Activity Report 2013

Team TOCCATA

Formally Verified Programs, Certified Tools and Numerical Computations

RESEARCH CENTER
Saclay - Île-de-France

THEME
Proofs and Verification

Table of contents

Team TOCCATA

Keywords: Proofs Of Programs, Automated Theorem Proving, Interactive Theorem Proving, Safety, Floating-point Numbers

The Toccata team 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.

1. Members

Research Scientists

Claude Marché [Team leader, Inria, Senior Researcher, HdR]
Sylvie Boldo [Team Vice-Leader, Inria, Researcher]
Arthur Charguéraud [Inria, Researcher]
Évelyne Contejean [CNRS, Researcher]
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 [Univ. Paris-Sud, Professor, HdR]

External Collaborator

Xavier Urbain [ENSIEE]

PhD Students

Martin Clochard [ENS Paris, from Sep. 2013]
Claire Dross [Univ. Paris-Sud, CIFRE Adacore]
Stefania Dumbrava [Univ. Paris-Sud]
Levs Gondelmans [Univ. Paris-Sud, from Oct. 2013]
Mohamed Iguernelala [Univ. Paris-Sud, until Aug 2013]
Catherine Lelay [Inria, Digiteo grant Coquelicot]
Alain Mebsout [Univ. Paris-Sud]

Post-Doctoral Fellows

Asma Tafat Bouzid [Univ. Paris-Sud, A.T.E.R]
Erik Martin-Dorel [Inria, ANR project Verasco, from Oct. 2013]

Administrative Assistant

Régine Bricquet [Inria]

2. Overall Objectives

2.1. Introduction

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 last decade. Such a software appears in various application domains like 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* [81].
- *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 [80]. 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 [53].
- *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 class of approaches above are compatible. We indeed have studied some way to combine 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 [61] and other railroad-related systems, a formally proved C compiler was developed using the *Coq* proof assistant [101], Microsoft's hypervisor for highly secure virtualization was verified using VCC [82] and the Z3 prover [120], 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 [60], JML [72]) and automatically producing a set of mathematical formulas (the *verification conditions*) which are typically proved using automatic provers (e.g. Z3 [120], *Alt-Ergo* [62], CVC3 [58]). The first family of approaches usually offers a higher level of assurance than the second, but also demands more work to perform the proofs (because of their interactive nature) and makes them less easy to adopt by industry. Moreover they do not allow to directly analyze a program written in a mainstream programming language like Java or C. The second kind of approaches has benefited in the past years from the tremendous 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 4 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 [63] 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 recent contributions. A shorter version appears in the STTT journal [88]. The ABZ'2012 paper [107] is a representative publication presenting an application of our *Why3* system to solving proof obligations coming from *Atelier B*. The VSTTE'2012 paper [90] 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 [119] presenting the use of the Gappa solver for proving numerical algorithms, a paper in the ISSE journal in 2011 [71] together with a paper at the CPP'2011 conference [111] 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 [110]. 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 [66] and in another part at ICALP'2009 [4] and fully in a paper in the JAR journal [15].

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] [89], 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 or automatic provers can be used to discharge the verification conditions.

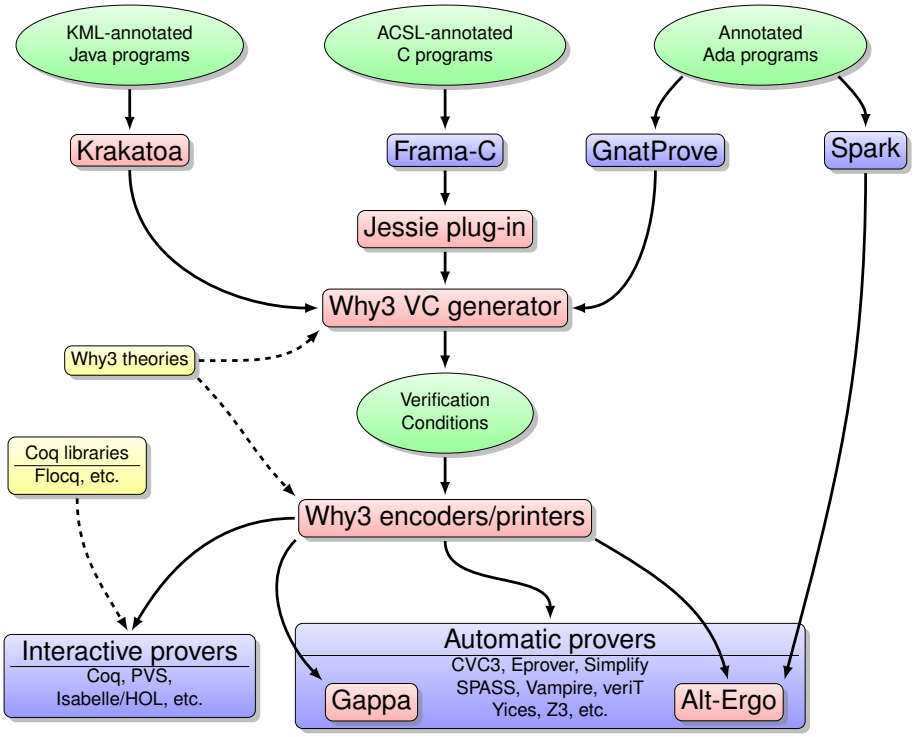

Figure 1. Interactions between our tools and with others

Among automated provers, our own tool *Alt-Ergo* [62] 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* [103] reads annotated Java source code and produces *Why3* code. The tool *Jessie* [109][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].

2.2. Highlights of the Year

- The *Castor informatique* http://castor-informatique.fr/, is an international competition to present computer science to pupils (from *6ème* to *terminale*). More than 170,000 teenagers played on more than 30 proposed exercises in November 2013. Two members of the Toccata team (S. Boldo and A. Charguéraud) belong to the organization committee (5 people).
- The full formalization of the JavaScript language specification (ECMAScript 5) was recently completed by the *JsCert* team [24], which includes A. Charguéraud and 7 collaborators from Imperial College and Inria Rennes (http://jscert.org). The formalization, which involves more than 10,000 lines of code and an inductive semantics with over 600 reduction rules, is the result of 2 years of effort. It lead to the discovery of bugs in the official standard, in the official test suites, and in all major browsers. In particular, it has raised the interest of several members of the ECMAScript standardization committee, and that of the developers of secure subsets for JavaScript.
- J.-C. Filliâtre was invited as keynote speaker ("One Logic To Use Them All" [19]) at the International Conference on Automated Deduction in 2013. It is the main conference of the year in the domain of Automated Reasoning. In this talk he presented the *Why3* approach for interacting with dozens of provers on the same theories and goals. This invited talk is a recognition by the community of this unique feature of *Why3*.
- Most 18-year old French students pass an exam called Baccalaureate which ends the high school and is required for attending the university. The idea was to try our Coq library Coquelicot on the 2013 mathematics test of the scientific Baccalaureate. C. Lelay went to the "Parc de Vilgénis" high school in Massy, France and took the 2013 test at the same time as the students, but had to formally prove the answers [45] (see also https://www.lri.fr/~lelay/).
- The Coq proof assistant received the ACM *Programming Languages Software Award* in 2013 http://www.sigplan.org/Awards/Software/Main. The development of Coq was initiated by Thierry Coquand and Gérard Huet in 1984. The current environment is the result of the work of more than 40 direct contributors, including major contributions by Christine Paulin-Mohring and Jean-Christophe Filliâtre from our team.

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 [67][5]. This is why formal verification of numerical programs is another axis of our research.

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.

3.2. Formally Verified Programs

This theme of research builds upon our expertise on the development of methods and tools for proving programs, from source codes annotated with specifications to proofs. In the past years, 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* [117], *implicit dynamic frames* [115], and *considerate reasoning* [116]. 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 [118] 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 [74], [96], [55].

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 [84], [85]. 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. There are a few solutions that were proposed yet, that amount to encode higher-order goals in first-order ones [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 [112]. 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 [73]. 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* [108] 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 in 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 [54]. We plan to use proof techniques on applications related to databases. We also have plans to support low-level programs such as assembly code [87], [111] 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* [107] (project BWare).

3.3. Certified Tools

The goal of this theme 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 [64], Gappalib [105], and now Flocq [6] which unifies the formers). Second we recently conducted the development of a certified decision procedure [102] that corresponds to a core part of *Alt-Ergo*, and a certified verification condition generator for a language [94] similar to *Why*. On-going work aims at building, still in *Coq*, a certified VC generator for C annotated in ACSL [60], based on the operational semantics formalized in the CompCert certified compiler project [101].

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 [52]).

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 [107]. 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.

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

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* [102] 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. Currently, *Gappa* can be asked to produce a *Coq* proof of its given goal, so as to check *a posteriori* its soundness. Indeed, the idea of producing a trace is not contradictory with certifying the tool. For very complex decision procedures, the goal of developing a certified proof search might be too ambitious, and the production of an internal trace is a general technique that might be used as a workaround: it suffices to instrument the proof search and to develop a certified trace checker to be used by the tool before it gives an answer. We used this approach in the past for certified proofs of termination of rewriting systems [79]. This is also a technique that is used internally in CompCert for some passes of compilation [101].

3.3.4. Certified VC Generation

The other kind of tools that we would like to certify are the VC generators. This will be a continuation of the on-going work on developing in *Coq* a certified VC generator for C code annotated in ACSL. We would like to develop such a generator in *Why3* instead of *Coq*. As before, this will build 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, it would be interesting 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 [75] 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 are not first-class objects in *Coq*. 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 [57].

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 [70], [119], [66], [114], [69], [65], [106], [104] but also 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, and to achieve them some automation is needed, and obtained by combining SMT solvers and *Gappa* [67], [83], [51][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 [71], [111].

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

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 smart 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 tend to be confused by floating-point arithmetic [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

The goal is to handle floating-point programs that are related to continuous quantities. This includes numerical analysis programs we have already worked on [15] [66][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 this kind of programs. 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 automation as possible. We wish this could be used by numerical analysts that are not or hardly familiar with formal methods, but are 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

- Transportation: aeronautics, railroad, space flight, automotive
- Communications: mobile phones, smart phones, Web applications
- Financial applications, banking
- Medicine: diagnostic devices, computer-assisted surgery
- Databases with confidentiality requirements (e.g. health records, electronic voting)

Currently our industrial collaborations mainly belong the first of these domains: transportation. These include, in the context of the ANR U3CAT project (Airbus France, Toulouse; Dassault Aviation, Saint-Cloud; Sagem Défense et Sécurité):

- proof of C programs via *Frama-C*/*Jessie*/*Why*;
- proof of floating-point programs;
- use of the *Alt-Ergo* prover via CAVEAT tool (CEA) or *Frama-C*/WP.

In the context of the FUI project Hi-Lite, the Adacore (Paris) uses *Why3* and *Alt-Ergo* as back-end to GnatProve, an environment for verification of Ada programs. This is applied in the domain of aerospace (Thales, EADS Astrium).

In the context of ANR project BWare, we investigate the use of *Why3* and *Alt-Ergo* as an alternative back-end for checking proof obligation generated by *Atelier B*, whose main applications are railroad-related software (http://www.methode-b.com/documentation_b/ClearSy-Industrial_Use_of_B.pdf, collaboration with Mitsubishi Electric R&D Centre Europe, Rennes; ClearSy, Aix-en-Provence)

Apart from the domain of transportation, the Cubicle model checker modulo theories based on the *Alt-Ergo* SMT prover (collaboration with Intel Strategic Cad Labs, Hillsboro, OR, USA) can be applied to verification of concurrent programs and protocols (http://cubicle.lri.fr/).

5. Software and Platforms

5.1. The CiME rewrite toolbox

Participants: Évelyne Contejean [contact], Claude Marché, Andrei Paskevich.

CiME is a rewriting toolbox. Distributed since 1996 as open source, at URL http://cime.lri.fr. Beyond a few dozens of users, CiME is used as back-end for other tools such as the TALP tool developed by Enno Ohlebusch at Bielefeld university for termination of logic programs; the MU-TERM tool (http://www.dsic.upv.es/~slucas/csr/termination/muterm/) for termination of context-sensitive rewriting; the CARIBOO tool (developed at Inria Nancy Grand-Est) for termination of rewriting under strategies; and the MTT tool (http://www.lcc.uma.es/~duran/MTT/) for termination of Maude programs. CiME2 is no longer maintained, and the currently developed version is CiME3, available at http://a3pat.ensiie.fr/pub. The main new feature of CiME3 is the production of traces for *Coq*. CiME3 is also developed by the participants of the A3PAT project at the CNAM, and is distributed under the Cecill-C license.

5.2. The Why platform

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

Criteria for Software Self-Assessment [5]: 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*.

[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)

The platform is distributed as open source, under GPL license, at http://why.lri.fr/. The internal VC generator and the translators to external provers are no longer under active development, as superseded by the *Why3*system described below.

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 [93], at Universidade do Minho in Portugal [54], at Moscow State University, Russia (http://journal.ub.tu-berlin.de/eceasst/article/view/255).

5.3. 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.

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/.

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 in a near future 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 (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.

5.4. 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 automatic, little engine of proof 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 in the *Why*'s annotation language; this means that *Alt-Ergo* fully supports first order polymorphic logic with quantifiers. *Alt-Ergo* also supports the standard [113] defined by the SMT-lib initiative.

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. *Alt-Ergo* is also called as an external prover by the Pangolin tool developed by Y. Regis Gianas, Inria project-team Gallium http://code.google.com/p/pangolin-programming-language/. Alt-Ergo is usable as a back-end prover in the SPARK verifier for ADA programs, since Oct 2010. It is planed to be integrated in next generation of Airbus development process.

Alt-Ergo is distributed as open source, under the CeCILL-C license, at URL http://alt-ergo.lri.fr/.

5.5. 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. This is a syntactically restricted class of parametrized transition systems with states represented as arrays indexed by an arbitrary number of processes. Cache coherence protocols and mutual exclusion algorithms are typical examples of such systems.

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.

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

Bibtex2html is also distributed as a package in most Linux distributions. Package popularity contests show that it is among the 20% most often installed packages.

5.7. 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/. It is also distributed as a package in several Linux distributions. OCamlgraph is now widely spread among the community of *OCaml* developers.

5.8. 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 [56].

5.9. 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 [92] and at TFP 2011 [91].

5.10. The Pff library

Participant: Sylvie Boldo [contact].

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

The Pff library for the *Coq* proof assistant is a formalization of floating-point arithmetic with high-level definitions and high-level properties [64].

It is distributed as open source, under a LGPL license, at http://lipforge.ens-lyon.fr/www/pff/, and is packaged in Debian and Ubuntu as "coq-float".

It was initiated by M. Daumas, L. Rideau and L. Théry in 2001, and then developed and maintained by S. Boldo since 2004. It is now only maintained by S. Boldo. The development has ended as this library is now subsumed by the Flocq library (see below).

5.11. 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.

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

5.12. 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 [83] [119].

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

5.13. The Interval package for Coq

Participant: Guillaume Melquiond [contact].

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*.

It is distributed as open source, under a LGPL license, at http://www.lri.fr/~melquion/soft/coq-interval/. Part of the work on this library was done while in the Mathematical Components team (Microsoft Research–Inria Joint Research Center).

5.14. 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.

It is distributed as open source, at http://www.lri.fr/~paulin/ALEA. It 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 [59]. It is also experimented in LABRI as a basis to study formal proofs of probabilistic distributed algorithms. ALEA version 8 distributed in May 2013 includes a module to reason with random variables with values in positive real numbers.

5.15. The Coccinelle library for term rewriting

Participant: Évelyne Contejean [contact].

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 some of the algorithms implemented in the CiME toolbox, such a matching, matching modulo associativity-commutativity, computation of the one-step reducts of a term, 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.

5.16. 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.

Coquelicot is a *Coq* library dedicated to real analysis: differentiation, integration, and so on. It is a conservative extension of the standard library of *Coq*, but with a strong focus on usability.

Coquelicot is available at http://coquelicot.saclay.inria.fr/.

5.17. CFML

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. 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/. The tool has been initially developed by A. Charguéraud in 2010, and has been maintained and improved since by the author.

6. New Results

6.1. Deductive Verification

- F. Bobot, J.-C. Filliâtre, C. Marché, G. Melquiond, and A. Paskevich have presented the proof session mechanism of *Why3* at VSTTE 2013 [23]. It is a technique to maintain a proof session against modification of verification conditions. It was successfully used in developing more than a hundred verified programs and in keeping them up to date along the evolution of *Why3* and its standard library. It also helps out with changes in the environment, *e.g.* prover upgrades.
- 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 will be presented at the PLPV conference in January 2014 [29]

- 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 [28]. 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, L. Gondelman, and A. Paskevich have formalized the notion of ghost code implemented in *Why3*, in a paper *The Spirit of Ghost Code* [49] to be submitted. This is an outcome of L. Gondelman's M2 internship (spring/summer 2013).
- In 2013, two public releases of *Why3* were launched, version 0.81 in March and version 0.82 in December [42]. A first important evolution relies on significant efficiency improvements both in terms of execution speed and of memory usage. The second major evolution is the support for many new provers, including interactive provers PVS 6 (used at NASA) and Isabelle2013-2 (planned to be used in the context of Ada program via Spark), and automated ones: CVC4, Mathematica, Metitarski, Metis, Beagle, Princess, and Yices2. The design of the programming language of *Why3*(WhyML) was presented during a tool demonstration at the ESOP conference [33].

6.2. Floating-Point and Numerical Programs

- S. Boldo, F. Clément, J.-C. Filliâtre, M. Mayero, G. Melquiond, and P. Weis, finished the formal proof of a numerical analysis program: the second order centered finite difference scheme for the one-dimensional acoustic wave [15].
- S. Boldo developed a formal proof of an algorithm for computing the area of a triangle, an improvement of its error bound and new investigations in case of underflow [25].
- S. Boldo, J.-H. Jourdan, X. Leroy, and G. Melquiond, extended CompCert to get the first formally verified compiler that provably preserves the semantics of floating-point programs [26].
- S. Boldo and G. Melquiond wrote a chapter of the book [38] that describes the current state of the Mathematics/Computer science research in France.
- C. Lelay worked on formalizing power series for the Coq proof assistant [35].
- Most 18-year old French students pass an exam called Baccalaureate which ends the high school and is required for attending the university. The idea was to try our Coq library Coquelicot on the 2013 mathematics test of the scientific Baccalaureate. C. Lelay went to the "Parc de Vilgénis" high school in Massy, France and took the 2013 test at the same time as the students, but had to formally prove the answers. There was therefore no possible cheating: the Coq library was already developed and it was tested as is during the four hours of the test. This experiment shows that Coquelicot is able to cope with basic real analysis: it has the necessary definitions and lemmas, and its usability and efficiency have been demonstrated in a test with a limited time [45] (see also https://www.lri.fr/~lelay/).
- D. Ishii and G. Melquiond applied methods of deductive program verification to ensure the safety of hybrid automata [34].
- É. Martin-Dorel, G. Hanrot, M. Mayero, L. Théry, showed how to generate and formally check certificates in the Coq proof assistant to solve myriads of instances of the Integer Small Value Problem (ISValP). This problem is directly related to solving the Table Maker's Dilemma with hardest-to-round computations [50]. A new version of the formalized library has been released (http://tamadi.gforge.inria.fr/CoqHensel/).
- É. Martin-Dorel, G. Melquiond, and J.-M. Muller, studied issues related to double rounding in the implementation of error-free transformations [16].

6.3. Automated Reasoning

- C. Dross, S. Conchon, J. Kanig, and A. Paskevich have proposed a new approach for handling quantified formulas in SMT solvers. Their framework is based on the notion of instantiation patterns, also known as triggers, that suggest instances which are more likely to be useful in proof search. This framework has been implemented in the Alt-Ergo SMT solver [48].
- S. Conchon, A. Goel, S. Krstic, A. Mebsout, and F. Zaïdi have designed a new model checking algorithm that is able to infer invariants strong enough to prove complex parameterized cache-coherence protocols [30].
- S. Conchon, A. Mebsout, and F. Zaïdi have presented a new SMT library called Alt-Ergo-Zero. This library is tightly integrated to the backward reachability algorithm of the Cubicle model checker [31].
- S. Conchon, M. Iguernelala, and A. Mebsout have designed a collaborative framework for reasoning modulo simple properties of non-linear arithmetic. This framework has been implemented in the Alt-Ergo SMT solver [47].
- J. C. Blanchette and A. Paskevich designed an extension to the TPTP TFF (Typed First-order Form) format of theorem proving problems to support rank-1 polymorphic types (also known as ML-style parametric polymorphism). This extension, named TFF1, was incorporated in the TPTP standard and was presented at the CADE-24 conference [22].

6.4. Certification of Languages, Tools and Systems

- A. Tafat and C. Marché developed a certified VC generator using Why3. The challenge was to formalize the operational semantics of an imperative language, and a corresponding weakest precondition calculus, without the possibility to use *Coq* advanced features such as dependent types nor higher-order functions. The classical issues with local bindings, names and substitutions were solved by identifying appropriate lemmas. It was shown that Why3 can offer a very significantly higher amount of proof automation compared to *Coq* [36]
- 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 – Bretagne Atlantique (see http://jscert.org).

  The formalization consists of a set of inductive rules translating the prose from the *ECMAScript Language Specification, version 5*. 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 2014 [24]. A key ingredient in this formalization is the use of the *pretty-big-step semantics*. This technique allows for representing evaluation rules in big-step style without suffering from a duplication of several premises across different rules. The pretty-big-step technique is described in a paper published by A. Charguéraud at ESOP 2013 [27].
- É. Contejean, together with V. Benzaken and their PhD student S. Dumbrava, have proposed a *Coq* formalization of the relational data model which underlies relational database systems [21]. Proposing such a formalization is the first, *essential* step, that will allow to *prove* that existing systems conform to their specifications and to *verify* both production implementations of database systems and database-backed applications. More precisely, they present and formalize the data definition part of the model including integrity constraints, attributes, tuples, relations, schemas and integrity constraints (including the so-called Armstrong's system and the chase). They model two different query language formalisms: relational algebra and conjunctive queries. The former is

the basis of the SQL commercial query language and the latter underlies graphical languages, such as Microsoft Access or Query By Example (QBE). They also present logical query optimization and prove the main "database theorems": algebraic equivalences, the homomorphism theorem and conjunctive query minimization.

6.5. Miscellaneous

- R. El Sibaie and J.-C. Filliâtre have developed *Combine*, an OCaml library for combinatorics. It provides two different solutions to the exact matrix cover problem: Knuth's dancing links and ZDDs, a variant of binary decision diagrams [32].

7. Bilateral Contracts and Grants with Industry

7.1. Bilateral Contracts with Industry

7.1.1. 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 bilateral collaboration contract, that started in January 2012 for 3 years.

The aim is 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 [85]. A focus is made on programs involving Ada containers [86].

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. The goal of this project was to develop a new version of Cubicle with significantly improved model-checking power. This required innovative algorithmic enhancements to be implemented and evaluated.

8. Partnerships and Cooperations

8.1. Regional Initiatives

8.1.1. Coquelicot

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

Coquelicot is a 3 years 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 [100], [68][35], [45]. This is sorely needed to ease the verification of numerical applications, especially those involving advanced mathematics.

Partners: LIX (Palaiseau), University Paris 13

8.2. National Initiatives

8.2.1. ANR BWare

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

This is a research project funded by the programme "Ingénierie Numérique & Sécurité" of the ANR. It is funded for a period of 4 years and started on September 1, 2012. http://bware.lri.fr.

It 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 guarantees 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 [107]. 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); Inria teams Gallium, Deducteam and Asap; Mitsubishi Electric R&D Centre Europe, the ClearSy company that develops and maintains *Atelier B* and the OCamlPro start-up.

8.2.2. ANR Verasco

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

This is a research project funded by the programme "Ingénierie Numérique & Sécurité" of the ANR. It is funded for a period of 4 years and started on January 1st, 2012. 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).

8.2.3. Systematic: Hi-Lite

Participants: Claude Marché [contact], Jean-Christophe Filliâtre, Sylvain Conchon, Évelyne Contejean, Andrei Paskevich, Alain Mebsout, Mohamed Iguernelala, Denis Cousineau.

The Hi-Lite project (http://www.open-do.org/projects/hi-lite/) is a project in the SYSTEMATIC Paris Region French cluster in complex systems design and management http://www.systematic-paris-region.org.

Hi-Lite is a project aiming at popularizing formal methods for the development of high-integrity software. It targets ease of adoption through a loose integration of formal proofs with testing and static analysis, that allows combining techniques around a common expression of specifications. Its technical focus is on modularity, that allows a divide-and-conquer approach to large software systems, as well as an early adoption by all programmers in the software life cycle.

Our involvements in that project include the use of the Alt-Ergo prover as back-end to already existing tools for SPARK/ADA, and the design of a verification chain for an extended SPARK/ADA language to verification conditions, via the Why3 VC generator.

The results of that project are the basis of SPARK2014, the next generation of the SPARK.

This project was funded by the French Ministry of industry (FUI), the Île-de-France region and the Essonne general council for 36 months from September 2010.

8.3. European Initiatives

8.3.1. FP7 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) is the principal investigator of this ERC-funded project. The other 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/.

8.3.2. Collaborations in European Programs, except FP7

Project acronym: JsCert

Project title: Certified JavaScript

Duration: Oct. 2011 - ...

Other partners: Imperial College and Inria Rennes – Bretagne Atlantique (Celtique project).

Abstract: This project aims at providing a formal semantics to the JavaScript language. It is joint work with Philippa Gardner, Sergio Maffeis, Gareth Smith, Daniele Filaretti and Daiva Naudziuniene from Imperial College, Alan Schmitt and Martin Bodin from Inria Rennes – Bretagne Atlantique, and Arthur Charguéraud from Inria Saclay –Île-de-France. Project website: http://jscert.org.

8.4. International Initiatives

8.4.1. Inria International Partners

8.4.1.1. Informal International Partners

- S. Conchon, A. Mebsout and F. Zaidi (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.

8.4.2. Participation In other International Programs

- C. Paulin is the representative of Univ. Paris-Sud for the education part of the EIT KIC ICT Labs. She contributed to the proposition of two master programs as well as the action on weaving Innovation and Entrepreneurship in Doctoral programs and the preparation of the Summer School “Imagine the future in ICT”.

9. Dissemination

9.1. Scientific Animation

9.1.1. Event Organization

- C. Marché organizer of the first DigiCosme Spring School (http://labex-digicosme.fr/Spring+School+2013 whose theme is *Program Analysis and Verification* in April 2013.
- C. Paulin, organizer with D. Pichardie and S. Blazy of the 4th Conference on Interactive Theorem Proving (http://itp2013.inria.fr/) in July 2013.
- C. Paulin, organizer with Zhong Shao (Yale Univ.) of the workshop "Certification of high-level and low-level programs" July 7-11, 2014, as part of the Institut Henri Poincaré thematic trimester on Semantics of proofs and certified mathematics https://ihp2014.pps.univ-paris-diderot.fr.

9.1.2. Editorial boards

- S. Boldo, member of the editorial committee of the popular science web site *interstices*, http://interstices.info/.
- J.-C. Filliâtre is member of the editorial board of the *Journal of Functional Programming*.
- C. Paulin, member of the editorial board of the *Journal of Formalized Reasoning*.

9.1.3. Learned societies

- J.-C. Filliâtre is a member of IFIP Working Group 1.9/2.15 (Verified Software)

9.1.4. Program committees

- É. Contejean, member of the program committee of the 24th International Conference on Automated Deduction (CADE 24, http://www.cade-24.info/),member of the program committee of the ACM SIGPLAN 2014 Workshop on Partial Evaluation and Program Manipulation (PEPM 2014, http://www.program-transformation.org/PEPM14), and member of the program committee of the 13th International Workshop on Termination (WST 2013, http://www.imn.htwk-leipzig.de/WST2013/).
- C. Marché, *Tool Chair* of the program committee of the 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2013, Rome, Italy, http://www.etaps.org/index.php/2013/tacas), part of the ETAPS joint Conference. The tool chair is responsible for the evaluation and selection of tool papers and tool demonstrations, following precise guidelines given in the call for papers. This initiative of TACAS aims at making the selection of such submissions more accurate (http://www.etaps.org/index.php/2013/tacas/tacas13-tool-papers-menu).
- C. Paulin, member of the program committees of the fourth and fifth conferences on Interactive Theorem Proving (ITP 2013, http://itp2013.inria.fr/ and ITP 2014 http://www.cs.uwyo.edu/~ruben/itp-2014/).
- J.-C. Filliâtre is a member of the program committees of the 5th NASA Formal Methods Symposium (NFM 2013), Certified Programs and Proofs (CPP 2013), Symposium on Languages, Applications and Technologies (SLATE 2013), VeriSure: Verification and Assurance (2013), and the 5th Working Conference on Verified Software: Theories, Tools and Experiments (VSTTE 2013).
- A. Paskevich is a member of the program committee of the 3rd International Workshop on Proof Exchange for Theorem Proving (PxTP 2013) affiliated with the CADE-24 conference.

9.1.5. Invited Presentations

- J.-C. Filliâtre, "One logic to use them all", CADE-24, Lake Placid, USA, June 2013 [19].
- J.-C. Filliâtre, "Deductive Program Verification", PLMW 2013, Rome, Italy, January 2013 [18].
- S. Boldo, "Formal proofs and the 1D wave equation", MOISE seminar, Grenoble, January 10th.
- S. Boldo, "Formal verification of numerical programs", long talk at the Journées du GDR IM, Lyon, January 21st.
- É. Contejean, "A first Coq mechanized course in relational databases", ANR Typex, Paris, December 17th.
- C. Lelay, "Real Analysis in Coq", LIX PhD seminar, Palaiseau, September 27th.

- G. Melquiond, “Formal Proof of Numerical Properties and Automation”, CEA LSL seminar, Gif-sur-Yvette, February 26th.
- G. Melquiond, “Formal Proof and Interval Arithmetic, a Virtuous Circle”, College of Engineering, University of Texas, El Paso, USA, April 12th.
- G. Melquiond, “What is in Store for Coq.Interval”, ANR Tamadi, Lyon, July 16th.
- G. Melquiond, “Automations for Verifying Floating-point Algorithms in Coq”, the 5th Coq Workshop, Rennes, July 22th.
- S. Conchon, “Cubicle: Design and Implementation of an SMT based Model Checker for Parameterized Systems”, SMT Workshop 2013, Helsinki, Finland, July 9th.
- A. Paskevich, “Deductive Program Verification with Why3”, IRISA seminar, Rennes, October 10th.
- S. Conchon and A. Paskevich, “Savoir-faire et logiciels open source — Toccata”, Open World Forum, “Rencontre Inria-Industrie sur la qualité logicielle”, Montrouge, October 4th.
- C. Dross, “Defining new theories in SMT solvers using fisrt-order axioms with triggers”, CEA LSL seminar, Gif-sur-Yvette, November 12th.

9.2. Interaction with the scientific community

9.2.1. Collective Responsibilities within Inria

- S. Boldo, elected member of the Inria Evaluation Committee. She was in the committee in charge of selecting the Inria permanent researchers (CR2) in Sophia and Saclay.
- S. Boldo was in the committee in charge of upgrading an Inria support staff at the IR level (*ingénieur de recherche*), which is the highest level for support staff.
- S. Boldo, member of the CLFP, *comité local de formation permanente*.
- S. Boldo and A. Charguéraud, members of the committee for the monitoring of PhD students (*commission de suivi des doctorants*).
- S. Boldo, scientific head for Saclay for the MECSI group for networking about computer science popularization inside Inria.
- S. Boldo, member of the popularization committee, *comité de médiation scientifique*, of Inria.

9.2.2. Collective Responsibilities outside 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. France-IOI also provides online exercises in programming and algorithmics — in average, more than 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.
- É. Contejean and C. Marché, nominated members of the *“conseil du laboratoire”* of LRI since April 2010.
- É. Contejean, elected member of the *“section 6 du Comité National de la Recherche Scientifique”* since September 2012.
- C. Lelay, elected member of the *“conseil du laboratoire”* of LRI since November 2011.
- C. Lelay, elected representative of the students at the Doctoral School in Computer Science at University Paris-Sud from November 2011 to November 2013.
- 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 and S. Boldo, members of the *"jury de l'agrégation externe de mathématiques"* as experts in computer science, since 2012.
- G. Melquiond and C. Paulin, members of the *"commission consultative de spécialistes de l'université"*, Section 27, University Paris-Sud since April 2010.
- 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.
- 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.
- A. Paskevich is 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.

9.3. Teaching - Supervision - Juries

9.3.1. Teaching

Licence (L2): "Principes d'interprétation des langages", C. Dross (30h), Université Paris-Sud, France

Licence (L2): "Mathématiques pour l'Informatique" , C. Paulin (64h), M. Iguernelala (10h), A. Tafat (10h), Université Paris-Sud, France

Licence (L3): "Eléments de logique pour l'informatique", C. Paulin (32h), Université Paris-Sud, France

Licence (L3): "Programmation fonctionnelle", C. Dross (4h), Université Paris-Sud, France

Licence Professionnelle «Programmation en environnements répartis» (LP PER): "Programmation concurrente" (L3Pro), A. Paskevich (36h), IUT d'Orsay, Université Paris-Sud, France

Licence (L3): "Programmation fonctionnelle", M. Clochard (15h), ENSIIE, France

Master (M1): "Projet de Programmation", A. Tafat (42h), Université Paris-Sud, France

Master (M1): "Compilation", A. Tafat (28h), Université Paris-Sud, France

Master (M1): "Complément objet", A. Tafat (12h), Polytech, Université Paris-Sud, France

Master (M1): "Compilation", A. Tafat (12h), Polytech, Université Paris-Sud, France

Master (M1): "Projet de Compilation", A. Tafat (12h), Polytech, Université Paris-Sud, France

Master (M2Pro): "XML et Programmation Internet", A. Tafat (13h), Université Paris-Sud, France

Master (M2-agrégation): "Logique", C. Paulin (21h), Université Paris-Sud and ENS Cachan, France

Master Parisien de Recherche en Informatique (MPRI) https://wikimpri.dptinfo.ens-cachan.fr/doku.php: "Automated Deduction" (M2-5), S. Conchon (9h), É. Contejean (3h), Université Paris 7, France.

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), G. Melquiond (12h), Université Paris 7, France.

DUT (Diplôme Universitaire de Technologie): “Structures de données et algorithmique fondamentale” (S1), C. Lelay (38h, “moniteur” position), IUT d’Orsay, Université Paris-Sud, France.

DUT (Diplôme Universitaire de Technologie): “Introduction aux systèmes informatiques” (S1), A. Paskevich (97h), C. Lelay (30h), IUT d’Orsay, Université Paris-Sud, France.

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

Teaching teachers (“Formation de formateurs”) S. Boldo (3h) January 17th

École Jeunes Chercheurs en Programmation (EJCP 2013): J.-C. Filliâtre ,“Deductive Program Verification with Why3” (4h) http://why3.lri.fr/ejcp-2013/.

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

Licence: “INF421: Les bases de l’algorithmique et de la programmation” (L3) et “INF431” (L3), J.-C. Filliâtre (70h), École Polytechnique, France

9.3.2. Supervision

PhD: P. Herms, “Certification of a Tool Chain for Verification of C programs” [12], Univ. Paris-Sud Jan. 14, 2013, C. Marché, B. Monate (CEA-LIST)

PhD: M. Iguernelala, “Strengthening the heart of an SMT-solver: Design and implementation of efficient decision procedures” [13], Univ. Paris-Sud, June 10, 2013, S. Conchon, É. Contejean

PhD: A. Tafat, “Preuve par raffinement de programmes avec pointeurs” [14], Univ. Paris-Sud, Sep. 6, 2013, C. Marché

PhD in progress: C. Dross, “Theories and Techniques for Automated Proof of programs”, since Jan. 2011, C. Marché, A. Paskevich, and industrial supervisors Y. Moy and J. Kanig (AdaCore company)

PhD in progress: A. Mebsout, “SMT-based Model-Checking”, since Sep. 2011, F. Zaidi, S. Conchon

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

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

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

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

9.3.3. Juries

C. Marché: president of the PhD committee of C. Keller, “A Matter of Trust: Skeptical Communication Between Coq and External Provers”, (École Polytechnique, LIX laboratory, June 19, 2013)

C. Marché: reviewer, PhD committee of X. Shi “Certification of an Instruction Set Simulator” (University Grenoble, Verimag laboratory, July 10th, 2013)

C. Marché: president of the PhD committee of E. Tushkanova “Schematic calculi for the analysis of decision procedures” (University Besançon, FEMTO-ST laboratory, July 19th, 2013)

C. Marché: reviewer, PhD committee of H. Debrat “Certification formelle de la correction d’algorithmes de Consensus” (University Nancy, LORIA laboratory, Dec 6th, 2013)

C. Marché: president of HDR committee of S. Gérard “Ingénierie dirigée par les modèles” (University Paris-Sud, LISE laboratory of CEA-LIST, Dec 17th, 2013)

S. Conchon: president of the PhD committee of L. Gerard “Programmer le parallélisme avec des futures en Heptagon un langage synchrone flot de données et étude des réseaux de Kahn en vue d’une compilation synchrone” (University Paris-Sud, ENS, Sept. 25th, 2013)

S. Conchon: president of the PhD committee of J. Cheng “Stochastic Combinatorial Optimization” (University Paris-Sud, Nov. 8th, 2013)

S. Conchon: reviewer, PhD committee of M. Farooque “Automated reasoning techniques as proof-search in sequent calculus” (École Polytechnique, Dec. 19th, 2013)

J.-C. Filliâtre: reviewer, PhD committee of David Miguel Ramalho Pereira “Towards certified program logics for the verification of imperative programs” (Universidade do Porto, April 18, 2013)

J.-C. Filliâtre: reviewer, PhD committee of Jean Fortin “BSP-Why: a Tool for Deductive Verification of BSP Programs” (Université Paris Est, October 14, 2013)

J.-C. Filliâtre: reviewer, PhD committee of Maxime Denès “Formal study of efficient algorithms in linear algebra” (Université de Nice - Sophia Antipolis, November 20, 2013)

9.4. Industrial Dissemination

- As a final result of the Hi-Lite project, the Adacore company (Paris) implemented the new environment Spark2014 for the development of critical Ada software (http://www.spark-2014.org/), the successor of Spark, to be released in 2014. Part of this environment is the tool GnatProve which aims at formal verification. It translates annotated Ada code into the *Why3* intermediate language and then use the *Why3* system to generate proof obligations and discharge them with Alt-Ergo, or other available back-end provers.

9.5. Education, Popularization

- S. Conchon and J.-C. Filliâtre were involved in the writing of a new book supporting the new teaching program for the *“Classes préparatoires aux grandes écoles”* [39].
- S. Boldo and A. Charguéraud belong to the organization committee of the *Castor informatique* http://castor-informatique.fr/, an international competition to present computer science to pupils (from *6ème* to *terminale*). More than 170,000 teenagers played on the more than 30 proposed exercises in November 2013.
- Since April 2008, S. Boldo is member of the editorial committee of the popular science web site)i(: http://interstices.info/.
- S. Boldo, scientific head for Saclay for the MECSI group for networking about computer science popularization inside Inria.
- S. Boldo, member of the popularization committee, *comité de médiation scientifique*, of Inria.
- S. Boldo was among the authors of a document [41] that describes the present and future of popularization at Inria.
- S. Boldo is responsible for a *mission doctorale* for popularization. She is in charge of Li Gong of the LIMSI laboratory: he wrote an Interstices article: http://interstices.info/traduction-automatique-statistique.
- S. Boldo, talk at the *Fête de la science* 2013 for the laboratory, October 11th.
- S. Boldo, talk for a general audience at the Courbevoie library, February 9th
- S. Boldo, talk for teenagers at the lycée Talma de Brunoy, April 23th
- S. Boldo, talk for mathematics teachers at Rocquencourt, June 5th
- S. Boldo, “speed-dating” with teenagers at the Halle Forum, October 18th, in an event called “Science au carré(e)”.
- S. Boldo, article for the French blog celebrating 2013 as the “Mathematics of Planet Earth” year: http://mpt2013.fr/meme-les-ordinateurs-font-des-erreurs/.
- C. Lelay tried the 2013 mathematics test of the scientific Baccalaureate in Coq. After the test, a meeting was organized with some teachers and the produced proofs and results were presented.

- C. Paulin organised at the *Fête de la science* 2013 an action of Labex DigiCosme to promote the new course “Informatique and Sciences du Numériques” in high-school, a few selected projects developed by students as part of their curriculum were exhibited.

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] P. HERMS. , *Certification of a Tool Chain for Deductive Program Verification*, Université Paris Sud - Paris XI, January 2013, http://hal.inria.fr/tel-00789543

[13] M. IGUERNELALA. , *Strengthening the heart of an SMT-solver : Design and implementation of efficient decision procedures*, Université Paris Sud - Paris XI, June 2013, http://hal.inria.fr/tel-00842555

[14] A. TAFAT. , *Preuves par raffinement de programmes avec pointeurs*, Université Paris Sud - Paris XI, September 2013, http://hal.inria.fr/tel-00874679

Articles in International Peer-Reviewed Journals

[15] S. BOLDO, F. CLÉMENT, J.-C. FILLIÂTRE, M. MAYERO, G. MELQUIOND, P. WEIS. *Wave Equation Numerical Resolution: a Comprehensive Mechanized Proof of a C Program*, in "Journal of Automated Reasoning", April 2013, vol. 50, n^{o} 4, pp. 423-456 [*DOI :* 10.1007/s10817-012-9255-4], http://hal.inria.fr/hal-00649240

[16] É. MARTIN-DOREL, G. MELQUIOND, J.-M. MULLER. *Some issues related to double roundings*, in "BIT Numerical Mathematics", December 2013, vol. 53, n^{o} 4, pp. 897-924 [*DOI :* 10.1007/s10543-013-0436-2], http://hal.inria.fr/ensl-00644408

[17] G. MELQUIOND, W. G. NOWAK, P. ZIMMERMANN. *Numerical Approximation of the Masser-Gramain Constant to Four Decimal Digits: delta=1.819...*, in "Mathematics of Computation", 2013, vol. 82, pp. 1235-1246 [*DOI :* 10.1090/S0025-5718-2012-02635-4], http://hal.inria.fr/hal-00644166

Invited Conferences

[18] J.-C. FILLIÂTRE. *Deductive Program Verification*, in "Programming Languages Mentoring Workshop (PLMW 2013)", Rome, Italy, N. FOSTER, P. GARDNER, A. SCHMITT, G. SMITH, P. THIEMAN, T. WRIGSTAD (editors), Nate Foster and Philippa Gardner and Alan Schmitt and Gareth Smith and Peter Thieman and Tobias Wrigstad, January 2013, http://hal.inria.fr/hal-00799190

[19] J.-C. FILLIÂTRE. *One Logic To Use Them All*, in "CADE 24 - the 24th International Conference on Automated Deduction", Lake Placid, NY, United States, M. P. BONACINA (editor), Springer, April 2013, http://hal.inria.fr/hal-00809651

International Conferences with Proceedings

[20] U. ACAR, A. CHARGUÉRAUD, M. RAINEY. *Scheduling Parallel Programs by Work Stealing with Private Deques*, in "PPOPP - 18th ACM SIGPLAN symposium on Principles and practice of parallel programming", Shenzhen, China, ACM New York, NY, USA, February 2013, pp. 219-228, http://hal.inria.fr/hal-00863028

[21] 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, 2014, http://hal.inria.fr/hal-00924156

[22] J. BLANCHETTE, A. PASKEVICH. *TFF1: The TPTP Typed First-Order Form with Rank-1 Polymorphism*, in "CADE - 24th International Conference on Automated Deduction - 2013", Lake Placid, NY, United States, 2013, http://hal.inria.fr/hal-00825086

[23] F. BOBOT, J.-C. FILLIÂTRE, C. MARCHÉ, G. MELQUIOND, A. PASKEVICH. *Preserving User Proofs Across Specification Changes*, in "Fifth Working Conference on Verified Software: Theories, Tools and Experiments", Atherton, United States, E. COHEN, A. RYBALCHENKO (editors), Springer, 2013, vol. 8164, pp. 191-201, http://hal.inria.fr/hal-00875395

[24] 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, November 2013, http://hal.inria.fr/hal-00910135

[25] S. BOLDO. *How to Compute the Area of a Triangle: a Formal Revisit*, in "21st IEEE International Symposium on Computer Arithmetic", Austin, TX, United States, A. NANNARELLI, P.-M. SEIDEL, P. T. P. TANG (editors), IEEE, April 2013, pp. 91-98 [*DOI :* 10.1109/ARITH.2013.29], http://hal.inria.fr/hal-00790071

[26] S. BOLDO, J.-H. JOURDAN, X. LEROY, G. MELQUIOND. *A Formally-Verified C Compiler Supporting Floating-Point Arithmetic*, in "Arith - 21st IEEE Symposium on Computer Arithmetic", Austin, United States, A. NANNARELLI, P.-M. SEIDEL, P. T. P. TANG (editors), IEEE, 2013, pp. 107-115, http://hal.inria.fr/hal-00743090

[27] A. CHARGUÉRAUD. *Pretty-Big-Step Semantics*, in "22nd European Symposium on Programming (ESOP)", Rome, Italy, M. FELLEISEN, P. GARDNER (editors), Springer, March 2013, http://hal.inria.fr/hal-00798227

[28] 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, http://hal.inria.fr/hal-00920955

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

[30] S. CONCHON, A. GOEL, S. KRSTIĆ, A. MEBSOUT, F. ZAÏDI. *Invariants for Finite Instances and Beyond*, in "Formal Methods in Computer-Aided Design (FMCAD)", Portland, Oregon, United States, October 2013, pp. 61-68 [*DOI :* 10.1109/FMCAD.2013.6679392], http://hal.inria.fr/hal-00924640

[31] S. CONCHON, A. MEBSOUT, F. ZAÏDI. *Vérification de systèmes paramétrés avec Cubicle*, in "JFLA - Journées francophones des langages applicatifs - 2013", Aussois, France, D. POUS, C. TASSON (editors), Damien Pous and Christine Tasson, February 2013, http://hal.inria.fr/hal-00778832

[32] R. EL SIBAÏE, J.-C. FILLIÂTRE. *combine : une bibliothèque OCaml pour la combinatoire*, in "JFLA - Journées francophones des langages applicatifs", Aussois, France, D. POUS, C. TASSON (editors), Damien Pous and Christine Tasson, February 2013, http://hal.inria.fr/hal-00779431

[33] J.-C. FILLIÂTRE, A. PASKEVICH. *Why3 – Where Programs Meet Provers*, in "ESOP'13 22nd European Symposium on Programming", Rome, Italy, LNCS, Springer, March 2013, vol. 7792, http://hal.inria.fr/hal-00789533

[34] D. ISHII, G. MELQUIOND, S. NAKAJIMA. *Inductive Verification of Hybrid Automata with Strongest Post-condition Calculus*, in "iFM - 10th International Conference on Integrated Formal Methods - 2013", Turku, Finland, E. B. JOHNSEN, L. PETRE (editors), Lecture Notes in Computer Science, Springer, 2013, vol. 7940, pp. 139-153 [*DOI :* 10.1007/978-3-642-38613-8_10], http://hal.inria.fr/hal-00806701

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

[36] C. MARCHÉ, A. TAFAT. *Calcul de plus faible précondition, revisité en Why3*, in "JFLA - Journées francophones des langages applicatifs - 2013", Aussois, France, D. POUS, C. TASSON (editors), Damien Pous and Christine Tasson, February 2013, http://hal.inria.fr/hal-00778791

Scientific Books (or Scientific Book chapters)

[37] S. BLAZY, C. PAULIN-MOHRING, D. PICHARDIE (editors). , *Interactive Theorem Proving - 4th International Conference, ITP 2013, Rennes, France, July 22-26, 2013. Proceedings.*, Lecture Notes in Computer Science, Springer, 2013, vol. 7998, 500 p. [*DOI :* 10.1007/978-3-642-39634-2], http://hal.inria.fr/hal-00908865

[38] E. BEFFARA, J.-D. BOISSONNAT, S. BOLDO, F. CHAZAL, E. GIOAN, M. MARTEL, C. PAUL, G. MELQUIOND, J. RAMIREZ ALFONSIN, L. VAUX, M. YVINEC. , P. LANGLOIS (editor), *Informatique Mathématique : une photographie en 2013*, Etudes, Presses Universitaires de Perpignan, April 2013, 283 p. , http://hal.inria.fr/lirmm-00835506

[39] J. COURANT, M. DE FALCO, S. GONNORD, J.-C. FILLIÂTRE, S. CONCHON, G. DOWEK, B. WACK. , *Informatique pour tous en classes préparatoires aux grandes écoles : Manuel d'algorithmique et programmation structurée avec Python*, Eyrolles, 2013, 408 p. , http://hal.inria.fr/hal-00880268

Research Reports

[40] U. ACAR, A. CHARGUÉRAUD, S. MULLER, M. RAINEY. , *Atomic Read-Modify-Write Operations are Unnecessary for Shared-Memory Work Stealing*, September 2013, http://hal.inria.fr/hal-00910130

[41] A. ROUSSEAU, A. DARNAUD, B. GOGLIN, C. ACHARIAN, C. LEININGER, C. GODIN, C. HOLIK, C. KIRCHNER, D. RIVES, E. DARQUIE, E. KERRIEN, F. NEYRET, F. MASSEGLIA, F. DUFOUR, G. BERRY, G. DOWEK, H. ROBAK, H. XYPAS, I. ILLINA, I. GNAEDIG, J. JONGWANE, J. EHREL, L. VIENNOT, L. GUION, L. CALDERAN, L. KOVACIC, M. COLLIN, M.-A. ENARD, M.-H. COMTE, M. QUINSON, M. OLIVI, M. GIRAUD, M. DORÉMUS, M. OGOUCHI, M. DROIN, N. LACAUX, N. ROUGIER, N. ROUSSEL, P. GUITTON, P. PETERLONGO, R.-M. CORNUS, S. VANDERMEERSCH, S. MAHEO, S. LEFEBVRE, S. BOLDO, T. VIÉVILLE, V. POIREL, A. CHABREUIL, A. FISCHER, C. FARGE, C. VADEL, I. ASTIC, J.-P. DUMONT, L. FÉJOZ, P. RAMBERT, P. PARADINAS, S. DE QUATREBARBES, S. LAURENT. , *Médiation Scientifique : une facette de nos métiers de la recherche*, March 2013, 34 p. , http://hal.inria.fr/hal-00804915

Other Publications

[42] F. BOBOT, J.-C. FILLIÂTRE, C. MARCHÉ, G. MELQUIOND, A. PASKEVICH. , *The Why3 platform 0.81*, March 2013, Tutorial and Reference Manual, http://hal.inria.fr/hal-00822856

[43] S. BOLDO. , *How to Compute the Area of a Triangle: a Formal Revisit with a Tighter Error Bound*, 2013, http://hal.inria.fr/hal-00862653

[44] S. BOLDO, J.-H. JOURDAN, X. LEROY, G. MELQUIOND. , *A Formally-Verified C Compiler Supporting Floating-Point Arithmetic*, 2013, http://hal.inria.fr/hal-00862689

[45] S. BOLDO, C. LELAY, G. MELQUIOND. , *Coquelicot: A User-Friendly Library of Real Analysis for Coq*, September 2013, http://hal.inria.fr/hal-00860648

[46] S. BOLDO, C. LELAY, G. MELQUIOND. , *Formalization of Real Analysis: A Survey of Proof Assistants and Libraries*, April 2013, http://hal.inria.fr/hal-00806920

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

[48] C. DROSS, S. CONCHON, J. KANIG, A. PASKEVICH. , *Adding Decision Procedures to SMT Solvers using Axioms with Triggers*, 2013, http://hal.inria.fr/hal-00915931

[49] J.-C. FILLIÂTRE, L. GONDELMAN, A. PASKEVICH. , *The Spirit of Ghost Code*, October 2013, http://hal.inria.fr/hal-00873187

[50] É. MARTIN-DOREL, G. HANROT, M. MAYERO, L. THÉRY. , *Formally verified certificate checkers for hardest-to-round computation*, December 2013, http://hal.inria.fr/hal-00919498

References in notes

[51] 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

[52] 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

[53] 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

[54] 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

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

[56] 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

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

[58] 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

[59] 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

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

[61] 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, September 1999, pp. 369–387

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

[63] 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

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

[65] 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/

[66] 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/

[67] 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

[68] 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

[69] 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/

[70] 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/

[71] 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

[72] L. BURDY, Y. CHEON, D. R. COK, M. D. ERNST, J. R. KINIRY, G. T. LEAVENS, K. R. M. LEINO, E. POLL. *An overview of JML tools and applications*, in "International Journal on Software Tools for Technology Transfer (STTT)", June 2005, vol. 7, n^{o} 3, pp. 212–232

[73] 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

[74] 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

[75] 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

[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] B. COOK. , *Static Driver Verifier*

[81] 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

[82] 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

[83] 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/

[84] 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

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

[86] 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

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

[88] 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

[89] 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

[90] 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

[91] 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

[92] 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

[93] 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

[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

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

[100] 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/

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

[102] 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

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

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

[105] 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

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

[107] 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/

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

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

[110] 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

[111] 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

[112] 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

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

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

[115] 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

[116] 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

[117] 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

[118] 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/

[119] 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° 2, pp. 242–253, http://hal.inria.fr/inria-00533968/en/

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