IN PARTNERSHIP WITH:
Université Rennes 1

Ecole normale supérieure de Cachan

Activity Report 2012

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
Programs, Verification and Proofs

Table of contents

1. Members **1**
2. Overall Objectives **1**
3. Scientific Foundations **2**
3.1. Static program analysis 2
3.1.1. Static analysis of Java 3
3.1.2. Quantitative aspects of static analysis 3
3.1.3. Semantic analysis for test case generation 4
3.2. Software certification 4
3.2.1. Process-oriented software certification 5
3.2.2. Semantic software certificates 5
3.2.3. Certified static analysis 6
4. Software **7**
4.1. Javalib 7
4.2. SAWJA 7
4.3. Jacal 7
4.4. Timbuk 7
5. New Results **8**
5.1. Control-Flow Analysis by Abstract Interpretation 8
5.2. Secure the Clones: Static Enforcement of Policies for Secure Object Copying 8
5.3. A formally verified SSA-based middle-end 8
5.4. Non linear analysis: fast inference of polynomial invariants 9
5.5. Result Certification of Static Analysis Results 9
5.6. Towards efficient abstract domains for regular language based static analysis 10
5.7. Cryptography 10
6. Bilateral Contracts and Grants with Industry **11**
6.1. Bilateral Project with FIME 11
6.2. The FRAE ASCERT project 11
7. Partnerships and Cooperations **11**
7.1. National Initiatives 11
7.1.1. The PiCoq ANR project 11
7.1.2. The ANR VERASCO project 12
7.1.3. ANR DECERT project 12
7.1.4. Labex COMIN Labs Seccloud project 12
7.2. European Initiatives 12
7.3. International Initiatives 12
7.4. International Research Visitors 13
8. Dissemination **13**
8.1. Scientific Animation 13
8.2. Teaching - Supervision - Juries 13
8.2.1. Teaching 13
8.2.2. Supervision 14
8.2.3. Juries 14
8.3. Popularization 14
9. Bibliography **15**

Project-Team CELTIQUE

Keywords: Programming Languages, Static Program Analysis, Security, Interactive Theorem Proving, Formal Methods, Proofs Of Programs

Creation of the Project-Team: July 01, 2009 .

1. Members

Research Scientists

Thomas Jensen [Team leader, DR, Inria, HdR]
Frédéric Besson [Inria, CR]
Arnaud Gotlieb [Inria, CR, HdR]
David Pichardie [Inria, CR, HdR]
Alan Schmitt [Inria, CR, HdR]

Faculty Members

Sandrine Blazy [University of Rennes 1, Prof, HdR]
Thomas Genet [University of Rennes 1, HdR]
David Cachera [Ens Cachan, HdR]
Pierre-Alain Fouque [ENS, HdR]
Arnaud Jobin [Ens Cachan, ATER]

Engineers

Vincent Monfort [Software Engineer]
Pierre Vittet [Software Engineer, from 15/11/2011]
Ronan Saillard [Software Engineer, from 1/12/2011]

PhD Students

Valérie Murat [MENRT grant, joint PhD with Distribcom team]
Yann Salmon [MENRT grant]
Delphine Demange [MENRT grant]
Pierre-Emmanuel Cornilleau [INRIA grant]
Zhoulai Fu [POLYTECHNIQUE grant]
André Oliveira Maroneze [MENRT grant, from 01/09/2010]
Stéphanie Riaud [INRIA-DGA grant, from 01/09/2011]
Jean-Christophe Zapalowicz [INRIA-DGA grant, from 01/12/2011]
Vincent Laporte [ANR grant]
Martin Bodin [ENS grant, from 01/09/2012]

Administrative Assistant

Lydie Mabil [Inria, TR]

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.

3. Scientific Foundations

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 [71].
- 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 [51], 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** [49], [74]. 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 [50]. 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 [3].

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) [42] 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 [72], [73] 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.,* [60], [59] have explored the algorithmic design space of contexts for benchmarks of significant size. Latter on, Milanova *et. al.,* [66] 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 [64] have extended the empiric evaluation of object-sensitivity using a BDD implementation allowing to cope with benchmarks otherwise out-of-scope. Besson and Jensen [46] 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 [65] and the race analysis of Naik, Aiken [67] and Whaley [68] 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 [55] 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 [63]. The proposed technique has been implemented in a tool called NIT [62]. Null pointer detection is also done by bug-detection tools such as FindBugs [62]. 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 [61], [41], linear logic approaches to implicit computational complexity [43], cost model for Java byte code [37] based on size relation inference, and WCET computation by abstract interpretation based loop bound interval analysis techniques [52].

We have proposed an original approach for designing static analyses computing program costs: inspired from a probabilistic approach [75], 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 [48]. An example of cache miss analysis has been developed within this framework [79].

3.1.3. Semantic analysis for test case generation

The semantic analysis of programs can be combined with efficient constraint solving techniques in order to extract specific information about the program, *e.g.*, concerning the accessibility of program points and feasibility of execution paths [76], [54]. As such, it has an important use in the automatic generation of test data. Automatic test data generation received considerable attention these last years with the development of efficient and dedicated constraint solving procedures and compositional techniques [58].

We have made major contributions to the development of **constraint-based testing**, which is a two-stage process consisting of first generating a constraint-based model of the program's data flow and then, from the selection of a testing objective such as a statement to reach or a property to invalidate, to extract a constraint system to be solved. Using efficient constraint solving techniques allows to generate test data that satisfy the testing objective, although this generation might not always terminate. In a certain way, these constraint techniques can be seen as efficient decision procedures and so, they are competitive with the best software model checkers that are employed to generate test data.

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.

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], [45]. 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 [47]. **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) [69] 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" [70]. Many solutions have been proposed to reduce the size of the trusted computing base. In the *foundational*

[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 ?

proof carrying code of Appel and Felty [40], [39], 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 [70] 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.* [38] 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 [36] 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 [5].

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 [56], [57]. We defined a simple and efficient algorithm [53] 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 [3]. This checker is now used to certify all analysis results produced by the regular completion tool as well as the optimised version of [44].

4. Software

4.1. Javalib

Participants: Frédéric Besson [correspondant], David Pichardie, Vincent Monfort.

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: 2.2
- Programming language: Ocaml

4.2. SAWJA

Participants: Frédéric Besson [correspondant], David Pichardie, Vincent Monfort.

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: 1.2
- Programming language: Ocaml

4.3. Jacal

Participants: Frédéric Besson [correspondant], Thomas Jensen, David Pichardie, Delphine Demange, Vincent Monfort, Pierre Vittet.

Static program analysis, Javacard, Certification, AFSCM

`Jacal` is a JAvaCard AnaLyseur developed on top of the SAWJA4.2 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 provide a CounterExample Guided Abstraction Refinement (CEGAR) tool for tree automata completion. The CEGAR part is based on the Buddy BDD library.

- Version: 3.1
- Programming language: Ocaml

5. New Results

5.1. Control-Flow Analysis by Abstract Interpretation

Control-flow analysis (CFA) of functional programs is concerned with determining how the program's functions call each other. In the case of the lambda calculus, this amounts to computing the flow of lambda expressions in order to determine what functions are effectively called in an application $(e_1\ e_2)$. This work shows that it is possible to use abstract interpretation techniques to derive systematically a control-flow analysis for a simple higher-order functional language. The analysis approximates the interprocedural control-flow of both function calls and returns in the presence of first-class functions and tail-call optimization. A number of advantages follow from taking this approach:

- The systematic derivation of a CFA for a higher-order functional language from a well-known operational semantics provides the resulting analysis with strong mathematical foundations. Its correctness follows directly from the general theorems of abstract interpretation.
- The approach is easily adapted to different variants of the source language. We demonstrate this by deriving a CFA for functional programs written in continuation-passing style.
- The common framework of these analyses enables their comparison. We take advantage of this to settle a question about the equivalence between the analysis of programs in direct and continuation-passing style.
- The resulting equations can be given an equivalent constraint-based presentation, providing *ipso facto* a rational reconstruction and a correctness proof of constraint-based CFA.

This work was published in the journal Information and Computation [14]

5.2. Secure the Clones: Static Enforcement of Policies for Secure Object Copying

Participants: Thomas Jensen, David Pichardie.

Exchanging mutable data objects with untrusted code is a delicate matter because of the risk of creating a data space that is accessible by an attacker. Consequently, secure programming guidelines for Java stress the importance of using defensive copying before accepting or handing out references to an internal mutable object.

However, implementation of a copy method (like clone()) is entirely left to the programmer. It may not provide a sufficiently deep copy of an object and is subject to overriding by a malicious sub-class. Currently no language-based mechanism supports secure object cloning.

We propose a type-based annotation system for defining modular copy policies for class-based object-oriented programs. A copy policy specifies the maximally allowed sharing between an object and its clone. We provide a static enforcement mechanism that will guarantee that all classes fulfill their copy policy, even in the presence of overriding of copy methods, and establish the semantic correctness of the overall approach in Coq.

The mechanism has been implemented and experimentally evaluated on clone methods from several Java libraries. The work as been presented at ESOP 2011. In 2012 a journal special issue has been published in Logical Methods in Computer Science [13].

5.3. A formally verified SSA-based middle-end

Participants: Delphine Demange, David Pichardie.

CompCert is a formally verified compiler that generates compact and efficient PowerPC, ARM and x86 code for a large and realistic subset of the C language. However, CompCert foregoes using Static Single Assignment (SSA), an intermediate representation that allows for writing simpler and faster optimizers, and is used by many compilers. In fact, it has remained an open problem to verify formally a SSA-based compiler middle-end.

We report on a formally verified, SSA-based, middle-end for CompCert. Our middle-end performs conversion from CompCert intermediate form to SSA form, optimization of SSA programs, including Global Value Numbering, and transforming out of SSA to intermediate form.

In addition to provide the first formally verified SSA-based middle-end, we address two problems raised by Leroy: giving a simple and intuitive formal semantics to SSA, and leveraging the global properties of SSA to reason locally about program optimizations. The work as been presented at ESOP 2012 [16].

5.4. Non linear analysis: fast inference of polynomial invariants

Participants: Thomas Jensen, David Cachera, Arnaud Jobin.

The problem of automatically inferring non-linear (polynomial) invariants of programs is still a challenge in program verification. A central observation in existing work on generating polynomial invariants is that n-ary relations between variables that can be described as the zeroes of a set of polynomials, correspond to a lattice of polynomials ideals. Such ideals are finitely generated , and all the approaches proposed so far in the literature rely on Gröbner base computations for computing ideal intersection or inclusion, or analysing the effects of polynomial assignments to variables. Computing Gröbner bases however slows down considerably the overall analysis.

We have proposed an abstract interpretation based method for inferring polynomial invariants that entirely avoids computing Gröbner bases. The method is precise and efficient, and is obtained without restricting the expressiveness of the polynomial programming language. Our analysis handles a general polynomial structured programming language that includes if and while constructs where branching conditions are both polynomial equalities and disequalities. Our analysis uses a form of weakest precondition calculus for showing that a polynomial relation $g = 0$ holds at the end of a program. We show that this backward approach, which was already observed to be well adapted to polynomial disequality guards can be extended to equality guards by using parameterized polynomial division.

Based on this anlysis, we have designed a constraint-based algorithm for inferring polynomial invariants. Such constraint-based techniques (rather than iteration) when dealing with loops means that it becomes feasible to analyse conditionals precisely, using parameterized polynomial division. A salient feature of this analysis, which distinguishes it from previous analyses, is that it does not require the use of Gröbner base computations. We have implemented this algorithm in Maple and 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 [19].

5.5. Result Certification of Static Analysis Results

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

Result Certification, Static program analysis, Decision procedures

We develop a lightweight approach for verifying *a posteriori* that the result of a static analysis is correct. The approach consists in encoding the program semantics directly inside an Intermediate Verification Language e.g., Why3 as an executable program interpreter. Running the standard VcGen of the IVL for the interpreter specialised for a program annotated with analysis results therefore amounts to generating program specific verification conditions [20]. This approach has the advantage of reducing the size of the Trusted Computing Base (TCB) because the VcGen is generic and language agnostic. Moreover, unlike traditional approaches, our TCB does not embed a compiler from the source code to the language of the IVL.

Verification conditions are usually discharged by Satisfiability Modulo Theory (SMT) provers that are therefore part of the TCB. To reduce further the TCB, we advocate for proof-generating SMT provers which results can be independently verified by reflexive Coq proof-checkers. For the EUF logic, we have proposed a novel compact format and proved correct an efficient Coq checker [17].

5.6. 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 properties on the program by showing that some "bad" terms, encoding dangerous or problematic configurations, are not in the superset and thus not reachable. With such technique, like with any approximated technique, is when the "bad" terms are in the superset. We proposed a new CounterExample Guided Abstraction Refinement (CEGAR) algorithm for tree automata completion. Our approach relies on a new equational-abstraction based completion algorithm to compute a regular overapproximation of the set of reachable states in finite time. This set is represented by, so-called, R/E-automata, a new extended tree automaton formalism whose structure can be exploited to detect and remove false positives in an efficient manner. Our approach has been implemented in Timbuk and used to analyze Java programs by exploiting a translation from the Java byte code to term rewriting systems. These results have been published in [18]. Now, we aim at applying this technique to the static analysis of programming languages whose semantics is based on terms, like functional programming languages. The first step in this direction is to take into account the evaluation strategy of the language when approximating the set of reachable terms [30].

5.7. Cryptography

Participants: Pierre-Alain Fouque, Jean-Christophe Zapalowicz.

Pierre-Alain Fouque joined the team Celtique from September 2011 to August 2012. As a cryptographer, he still worked on symmetric cryptography with his PhD and postdoc students and proposed new security analysis of the block-ciphers AES and Camellia using meet-in-the-middle techniques in [27], [22] at IWSEC'12 and Indocrypt'12 and new security proofs for signature schemes AbdallaFLT12 at Eurocrypt'12 and elliptic-curve hash function [25] at LatinCrypt'12 with nice properties.

With Pierre-Alain, we also worked on more practical security aspects since his delegation in the Celtique team was to study side-channel attacks and formal methods. In side-channel attacks, we work with people from DGA and NTT in Japan to present new efficient attacks on one well-known implementation of RSA in many smartcards. Our attack targets any implementation of RSA using the Chinese Remainder Theorem in order to speed-up the computation, any exponentiation algorithm and the Montgomery multiplication. Usually, public-key cryptography requires large integer arithmetic and in order to accelerate the computation of the modulo, Montgomery proposed a new algorithm that avoids the need of arbitrary euclidean division which is the most consuming part of the exponentiation algorithm. This algorithm uses a small register (8, 16 or 32 bits depending on the architecture) during the computation and if a fault makes the value of this register much shorter, we show that we can recover the factorization of the RSA modulus in polynomial time. Furthermore, we describe on many proposed hardware architectures that our attack can indeed be used in practice if a laser is used to provoke the fault. This article has been published at CHES'12.

With people from DGA, we also studied how fault attack can be used to have buffer overflow effects. Indeed, by accelerating the clock, it is possible to avoid some instruction in the assembler code of a function. Consequently, if a fault avoids the function epilogue that restores the stack and registers to the state they were in before the function was called, then the stack pointer is changed and we can execute another function. Such attacks show that code executed in embedded processor have to be protected using buffer overflow techniques.

Finally, we also worked with people from DGA and Grenoble University to study security proofs in a computational logic. We show that the mode of operations of some hash functions is secure in [21] and published at CSF'12. In particular, we show a small bug in the security proof of the sponge construction used in the new SHA-3 candidate and winner of the competition Keccak.

6. Bilateral Contracts and Grants with Industry

6.1. Bilateral Project with FIME

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

Static program analysis, Javacard, Certification, AFSCM

- Partner : FIME
- Period: Starting January 2012; ending February 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 outcome of the project is the Jacal (JAvaCard AnaLyser) (4.3.

6.2. The FRAE ASCERT project

Participants: Frédéric Besson, Sandrine Blazy, David Cachera, Thomas Jensen, David Pichardie, Pierre-Emmanuel Cornilleau.

Static program analysis, Certified static analysis

The ASCERT project (2009–2012) is founded by the *Fondation de Recherche pour l'Aéronautique et l'Espace*. It aims at studying the formal certification of static analysis using and comparing various approaches like certified programming of static analysers, checking of static analysis result and deductive verification of analysis results. It is a joint project with the Inria teams ABSTRACTION, GALLIUM and POP-ART.

7. Partnerships and Cooperations

7.1. National Initiatives

7.1.1. The PiCoq ANR project

Participant: Alan Schmitt.

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.

7.1.2. 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 founded 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.

7.1.3. ANR DECERT project

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

The DECERT project (2009–2012) is funded by the call Domaines Emergents 2008, a program of the Agence Nationale de la Recherche.

The objective of the DECERT project has been to design an architecture for cooperating decision procedures, with a particular emphasis on fragments of arithmetic, including bounded and unbounded arithmetic over the integers and the reals, and on their combination with other theories for data structures such as lists, arrays or sets. To ensure trust in the architecture, the decision procedures will either be proved correct inside a proof assistant or produce proof witnesses allowing external checkers to verify the validity of their answers.

This is a joint project with Systerel, CEA List and Inria teams Mosel, Cassis, Marelle, Proval and Celtique (coordinator).

7.1.4. Labex COMIN Labs Seccloud project

Participants: Frédéric Besson, 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.

7.2. European Initiatives

7.2.1. Collaborations with Major European Organizations

Imperial College (UK)

The JScert project (http://jscert.org) aims to really understand JavaScript by building models of ECMAScript semantics in the Coq proof assistant, and automated logical reasoning tools built on those semantics.

7.3. International Initiatives

7.3.1. Inria International Partners

Delphine Demange and David Pichardie have been working with Gilles Barthe from IMDEA Software, Madrid, Spain about the new verified SSA middle-end.

7.4. International Research Visitors

7.4.1. Visits to International Teams

David Pichardie has spent one year at Purdue University, Indiana, US (from September 2011 to August 2012) working with Jan Vitek and Suresh Jagannathan. This was a one year Inria sabbatical leave. The collaboration deals with the formal verification of a Java compiler, taking into account concurrency. As a first result, a paper will appear at POPL 2013 where we provide a new intermediate memory model for the Java language.

8. Dissemination

8.1. Scientific Animation

David Pichardie served on the program committees of the international conferences IFM 2012 and ITP 2012 and the international workshop SVARM & VERIFY 2012. He co-chaired the international workshop PxTP 2012. A. Schmitt is a member of the steering committee of the Journées Françaises des Langages Applicatifs (JFLA). Thomas Jensen and Alan Schmitt organized the École Jeunes Chercheurs en Programmation (EJCP 2012). Sandrine Blazy served on the organizing committee of the annual meeting of the GDR GPL French community. Sandrine Blazy is a member of the steering committe of the ITP conference. Sandrine Blazy is scientific director of the "Languages and software engineering department" of IRISA.

8.2. Teaching - Supervision - Juries

8.2.1. Teaching

Alan Schmitt, Caml, 43h, L3, Insa Rennes, France

Alan Schmitt, Méthodes Formelles pour le développement de logiciels sûrs, 12h, M1, Rennes 1, France

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

Sandrine Blazy, Software vulnerabilities, 24h, M2PRO, Rennes 1

Sandrine Blazy, Mechanised semantics, 30h, M2R, Rennes 1, France

Sandrine Blazy, The Coq proof assistant, 6h, 4th Asian-Pacific summer school on formal methods, Shanghai, China

Sandrine Blazy, Functional programming, 30h, L3, Rennes 1, France

Frédéric Besson, Compilation, 30h, M1, Insa Rennes, France

Thomas Jensen, Program analysis and Semantics, 30h, M2R, Rennes 1, France

Thomas Jensen, Sécurité de logiciel, 30h, M2R, Rennes 1, France

Pierre-Alain Fouque, Cryptography, 58h, M2PRO, Rennes 1, France

Thomas Jensen, Static program analysis, 5h, Ecole Jeunes Chercheurs en Programmation.

Thomas Genet, Cryptography, 18h, M1, Rennes 1, France

Thomas Genet, Object-oriented software engineering, 48h, M1, Rennes 1, France

Thomas Genet, Software Formal Analysis and Design, 65h, M1, Rennes 1, France

Thomas Genet, Functional programming, 44h, L3, Rennes 1, France

David Cachera, Programming language semantics, 24h, M1, Rennes 1, France

David Cachera, Computability and Logic, 24h, L3, ENS Cachan Bretagne, France

David Cachera, Formal Languages, 24h, L3, ENS Cachan Bretagne, France

David Cachera, Algorithmics, 10h, M2FES, ENS Cachan Bretagne, France

8.2.2. Supervision

PhD & HdR :

HdR : David Pichardie, Toward a Verified Software Toolchain for Java, ENS Cachan, November 19th 2012

PhD : Delphine Demange, Semantic foundations of intermediate program representations, ENS Cachan, October 19th 2012, Thomas Jensen and David Pichardie

PhD : Arnaud Jobin, Dioïdes et idéaux de polynômes en analyse statique, ENS Cachan, January 16th 2012, Thomas Jensen and David Cachera

PhD in progress: Zhoulai Fu, Abstract interpretation and memory analysis, octobre 2009, Thomas Jensen and David Pichardie

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

PhD in progress: Martin Bodin, Certified Analyses for JavaScript, September 2012, Thomas Jensen and Alan Schmitt

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

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

PhD in progress: J-C Zapalowicz, Formal methods for identifying security vulnerabilities, September 2011, Thomas Jensen and Pierre-Alain Fouque.

Pierre-Emmanuel Cornilleau: Result certification of static analyses using SMTs, October 2009, Frédéric Besson and Thomas Jensen.

PhD in progress: Valérie Murat, Automatic verification of infinite state systems using tree automata completion, septembre 2010, Thomas Genet

PhD in progress: Yann Salmon, Static Analysis of functional programs using tree automata, septembre 2011, Thomas Genet

8.2.3. Juries

Alan Schmitt, jury member for the HdR defense of Ludovic Henrio, July 19th 2012

Alan Schmitt, jury member (reviewer) for the PhD defense of Federico Ulliana, December 12 2012

Sandrine Blazy, jury member for the HDR defense of Eric Totel, December 2012, Supélec, Rennes

Sandrine Blazy, jury member (reviewer) for the PhD defense of Vincent Filou, December 2012, LABRI, Bordeaux

Thomas Jensen, jury member (reviewer) for the PhD defense of Tahina Ramananandro, January 2012, ENS Paris

Thomas Jensen, jury member (examiner) for the PhD defense of Jérémy Planul, February 2012, Ecole polytechnique

Pierre-Alain Fouque, jury member (reviewer) for the PhD defense of Olivier Meynard, January 2012, ENST Paris

Pierre-Alain Fouque, jury member (reviewer) for the PhD defense of Sylvain Heraud, February 2012, Nice-Sophia Antipolis

Pierre-Alain Fouque, jury member (reviewer) for the PhD defense of Christina Boura, January 2012, Paris 6 University

8.3. Popularization

David Pichardie organised the second 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 2012, there was about 90.000 participants.

Sandrine Blazy was interviewed for a 10 minutes podcast (in French) about compiler verification. The podcast is available from the Interstices website.

9. Bibliography

Major publications by the team in recent years

[1] 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, p. 273–291.

[2] 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, p. 642-666.

[3] 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, p. 347–362.

[4] 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, p. 589-624.

[5] 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, p. 56–78.

[6] F. CHARRETEUR, B. BOTELLA, A. GOTLIEB. *Modelling dynamic memory management in Constraint-Based Testing*, in "The Journal of Systems and Software", Nov. 2009, vol. 82, n^{o} 11, p. 1755–1766.

[7] 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, p. 574-597, http://hal.inria.fr/inria-00495405.

[8] 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, p. 1030–1044.

[9] 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, p. 101-115, http://hal.inria.fr/inria-00503953.

Publications of the year

Doctoral Dissertations and Habilitation Theses

[10] D. DEMANGE. *Semantic foundations of intermediate program representations*, ENS Cachan, 2012.

[11] A. JOBIN. *Diïdes et idéaux de polynômes en analyse statique*, ENS Cachan, 2012.

[12] D. PICHARDIE. *Toward a Verified Software Toolchain for Java*, ENS Cachan, 2012, Habilitation à Diriger des Recherches.

Articles in International Peer-Reviewed Journals

[13] T. JENSEN, F. KIRCHNER, D. PICHARDIE. *Secure the Clones: Static Enforcement of Policies for Secure Object Copying*, in "Logical Methods in Computer Science (LMCS)", 2012, vol. 8, n° 2.

[14] J. MIDTGAARD, T. JENSEN. *Control Flow Analysis of Function Calls and Returns by Abstract Interpretation*, in "Information and Computation", 2012, vol. 211, p. 49–76.

International Conferences with Proceedings

[15] M. ABDALLA, P.-A. FOUQUE, V. LYUBASHEVSKY, M. TIBOUCHI. *Tightly-Secure Signatures from Lossy Identification Schemes*, in "Advances in Cryptology - EUROCRYPT 2012", Lecture Notes in Computer Science, Springer, 2012, vol. 7237, p. 572-590, http://www.di.ens.fr/~fouque.

[16] G. BARTHE, D. DEMANGE, D. PICHARDIE. *A formally verified SSA-based middle-end - Static Single Assignment meets CompCert*, in "Proc. of 21th European Symposium on Programming (ESOP 2012)", Lecture Notes in Computer Science, Springer-Verlag, 2012, vol. 7211, p. 47-66.

[17] F. BESSON, P.-E. CORNILLEAU, R. SAILLARD. *Walking through the Forest: a Fast EUF Proof-Checking Algorithm*, in "Second International Workshop on Proof eXchange for Theorem Proving - PxTP 2012", 2012.

[18] Y. BOICHUT, B. BOYER, T. GENET, A. LEGAY. *Equational Abstraction Refinement for Certified Tree Regular Model Checking*, in "ICFEM'12", Kyoto, Japon, LNCS, Springer-Verlag, 2012, n° 7635, p. 299-315, http://hal.archives-ouvertes.fr/hal-00759149.

[19] D. CACHERA, T. JENSEN, A. JOBIN, F. KIRCHNER. *Inference of polynomial invariants for imperative programs: a farewell to Gröbner bases*, in "SAS - 19th International Static Analysis Symposium - 2012", Deauville, France, September 2012, Projet Région Bretagne CertLogs, http://hal.inria.fr/hal-00758890.

[20] P.-E. CORNILLEAU. *Prototyping Static Analysis Certification using Why3*, in "Boogie 2012: Second International Workshop on Intermediate Verification Languages", 2012.

[21] M. DAUBIGNARD, P.-A. FOUQUE, Y. LAKHNECH. *Generic Indifferentiability Proofs of Hash Designs*, in "25th IEEE Computer Security Foundations Symposium, CSF 2012", IEEE, 2012, p. 340-353, http://www.di.ens.fr/~fouque.

[22] P. DERBEZ, P.-A. FOUQUE, J. JEAN. *Faster Chosen-Key Distinguishers on Reduced-Round AES*, in "Indocrypt 2012", Lecture Notes in Computer Science, Springer, 2012, vol. 7668, p. 225-243, http://www.di.ens.fr/~fouque.

[23] P.-A. FOUQUE, N. GUILLERMIN, D. LERESTEUX, M. TIBOUCHI, J.-C. ZAPALOWICZ. *Attacking RSA-CRT Signatures with Faults on Montgomery Multiplication*, in "Cryptographic Hardware and Embedded Systems - CHES 2012", Lecture Notes in Computer Science, Springer, 2012, vol. 7428, p. 447-462, http://www.di.ens.fr/~fouque.

[24] P.-A. FOUQUE, D. LERESTEUX, F. VALETTE. *Using faults for buffer overflow effects*, in "ACM Symposium on Applied Computing, SAC 2012", ACM, 2012, p. 1638-1639, http://www.di.ens.fr/~fouque.

[25] P.-A. FOUQUE, M. TIBOUCHI. *Indifferentiable Hashing to Barreto-Naehrig Curves*, in "LATINCRYPT 2012", Lecture Notes in Computer Science, Springer, 2012, vol. 7533, p. 1-17, http://www.di.ens.fr/~fouque.

[26] A. HERVIEU, B. BAUDRY, A. GOTLIEB. *Managing Execution Environment Variability during Software Testing: an industrial experience*, in "International Conference on Testing Software and Systems", Aalborg, Denmark, Springer, November 2012, http://hal.inria.fr/hal-00726137.

[27] J. LU, Y. WEI, E. PASALIC, P.-A. FOUQUE. *Meet-in-the-Middle Attack on Reduced Versions of the Camellia Block Cipher*, in "International Workshop on Security, IWSEC 2012", Lecture Notes in Computer Science, Springer, 2012, vol. 7631, p. 197-215, http://www.di.ens.fr/~fouque.

National Conferences with Proceeding

[28] S. BOULIER, A. SCHMITT. *Formalisation de HOCore en Coq*, in "JFLA - Journées Francophones des Langages Applicatifs - 2012", Carnac, France, February 2012, http://hal.inria.fr/hal-00665945.

Conferences without Proceedings

[29] R. BEDIN FRANÇA, S. BLAZY, D. FAVRE-FELIX, X. LEROY, M. PANTEL, J. SOUYRIS. *Formally verified optimizing compilation in ACG-based flight control software*, in "ERTS2 2012: Embedded Real Time Software and Systems", Toulouse, France, AAAF, SEE, February 2012, http://hal.inria.fr/hal-00653367.

[30] T. GENET, Y. SALMON. *Proving Reachability Properties on Term Rewriting Systems with Strategies*, in "2nd Joint International Workshop on Strategies in Rewriting, Proving and Programming, IWS'12", London, 2012.

Research Reports

[31] T. GENET, T. LE GALL, A. LEGAY, V. MURAT. *Tree Regular Model Checking for Lattice-Based Automata*, Inria, April 2012, n^{o} RT-0424, 33, http://hal.inria.fr/hal-00687310.

[32] P. GENEVÈS, N. LAYAÏDA, A. SCHMITT. *Logical Combinators for Rich Type Systems*, Inria, July 2012, n^{o} RR-8010, 18, http://hal.inria.fr/hal-00714353.

[33] X. LEROY, ANDREW W. APPEL, S. BLAZY, G. STEWART. *The CompCert Memory Model, Version 2*, Inria, June 2012, n^{o} RR-7987, 26, http://hal.inria.fr/hal-00703441.

Other Publications

[34] A. BRIDE. *Vérification probabiliste de résultats d'analyse statique*, June 2012, http://dumas.ccsd.cnrs.fr/dumas-00725213.

[35] A. KUMAR, P.-A. FOUQUE, T. GENET, M. TIBOUCHI. *Proofs as Cryptography: a new interpretation of the Curry-Howard isomorphism for software certificates*, 2012, 19 pages, http://hal.inria.fr/hal-00715726.

References in notes

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

[37] 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, p. 113-132.

[38] 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, p. 380-397.

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

[40] 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.

[41] 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, p. 34–49.

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

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

[44] 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, p. 67-82.

[45] 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, p. 125-140.

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

[47] 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, p. 971-998, http://hal.inria.fr/inria-00537821.

[48] 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, p. 122-138.

[49] 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, p. 56–78.

[50] 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.

[51] 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, p. 238–252.

[52] 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.

[53] G. FEUILLADE, T. GENET, V. VIET TRIEM TONG. *Reachability Analysis over Term Rewriting Systems*, in "Journal of Automated Reasoning", 2004, vol. 33, n° 3–4, p. 341–383.

[54] C. FLANAGAN. *Automatic software model checking via constraint logic*, in "Sci. Comput. Program.", 2004, vol. 50, n° 1-3, p. 253-270.

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

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

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

[58] P. GODEFROID. *Compositional dynamic test generation*, in "POPL'07", 2007, p. 47-54.

[59] D. GROVE, C. CHAMBERS. *A framework for call graph construction algorithms*, in "Toplas", 2001, vol. 23, n° 6, p. 685–746.

[60] D. GROVE, G. DEFOUW, J. DEAN, C. CHAMBERS. *Call graph construction in object-oriented languages*, in "ACM SIGPLAN Notices", 1997, vol. 32, n° 10, p. 108–124.

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

[62] 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.

[63] 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, p. 132-149.

[64] 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° 1.

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

[66] 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° 1, p. 1–41.

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

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

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

[70] 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, p. 248-260.

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

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

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

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

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

[76] A. PODELSKI. *Model Checking as Constraint Solving*, in "SAS'00", 2000, p. 22-37.

[77] E. ROSE. *Lightweight Bytecode Verification*, in "Journal of Automated Reasoning", 2003, vol. 31, n^{o} 3–4, p. 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, p. 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, p. 153-167.

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

[81] J. WHALEY, M. S. LAM. *Cloning-based context-sensitive pointer alias analysis using binary decision diagrams*, in "PLDI '04", ACM, 2004, p. 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, p. 333–347.