J.-C. FILLIÂTRE, J. KANIG, S. LESCUYER. *Faire bonne figure avec Mlpost*, in "Vingtièmes Journées Francophones des Langages Applicatifs", Saint-Quentin sur Isère, Inria, January 2009

[53] B. BARRAS, B. WERNER. *Coq in Coq*, 1997

[54] 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

[55] G. BARTHE, B. GRÉGOIRE, S. Z. BÉGUELIN. *Formal certification of code-based cryptographic proofs*, in "POPL", Savannah, GA, USA, Z. SHAO, B. C. PIERCE (editors), ACM Press, January 2009, pp. 90-101

[56] P. BAUDIN, J.-C. FILLIÂTRE, C. MARCHÉ, B. MONATE, Y. MOY, V. PREVOSTO. *ACSL: ANSI/ISO C Specification Language, version 1.4*, 2009

[57] 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

[58] F. BOBOT, S. CONCHON, É. CONTEJEAN, M. IGUERNELALA, S. LESCUYER, A. MEBSOUT. *The Alt-Ergo Automated Theorem Prover*, 2008

[59] 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

[60] S. BOLDO. *Preuves formelles en arithmétiques à virgule flottante*, École Normale Supérieure de Lyon, 2004

[61] S. BOLDO. *Kahan's algorithm for a correct discriminant computation at last formally proven*, in "IEEE Transactions on Computers", February 2009, vol. 58, n° 2, pp. 220-225, http://hal.inria.fr/inria-00171497/en/

[62] 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/

[63] S. BOLDO, F. CLÉMENT, J.-C. FILLIÂTRE, M. MAYERO, G. MELQUIOND, P. WEIS. *Wave Equation Numerical Resolution: Mathematics and Program*, Inria, December 2011, n° 7826, 30 p. , http://hal.inria.fr/hal-00649240/en/

[64] 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

[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, G. MELQUIOND. *Emulation of FMA and Correctly-Rounded Sums: Proved Algorithms Using Rounding to Odd*, in "IEEE Transactions on Computers", 2008, vol. 57, n° 4, pp. 462–471, http://hal.inria.fr/inria-00080427/

[67] S. BOLDO, J.-M. MULLER. *Exact and Approximated error of the FMA*, in "IEEE Transactions on Computers", February 2011, vol. 60, n° 2, pp. 157–164, http://hal.inria.fr/inria-00429617/en/

[68] 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

[69] 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° 3, pp. 212–232

[70] S. BÖHME, T. NIPKOW. *Sledgehammer: Judgement Day*, in "IJCAR", J. GIESL, R. HÄHNLE (editors), Lecture Notes in Computer Science, Springer, 2010, vol. 6173, pp. 107-121

[71] 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

[72] A. CHARGUÉRAUD, F. POTTIER. *Functional Translation of a Calculus of Capabilities*, in "ACM SIGPLAN International Conference on Functional Programming (ICFP)", September 2008, pp. 213–224

[73] A. CHARGUÉRAUD. *Characteristic formulae for the verification of imperative programs*, in "Proceeding of the 16th ACM SIGPLAN international conference on Functional Programming (ICFP)", Tokyo, Japan, M. M. T. CHAKRAVARTY, Z. HU, O. DANVY (editors), ACM, September 2011, pp. 418-430

[74] A. CHARGUÉRAUD. *Pretty-Big-Step Semantics*, 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. 41–60, http://hal.inria.fr/hal-00798227

[75] 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

[76] 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

[77] 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

[78] É. CONTEJEAN. *Coccinelle, a Coq library for rewriting*, in "Types", Torino, Italy, March 2008

[79] É. CONTEJEAN, P. COURTIEU, J. FOREST, A. PASKEVICH, O. PONS, X. URBAIN. *A3PAT, an Approach for Certified Automated Termination Proofs*, in "Partial Evaluation and Program Manipulation", Madrid, Spain, J. P. GALLAGHER, J. VOIGTLÄNDER (editors), ACM Press, January 2010, pp. 63-72

[80] P. COUSOT, R. COUSOT, J. FERET, L. MAUBORGNE, A. MINÉ, D. MONNIAUX, X. RIVAL. *The ASTRÉE Analyzer*, in "ESOP", Lecture Notes in Computer Science, 2005, n^{o} 3444, pp. 21–30

[81] 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

[82] M. DAUMAS, G. MELQUIOND. *Certification of bounds on expressions involving rounded operators*, in "Transactions on Mathematical Software", 2010, vol. 37, n^{o} 1, http://hal.archives-ouvertes.fr/inria-00534350/fr/

[83] C. DROSS, S. CONCHON, J. KANIG, A. PASKEVICH. *Reasoning with Triggers*, Inria, June 2012, n^{o} RR-7986, 29 p. , http://hal.inria.fr/hal-00703207

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

[85] C. DROSS, J.-C. FILLIÂTRE, Y. MOY. *Correct Code Containing Containers*, in "5th International Conference on Tests and Proofs (TAP'11)", Zurich, Lecture Notes in Computer Science, Springer, June 2011, vol. 6706, pp. 102–118, http://hal.inria.fr/hal-00777683

[86] J.-C. FILLIÂTRE. *Formal Verification of MIX Programs*, in "Journées en l'honneur de Donald E. Knuth", Bordeaux, France, October 2007

[87] J.-C. FILLIÂTRE. *Deductive Software Verification*, in "International Journal on Software Tools for Technology Transfer (STTT)", August 2011, vol. 13, n^{o} 5, pp. 397-403

[88] 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

[89] 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

[90] J.-C. FILLIÂTRE, K. KALYANASUNDARAM. *Functory: A Distributed Computing Library for Objective Caml*, in "Trends in Functional Programming", Madrid, Spain, Lecture Notes in Computer Science, May 2011, vol. 7193, pp. 65–81

[91] J.-C. FILLIÂTRE, K. KALYANASUNDARAM. *Une bibliothèque de calcul distribué pour Objective Caml*, in "Vingt-deuxièmes Journées Francophones des Langages Applicatifs", La Bresse, France, S. CONCHON (editor), Inria, January 2011

[92] J. GERLACH, J. BURGHARDT. *An Experience Report on the Verification of Algorithms in the C++ Standard Library using Frama-C*, in "Formal Verification of Object-Oriented Software, Papers Presented at the International Conference", Paris, France, B. BECKERT, C. MARCHÉ (editors), Karlsruhe Reports in Informatics, June 2010, pp. 191–204, http://hal.inria.fr/hal-00772519

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

[94] 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

[95] K. KALYANASUNDARAM, C. MARCHÉ. *Automated Generation of Loop Invariants using Predicate Abstraction*, Inria, August 2011, n^{o} 7714, http://hal.inria.fr/inria-00615623/en/

[96] J. KANIG, J.-C. FILLIÂTRE. *Who: A Verifier for Effectful Higher-order Programs*, in "ACM SIGPLAN Workshop on ML", Edinburgh, Scotland, UK, August 2009, http://hal.inria.fr/hal-00777585

[97] 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

[98] G. T. LEAVENS, K. R. M. LEINO, P. MÜLLER. *Specification and verification challenges for sequential object-oriented programs*, in "Formal Aspects of Computing", 2007, pp. 159–189

[99] K. R. M. LEINO. *Efficient weakest preconditions*, in "Information Processing Letters", 2005, vol. 93, n^{o} 6, pp. 281-288

[100] 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

[101] 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/

[102] 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/

[103] S. LESCUYER. *Formalisation et développement d'une tactique réflexive pour la démonstration automatique en Coq*, Université Paris-Sud, January 2011, http://tel.archives-ouvertes.fr/tel-00713668

[104] C. MARCHÉ. *The Krakatoa tool for Deductive Verification of Java Programs*, January 2009, Winter School on Object-Oriented Verification, Viinistu, Estonia

[105] 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

[106] G. MELQUIOND. *De l'arithmétique d'intervalles à la certification de programmes*, École Normale Supérieure de LyonLyon, France, 2006

[107] G. MELQUIOND. *Floating-point arithmetic in the Coq system*, in "Proceedings of the 8th Conference on Real Numbers and Computers", Santiago de Compostela, Spain, 2008, pp. 93–102

[108] G. MELQUIOND, S. PION. *Formally certified floating-point filters for homogeneous geometric predicates*, in "Theoretical Informatics and Applications", 2007, vol. 41, n^{o} 1, pp. 57–70

[109] 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/

[110] C. MORGAN. *Programming from specifications (2nd ed.)*, Prentice Hall International (UK) Ltd., 1994

[111] Y. MOY, C. MARCHÉ. *The Jessie plugin for Deduction Verification in Frama-C — Tutorial and Reference Manual*, Inria & LRI, 2011

[112] 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

[113] 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

[114] 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

[115] C. PAULIN-MOHRING. *Introduction to the Calculus of Inductive Constructions*, in "All about Proofs, Proofs for All", London, UK, D. DELAHAYE, B. WOLTZENLOGEL PALEO (editors), Mathematical Logic and Foundations, College Publications, 01 2015, http://hal.inria.fr/hal-01094195

[116] S. RANISE, C. TINELLI. *The Satisfiability Modulo Theories Library (SMT-LIB)*, 2006

[117] S. M. RUMP, P. ZIMMERMANN, S. BOLDO, G. MELQUIOND. *Computing predecessor and successor in rounding to nearest*, in "BIT", June 2009, vol. 49, n^{o} 2, pp. 419–431, http://hal.inria.fr/inria-00337537/

[118] J. SMANS, B. JACOBS, F. PIESSENS. *Implicit Dynamic Frames: Combining Dynamic Frames and Separation Logic*, in "ECOOP 2009 — Object-Oriented Programming", S. DROSSOPOULOU (editor), Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2009, pp. 148-172

[119] B. SPITTERS, S. VICKERS, S. WOLTERS. *Gelfand spectra in Grothendieck toposes using geometric mathematics*, in "Proceedings 9th Workshop on Quantum Physics and Logic", R. DUNCAN, P. PANANGADEN (editors), EPTCS, 2014, vol. 158, pp. 77–107

[120] A. J. SUMMERS, S. DROSSOPOULOU. *Considerate Reasoning and the Composite Design Pattern*, in "VMCAI", G. BARTHE, M. V. HERMENEGILDO (editors), Lecture Notes in Computer Science, Springer, 2010, vol. 5944, pp. 328-344

[121] H. TUCH, G. KLEIN, M. NORRISH. *Types, Bytes, and Separation Logic*, in "Proc. 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'07)", Nice, France, M. HOFMANN, M. FELLEISEN (editors), January 2007, pp. 97-108

[122] E. TUSHKANOVA, A. GIORGETTI, C. MARCHÉ, O. KOUCHNARENKO. *Specifying Generic Java Programs: two case studies*, in "Tenth Workshop on Language Descriptions, Tools and Applications", C. BRABRAND, P.-E. MOREAU (editors), ACM Press, 2010, http://hal.inria.fr/inria-00525784/en/

[123] 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/

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