IN PARTNERSHIP WITH:
Université Rennes 1

Ecole normale supérieure de Cachan

Activity Report 2013

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 [ENS Cachan, Associate Professor, HdR]
Delphine Demange [Univ. Rennes I, Associate Professor]
Thomas Genet [Univ. Rennes I, Associate Professor, HdR]
Arnaud Jobin [ENS Cachan, until Aug 2013]
David Pichardie [ENS Cachan, Professor, HdR]

External Collaborator
Colas Le Guernic [DGA, from Jul 2013]

Engineers
Pierre Vittet [Inria, until Nov 2013]
Laurent Guillo [CNRS, from Nov 2013]

PhD Students
Oana Fabiana Andreescu [CIFRE grant, from Oct 2013]
Martin Bodin [Univ. Rennes I]
Pauline Bolignano [CIFRE grant, from Oct 2013]
David Buhler [CEA, from Nov 2013]
Pierre-Emmanuel Cornilleau [Inria, until Mar 2013]
Vincent Laporte [Univ. Rennes I]
Valérie Murat [Univ. Rennes I]
André Oliveira Maroneze [Univ. Rennes I]
Stéphanie Riaud [DGA]
Yann Salmon [Univ. Rennes I]
Pierre Wilke [Univ. Rennes I, from Aug 2013]
Jean-Christophe Zapalowicz [Inria, DGA grant]

Post-Doctoral Fellows
Nataliia Bielova [Inria, until Dec 2013]
Petar Maksimovic [Inria, from Nov 2013]
Daniela Petrisan [Inria, grant by ANR PiCoq project, from Nov 2013]
Thomas Seiller [Inria, grant by ANR PiCoq project, until Nov 2013]

Administrative Assistant
Lydie Mabil [Inria]

2. Overall Objectives

2.1. Project overview

The goal of the CELTIQUE project is to improve the security and reliability of software through software certificates that attest to the well-behavedness of a given software. Contrary to certification techniques based on cryptographic signing, we are providing certificates issued from semantic software analysis. 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 data race freedom of multi-threaded code.

Existing software certification procedures make extensive use of systematic test case generation. Semantic analysis can serve to improve these testing techniques by providing precise software models from which test suites for given test coverage criteria can be manufactured. Moreover, an emerging trend in mobile code security is to equip mobile code with proofs of well-behavedness that can then be checked by the code receiver before installation and execution. A prominent example of such proof-carrying code is the stack maps for Java byte code verification. We propose to push this technique much further by designing certifying analyses for Java byte code that can produce compact certificates of a variety of properties. Furthermore, we will develop efficient and verifiable checkers for these certificates, relying on proof assistants like Coq to develop provably correct checkers. We target two application domains: Java software for mobile devices (in particular mobile telephones) and embedded C programs.

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

2.2. Highlights of the Year

The European Association for Programming Languages and Systems (EAPLS) Best PhD Dissertation Award 2012 has been won by Delphine Demange (ENS Cachan - Brittany Extension and the Celtique team at IRISA / Inria Rennes, advisors Thomas Jensen and David Pichardie), for her dissertation on "Semantic Foundations of Intermediate Program Representations".

The thesis prize Gilles Kahn 2013, awarded by the Société Informatique de France (SiF) and sponsored by Academy of Sciences, was awarded to Delphine Demange for her dissertation "Semantic Foundations of Intermediate Program Representations" (ENS Cachan - Brittany Extension and the Celtique team at IRISA / Inria Rennes, advisors Thomas Jensen and David Pichardie).

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 [75].
- 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 [57], 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** [55], [78]. 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 [56]. 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) [48] is a scalable algorithm for constructing an initial call-graph of the program. Tip and Palsberg [83] 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 [76], [77] 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.,* [64], [63] have explored the algorithmic design space of contexts for benchmarks of significant size. Latter on, Milanova *et. al.,* [70] 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 [68] have extended the empiric evaluation of object-sensitivity using a BDD implementation allowing to cope with benchmarks otherwise out-of-scope. Besson and Jensen [52] proposed to use DATALOG in order to specify context-sensitive analyses. Whaley and Lam [84] 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 [69] and the race analysis of Naik, Aiken [71] and Whaley [72] 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 [60] 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 [67]. The proposed technique has been implemented in a tool called NIT [66]. Null pointer detection is also done by bug-detection tools such as FindBugs [66]. 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 [65], [47], linear logic approaches to implicit computational complexity [49], cost model for Java byte code [43] based on size relation inference, and WCET computation by abstract interpretation based loop bound interval analysis techniques [58].

We have proposed an original approach for designing static analyses computing program costs: inspired from a probabilistic approach [79], 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 [54]. An example of cache miss analysis has been developed within this framework [82].

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

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. 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. 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.

So far, static analysis tools are not a part of the approved certification procedures. This would require the acceptance of the static analyzers by the certification authorities 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.

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 [81], [51]. Another kind of problems concern the way computational resources are being accessed and used, in order to ensure the correct implementation of a given access policy, 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 [53]. **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) [73] 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.

[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 ?

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" [74]. Many solutions have been proposed to reduce the size of the trusted computing base. In the *foundational proof carrying code* of Appel and Felty [46], [45], the code producer gives a direct proof that, in some "foundational" higher-order logic, the code respects a given security policy. Wildmoser and Nipkow [86], [85]. prove the soundness of a *weakest precondition* calculus for a reasonable subset of the Java bytecode. Necula and Schneck [74] 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 [80], 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.* [44] 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 [42] 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 [61], [62]. We defined a simple and efficient algorithm [59] 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 [50].

4. 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, Pierre Vittet.

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: Martin Bodin, Alan Schmitt.

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 is available at http://jscert.org/.

5. New Results

5.1. Information Flow Tracking

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

We investigate different approaches for dynamically tracking information flows.

The first track of work is motivated by web-browser security. In a survey [15], we have classified JavaScript security policies and their enforcement mechanisms in a web-browser. We have identified the problem of stateless web tracking (fingerprinting) and have proposent a novel approach to hybrid information flow monitoring by tracking the knowledge about secret variables using logical formulae. A logic formula quantifies the amount of knowledge stored in a variable. This knowledge representation helps to compare and improve precision of hybrid information flow monitors. We define a generic hybrid monitor parametrised by a static analysis and derive sufficient conditions on the static analysis for soundness and relative precision of hybrid monitors. We instantiate the generic monitor with a combined static constant and dependency analysis. Several other hybrid monitors including those based on well-known hybrid techniques for information flow control are formalised as instances of our generic hybrid mon- itor. These monitors are organised into a hierarchy that establishes their relative precision. The whole framework is accompanied by a formalisation of the theory in the Coq proof assistant [19].

Our second activity is related to SAFE, 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].

5.2. Towards efficient abstract domains for regular language based static analysis

Participants: Thomas Genet, Valérie Murat, 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 safety 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[27]. As a first instance, we implemented in Timbuk a completion with an interval abstract domain. We shown that this implementation permits to scale up regular tree model-checking of Java programs dealing with integer arithmetics. Now, we aim at applying this technique to the static analysis of programming languages whose semantics is based on terms, like functional programming languages [38].

5.3. Result Certification of Static Program Analysers with Automated Theorem Provers

Participants: Frédéric Besson, Pierre-Emmanuel Cornilleau, Thomas Jensen.

The automation of the deductive approach to program verification crucially depends on the ability to efficiently infer and discharge program invariants. In an ideal world, user-provided invariants would be strengthened by incorporating the result of static analysers as untrusted annotations and discharged by automated theorem provers. However, the results of object-oriented analyses are heavily quantified and cannot be discharged, within reasonable time limits, by state-of-the-art auto- mated theorem provers. In the present work, we investigate an original approach for verifying automatically and efficiently the result of certain classes of object-oriented static analyses using off-the-shelf automated theorem provers. We propose to generate verification conditions that are generic enough to capture, not a single, but a family of analyses which encompasses Java bytecode verification and Fähndrich and Leino type-system for checking null pointers. For those analyses, we show how to generate tractable verification conditions that are still quantified but fall in a decidable logic fragment that is reducible to the Effectively Propositional logic. Our experiments confirm that such verification conditions are efficiently discharged by off-the-shelf automated theorem provers [20].

5.4. Formal Semantics for Multi-threaded Java

Participants: Delphine Demange, Vincent Laporte, David Pichardie.

Recent advances in verification have made it possible to envision trusted implementations of real-world languages. Java with its type-safety and fully specified semantics would appear to be an ideal candidate; yet, the complexity of the translation steps used in production virtual machines have made it a challenging target for verifying compiler technology. One of Java's key innovations, its memory model, poses significant obstacles to such an endeavor. The Java Memory Model is an ambitious attempt at specifying the behavior of multithreaded programs in a portable, hardware agnostic, way. While experts have an intuitive grasp of the properties that the model should enjoy, the specification is complex and not well-suited for integration within a verifying compiler infrastructure. Moreover, the specification is given in an axiomatic style that is distant from the intuitive reordering-based reasonings traditionally used to justify or rule out behaviors, and ill suited to the kind of operational reasoning one would expect to employ in a compiler. We take a step back,

and introduces a *Buffered Memory Model* (BMM) for Java [26]. We choose a pragmatic point in the design space sacrificing generality in favor of a model that is fully characterized in terms of the reorderings it allows, amenable to formal reasoning, and which can be efficiently applied to a specific hardware family, namely x86 multiprocessors. Although the BMM restricts the reorderings compilers are allowed to perform, it serves as the key enabling device to achieving a verification pathway from bytecode to machine instructions. Despite its restrictions, we show that it is backwards compatible with the Java Memory Model and that it does not cripple performance on TSO architectures.

5.5. Formal Verification of Static Analysis

Participants: Sandrine Blazy, Martin Bodin, Thomas Jensen, Vincent Laporte, André Oliveira Maroneze, David Pichardie, Alan Schmitt.

Static analyzers based on abstract interpretation are complex pieces of software implementing delicate algorithms. Even if static analysis techniques are well understood, their implementation on real languages is still error-prone.

Using the Coq proof assistant, we formalized of a value analysis (based on abstract interpretation), and a soundness proof of the value analysis. The formalization relies on generic interfaces. The mechanized proof is facilitated by a translation validation of a Bourdoncle fixpoint iterator. The work has been integrated into the CompCert verified C-compiler. Our verified analysis directly operates over an intermediate language of the compiler having the same expressiveness as C. The automatic extraction of our value analysis into OCaml yields a program with competitive results, obtained from experiments on a number of benchmarks and comparisons with the Frama-C tool [21]. The value analysis was applied to a loop bound estimation tool for WCET analysis [22] relying also on program slicing and loop bound calculation.

Moreover, we formalized static analyses for logic programming, relying on results about the relative correctness of semantics in different styles; forward and backward, top-down and bottom-up. The results chosen are paradigmatic of the kind of correctness theorems that semantic analyses rely on and are therefore well-suited to explore the possibilities afforded by the application of interactive theorem provers to this task, as well as the difficulties likely to be encountered in the endeavour [29].

We also study the development of certified information flow analyses based on a formal semantics of JavaScript. We have in particular presented a technique for deriving semantic program analyses from a natural semantics specification of the programming language. The technique is based on the pretty-big-step semantics approach applied to a language with simple objects called O'While. We have specified a series of instrumentations of the semantics that makes explicit the flows of values in a program. This leads to a semantics-based dependency analysis, at the core, e.g., of tainting or information flow analyses in software security [32].

5.6. Certified JavaScript Semantics

Participants: Martin Bodin, Alan Schmitt.

JavaScript is the most widely used web language for client-side applications. Whilst the development of JavaScript was initially just led by implementation, there is now increasing momentum behind the ECMA standardisation process. The time is ripe for a formal, mechanised specification of JavaScript, to clarify ambiguities in the ECMA standards, to serve as a trusted reference for high-level language compilation and JavaScript implementations, and to provide a platform for high-assurance proofs of language properties. We present JScert, a formalisation of the current ECMA standard in the Coq proof assistant, and JSref, a reference interpreter for JavaScript extracted from Coq to OCaml. We give a Coq proof that JSref is correct with respect to JScert and assess JSref using test262, the ECMA conformance test suite. Our methodology ensures that JScert is a comparatively accurate formulation of the English standard, which will only improve as time goes on. We have demonstrated that modern techniques of mechanised specification can handle the complexity of JavaScript [25], [24].

5.7. Concurrent Reversibility

Participant: Alan Schmitt.

Concurrent reversibility has been studied in different areas, such as biological or dependable distributed systems. However, only "rigid" reversibility has been considered, allowing to go back to a past state and restart the exact same computation, possibly leading to divergence. We present a concurrent calculus featuring *flexible reversibility*, allowing the specification of alternatives to a computation to be used upon rollback. Alternatives in processes of this calculus are attached to messages. We show the robustness of this mechanism by encoding more complex idioms for specifying flexible reversibility, and we illustrate the benefits of our approach by encoding a calculus of communicating transactions [30].

5.8. Non linear analysis: fast inference of polynomial invariants

Participants: Thomas Jensen, David Cachera, Arnaud Jobin.

We have proposed an abstract interpretation based method for inferring polynomial invariants.Our analysis uses a form of weakest precondition calculus which was already observed to be well adapted to polynomial disequality guards, and which we extend to equality guards by using parameterized polynomial division. We have shown that the choice of a suitable division operation is crucial at each iteration step in order to compute the invariant. Based on this analysis, we have designed a constraint-based algorithm for inferring polynomial invariants. We have identified heuristics to solve equality constraints between ideals, and implemented the whole analysis algorithm in Maple.A salient feature of this analysis, which distinguishes it from the approaches proposed so far in the literature, is that it does not require the use of Gröbner base computations, which are known to be costly on parameterized polynomials. Our benchmarks show that our analyzer can successfully infer invariants on a sizeable set of examples, while performing two orders of magnitude faster than other existing implementations [16].

6. Bilateral Contracts and Grants with Industry

6.1. Bilateral Project with FIME

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

Static program analysis, Javacard, Certification, AFSCM

- Partner : FIME
- Period: Starting January 2012; ending June 2013

The FIME contract consists in an industrial transfer of the Sawja platform 4.2 adapted to analyse Javacard programs according to AFSCM (Association Française du Sans Contact Mobile) security guidelines. The rules specify syntactic constraints but also more semantics properties such as the absence of certain runtime exceptions. FIME aims at automating the process of validating that Javacard applications are conformant to the rules. The outcome of the project is the Jacal (JAvaCard AnaLyser) (4.3 which takes a binary Javacard application; performs static analysis and output statuses for the different rules. Pierre Vittet has recently been recruited by FIME and the operational deployment of Jacal is in progress.

7. Partnerships and Cooperations

7.1. Regional Initiatives

The Celtique team collaborates with DGA-MI, a research laboratory belonging to the French army, and located in Rennes. The collaboration has several facets.

- We run a joint bi-monthly seminar on Security and Formal Methods. This seminar attracts attendance from academia and industry.
- DGA-MI is funding a PhD thesis, supervised jointly, on code obfuscation.
- Colas Le Guernic, a DGA-MI researcher, is external collaborator of Celtique on our activities on analysis of binary code.

7.2. National Initiatives

7.2.1. 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 November 2010 to October 2014.

7.2.2. 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.

7.2.3. The ANR Binsec project

Participants: Frédéric Besson, Sandrine Blazy, Pierre Wilke.

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.

7.2.4. Labex COMIN Labs Seccloud project

Participants: Frédéric Besson, Nataliia Bielova, Thomas Jensen, Alan Schmitt, 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.

7.3. International Initiatives

7.3.1. Inria International Partners

7.3.1.1. Informal International Partners

A strong collaboration is ongoing with researchers from Imperial College (UK) in the setting of the JSCert project (http://jscert.org/). This project aims at really understanding JavaScript by building models of ECMAScript semantics in the Coq proof assistant, and certifying automated logical reasoning tools built on those semantics. We are closely working with Philippa Gardner and Sergio Maffeis. This collaboration has resulted in a large Coq development including a formal semantics for JavaScript and a certified JavaScript interpreter. These results are described in our POPL 2014 paper [24].

In 2013, Martin Bodin, Thomas Jensen, and Alan Schmitt visited Imperial College twice. Daiva Naudziuniene, a PhD student of Philippa Gardner, also did a one month internship in the Celtique team in the setting of this collaboration.

David Pichardie was on sabbatical in 2012, in Jan Vitek's group at Purdue University, Indiana, USA. The strong collaboration is still ongoing, and an Associate Team proposal for 2014-2016 has been submitted in 2013 as part of an Inria International program. The JCert project research aims at verifying the compilation of concurrent managed languages, following the previous outcomes of the informal collaboration – a new memory model for concurrent Java that is more suitable to formal verification [26], as well as refinement-based proof methodology (under submission) that allows to reason compositionally about the atomicity of low-level concurrent code fragments. If the proposal is accepted, David Pichardie would be the Inria principal investigator of the JCert project, and Delphine Demange, Thomas Jensen, and Vincent Laporte will also be active participants.

7.4. International Research Visitors

7.4.1. Visits of International Scientists

7.4.1.1. Internships

Patricio Palladino

Subject: Protection from Web Tracking: Analysis of web browser fingerprints

Date: from Mar 2013 until Apr 2013

Institution: University of Buenos Aires (Argentina)

7.4.2. Visits to International Teams

David Pichardie took a sabbatical year and visited Greg Morrisett's group at Harvard University, Cambridge, USA in 2013. During this sabbatical, he worked on the DARPA SAFE project with Harvard University and UPenn University [17].

8. Dissemination

8.1. Scientific Animation

Sandrine Blazy and David Pichardie co-chaired and organized in Rennes the international conference ITP 2013 and its 2 related workshops. Sandrine Blazy is a member of the steering committee of the ITP conference. Sandrine Blazy served on the organizing committee of the international conferences VMCAI 2013 and LPAR 2013. Sandrine Blazy was scientific director of the "Languages and software engineering department" of IRISA until September 2013. Sandrine Blazy gave an invited talk at the VSTTE 2013 conference in Menlo Park, USA. David Pichardie served on the external reviewing committee of the international conference PLDI 2013 and the organizing committee of the international workshops BYTECODE 2013, PxTP 2013 and Coq Workshop 2013. David Pichardie is chair of the department of Computer Science at ENS Rennes. Alan Schmitt co-chaired and organized the International Symposium on Database Programming Languages, co-located with VLDB, in Trento, Italy (http://dbpl2013.inria.fr/). He also participated in the organization of the Programming Languages Mentoring Workshop, co-located with POPL, in Roma, Italy (http://www.doc.ic.ac.uk/~gds/PLMW/index.html). Alan Schmitt is a member of the steering committee of the Journées Francophones des Langages Applicatifs. Thomas Jensen and Alan Schmitt organized the École Jeunes Chercheurs en Programmation (EJCP 2013).

8.2. Teaching - Supervision - Juries

8.2.1. Teaching

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

Licence : Sandrine Blazy, Programmation fonctionnelle, 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 and Computability, 24h, L3, ENS Rennes

Licence : David Cachera, Logics, 24h, L3, ENS Rennes

Licence : David Cachera, Algorithmics, 12h, L3, ENS Rennes

Master : David Cachera, Programming Language Semantics, 24h, M1, Rennes 1

Licence : Delphine Demange, Algorithmique et Programmation Fonctionnelle, 80h, level L1, Université de Rennes 1 / Istic, France

Master : Delphine Demange, Sémantique, 8h, level M1, Université de Rennes 1 / Istic / ENS Cachan Antenne Bretagne, France

Licence : Thomas Genet, Programmation fonctionnelle, 32h, niveau L3, Université de Rennes 1 / Istic, France

Master : Thomas Genet, Conception et vérification formelle, 90h, niveau M1, Université de Rennes 1 / Istic, France

Master : Thomas Genet, Protocoles cryptographiques, 24h, niveau M2, Université de Rennes 1 / Istic, France

Master : Thomas Jensen, Program analysis, 14h, M2, Rennes

Master : Thomas Jensen, Software security, 20h, M2, Rennes

Licence : David Pichardie, Algorithms, 60h, L3, ENS Rennes

Master : David Pichardie, Mechanised semantics, 15h, M2, Rennes 1, France

Master : David Pichardie, Program analysis, 6h, M2, Rennes

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

8.2.2. Supervision

PhD : Pierre-Emmanuel Cornilleau, Certification of static analysis in many-sorted first-order logic, ENS Cachan - antenne de bretagne, 25 march 2013, Thomas Jensen and Frédéric Besson

PhD in progress : Pierre Wilke, Retro-engineering of auto-modifying code by static analysis, 1st august 2013, Sandrine Blazy and Frédéric Besson

PhD in progress : Valérie Murat, Automatic Verification of infinite state systems by tree automata completion, 1st august 2011, Thomas Genet

PhD in progress : Yann Salmon, Reachability for Term Rewriting Systems under Strategies, 1st august 2012, Thomas Genet

PhD in progress: Andre Oliveira Maroneze, Compilation vérifiée et calcul de temps d'exécution au pire cas, septembre 2010, Sandrine Blazy, David Pichardie and Isabelle Puaut

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

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

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

PhD in progress : Oana Andreescu, Proof reusability in the formal modeling of secure operating systems, 1st September 2013, Thomas Jensen and Stephane Lescuyer (Prove & Run)

PhD in progress : Pauline Bolignano, Formal methods for minimizing a trusted computing base in an operating system, 1st October 2013, Thomas Jensen and Vincent Siles (Prove & Run)

8.2.3. Juries

Frédéric Besson, jury member for the PhD defense of Mohamed Iguernelala, July 10th 2013

Sandrine Blazy, jury member (reviewer) for the PhD defense of Cédric Auger, February 2013, University Paris Sud

Sandrine Blazy, jury member (reviewer) for the PhD defense of Suman Saha, March 2013, University Paris 6

Sandrine Blazy, jury member (president) for the PhD defense of Benjamin Lesage, May 2013, University Rennes 1

Sandrine Blazy, jury member (reviewer) for the PhD defense of Xiaomu Shi, July 2013, Grenoble University

Sandrine Blazy, jury member for the PhD defense of Pierre Néron, October 2013, École Polytechnique

Sandrine Blazy, jury member (reviewer) for the PhD defense of Jean Fortin, October 2013, University Paris East

Thomas Jensen, jury member (examiner) for the PhD defense of Andreas lundblad, March 2013, Royal Technological University, Stockholm, Sweden.

Thomas Jensen, jury member (reviewer) for the PhD defense of Michal Terepeta, October 2013, Technical University, Denmark.

Thomas Jensen, jury member (president) for the PhD defense of Yassamine Seladji, November 2013, Ecole Polytechnique.

Thomas Jensen, jury member (reviewer,president) for the PhD defense of Quentin Sabah, December 2013, University of Grenoble.

David Pichardie, jury member (president) for the PhD defense of Sebastien Chédor, December 2013, University Rennes 1

8.3. Popularization

Frédéric Besson and Nataliia Bielova have presented their approach for quantifying web-fingerprinting in the "Du côté de la recherche" section of the November issue of Ouest Inria.

David Pichardie organised the third edition of the french Castor Informatique contest. This contest promotes Computer Science in secondary schools and high schools. It is organised by Inria, ENS Cachan and the France IOI association and supported by CNRS, Pascaline, the SIF and API associations. In 2013, there was about 170.000 participants.

9. 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, http://hal.inria.fr/inria-00495405

[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, http://hal.inria.fr/inria-00503953

Publications of the year

Doctoral Dissertations and Habilitation Theses

[12] P.-E. CORNILLEAU. , *Certification of static analysis in many-sorted first-order logic*, École normale supérieure de Cachan - ENS Cachan, March 2013, http://hal.inria.fr/tel-00846347

[13] Z. FU. , *Static analysis of numerical properties in the presence of pointers*, Université Rennes 1 and Université européenne de Bretagne, July 2013, http://hal.inria.fr/tel-00918593

Articles in International Peer-Reviewed Journals

[14] G. BARTHE, D. PICHARDIE, T. REZK. *A certified lightweight non-interference Java bytecode verifier*, in "Mathematical Structures in Computer Science", June 2013, vol. 23, n^{o} 5, pp. 1032-1081 [*DOI :* 10.1017/S0960129512000850], http://hal.inria.fr/hal-00915189

[15] N. BIELOVA. *Survey on JavaScript Security Policies and their Enforcement Mechanisms in a Web Browser*, in "Journal of Logic and Algebraic Programming", 2013, vol. 82, n^{o} 8, pp. 243-262 [*DOI :* 10.1016/J.JLAP.2013.05.001], http://hal.inria.fr/hal-00932730

[16] 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, To appear, http://hal.inria.fr/hal-00932351

International Conferences with Proceedings

[17] A. A. 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, To appear [*DOI :* 10.1145/2535838.2535839], http://hal.inria.fr/hal-00918847

[18] R. BAGNARA, M. CARLIER, R. GORI, A. GOTLIEB. *Symbolic Path-Oriented Test Data Generation for Floating-Point Programs*, in "Proc. of the 6th IEEE Int. Conf. on Software Testing, Verification and Validation (ICST'13)", Luxembourg, Luxembourg, 2013, http://hal.inria.fr/hal-00807884

[19] 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

[20] F. BESSON, P.-E. CORNILLEAU, T. JENSEN. *Result Certification of Static Program Analysers with Automated Theorem Provers*, in "VSTTE 2013 - Fifth Working Conference on Verified Software: Theories, Tools and Experiments", Atherthon, United States, 2013, http://hal.inria.fr/hal-00924167

[21] S. BLAZY, V. LAPORTE, A. MARONEZE, D. PICHARDIE. *Formal Verification of a C Value Analysis Based on Abstract Interpretation*, in "SAS - 20th Static Analysis Symposium", Seattle, United States, M. FAHNDRICH, F. LOGOZZO (editors), Springer, 2013, vol. Lecture Notes in Computer Science, pp. 324-344, http://hal.inria.fr/hal-00812515

[22] S. BLAZY, A. MARONEZE, D. PICHARDIE. *Formal Verification of Loop Bound Estimation for WCET Analysis*, in "VSTTE - Verified Software: Theories, Tools and Experiments", Menlo Park, United States, E. COHEN, A. RYBALCHENKO (editors), Lecture Notes in Computer Science, Springer, 2013, vol. 8164, pp. 281-303, http://hal.inria.fr/hal-00848703

[23] 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 2014", San Antonio, United States, 2014, http://hal.inria.fr/hal-00927427

[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] M. BODIN, A. SCHMITT. *A Certified JavaScript Interpreter*, 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-00779459

[26] 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

[27] T. GENET, T. LE GALL, A. LEGAY, V. MURAT. *Tree Regular Model Checking for Lattice-Based Automata*, in "CIAA - 18th International Conference on Implementation and Application of Automata", Halifax, Canada, LNCS, Springer, 2013, vol. 7982, http://hal.inria.fr/hal-00924849

[28] A. GOTLIEB, T. DENMAT, N. LAZAAR. *Constraint-based reachability*, in "Infinity workshop 2012", Paris, France, 2013 [*DOI :* 10.4204/EPTCS.107.4, SOURCE: ARXIV], http://hal.inria.fr/hal-00807856

[29] J. KRIENER, A. KING, S. BLAZY. *Proofs You Can Believe In. Proving Equivalences Between Prolog Semantics in Coq*, in "15th International Symposium on Principles and Practice of Declarative Programming (PPDP)", Madrid, Spain, T. SCHRIJVERS (editor), ACM, September 2013, pp. 37-48, http://hal.inria.fr/hal-00908848

[30] I. LANESE, M. LIENHARDT, C. MEZZINA, A. SCHMITT, J.-B. STEFANI. *Concurrent Flexible Reversibility*, in "22nd European Symposium on Programming, ESOP 2013", Rome, Italy, M. FELLEISEN, P. GARDNER (editors), Lecture Notes in Computer Science (LNCS), Springer, March 2013, vol. 7792, pp. 370-390 [*DOI :* 10.1007/978-3-642-37036-6_21], http://hal.inria.fr/hal-00811629

[31] D. MARIJAN, A. GOTLIEB, S. SEN, A. HERVIEU. *Practical Pairwise Testing for Software Product Lines*, in "SPLC 2013", Tokyo, Japan, August 2013, http://hal.inria.fr/hal-00859438

National Conferences with Proceedings

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

Scientific Books (or Scientific Book chapters)

[33] 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, http://hal.inria.fr/hal-00905435

Books or Proceedings Editing

[34] 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

Research Reports

[35] R. ABDALLAH, A. GOTLIEB, L. HÉLOUËT, C. JARD. , *Scenario realizability with constraint optimization*, January 2013, http://hal.inria.fr/hal-00769656

[36] T. GENET. , *Towards Static Analysis of Functional Programs using Tree Automata Completion*, December 2013, 15 p. , http://hal.inria.fr/hal-00921814

[37] T. GENET, Y. SALMON. , *Reachability Analysis of Innermost Rewriting*, July 2013, http://hal.inria.fr/hal-00848260

[38] T. GENET, Y. SALMON. , *Tree Automata Completion for Static Analysis of Functional Programs*, January 2013, http://hal.inria.fr/hal-00780124

Other Publications

[39] C. EBERHART, T. HIRSCHOWITZ, T. SEILLER. , *Fully-abstract concurrent games for pi*, 2013, 20 pages, submitted, http://hal.inria.fr/hal-00873626

[40] P. GENEVÈS, N. LAYAÏDA, A. SCHMITT. , *Expressive Logical Combinators for Free*, October 2013, http://hal.inria.fr/hal-00868724

[41] P. GENEVÈS, N. LAYAÏDA, A. SCHMITT. , *Efficiently Deciding Mu-calculus with Converse over Finite Trees*, January 2014, http://hal.inria.fr/hal-00868722

References in notes

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

[43] 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

[44] 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

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

[46] 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

[47] 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

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

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

[50] 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

[51] 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

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

[53] 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, http://hal.inria.fr/inria-00537821

[54] 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

[55] 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

[56] 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

[57] 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

[58] 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

[59] 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

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

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

[62] 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

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

[64] 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

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

[66] 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, To appear

[67] 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

[68] 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

[69] 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

[70] 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

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

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

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

[74] 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

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

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

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

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

[79] 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

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

[81] 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

[82] 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

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

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

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

[86] 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