IN PARTNERSHIP WITH:
Université Rennes 1

Ecole normale supérieure de Cachan

Activity Report 2014

Project-Team CELTIQUE

Software certification with semantic analysis

IN COLLABORATION WITH: Institut de recherche en informatique et systèmes aléatoires (IRISA)

RESEARCH CENTER
Rennes - Bretagne-Atlantique

THEME
Proofs and Verification

Table of contents

Project-Team CELTIQUE

Keywords: Programming Languages, Static Analysis, Abstract Interpretation, Security, Interactive Theorem Proving, Formal Methods

Creation of the Project-Team: 2009 July 01.

1. Members

Research Scientists
Thomas Jensen [Team leader, Inria, Senior Researcher, HdR]
Frédéric Besson [Inria, Researcher]
Alan Schmitt [Inria, Researcher, HdR]

Faculty Members
Sandrine Blazy [Univ. Rennes I, Professor, HdR]
David Cachera [Normale Sup Rennes, Associate Professor, HdR]
Delphine Demange [Univ. Rennes I, Associate Professor]
Thomas Genet [Univ. Rennes I, Associate Professor, HdR]
Barbara Kordy [INSA Rennes, Associate Professor, from Sep 2014]
David Pichardie [Normale Sup Rennes, Professor, HdR]

Engineer
Laurent Guillo [CNRS]

Post-Doctoral Fellows
Petar Maksimovic [Inria]
Daniela Petrisan [Inria, until Oct 2014, granted by ANR PiCoq project]

Administrative Assistant
Lydie Mabil [Inria]

Others
Oana Fabiana Andreescu [Prove&Run]
Martin Bodin [Univ. Rennes I]
Pauline Bolignano [Prove&Run]
David Buhler [CEA]
Vincent Laporte [Univ. Rennes I]
Colas Le Guernic [DGA]
André Oliveira Maroneze [Univ. Rennes I, until Aug 2014]
Stéphanie Riaud [DGA]
Yann Salmon [Univ. Rennes I]
Pierre Wilke [Univ. Rennes I]
Yannick Zakowski [Normale Sup Rennes]
Jean-Christophe Zapalowicz [Inria, until Nov 2014]

2. Overall Objectives

2.1. Project overview

The overall goal of the CELTIQUE project is to improve the security and reliability of software with semantic certification that attests to its well-behavedness. The semantic analyses extract approximate but sound descriptions of software behaviour from which a proof of security can be constructed. The analyses of relevance include numerical data flow analysis, control flow analysis for higher-order languages, alias and points-to analysis for heap structure manipulation, and various kinds of information flow analysis.

To achieve this goal, the project conducts work on improving semantic analysis techniques, as well as work on using proof assistants such as Coq to develop and prove properties of these analyses. We are applying such techniques to a variety of source languages, including Java, C, and JavaScript. We also study how these techniques apply to low-level languages, and how they can be combined with certified compilation.

We target three application domains: Java software for small devices (in particular smart cards and mobile telephones), embedded C programs, and web applications.

CELTIQUE is a joint project with the CNRS, the University of Rennes 1 and ENS Rennes.

3. Research Program

3.1. Static program analysis

Static program analysis is concerned with obtaining information about the run-time behaviour of a program without actually running it. This information may concern the values of variables, the relations among them, dependencies between program values, the memory structure being built and manipulated, the flow of control, and, for concurrent programs, synchronisation among processes executing in parallel. Fully automated analyses usually render approximate information about the actual program behaviour. The analysis is correct if the information includes all possible behaviour of a program. Precision of an analysis is improved by reducing the amount of information describing spurious behaviour that will never occur.

Static analysis has traditionally found most of its applications in the area of program optimisation where information about the run-time behaviour can be used to transform a program so that it performs a calculation faster and/or makes better use of the available memory resources. The last decade has witnessed an increasing use of static analysis in software verification for proving invariants about programs. The Celtique project is mainly concerned with this latter use. Examples of static analysis include:

- Data-flow analysis as it is used in optimising compilers for imperative languages. The properties can either be approximations of the values of an expression ("the value of variable x is greater than 0" or x is equal to y at this point in the program") or more intensional information about program behaviour such as "this variable is not used before being re-defined" in the classical "dead-variable" analysis [72].
- Analyses of the memory structure includes shape analysis that aims at approximating the data structures created by a program. Alias analysis is another data flow analysis that finds out which variables in a program addresses the same memory location. Alias analysis is a fundamental analysis for all kinds of programs (imperative, object-oriented) that manipulate state, because alias information is necessary for the precise modelling of assignments.
- Control flow analysis will find a safe approximation to the order in which the instructions of a program are executed. This is particularly relevant in languages where parameters or functions can be passed as arguments to other functions, making it impossible to determine the flow of control from the program syntax alone. The same phenomenon occurs in object-oriented languages where it is the class of an object (rather than the static type of the variable containing the object) that determines which method a given method invocation will call. Control flow analysis is an example of an analysis whose information in itself does not lead to dramatic optimisations (although it might enable in-lining of code) but is necessary for subsequent analyses to give precise results.

Static analysis possesses strong **semantic foundations**, notably abstract interpretation [54], that allow to prove its correctness. The implementation of static analyses is usually based on well-understood constraint-solving techniques and iterative fixpoint algorithms. In spite of the nice mathematical theory of program analysis and the solid algorithmic techniques available one problematic issue persists, *viz.*, the *gap* between the analysis that is proved correct on paper and the analyser that actually runs on the machine. While this gap might be small for toy languages, it becomes important when it comes to real-life languages for which the implementation and maintenance of program analysis tools become a software engineering task. A *certified static analysis* is an analysis that has been formally proved correct using a proof assistant.

In previous work we studied the benefit of using abstract interpretation for developing **certified static analyses** [52], [75]. The development of certified static analysers is an ongoing activity that will be part of the Celtique project. We use the Coq proof assistant which allows for extracting the computational content of a constructive proof. A Caml implementation can hence be extracted from a proof of existence, for any program, of a correct approximation of the concrete program semantics. We have isolated a theoretical framework based on abstract interpretation allowing for the formal development of a broad range of static analyses. Several case studies for the analysis of Java byte code have been presented, notably a memory usage analysis [53]. This work has recently found application in the context of Proof Carrying Code and have also been successfully applied to particular form of static analysis based on term rewriting and tree automata [5].

3.1.1. Static analysis of Java

Precise context-sensitive control-flow analysis is a fundamental prerequisite for precisely analysing Java programs. Bacon and Sweeney's Rapid Type Analysis (RTA) [45] is a scalable algorithm for constructing an initial call-graph of the program. Tip and Palsberg [80] have proposed a variety of more precise but scalable call graph construction algorithms *e.g.,* MTA, FTA, XTA which accuracy is between RTA and 0'CFA. All those analyses are not context-sensitive. As early as 1991, Palsberg and Schwartzbach [73], [74] proposed a theoretical parametric framework for typing object-oriented programs in a context-sensitive way. In their setting, context-sensitivity is obtained by explicit code duplication and typing amounts to analysing the expanded code in a context-insensitive manner. The framework accommodates for both call-contexts and allocation-contexts.

To assess the respective merits of different instantiations, scalable implementations are needed. For Cecil and Java programs, Grove *et al.,* [61], [60] have explored the algorithmic design space of contexts for benchmarks of significant size. Latter on, Milanova *et. al.,* [67] have evaluated, for Java programs, a notion of context called *object-sensitivity* which abstracts the call-context by the abstraction of the `this` pointer. More recently, Lhotak and Hendren [65] have extended the empiric evaluation of object-sensitivity using a BDD implementation allowing to cope with benchmarks otherwise out-of-scope. Besson and Jensen [49] proposed to use DATALOG in order to specify context-sensitive analyses. Whaley and Lam [81] have implemented a context-sensitive analysis using a BDD-based DATALOG implementation.

Control-flow analyses are a prerequisite for other analyses. For instance, the security analyses of Livshits and Lam [66] and the race analysis of Naik, Aiken [68] and Whaley [69] both heavily rely on the precision of a control-flow analysis.

Control-flow analysis allows to statically prove the absence of certain run-time errors such as "message not understood" or cast exceptions. Yet it does not tackle the problem of "null pointers". Fahnrich and Leino [57] propose a type-system for checking that after object creation fields are non-null. Hubert, Jensen and Pichardie have formalised the type-system and derived a type-inference algorithm computing the most precise typing [64]. The proposed technique has been implemented in a tool called NIT [63]. Null pointer detection is also done by bug-detection tools such as FindBugs [63]. The main difference is that the approach of findbugs is neither sound nor complete but effective in practice.

3.1.2. Quantitative aspects of static analysis

Static analyses yield qualitative results, in the sense that they compute a safe over-approximation of the concrete semantics of a program, w.r.t. an order provided by the abstract domain structure. Quantitative aspects of static analysis are two-sided: on one hand, one may want to express and verify (compute) quantitative properties of programs that are not captured by usual semantics, such as time, memory, or energy consumption; on the other hand, there is a deep interest in quantifying the precision of an analysis, in order to tune the balance between complexity of the analysis and accuracy of its result.

The term of quantitative analysis is often related to probabilistic models for abstract computation devices such as timed automata or process algebras. In the field of programming languages which is more specifically addressed by the Celtique project, several approaches have been proposed for quantifying resource usage: a non-exhaustive list includes memory usage analysis based on specific type systems [62], [44], linear

logic approaches to implicit computational complexity [46], cost model for Java byte code [40] based on size relation inference, and WCET computation by abstract interpretation based loop bound interval analysis techniques [55].

We have proposed an original approach for designing static analyses computing program costs: inspired from a probabilistic approach [76], a quantitative operational semantics for expressing the cost of execution of a program has been defined. Semantics is seen as a linear operator over a dioid structure similar to a vector space. The notion of long-run cost is particularly interesting in the context of embedded software, since it provides an approximation of the asymptotic behaviour of a program in terms of computation cost. As for classical static analysis, an abstraction mechanism allows to effectively compute an over-approximation of the semntics, both in terms of costs and of accessible states [51]. An example of cache miss analysis has been developed within this framework [79].

3.2. Software certification

The term "software certification" has a number of meanings ranging from the formal proof of program correctness via industrial certification criteria to the certification of software developers themselves! We are interested in two aspects of software certification:

- industrial, mainly process-oriented certification procedures
- software certificates that convey semantic information about a program

Semantic analysis plays a role in both varieties.

Criteria for software certification such as the Common criteria or the DOA aviation industry norms describe procedures to be followed when developing and validating a piece of software. The higher levels of the Common Criteria require a semi-formal model of the software that can be refined into executable code by traceable refinement steps. The validation of the final product is done through testing, respecting criteria of coverage that must be justified with respect to the model. The use of static analysis and proofs has so far been restricted to the top level 7 of the CC and has not been integrated into the aviation norms.

3.2.1. Process-oriented software certification

The testing requirements present in existing certification procedures pose a challenge in terms of the automation of the test data generation process for satisfying functional and structural testing requirements. For example, the standard document which currently governs the development and verification process of software in airborne system (DO-178B) requires the coverage of all the statements, all the decisions of the program at its higher levels of criticality and it is well-known that DO-178B structural coverage is a primary cost driver on avionics project. Although they are widely used, existing marketed testing tools are currently restricted to test coverage monitoring and measurements [1] but none of these tools tries to find the test data that can execute a given statement, branch or path in the source code. In most industrial projects, the generation of structural test data is still performed manually and finding automatic methods for this problem remains a challenge for the test community. Building automatic test case generation methods requires the development of precise semantic analysis which have to scale up to software that contains thousands of lines of code.

Static analysis tools are so far not a part of the approved certification procedures. For this to change, the analysers themselves must be accepted by the certification bodies in a process called “Qualification of the tools” in which the tools are shown to be as robust as the software it will certify. We believe that proof assistants have a role to play in building such certified static analysis as we have already shown by extracting provably correct analysers for Java byte code.

[1] Coverage monitoring answers to the question: what are the statements or branches covered by the test suite ? While coverage measurements answers to: how many statements or branches have been covered ?

3.2.2. Semantic software certificates

The particular branch of information security called "language-based security" is concerned with the study of programming language features for ensuring the security of software. Programming languages such as Java offer a variety of language constructs for securing an application. Verifying that these constructs have been used properly to ensure a given security property is a challenge for program analysis. One such problem is confidentiality of the private data manipulated by a program and a large group of researchers have addressed the problem of tracking information flow in a program in order to ensure that *e.g.*, a credit card number does not end up being accessible to all applications running on a computer [78], [48]. Another kind of problems concern the way that computational resources are being accessed and used, in order to ensure that a given access policy is being implemented correctly and that a given application does not consume more resources that it has been allocated. Members of the Celtique team have proposed a verification technique that can check the proper use of resources of Java applications running on mobile telephones [50]. **Semantic software certificates** have been proposed as a means of dealing with the security problems caused by mobile code that is downloaded from foreign sites of varying trustworthiness and which can cause damage to the receiving host, either deliberately or inadvertently. These certificates should contain enough information about the behaviour of the downloaded code to allow the code consumer to decide whether it adheres to a given security policy.

Proof-Carrying Code (PCC) [70] is a technique to download mobile code on a host machine while ensuring that the code adheres to a specified security policy. The key idea is that the code producer sends the code along with a proof (in a suitably chosen logic) that the code is secure. Upon reception of the code and before executing it, the consumer submits the proof to a proof checker for the logic. Our project focus on two components of the PCC architecture: the proof checker and the proof generator.

In the basic PCC architecture, the only components that have to be trusted are the program logic, the proof checker of the logic, and the formalization of the security property in this logic. Neither the mobile code nor the proposed proof—and even less the tool that generated the proof—need be trusted.

In practice, the *proof checker* is a complex tool which relies on a complex Verification Condition Generator (VCG). VCGs for real programming languages and security policies are large and non-trivial programs. For example, the VCG of the Touchstone verifier represents several thousand lines of C code, and the authors observed that "there were errors in that code that escaped the thorough testing of the infrastructure" [71]. Many solutions have been proposed to reduce the size of the trusted computing base. In the *foundational proof carrying code* of Appel and Felty [43], [42], the code producer gives a direct proof that, in some "foundational" higher-order logic, the code respects a given security policy. Wildmoser and Nipkow [83], [82]. prove the soundness of a *weakest precondition* calculus for a reasonable subset of the Java bytecode. Necula and Schneck [71] extend a small trusted core VCG and describe the protocol that the untrusted verifier must follow in interactions with the trusted infrastructure.

One of the most prominent examples of software certificates and proof-carrying code is given by the Java byte code verifier based on *stack maps*. Originally proposed under the term “lightweight Byte Code Verification” by Rose [77], the techniques consists in providing enough typing information (the stack maps) to enable the byte code verifier to check a byte code in one linear scan, as opposed to inferring the type information by an iterative data flow analysis. The Java Specification Request 202 provides a formalization of how such a verification can be carried out.

Inspired by this, Albert *et al.* [41] have proposed to use static analysis (in the form of abstract interpretation) as a general tool in the setting of mobile code security for building a proof-carrying code architecture. In their *abstraction-carrying code* framework, a program comes equipped with a machine-verifiable certificate that proves to the code consumer that the downloaded code is well-behaved.

3.2.3. Certified static analysis

In spite of the nice mathematical theory of program analysis (notably abstract interpretation) and the solid algorithmic techniques available one problematic issue persists, *viz.*, the *gap* between the analysis that is proved correct on paper and the analyser that actually runs on the machine. While this gap might be small for

toy languages, it becomes important when it comes to real-life languages for which the implementation and maintenance of program analysis tools become a software engineering task.

A *certified static analysis* is an analysis whose implementation has been formally proved correct using a proof assistant. Such analysis can be developed in a proof assistant like Coq [39] by programming the analyser inside the assistant and formally proving its correctness. The Coq extraction mechanism then allows for extracting a Caml implementation of the analyser. The feasibility of this approach has been demonstrated in [7].

We also develop this technique through certified reachability analysis over term rewriting systems. Term rewriting systems are a very general, simple and convenient formal model for a large variety of computing systems. For instance, it is a very simple way to describe deduction systems, functions, parallel processes or state transition systems where rewriting models respectively deduction, evaluation, progression or transitions. Furthermore rewriting can model every combination of them (for instance two parallel processes running functional programs).

Depending on the computing system modelled using rewriting, reachability (and unreachability) permits to achieve some verifications on the system: respectively prove that a deduction is feasible, prove that a function call evaluates to a particular value, show that a process configuration may occur, or that a state is reachable from the initial state. As a consequence, reachability analysis has several applications in equational proofs used in the theorem provers or in the proof assistants as well as in verification where term rewriting systems can be used to model programs.

For proving unreachability, i.e. safety properties, we already have some results based on the over-approximation of the set of reachable terms [58], [59]. We defined a simple and efficient algorithm [56] for computing exactly the set of reachable terms, when it is regular, and construct an over-approximation otherwise. This algorithm consists of a *completion* of a *tree automaton*, taking advantage of the ability of tree automata to finitely represent infinite sets of reachable terms.

To certify the corresponding analysis, we have defined a checker guaranteeing that a tree automaton is a valid fixpoint of the completion algorithm. This consists in showing that for all term recognised by a tree automaton all his rewrites are also recognised by the same tree automaton. This checker has been formally defined in Coq and an efficient Ocaml implementation has been automatically extracted [5]. This checker is now used to certify all analysis results produced by the regular completion tool as well as the optimised version of [47].

4. New Software and Platforms

4.1. Javalib

Participants: Frédéric Besson [correspondant], David Pichardie, Pierre Vittet, Laurent Guillo.

Javalib is an efficient library to parse Java .class files into OCaml data structures, thus enabling the OCaml programmer to extract information from class files, to manipulate and to generate valid .class files.
See also the web page http://sawja.inria.fr/.

- Version: 2.3
- Programming language: Ocaml

4.2. SAWJA

Participants: Frédéric Besson [correspondant], David Pichardie, Pierre Vittet, Laurent Guillo.

Sawja is a library written in OCaml, relying on Javalib to provide a high level representation of Java bytecode programs. It name comes from Static Analysis Workshop for JAva. Whereas Javalib is dedicated to isolated classes, Sawja handles bytecode programs with their class hierarchy and with control flow algorithms.

Moreover, Sawja provides some stackless intermediate representations of code, called JBir and A3Bir. The transformation algorithm, common to these representations, has been formalized and proved to be semantics-preserving.

See also the web page http://sawja.inria.fr/.

- Version: 1.5
- Programming language: Ocaml

4.3. Jacal

Participants: Frédéric Besson [correspondant], Thomas Jensen, David Pichardie, Delphine Demange.

Static program analysis, Javacard, Certification, AFSCM

`Jacal` is a JAvaCard AnaLyseur developed on top of the SAWJA (see Section 4.2) platform. This proprietary software verifies automatically that Javacard programs conform with the security guidelines issued by the AFSCM (Association Française du Sans Contact Mobile). `Jacal` is based on the theory of abstract interpretation and combines several object-oriented and numeric analyses to automatically infer sophisticated invariants about the program behaviour. The result of the analysis is thereafter harvest to check that it is sufficient to ensure the desired security properties.

4.4. Timbuk

Participant: Thomas Genet [correspondant].

Timbuk is a library of OCAML functions for manipulating tree automata. More precisely, Timbuk deals with finite bottom-up tree automata (deterministic or not). This library provides the classical operations over tree automata (intersection, union, complement, emptiness decision) as well as exact or approximated sets of terms reachable by a given term rewriting system. This last operation can be certified using a checker extracted from a Coq specification. The checker is now part of the Timbuk distribution. Timbuk distribution now also provides a CounterExample Guided Abstraction Refinement (CEGAR) tool for tree automata completion. The CEGAR part is based on the Buddy BDD library. Timbuk also provides an implementation of Lattice Tree Automata to (efficiently) represent built-in values such as integers, strings, etc. in recognized tree languages. See also the web page http://www.irisa.fr/celtique/genet/timbuk/.

- Version: 3.1
- Programming language: Ocaml

4.5. JSCert

Participants: Alan Schmitt [correspondant], Martin Bodin.

The JSCert project aims to really understand JavaScript. JSCert itself is a mechanised specification of JavaScript, written in the Coq proof assistant, which closely follows the ECMAScript 5 English standard. JSRef is a reference interpreter for JavaScript in OCAML, which has been proved correct with respect to JSCert and tested with the Test 262 test suite.

We plan to build other verification and analysis projects on top of JSCert and JSRef, in particular the certification of derivations in program logics or static analyses.

This project is an ongoing collaboration between Inria and Imperial College. More information, including the source code, is available at http://jscert.org/.

5. New Results

5.1. Browser randomization against web tracking

Participants: Frédéric Besson, Thomas Jensen.

We have investigated different approaches for dynamically tracking information flows in order to improve web browser security. We have identified the problem of stateless web tracking (fingerprinting) and have proposed a novel approach to hybrid information flow monitoring by tracking the knowledge about secret variables using logical formulae. In a follow-up work we investigated how to enforce browser anonymity in the presence of finger-printing web trackers. One way to protect the users' privacy is to make them switch between different machine and browser configurations. We propose a formalisation of this privacy enforcement mechanism. We use information-theoretic channels to model the knowledge of the tracker and the fingerprinting program, and show how to synthesise a randomisation mechanism that defines the distribution of configurations for each user. This mechanism provides a strong guarantee of *privacy* (the probability of identifying the user is bounded by a given threshold) while maximising *usability* (the user switches to other configurations rarely). To find an optimal solution, we express the enforcement problem of randomisation by a linear program. We investigate and compare several approaches to randomisation and find that more efficient privacy enforcement would often provide lower usability. Finally, we relax the requirement of knowing the fingerprinting program in advance, by proposing a randomisation mechanism that guarantees privacy for an arbitrary program.

5.2. Static analysis of functional programs using tree automata and term rewriting

Participants: Thomas Genet, Barbara Kordy, Yann Salmon.

We develop a specific theory and the related tools for analyzing programs whose semantics is defined using term rewriting systems. The analysis principle is based on regular approximations of infinite sets of terms reachable by rewriting. The tools we develop use, so-called, Tree Automata Completion to compute a tree automaton recognizing a superset of all reachable terms. This over-approximation is then used to prove properties on the program by showing that some "bad" terms, encoding dangerous or problematic configurations, are not in the superset and thus not reachable. This is a specific form of, so-called, Regular Tree Model Checking. However, when dealing with infinite-state systems, Regular Tree Model Checking approaches may have some difficulties to represent infinite sets of data. We proposed Lattice Tree Automata, an extended version of tree automata to represent complex data domains and their related operations in an efficient manner. Moreover, we introduce a new completion-based algorithm for computing the possibly infinite set of reachable states in a finite amount of time. This algorithm is independent of the lattice making it possible to seamlessly plug abstract domains into a Regular Tree Model Checking algorithm. These results are part of Valérie Murat's PhD thesis [13]. Now, we aim at applying this technique to the static analysis of programming languages whose semantics is based on terms, like functional programming languages. We already shown that static analysis of first order functional programs can be automated using tree automata completion [28]. Now, one of the objective is to lift those results to the static analysis of higher-order functions. This was precisely the purpose of Yann Salmon's visit to Pr. Luke Ong. Barbara Kordy who joined Celtique in September 2014 is also going to work on this subject.

5.3. Certified JavaScript

Participants: Martin Bodin, Alan Schmitt.

We have completed our first milestone in the development of a certified JavaScript semantics. We have finished a first version of JSCert, a formalization of the current ECMA standard in the Coq proof assistant, and JSRef, a reference interpreter for JavaScript extracted from Coq to OCaml. We have also given a Coq proof that JSRef is correct with respect to JSCert and assessed JSRef using test262, the ECMA conformance test suite. Our methodology ensures that JSCert is a comparatively accurate formulation of the English standard. We have demonstrated that modern techniques of mechanized specification can handle the complexity of JavaScript. This result, obtained in the setting of a collaboration with Philippa Gardner and Sergio Maffeis of Imperial College, and Arthur Charguéraud of Inria Saclay, have been published in the conference Principles of Programming Languages [25].

5.4. SawjaCard: a static analysis tool for certifying Java Card applications

Participants: Frédéric Besson, Thomas Jensen, David Pichardie, Delphine Demange.

We have transfered to the FIME company a static analysis tool for certifying *Java Card* applications, according to security rules defined by the smart card industry. *Java Card* is a dialect of Java designed for programming multi-application smart cards and the tool, called *SawjaCard*, has been specialised for the particular *Java Card* programming patterns. The tool is built around a static analysis engine which uses a combination of numeric and heap analysis. It includes a model of the *Java Card* libraries and the *Java Card* firewall. The tool has been evaluated on a series of industrial applets and is shown to automate a substantial part of the validation process [21].

5.5. Semantics for C programs

Participants: Frédéric Besson, Sandrine Blazy, Pierre Wilke.

Real life C programs are often written using C dialects which, for the ISO C standard, have undefined behaviours. In particular, according to the ISO C standard, reading an uninitialised variable has an undefined behaviour and low-level pointer operations are implementation defined. We propose a formal semantics which gives a well-defined meaning to those behaviours for the C dialect of the CompCert compiler. Our semantics builds upon a novel memory model leveraging a notion of symbolic values. Symbolic values are used by the semantics to delay the evaluation of operations and are normalised lazily to genuine values when needed. We show that the most precise normalisation is computable and that a slightly relaxed normalisation can be efficiently implemented using an SMT solver. The semantics is executable and our experiments show that the enhancements of our semantics are mandatory to give a meaning to low-levels idioms such as those found in the allocation functions of a C standard library [21].

5.6. Fast inference of polynomial invariants

Participants: David Cachera, Thomas Jensen.

We have developed our static analysis techniques for computing polynomial invariants for imperative programs. The analysis is derived from an abstract interpretation of a backwards semantics, and computes preconditions for equalities of the form g = 0 to hold at the end of execution. A distinguishing feature of the technique is that it computes polynomial loop invariants without resorting to Gro¨bner base computations. The analysis uses remainder computations over parameterized polynomials in order to handle conditionals and loops efficiently. The algorithm can analyze and find a large majority of loop invariants reported previously in the literature, and executes significantly faster than implementations using Gro¨bner bases [15].

5.7. Quantitative analysis of security

Participant: Barbara Kordy.

Graphical models for security is a young but rapidly growing research field. Security models based on graphs combine intuitive, visual representation with rigorous, mathematical foundations. In [30] we address the growing need of performing meaningful probabilistic analysis of security using graphical models. We propose a framework that integrates the modeling technique of attack–defense trees with probabilistic information expressed in terms of Bayesian networks. This allows us to perform probabilistic evaluation of attack–defense scenarios involving dependent actions. To improve the efficiency of our computations, we make use of inference algorithms from Bayesian networks and encoding techniques from constraint reasoning. We discuss the algebraic theory underlying our framework and point out several generalizations which are possible thanks to the use of semiring theory

5.8. Formal Verification of an SSA-Based Middle-End for CompCert

Participants: Delphine Demange, David Pichardie.

CompCert is a formally verified compiler that generates compact and efficient code for a large subset of the C language. However, CompCert foregoes using SSA, an intermediate representation employed by many compilers that enables writing simpler, faster optimizers. In fact, it has remained an open problem to verify formally an SSA-based compiler. We report in [14] on a formally verified, SSA-based middle-end for CompCert. In addition to providing a formally verified SSA-based middle-end, we address two problems raised by Leroy in 2009: giving an intuitive formal semantics to SSA, and leveraging its global properties to reason locally about program optimizations. Joint work with Gilles Barthe.

5.9. A verified information-flow architecture

Participants: Delphine Demange, David Pichardie.

SAFE is a clean-slate design for a highly secure computer system, with pervasive mechanisms for tracking and limiting information flows. At the lowest level, the SAFE hardware supports fine-grained programmable tags, with efficient and flexible propagation and combination of tags as instructions are executed. The operating system virtualizes these generic facilities to present an information-flow abstract machine that allows user programs to label sensitive data with rich confidentiality policies. We present a formal, machine-checked model of the key hardware and software mechanisms used to control information flow in SAFE and an end-to-end proof of noninterference for this model in the Coq proof assistant [17]. This work has been obtained in collaboration with colleagues from University of Pennsylvania, Portland State University, and Harvard University, as part of the CRASH-SAFE project, funded by DARPA.

5.10. Formal Verification of Static Analysis

Participants: Sandrine Blazy, Vincent Laporte, David Pichardie.

Static analysis of binary code is challenging for several reasons. In particular, standard static analysis techniques operate over control flow graphs, which are not available when dealing with self-modifying programs which can modify their own code at runtime. We formalized in the Coq proof assistant some key abstract interpretation techniques that automatically extract memory safety properties and control flow graphs from binary code [22], and operate over a small subset of the x86 assembly. Our analyzer is formally proved correct and has been run on several self-modifying challenges, provided by Cai et al. in their PLDI 2007 paper.

6. Partnerships and Cooperations

6.1. National Initiatives

6.1.1. The PiCoq ANR project

Participants: Alan Schmitt, Petar Maksimovic.

Process calculi, Verification, Proof Assistants

The goal of the PiCoq project is to develop an environment for the formal verification of properties of distributed, component-based programs. The project's approach approach lies at the interface between two research areas: concurrency theory and proof assistants. Achieving this goal relies on three scientific advances, which the project intends to address:

- Finding mathematical frameworks that ease modular reasoning about concurrent and distributed systems: due to their large size and complex interactions, distributed systems cannot be analysed in a global way. They have to be decomposed into modular components, whose individual behaviour can be understood.
- Improving existing proof techniques for distributed/modular systems: while behavioural theories of first-order concurrent languages are well understood, this is not the case for higher-order ones. We also need to generalise well-known modular techniques that have been developed for first-order languages to facilitate formalization in a proof assistant, where source code redundancies should be avoided.
- Defining core calculi that both reflect concrete practice in distributed component programming and enjoy nice properties w.r.t. behavioural equivalences.

The project partners include Inria, LIP, and Université de Savoie. The project runs from December 2010 to November 2014.

6.1.2. The ANR VERASCO project

Participants: Sandrine Blazy, Delphine Demange, Vincent Laporte, André Oliveira Maroneze, David Pichardie.

Static program analysis, Certified static analysis

The VERASCO project (2012–2015) is funded by the call ISN 2011, a program of the Agence Nationale de la Recherche. It investigates 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. It is a joint project with the Inria teams ABSTRACTION, GALLIUM, The VERIMAG laboratory and the Airbus company.

6.1.3. The ANR Binsec project

Participants: Frédéric Besson, Sandrine Blazy, Pierre Wilke, Colas Le Guernic.

Binary code, Static program analysis

The Binsec project (2013–2017) is founded by the call ISN 2012, a program of the Agence Nationale de la Recherche. The goal of the BINSEC project is to develop static analysis techniques and tools for performing automatic security analyses of binary code. We target two main applicative domains: vulnerability analysis and virus detection.

Binsec is a joint project with the Inria CARTE team, CEA LIS, VERIMAG, EADS IW and VUPEN SECURITY. ABSTRACTION, The VERIMAG laboratory and the Airbus company.

6.1.4. The ANR MALTHY project

Participant: David Cachera.

The MALTHY project, funded by ANR in the program INS 2013, aims at advancing the state-of-the-art in real-time and hybrid model checking by applying advanced methods and tools from linear algebra and algebraic geometry. MALTHY is coordinated by VERIMAG, involving CEA-LIST, Inria Rennes (Estasys and Celtique), Inria Saclay (MAXPLUS) and VISEO/Object Direct.

6.1.5. The ANR AJACS project

Participants: Martin Bodin, Thomas Jensen, Alan Schmitt.

The goal of the AJACS project is to provide strong security and privacy guarantees on the client side for web application scripts. To this end, we propose to define a mechanized semantics of the full JavaScript language, the most widely used language for the Web. We then propose to develop and prove correct analyses for JavaScript programs, in particular information flow analyses that guarantee no secret information is leaked to malicious parties. The definition of sub-languages of JavaScript, with certified compilation techniques targeting them, will allow us to derive more precise analyses. Finally, we propose to design and certify security and privacy enforcement mechanisms for web applications, including the APIs used to program real-world applications.

The project partners include the following Inria teams: Celtique, Indes, Prosecco, and Toccata; it also involves researchers from Imperial College as external collaborators. The project runs from December 2014 to June 2018.

6.1.6. The ANR DISCOVER project

Participants: Sandrine Blazy, Delphine Demange, Thomas Jensen, David Pichardie.

The DISCOVER project project aims at leveraging recent foundational work on formal verification and proof assistants to design, implement and verify compilation techniques used for high-level concurrent and managed programming languages. The ultimate goal of DISCOVER is to devise new formalisms and proof techniques able to scale to the mechanized correctness proof of a compiler involving a rich class of optimizations, leading to efficient and scalable applications, written in higher-level languages than those currently handled by cutting-edge verified compilers.

In the light of recent work in optimizations techniques used in production compilers of high-level languages, control-flow-graph based intermediate representations seems too rigid. Indeed, the analyses and optimizations in these compilers work on more abstract representations, where programs are represented with data and control dependencies. The most representative representation is the sea-of-nodes form, used in the Java Hotspot Server Compiler, and which is the rationale behind the highly relaxed definition of the Java memory model. DISCOVER proposes to tackle the problem of verified compilation for shared-memory concurrency with a resolute language-based approach, and to investigate the formalization of adequate program intermediate representations and associated correctness proof techniques.

The project runs from October 2014 to September 2018.

6.1.7. Labex COMIN Labs Seccloud project

Participants: Frédéric Besson, Thomas Jensen, Alan Schmitt, Thomas Genet, Martin Bodin.

The SecCloud project, started in 2012, will provide a comprehensive language-based approach to the definition, analysis and implementation of secure applications developed using Javascript and similar languages. Our high level objectives is to enhance the security of devices (PCs, smartphones, ect.) on which Javascript applications can be downloaded, hence on client-side security in the context of the Cloud. We will achieve this by focusing on three related issues: declarative security properties and policies for client-side applications, static and dynamic analysis of web scripting programming languages, and multi-level information flow monitoring.

This is a joint project with Supelec Rennes and Ecole des Mines de Nantes.

6.2. International Initiatives

6.2.1. Inria Associate Teams

6.2.1.1. JCERT

Title: Verified Compilation of Concurrent Managed Languages
International Partner (Institution - Laboratory - Researcher):
Purdue University (ÉTATS-UNIS)
Duration: 2014 -
See also: http://www.irisa.fr/celtique/ea/jcert/
Safety-critical applications demand rigorous, unambiguous guarantees on program correctness. While a combination of testing and manual inspection is typically used for this purpose, bugs latent in other components of the software stack, especially the compiler and the runtime system, can invalidate these hard-won guarantees. To address such concerns, additional laborious techniques such as manual code reviews of generated assembly code are required by certification agencies. Significant restrictions are imposed on compiler optimizations that can be performed, and the scope of runtime and operating system services that can be utilized. To alleviate this burden, the JCert project is implementing a verified compiler and runtime for managed concurrent languages like Java or C#.

6.2.2. Inria International Partners

6.2.2.1. Informal International Partners

Yann Salmon spent one month in Luke Ong's group at Oxford University (UK) between january and february. The objective of this stay was, on the one side, to promote Yann's work on strategy-dependant analysis of functional programs and, on the other side, to learn from Luke Ong's group on the analysis principles for higher-order functions.

6.2.2.1.1. JSCert

The JSCert project is an informal collaboration between Inria (Celtique and Toccata teams) and Imperial College. Alan Schmitt (Celtique) and Arthur Charguéraud (Toccata) are external collaborators for the "Certified Verification of Client-Side Web Programs" EPSRC project, led by Imperial College. Sergio Maffeis and Philippa Gardner are external collaborators for the "AJACS" ANR project, led by Inria.

6.3. International Research Visitors

6.3.1. Visits to International Teams

6.3.1.1. Sabbatical programme

Jensen Thomas

Date: Sep 2014 - Aug 2015

Institution: University of Copenhagen, Denmark

Pichardie David

Date: Sep 2011 - Aug 2012

Institution: Purdue University (PAYS???)

6.3.1.2. Explorer programme

Salmon Yann

Date: Jan 2014 - Feb 2014

Institution: University of Oxford (UK)

7. Dissemination

7.1. Promoting Scientific Activities

7.1.1. Scientific events organisation

7.1.1.1. general chair, scientific chair

- EJCP 2014 (École jeunes chercheurs en programmation): Thomas Jensen, Alan Schmitt.

7.1.1.2. member of the organizing committee

- PLMW 2014 (Programming Languages Mentoring Workshop): Alan Schmitt.

7.1.2. Scientific events selection

7.1.2.1. responsible of the conference program committee

- GraMSec 2014 (International Workshop on Graphical Models for Security): Barbara Kordy.

7.1.2.2. member of the conference program committee

- ESOP 2014 (European Symposium on Programming): David Pichardie.
- ITP 2014 (Interactive Theorem Proving): David Pichardie.
- CC 2014 (Compiler Construction): Sandrine Blazy
- VSTTE 2014 (Verified Software: Theories, Tools and Experiments): Sandrine Blazy
- AFADL 2014 (Approches Formelles dans l'Assistance au Développement de Logiciels) : Sandrine Blazy
- WASMAS (Active Security Through Multi-Agent Systems): Barbara Kordy.

7.1.2.3. reviewer

- ESOP 2014 (European Symposium on Programming): Delphine Demange, Alan Schmitt, Thomas Jensen.
- FLOPS 2014 (Symposium on Functional and Logic Programming): Alan Schmitt.
- ICFP 2014 (International Conference on Functional Programming): Alan Schmitt.
- IFIP SEC 2014 (International Information Security and Privacy Conference): Frédéric Besson, Thomas Jensen.
- ITP 2014 (Interactive Theorem Proving): Delphine Demange.
- LICS 2014 (Logic In Computer Science): Delphine Demange.
- POPL 2014 (Symposium on Principles of Programming Languages): David Pichardie
- PPDP 2014 (Principles and Practice of Declarative Programming): Alan Schmitt.
- PEPM 2015 (Partial Evaluation and Program Manipulation): Thomas Jensen.
- CC 2014 (Compiler Construction): Sandrine Blazy
- VSTTE 2014 (Verified Software: Theories, Tools and Experiments): Sandrine Blazy
- AFADL 2014 (Approches Formelles dans l'Assistance au Développement de Logiciels) : Sandrine Blazy
- POST 2014 (Conference on Principles of Security and Trust): Barbara Kordy.
- TACAS 2014 (International Conference on Tools and Algorithms for the Construction and Analysis of Systems): Barbara Kordy.
- GraMSec 2014 (International Workshop on Graphical Models for Security): Barbara Kordy.
- WASMAS 2014 (Active Security Through Multi-Agent Systems): Barbara Kordy.
- FORTE 2014 (International Conference on Formal Techniques for Distributed Objects, Components and Systems): Barbara Kordy.
- PRDC 2014 (IEEE Pacific Rim International Symposium on Dependable Computing): Barbara Kordy.
- STM 2014 (International Workshop on Security and Trust Management): Barbara Kordy

7.1.3. Journal

7.1.3.1. reviewer

- Logical Methods in Computer Science: Alan Schmitt.
- Science of Computer Programming: Alan Schmitt.
- Journal of Computer Security: Sandrine Blazy
- Journal of Formalized Reasoning: Frédéric Besson, Sandrine Blazy.

7.2. Teaching - Supervision - Juries

7.2.1. Teaching

Licence: Delphine Demange, Functional Programming, 70h, L1, Université de Rennes 1 / Istic, France

Licence: Thomas Genet, Software Engineering, 58h, L2, Université de Rennes 1 / Istic, France

Licence: Delphine Demange, Software Engineering, 40h, L2, Université de Rennes 1 / Istic, France

Licence : Sandrine Blazy, Functional programming in OCaml, 30h, L3, Rennes 1, France

Master : Sandrine Blazy, Méthodes Formelles pour le développement de logiciels sûrs, 53h, M1, Rennes 1, France

Master : Sandrine Blazy, Software vulnerabilities, 26h, M2, Rennes 1

Master : Sandrine Blazy, Mechanised semantics, 15h, M2, Rennes 1, France

Licence : David Cachera, Formal Languages, 36h, L3, ENS Rennes, France

Licence : David Cachera, Logic, 36h, L3, ENS Rennes, France

Licence : David Pichardie, Algorithms, 36h, L3, ENS Rennes, France

Licence : Alan Schmitt, Programmation Fonctionnelle, 37h, L3, Insa Rennes, France

Licence : Barbara Kordy, Formal Languages and Grammars, 36h , L3, INSA Rennes, France

Master : Thomas Genet, Formal Design and Verification, 108h, M1, Université de Rennes 1 / Istic, France

Master : Thomas Genet, Cryptographic Protocols, 24h, M2, Université de Rennes 1 / Istic, France

Master : Frédéric Besson, Compilation, 68h, M1, Insa Rennes, France

Master : Alan Schmitt, Méthodes Formelles pour le développement de logiciels sûrs, 22h, M1, Rennes 1, France

Master : David Cachera, Semantics of Programming Languages, 36h, M1, Université Rennes 1, France

Master : Delphine Demange, Software Security, 9h, M2, Université Rennes 1, France

Master : David Pichardie, Software Security, 21h, M2, Université Rennes 1, France

Master : David Pichardie, Mechanized Semantics, 15h, M2, Université Rennes 1, France

Master : Thomas Jensen, Static Analysis and Software Security, 20h, University of Copenhagen, Denmark.

Master : Barbara Kordy, Security, 17h, M2, INSA Rennes, France

Master : Barbara Kordy, 4th year Project, 55h, M1, INSA Rennes, France

7.2.2. Supervision

PhD in progress : Martin Bodin, Certified Analyses of JavaScript, 1st september 2012, Thomas Jensen and Alan Schmitt

PhD in progress : Vincent Laporte, Formal verification of static analyses for low level langages, 1st septembre 2012, Sandrine Blazy and David Pichardie

PhD in progress : Yannick Zakowski, Programs Logics for Concurrency, 1st september 2014, David Pichardie and David Cachera

PhD in progress: Pierre Wilke, Low-level memory models for compilers and static analysers, 1st august 1013, Sandrine Blazy and Frédéric Besson

PhD in progress: Yann Salmon, Reachability for Term Rewriting Systems under Strategies, from Sept 2012, Thomas Genet

PhD in progress: David Bühler, Communication between analyses by deductive verification and abstract interpretation, November 2013, Sandrine Blazy and Boris Yakobowski (CEA)

PhD in progress: Stéphanie Riaud, Transformations de programmes pertinentes pour la sécurité du logiciel, septembre 2011, Sandrine Blazy

PhD: Valérie Murat, Tree Automata Extensions for the Verification of Infinite State Systems, Université de Rennes 1, Thomas Genet

PhD: Andre Oliveira Maroneze, Verified Compilation and Worst-Case Execution Time Estimation, Université de Rennes 1, Sandrine Blazy, David Pichardie and Isabelle Puaut

7.2.3. Juries

Thomas Genet, jury member (reviewer) for the PhD defense of Marc Sylvestre, October 2014, University Bordeaux 1

Alan Schmitt, jury member for the PhD defense of Alain Mebsout, September 2014, Laboratoire de Recherche en Informatique

Alan Schmitt, jury member (reviewer) for the PhD defense of Nuno Gaspar, December 2014, Université de Sophia-Antipolis

Thomas Jensen, jury member (reviewer) for the PhD defense of Jael Kriener, March 2014, University of Kent at Canterbury, UK

David Pichardie, jury member (chair) for the PhD defense of Sebastien Chedor, January 2014, University Rennes 1, France

David Pichardie, jury member (examiner) for the PhD defense of Jonathan Protzenko, September 2014, University Paris-Diderot, France

David Pichardie, jury member (chair) for the PhD defense of Hernan Vanzetto, December 2014, University of Lorraine, France

David Pichardie, jury member (chair) for the PhD defense of Radoniaina Andriatsimandefitra Ratsisahanana, December 2014, SUPELEC, France

Sandrine Blazy, jury member (reviewer) for the PhD defense of Vincent Benayoun, May 2014, CNAM, France

Sandrine Blazy, jury member (chair) for the PhD defense of Van Chan Ngo, July 2014, University Rennes 1, France

Sandrine Blazy, jury member (reviewer) for the PhD defense of Etienne Millon, July 2014, University Paris 6, France

Barbara Kordy, jury member, (reviewer) for the PhD defense of Hannes Holm, February 2014, KTH, Sweden.

7.2.4. Juries for competitive selections

Sandrine Blazy, jury member and external president for the selection of a Maître de Conférences, May 2014, University Paris Sud, France.

Sandrine Blazy, jury member for the selection of a Maître de Conférences, May 2014, INSA de Rennes, France.

Sandrine Blazy, jury member for the selection of a Professeur des Universités, May 2014, University Paris Diderot, France.

Sandrine Blazy, jury member for the final selection of Inria DR (senior researchers) candidates, June 2014, Inria, Paris, France.

Sandrine Blazy, member of the evaluation committee CES 28 of ANR for the competitive selection of ANR projects, June 2014, Paris, France.

Delphine Demange, jury member for the selection of a Maître de Conférences at University of Rennes 1 / ISTIC, May 2014, University of Rennes 1, France.

Delphine Demange, jury member of the Gilles Kahn PhD award committee, December 2014, Inria Paris - Rocquencourt, Antenne Parisienne, France.

7.3. Popularization

- Talk “Bug, Virus, Intrusion, Pirates... So many threats and no defense? Yes... maths.”, Thomas Genet, for high school teachers, ENS Rennes, Oct. 2014.

8. Bibliography

Major publications by the team in recent years

[1] F. BESSON, N. BIELOVA, T. JENSEN. *Hybrid Information Flow Monitoring Against Web Tracking*, in "CSF - 2013 IEEE 26th Computer Security Foundations Symposium", New Orleans, United States, 2013 [*DOI :* 10.1109/CSF.2013.23], http://hal.inria.fr/hal-00924138

[2] F. BESSON, T. JENSEN, D. PICHARDIE. *Proof-Carrying Code from Certified Abstract Interpretation to Fixpoint Compression*, in "Theoretical Computer Science", 2006, vol. 364, n^{o} 3, pp. 273–291

[3] F. BESSON, T. JENSEN, T. TURPIN. *Computing stack maps with interfaces*, in "Proc. of the 22nd European Conference on Object-Oriented Programming (ECOOP 2008)", LNCS, Springer-Verlag, 2008, vol. 5142, pp. 642-666

[4] 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

[5] B. BOYER, T. GENET, T. JENSEN. *Certifying a Tree Automata Completion Checker*, in "4th International Joint Conference, IJCAR 2008", Lectures Notes in Computer Science, Springer-Verlag, 2008, vol. 5195, pp. 347–362

[6] D. CACHERA, T. JENSEN, A. JOBIN, P. SOTIN. *Long-Run Cost Analysis by Approximation of Linear Operators over Dioids*, in "Mathematical Structures in Computer Science", 2010, vol. 20, n^{o} 4, pp. 589-624

[7] D. CACHERA, T. JENSEN, D. PICHARDIE, V. RUSU. *Extracting a Data Flow Analyser in Constructive Logic*, in "Theoretical Computer Science", 2005, vol. 342, n^{o} 1, pp. 56–78

[8] D. DEMANGE, V. LAPORTE, L. ZHAO, D. PICHARDIE, S. JAGANNATHAN, J. VITEK. *Plan B: A Buffered Memory Model for Java*, in "Proc. of the 40th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2013", Rome, Italy, ACM, 2013, http://hal.inria.fr/hal-00924716

[9] T. GENET, V. RUSU. *Equational Approximations for Tree Automata Completion*, in "Journal of Symbolic Computation", 2010, vol. 45(5):574-597, May 2010, n^{o} 5, pp. 574-597

[10] A. GOTLIEB, T. DENMAT, B. BOTELLA. *Goal-oriented test data generation for pointer programs*, in "Information and Software Technology", Sep. 2007, vol. 49, n^{o} 9-10, pp. 1030–1044

[11] L. HUBERT, T. JENSEN, V. MONFORT, D. PICHARDIE. *Enforcing Secure Object Initialization in Java*, in "15th European Symposium on Research in Computer Security (ESORICS)", Lecture Notes in Computer Science, Springer, 2010, vol. 6345, pp. 101-115

Publications of the year

Doctoral Dissertations and Habilitation Theses

[12] A. MARONEZE. *Verified compilation and worst-case execution time*, Université de Rennes 1, June 2014, https://hal.archives-ouvertes.fr/tel-01064869

[13] V. MURAT. *Tree automata extensions for verification of infinite states systems*, Université Rennes 1, June 2014, https://tel.archives-ouvertes.fr/tel-01065696

Articles in International Peer-Reviewed Journals

[14] G. BARTHE, D. DEMANGE, D. PICHARDIE. *Formal Verification of an SSA-based Middle-end for CompCert*, in "ACM Transactions on Programming Languages and Systems (TOPLAS)", 2014, 35 p. , https://hal.inria.fr/hal-01097677

[15] D. CACHERA, T. JENSEN, A. JOBIN, F. KIRCHNER. *Inference of polynomial invariants for imperative programs: a farewell to Gro¨bner bases*, in "Science of Computer Programming", 2014, vol. 93, 21 p. [*DOI :* 10.1016/J.SCICO.2014.02.028], https://hal.inria.fr/hal-00932351

[16] S. JAGANNATHAN, V. LAPORTE, G. PETRI, D. PICHARDIE, J. VITEK. *Atomicity Refinement for Verified Compilation*, in "ACM Transactions on Programming Languages and Systems (TOPLAS)", January 2014, 30 p. , https://hal.inria.fr/hal-01102435

International Conferences with Proceedings

[17] A. AZEVEDO DE AMORIM, N. COLLINS, A. DEHON, D. DEMANGE, C. HRITCU, D. PICHARDIE, B. C. PIERCE, R. POLLACK, A. TOLMACH. *A Verified Information-Flow Architecture*, in "41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)", San Diego, CA, United States, 2014 [*DOI :* 10.1145/2535838.2535839], https://hal.inria.fr/hal-00918847

[18] G. BARTHE, G. BETARTE, J. D. CAMPO, C. LUNA, D. PICHARDIE. *System-level Non-interference for Constant-time Cryptography*, in "ACM SIGSAC Conference on Computer and Communications Security, CCS'14", Scottsdale, United States, ACM, November 2014, pp. 1267 - 1279 [*DOI :* 10.1145/2660267.2660283], https://hal.inria.fr/hal-01101950

[19] F. BESSON, N. BIELOVA, T. JENSEN. *Browser Randomisation against Fingerprinting: A Quantitative Information Flow Approach*, in "NordSec - Nordic Conference on Secure IT Systems", Tromsø, Norway, October 2014 [*DOI :* 10.1007/978-3-319-11599-3_11], https://hal.inria.fr/hal-01081037

[20] F. BESSON, S. BLAZY, P. WILKE. *A Precise and Abstract Memory Model for C Using Symbolic Values*, in "APLAS 2014 - 12th Asian Symposium on Programming Languages and Systems", Singapore, Singapore, LNCS, Springer, 2014, vol. 8858, pp. 449 - 468 [*DOI :* 10.1007/978-3-319-12736-1_24], https://hal.inria.fr/hal-01093312

[21] F. BESSON, T. JENSEN, P. VITTET. *SawjaCard: A Static Analysis Tool for Certifying Java Card Applications*, in "SAS - 21st International Static Analysis Symposium", Munich, Germany, Springer, 2014, vol. 8858, pp. 51 - 67 [*DOI :* 10.1007/978-3-319-10936-7_4], https://hal.inria.fr/hal-01093327

[22] S. BLAZY, V. LAPORTE, D. PICHARDIE. *Verified Abstract Interpretation Techniques for Disassembling Low-level Self-modifying Code*, in "ITP - The 5th International Conference on Interactive Theorem Proving", Vienna, Austria, LNCS : Interactive Theorem Proving, Springer, 2014, vol. 8558, pp. 128 - 143 [*DOI :* 10.1007/978-3-319-08970-6_9], https://hal.inria.fr/hal-01102445

[23] S. BLAZY, A. MARONEZE, D. PICHARDIE. *Verified Validation of Program Slicing*, in "CPP : Conference on Certified Programs and Proofs", Mumbai, India, 2015, pp. 109-117 [*DOI :* 10.1145/2676724.2693169], https://hal.inria.fr/hal-01110821

[24] S. BLAZY, S. RIAUD. *Measuring the Robustness of Source Program Obfuscation - Studying the Impact of Compiler Optimizations on the Obfuscation of C Programs*, in "Fourth ACM Conference on Data and

Application Security and Privacy - SIGSAC ACM CODASPY", San Antonio, United States, March 2014, https://hal.inria.fr/hal-00927427

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

[26] M. BODIN, A. SCHMITT. *Certified Abstract Interpretation with Pretty-Big-Step Semantics*, in "CPP - Certified Programs and Proofs", Mumbai, India, Proceedings of the 2015 Conference on Certified Programs and Proofs, January 2015 [*DOI :* 10.1145/2676724.2693174], https://hal.inria.fr/hal-01111588

[27] D. DEMANGE, D. PICHARDIE, L. STEFANESCO. *Verifying Fast and Sparse SSA-based Optimizations in Coq* , in "CC - 24th International Conference on Compiler Construction", London, United Kingdom, 2015, https://hal.inria.fr/hal-01110779

[28] T. GENET. *Towards Static Analysis of Functional Programs Using Tree Automata Completion*, in "Workshop on Rewriting Logic and its Applications", Grenoble, France, 10th International Workshop on Rewriting Logic and its Applications, Springer, April 2014, vol. 8663, pp. 147 - 161 [*DOI :* 10.1007/978-3-319-12904-4_8], https://hal.archives-ouvertes.fr/hal-01089993

[29] J.-H. JOURDAN, V. LAPORTE, S. BLAZY, X. LEROY, D. PICHARDIE. *A formally-verified C static analyzer*, in "POPL : 42nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages", Mumbai, India, ACM, January 2015, pp. 247-259 [*DOI :* 10.1145/2676726.2676966], https://hal.inria.fr/hal-01078386

[30] B. KORDY, M. POULY, P. SCHWEITZER. *A Probabilistic Framework for Security Scenarios with Dependent Actions*, in "IFM - Integrated Formal Methods - 11th International Conference", Bertinoro, Italy, E. ALBERT, E. SEKERINSKI (editors), Springer, September 2014, vol. Lecture Notes in Computer Science, n^{o} 8739, pp. 256 - 271 [*DOI :* 10.1007/978-3-319-10181-1_16], https://hal.archives-ouvertes.fr/hal-01093276

[31] A. OLIVEIRA MARONEZE, S. BLAZY, D. PICHARDIE, I. PUAUT. *A Formally Verified WCET Estimation Tool*, in "14th International Workshop on Worst-Case Execution Time Analysis", Madrid, Spain, July 2014 [*DOI :* 10.4230/OASIcs.WCET.2014.11], https://hal.inria.fr/hal-01087194

[32] D. POUS, A. SCHMITT. *De la KAM avec un Processus d'Ordre Supérieur*, in "JFLA - 25ème Journées Francophones des Langages Applicatifs", Fréjus, France, January 2014, pp. 1-12, https://hal.archives-ouvertes.fr/hal-00966097

National Conferences with Proceedings

[33] M. BODIN, T. JENSEN, A. SCHMITT. *Pretty-big-step-semantics-based Certified Abstract Interpretation*, in "JFLA - 25ème Journées Francophones des Langages Applicatifs", Fréjus, France, January 2014, https://hal.inria.fr/hal-00927400

[34] M. ESCARRÁ, M. PETAR, A. SCHMITT. *HOCore in Coq*, in "JFLA - Vingt-sixièmes Journées Francophones des Langages Applicatifs", Le Val d'Ajol, France, D. BAELDE, J. ALGLAVE (editors), January 2015, https://hal.inria.fr/hal-01099130

Scientific Books (or Scientific Book chapters)

[35] X. LEROY, A. W. APPEL, S. BLAZY, G. STEWART. *The CompCert memory model*, in "Program Logics for Certified Compilers", A. W. APPEL (editor), Cambridge University Press, April 2014, pp. 237-271, https://hal.inria.fr/hal-00905435

Research Reports

[36] F. BESSON, N. BIELOVA, T. JENSEN. *Enforcing Browser Anonymity with Quantitative Information Flow*, 2014, n^{o} RR-8532, https://hal.inria.fr/hal-00984654

[37] T. GENET. *A Note on the Precision of the Tree Automata Completion*, IRISA, December 2014, 13 p. , https://hal.inria.fr/hal-01091393

Other Publications

[38] B. BONNEFOY-CLAUDET. *Security analysis of Android applications*, IRISA-Inria, Campus de Beaulieu, 35042 Rennes cedex, 2014, 35 p. , http://dumas.ccsd.cnrs.fr/dumas-01088788

References in notes

[39] *The Coq Proof Assistant*, 2009, http://coq.inria.fr/

[40] E. ALBERT, P. ARENAS, S. GENAIM, G. PUEBLA, D. ZANARDINI. *COSTA: Design and Implementation of a Cost and Termination Analyzer for Java Bytecode*, in "FMCO", 2007, pp. 113-132

[41] E. ALBERT, G. PUEBLA, M. HERMENEGILDO. *Abstraction-Carrying Code*, in "Proc. of 11th Int. Conf. on Logic for Programming Artificial Intelligence and Reasoning (LPAR'04)", Springer LNAI vol. 3452, 2004, pp. 380-397

[42] ANDREW W. APPEL. *Foundational Proof-Carrying Code*, in "Logic in Computer Science", J. HALPERN (editor), IEEE Press, June 2001, 247 p. , Invited Talk

[43] ANDREW W. APPEL, AMY P. FELTY. *A Semantic Model of Types and Machine Instructions for Proof-Carrying Code*, in "Principles of Programming Languages", ACM, 2000

[44] D. ASPINALL, L. BERINGER, M. HOFMANN, HANS-WOLFGANG. LOIDL, A. MOMIGLIANO. *A Program Logic for Resource Verification*, in "In Proceedings of the 17th International Conference on Theorem Proving in Higher-Order Logics, (TPHOLs 2004), volume 3223 of LNCS", Springer, 2004, pp. 34–49

[45] D. F. BACON, P. F. SWEENEY. *Fast Static Analysis of C++ Virtual Function Calls*, in "OOPSLA'96", 1996, pp. 324-341

[46] P. BAILLOT, P. COPPOLA, U. D. LAGO. *Light Logics and Optimal Reduction: Completeness and Complexity*, in "LICS", 2007, pp. 421-430

[47] E. BALLAND, Y. BOICHUT, T. GENET, P.-E. MOREAU. *Towards an Efficient Implementation of Tree Automata Completion*, in "Algebraic Methodology and Software Technology, 12th International Conference, AMAST 2008", Lectures Notes in Computer Science, Springer-Verlag, 2008, vol. 5140, pp. 67-82

[48] G. BARTHE, D. PICHARDIE, T. REZK. *A Certified Lightweight Non-Interference Java Bytecode Verifier*, in "Proc. of 16th European Symposium on Programming (ESOP'07)", Lecture Notes in Computer Science, Springer-Verlag, 2007, vol. 4421, pp. 125-140

[49] F. BESSON, T. JENSEN. *Modular Class Analysis with DATALOG*, in "SAS'2003", 2003, pp. 19-36

[50] F. BESSON, T. JENSEN, G. DUFAY, D. PICHARDIE. *Verifying Resource Access Control on Mobile Interactive Devices*, in "Journal of Computer Security", 2010, vol. 18, n^{o} 6, pp. 971-998

[51] D. CACHERA, T. JENSEN, A. JOBIN, P. SOTIN. *Long-Run Cost Analysis by Approximation of Linear Operators over Dioids*, in "Algebraic Methodology and Software Technology, 12th International Conference, AMAST 2008", Lectures Notes in Computer Science, Springer-Verlag, 2008, vol. 5140, pp. 122-138

[52] D. CACHERA, T. JENSEN, D. PICHARDIE, V. RUSU. *Extracting a Data Flow Analyser in Constructive Logic*, in "Theoretical Computer Science", 2005, vol. 342, n^{o} 1, pp. 56–78

[53] D. CACHERA, T. JENSEN, D. PICHARDIE, G. SCHNEIDER. *Certified Memory Usage Analysis*, in "Proc. of 13th International Symposium on Formal Methods (FM'05)", LNCS, Springer-Verlag, 2005

[54] P. COUSOT, R. COUSOT. *Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints*, in "Proc. of POPL'77", 1977, pp. 238–252

[55] A. ERMEDAHL, C. SANDBERG, J. GUSTAFSSON, S. BYGDE, B. LISPER. *Loop Bound Analysis based on a Combination of Program Slicing, Abstract Interpretation, and Invariant Analysis*, in "Seventh International Workshop on Worst-Case Execution Time Analysis, (WCET'2007)", July 2007, http://www.mrtc.mdh.se/index.php?choice=publications&id=1317

[56] G. FEUILLADE, T. GENET, V. VIET TRIEM TONG. *Reachability Analysis over Term Rewriting Systems*, in "Journal of Automated Reasoning", 2004, vol. 33, n^{o} 3–4, pp. 341–383

[57] M. FÄHNDRICH, K. R. M. LEINO. *Declaring and checking non-null types in an object-oriented language*, in "OOPSLA", 2003, pp. 302-312

[58] T. GENET. *Decidable Approximations of Sets of Descendants and Sets of Normal forms*, in "RTA'98", LNCS, Springer, 1998, vol. 1379, pp. 151–165

[59] T. GENET, V. VIET TRIEM TONG. *Reachability Analysis of Term Rewriting Systems with Timbuk*, in "LPAR'01", LNAI, Springer, 2001, vol. 2250, pp. 691-702

[60] D. GROVE, C. CHAMBERS. *A framework for call graph construction algorithms*, in "Toplas", 2001, vol. 23, n^{o} 6, pp. 685–746

[61] D. GROVE, G. DEFOUW, J. DEAN, C. CHAMBERS. *Call graph construction in object-oriented languages*, in "ACM SIGPLAN Notices", 1997, vol. 32, n^{o} 10, pp. 108–124

[62] M. HOFMANN, S. JOST. *Static prediction of heap space usage for first-order functional programs*, in "POPL", 2003, pp. 185-197

[63] L. HUBERT. *A Non-Null annotation inferencer for Java bytecode*, in "Proc. of the Workshop on Program Analysis for Software Tools and Engineering (PASTE'08)", ACM, 2008

[64] L. HUBERT, T. JENSEN, D. PICHARDIE. *Semantic foundations and inference of non-null annotations*, in "Proc. of the 10th International Conference on Formal Methods for Open Object-based Distributed Systems (FMOODS'08)", Lecture Notes in Computer Science, Springer-Verlag, 2008, vol. 5051, pp. 132-149

[65] O. LHOTÁK, L. J. HENDREN. *Evaluating the benefits of context-sensitive points-to analysis using a BDD-based implementation*, in "ACM Trans. Softw. Eng. Methodol.", 2008, vol. 18, n^{o} 1

[66] V. B. LIVSHITS, M. S. LAM. *Finding Security Errors in Java Programs with Static Analysis*, in "Proc. of the 14th Usenix Security Symposium", 2005, pp. 271–286

[67] A. MILANOVA, A. ROUNTEV, B. G. RYDER. *Parameterized object sensitivity for points-to analysis for Java*, in "ACM Trans. Softw. Eng. Methodol.", 2005, vol. 14, n^{o} 1, pp. 1–41

[68] M. NAIK, A. AIKEN. *Conditional must not aliasing for static race detection*, in "POPL'07", ACM, 2007, pp. 327-338

[69] M. NAIK, A. AIKEN, J. WHALEY. *Effective static race detection for Java*, in "PLDI'2006", ACM, 2006, pp. 308-319

[70] G. C. NECULA. *Proof-carrying code*, in "Proceedings of POPL'97", ACM Press, 1997, pp. 106–119

[71] G. C. NECULA, R. R. SCHNECK. *A Sound Framework for Untrusted Verification-Condition Generators*, in "Proc. of 18th IEEE Symp. on Logic In Computer Science (LICS 2003)", 2003, pp. 248-260

[72] F. NIELSON, H. NIELSON, C. HANKIN. *Principles of Program Analysis*, Springer, 1999

[73] J. PALSBERG, M. SCHWARTZBACH. *Object-Oriented Type Inference*, in "OOPSLA'91", 1991, pp. 146-161

[74] J. PALSBERG, M. SCHWARTZBACH. *Object-Oriented Type Systems*, John Wiley & Sons, 1994

[75] D. PICHARDIE. *Interprétation abstraite en logique intuitionniste : extraction d'analyseurs Java certiés*, Université Rennes 1Rennes, France, dec 2005

[76] A. D. PIERRO, H. WIKLICKY. *Operator Algebras and the Operational Semantics of Probabilistic Languages*, in "Electr. Notes Theor. Comput. Sci.", 2006, vol. 161, pp. 131-150

[77] E. ROSE. *Lightweight Bytecode Verification*, in "Journal of Automated Reasoning", 2003, vol. 31, n^{o} 3–4, pp. 303–334

[78] A. SABELFELD, A. C. MYERS. *Language-based Information-Flow Security*, in "IEEE Journal on Selected Areas in Communication", January 2003, vol. 21, n^{o} 1, pp. 5–19

[79] P. SOTIN, D. CACHERA, T. JENSEN. *Quantitative Static Analysis over semirings: analysing cache behaviour for Java Card*, in "4th International Workshop on Quantitative Aspects of Programming Languages (QAPL 2006)", Electronic Notes in Theoretical Computer Science, Elsevier, 2006, vol. 164, pp. 153-167

[80] F. TIP, J. PALSBERG. *Scalable propagation-based call graph construction algorithms*, in "OOPSLA", 2000, pp. 281-293

[81] J. WHALEY, M. S. LAM. *Cloning-based context-sensitive pointer alias analysis using binary decision diagrams*, in "PLDI '04", ACM, 2004, pp. 131–144

[82] M. WILDMOSER, A. CHAIEB, T. NIPKOW. *Bytecode Analysis for Proof Carrying Code*, in "Bytecode Semantics, Verification, Analysis and Transformation", 2005

[83] M. WILDMOSER, T. NIPKOW, G. KLEIN, S. NANZ. *Prototyping Proof Carrying Code*, in "Exploring New Frontiers of Theoretical Informatics, IFIP 18th World Computer Congress, TC1 3rd Int. Conf. on Theoretical Computer Science (TCS2004)", J.-J. LEVY, E. W. MAYR, J. C. MITCHELL (editors), Kluwer Academic Publishers, August 2004, pp. 333–347