INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team Celtique

Semantic analysis for software certification

Rennes - Bretagne-Atlantique

Theme : Programs, Verification and Proofs

Table of contents

1. Team

Research Scientist

Thomas Jensen [Team leader, DR, CNRS, HdR]
Frédéric Besson [Inria, CR]
Arnaud Gotlieb [Inria, CR]
David Pichardie [Inria, CR]
Olivier Heen [Industrial specialist]

Faculty Member

Sandrine Blazy [University of Rennes 1, from sept. 2009, HdR]
Thomas Genet [University of Rennes 1, HdR]
David Cachera [Ens Cachan, délégation Inria until 2009/08/31]

Technical Staff

Tiphaine Turpin [INRIA]
Christian Brunette [INRIA]
Nicolas Barré [Software Engineer, until 15/11/2009]
Benjamin Cama [Software Engineer, until 01/10/2009]
Nada Bendouro [Software Engineer, from 01/11/2009]
Vincent Monfort [Software Engineer, from 15/10/2009]

PhD Student

Florence Charreteur [MENRT grant]
Benoît Boyer [University Rennes 1 grant]
Laurent Hubert [BDI CNRS-Région Bretagne]
Mickael Delahaye [CEA grant]
Nadjib Lazaar [MENRT grant]
Arnaud Jobin [MENRT grant]
Delphine Demange [MENRT grant]
Pierre-Emmanuel Cornilleau [INRIA grant]
Zhoulai Fu [POLYTECHNIQUE grant]

Post-Doctoral Fellow

Frédéric Dabrowski [INRIA]
Florent Kirchner [INRIA, from 1/12/2009]
Matthieu Carlier [INRIA]

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.

2.2. Highlights of the year

CELTIQUE has achieved a rational reconstruction of standard control flow analysis techniques from basic abstract interpretation principles. The solution to this question—left open for more than ten years in the community—was obtained using a judicious combination of Galois connections and closure operators and was presented at this year's ACM International Conference on Functional Programming.

CELTIQUE contributed to the Javasec project, commissioned by the national information security agency (ANSSI), with an analysis of the intrinsic security of the Java language and a set of recommendations for how to enhance the security of a Java virtual machine. We also contributed to a "Developers guide to safe Java programming" to be published by ANSSI.

Euclide, the constraint-based test case generator for critical C programs developed by CELTIQUE, was presented at ICST 2009 (International Conference on Software Testing, Verification and Validation) at Denver, USA, in April, and the tool was also demonstrated at TAP 2009 (Test and Proofs) at Zurich, in July.

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 Celtiqueproject 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 [48], 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** [46], [77]. 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 [47]. 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 [44].

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) [37] is a scalable algorithm for constructing an initial call-graph of the program. Tip and Palsberg [84] 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 [75], [76] 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.,* [57], [56] have explored the algorithmic design space of contexts for benchmarks of significant size. Latter on, Milanova *et. al.,* [65] 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 [61] have extended the empiric evaluation of object-sensitivity using a BDD implementation allowing to cope with benchmarks otherwise out-of-scope. Besson and Jensen [41] proposed to use DATALOG in order to specify context-sensitive analyses. Whaley and Lam [85] 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 [62] 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 [51] 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 [60]. The proposed technique has been implemented in a tool called NIT [59]. Null pointer detection is also done by bug-detection tools such as FindBugs [59]. 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 Celtiqueproject, several approaches have been proposed for quantifying resource usage: a non-exhaustive list includes memory usage analysis based on specific type systems [58], [36], linear logic approaches to implicit computational complexity [38], cost model for Java byte code [31] based on size relation inference, and WCET computation by abstract interpretation based loop bound interval analysis techniques [49].

We have proposed an original approach for designing static analyses computing program costs: inspired from a probabilistic approach [78], 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 [45]. An example of cache miss analysis has been developed within this framework [82].

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 [79], [50]. 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 [55].

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 [81], [40]. 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 Celtiqueteam have proposed a verification technique that can check the proper use of resources of Java applications running on mobile telephones [10]. **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

[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 ?

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 proof carrying code* of Appel and Felty [34], [33], the code producer gives a direct proof that, in some "foundational" higher-order logic, the code respects a given security policy. Wildmoser and Nipkow [87], [86]. 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 [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.* [32] 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 [30] 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 [4].

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 [52], [53]. We defined a simple and efficient algorithm [6] 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 [39].

4. Software

4.1. Sawja: Static Analysis Workshop for Java Applications

Participants: Nicolas Barré, Frédéric Besson, Delphine Demange, Laurent Hubert, Vincent Monfort, David Pichardie, Tiphaine Turpin.

Javalib/Sawja is an OCaml platform for the development of static analyses of Java bytecode programs.

Javalib is a library to parse Java .class file into OCaml data structure, thus enabling the OCaml programmer to extract informations from class files, to manipulate and to generate valid class files. The library is maintained by the CELTIQUE team. It is distributed under the GNU General Public License.

On top of this library, we have developed the Sawja library that provides a high level representation of Java bytecode programs. Whereas Javalib is dedicated to isolated classes, Sawja handles bytecode programs with their class hierarchy and with control flow algorithms. Sawja provides some stackless intermediate representations of code. The transformation algorithm, common to these representations, has been formalized and proved to be semantics-preserving (see paragraph 5.1.2). This software is distributed under the GNU General Public License.

The Null-ability Inference Tool is based on this library. It is a tool [60], [59] to find suitable annotations for fields, method parameters and return values. It works at the bytecode level (on .class files or .jar files) so it can be used on programs where the source is not available. The tool has been presented by Laurent Hubert at JavaOne 2009 on the INRIA stand. This software is distributed under the GNU General Public License.

4.2. Constraint-Based Testing of critical C programs

Participant: Arnaud Gotlieb [contact point].

Euclide is software testing tool that features three main applications: structural test data generation, counter-example generation and partial program proving for critical C programs. The core algorithm of the tool takes as input a C program and a point to reach somewhere in the code. As a result, it outcomes either a test datum that reaches the selected point, or an “unreachable” indication showing that the selected point is unreachable. Optionally, the tool takes as input additional safety properties that can be given under the form of pre/post conditions or assertions directly written in the code. In this case, Euclide can either prove that these properties or assertions are verified according to an error-free semantics of the language or find a counter-example when there is one. As these problems are undecidable in the general case, Euclide only provides a semi-correct

procedure (when it terminates, it provides the right answer) for them. Hopefully, by restricting the subset of C that the tool can handle (no dynamic memory allocation, no recursion) these non-termination problems remain infrequent in practice. In addition, Euclide implements several procedures that combine atomic calls to the core algorithm. For example, by selecting appropriate points to reach in the source code, the tool can generate a complete test suite able to cover the all_statements or the all_decisions criteria.

4.3. Timbuk: a tree automata library

Participants: Thomas Genet, Benoît Boyer.

Timbuk [53] 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, *viz*, the boolean operations (intersection, union, complement), emptiness and inclusion checking, renaming, determinisation, transition normalisation, and a mechanism for building the tree automaton recognizing the set of irreducible terms for a left-linear TRS. This library also implements some more specific algorithms that we use for verification of cryptographic protocols and Java bytecode programs:

- exact computation of reachable terms for most of the known decidable classes of term rewriting systems,
- approximation of reachable terms and normal forms for any term rewriting system,
- matching in tree automata,
- the checker for approximations of reachable terms extracted from the Coq specification [44].

This software is distributed under the Gnu Library General Public License and is freely available at http://www.irisa.fr/lande/genet/timbuk/. Timbuk has been registered at the APP with number IDDN.FR.001.20005.00.S.P.2001.000.10600.

Timbuk is now in version 3.0 and provides tree automata completion with equational abstractions as proposed in 5.2.

Timbuk is used by other research groups to achieve cryptographic protocol verification. Frédéric Oehl and David Sinclair of Dublin University use it in an approach combining a proof assistant (Isabelle/HOL) and approximations (done with Timbuk) [74], [73]. Pierre-Cyrille Heam, Yohan Boichut and Olga Kouchnarenko of the Cassis Inria project use Timbuk as a verification back-end [42] for AVISPA [35]. AVISPA is a tool for verifying cryptographic protocols defined in high level protocol specification format. More recently, Timbuk was also used at LIAFA by Gael Patin, Mihaela Sighireanu and Tayssir Touili to design the SPADE tool whose purpose is to model-check multi-threaded and recursive programs.

5. New Results

5.1. Static Analysis of Object-Oriented Languages

Participants: Frédéric Dabrowski, Delphine Demange, Laurent Hubert, Thomas Jensen, David Pichardie.

The Celtiquegroup continues its investigation in various techniques for the static analysis of Object-Oriented Languages like Java.

5.1.1. Static Fields Static Analysis

Although in most cases class initialization works as expected, some static fields may be read before being initialized, despite being initialized in their corresponding class initializer. We propose an analysis [25] which computes, for each program point, the set of static fields that must have been initialized and discuss its soundness. We show that such an analysis can be directly applied to identify the static fields that may be read before being initialized and to improve the precision while preserving the soundness of a null-pointer analysis.

5.1.2. A Provably Correct Stackless Intermediate Representation For Java Bytecode

The Java virtual machine executes stack-based bytecode. The intensive use of an operand stack has been identified as a major obstacle for static analysis and it is now common for static analysis tools to manipulate a stackless intermediate representation (IR) of bytecode programs. Several algorithms have been proposed to achieve such a transformation, but only little attention has been paid to their formal semantic properties. In [28], we provide such a bytecode transformation, describes its semantic correctness and evaluates its performance with respect to the transformation time, the compactness of the obtained code and the impact on static analysis precision.

5.1.3. Datarace Analysis

A fundamental issue in multithreaded programming is detecting *data races*. A program is said to be well synchronised if it does not contain data races w.r.t. an interleaving semantics. Formally ensuring this property is central, because the Java Memory Model then guarantees that one can safely reason on the interleaved semantics of the program. In [20] we formalise in the Coq proof assistant a Java bytecode data race analyser based on the conditional must-not alias analysis of Naik and Aiken. The formalisation includes a context-sensitive points-to analysis and an instrumented semantics that counts method calls and loop iterations.

5.2. Static analysis based on rewriting and tree automata

Participants: Thomas Genet, Benoît Boyer, Olivier Heen.

5.2.1. Tree automata completion with equational abstractions

We have proposed a new language for defining regular approximations of set of reachable terms. Approximations are defined using equations which define equivalence classes of terms "similar" w.r.t. the approximation. The idea is close to the one developped with Valérie Viet Triem Tong [54] and more recently by José Meseguer, Miguel Palomino and Narciso Martí-Oliet [63]. With regards to this last work, the interest of our approach is that it imposes fewer restriction on the equations used to define approximation. Our only syntactical constraint is that equations have to be linear though [63] imposes that the term rewriting system and the set of equations have to be coherent which is a more drastic restriction. Our proposition, published in [12], consists in using the equations to detect equivalent terms recognized by the tree automata and merge the recognizing states so as to mimic the construction of equivalence classes. We have also proven a precision result showing that, under some retrictions on the initial language, our algorithm builds no more than terms reachable by rewriting modulo the set of equations.

5.2.2. Verification of Temporal Properties on Tree Automata

In the static analysis framework based on term rewriting systems and tree automata, we only consider the reachability and unreachability problem, i.e. is a term (representing a program configuration) reachable or not? This is closely related to so-called *safety properties*. In a recent work [18], we have achieved a step further and consider temporal properties, like *liveness properties*. From the tree automata produced by the new completion algorithm proposed in [12], we managed to extract a Büchi automaton representing the behaviour of the term rewriting system. The extracted Büchi automaton models exactly the rewriting steps at a given depth in a term. For the moment, our technique is only able to deal with term rewriting systems having a finite set of reachable terms, thus doing no more that usual finite model-checking. However, defining approximations is easy on the tree automata completion framework. Hence, we are currently improving this preliminary work so as to deal with verification of temporal properties on infinite-state models, using approximations.

5.2.3. Verification of cryptographic protocols

With respect to verification of cryptographic protocols, the last developments were done around the SPAN verification tool: http://www.irisa.fr/lande/genet/span/. We carried out verification of protocols for ad hoc network. Even with a few participants and a few messages, there is a loss of intuition that may lead to vulnerabilities in those particular protocols. We have automatically verified some security properties of the protocol designed for vehicular ad hoc networks [24], [23]. During all the verification process, SPAN was

useful to check the adequation between the model and the real protocol. It also provided a critical advantage for convincing automotive industry people of the validity of our approach. It is worth notifying that, during the IEEE VNC 2009 conference, three talks (including our talk) underlined the need of formal security verification in vehicular ah hoc network. We believe that this field will provide interesting uses cases and verification needs, exactly as the aviation industry did these last 20 years.

5.3. Certified Static Analysis and Compilation

Participants: Frédéric Besson, Sandrine Blazy, David Cachera, Thomas Jensen, David Pichardie.

A certified static analysis is an analysis whose semantic validity has been formally proved correct with a proof assistant. The recent increasing interest in using proof assistants for mechanizing programming language metatheory has given rise to several approaches for certification of static analysis. We propose in [19] a panorama of these techniques and compare their respective strengths and weaknesses.

In [27] we propose a tutorial on building a certified static analysis in Coq. We study a simple bytecode language for which we propose an interval analysis that allows to verify statically that no array-out-of-bounds accesses will occur.

Proving the correctness of an analyzer is based on semantic properties, and becomes difficult to ensure when complex analysis techniques are involved. In [14] we propose to adapt the general theory of static analysis by abstract interpretation to the framework of constructive logic. Implementing this formalism into the Coq proof assistant then allows for automatic extraction of certified analyzers. We focus in this work on a simple imperative language and present the computation of fixpoints by widening/narrowing and syntax-directed iteration techniques.

Iterated Register Coalescing (IRC) is a widely used heuristic for performing register allocation via graph coloring. Many implementations in existing compilers follow the imperative algorithm published in 1996. In [16], we present a formal verification of the whole IRC algorithm, that can be used as a reference for IRC. We also define the theory of register-interference graphs in Coq; we implement a purely functional version of the IRC algorithm, and we prove its total correctness. The automatic extraction of our IRC algorithm yields a program with competitive performance. This work has been integrated into the CompCert verified compiler.

In [17], we focus on optimal register allocation and we present two compiler optimizations for reducing interference graphs, while preserving optimality. This work has been done while Sandrine Blazy was a member of the Gallium group, as well as the definition of a formal semantics for the Clight source language of the CompCert compiler [11].

5.4. Control-flow analysis

Participants: Frédéric Besson, Thomas Jensen, Tiphaine Turpin.

Control-flow analysis (CFA) is a fundamental static analysis on which many other analyses rely. As such it has been the focus of researchers throughout the past two decades.

5.4.1. A Calculational Approach to Control-Flow Analysis by Abstract Interpretation

Surprisingly, very few formulate CFA within the classical abstract interpretation methodology. Such a formulation of CFA is advantageous in that it is constructive: Rather than proving CFA safe a priori, CFA is induced by systematically composing and calculating with Galois connections. Unfortunately it has remained an open problem of how to exploit Galois connections and widenings for CFA since its formulation by Nielson and Nielson [72]. The work [26] represents a complete answer to this question for 0-CFA of higher-order functional languages.

We present a derivation of a control-flow analysis by abstract interpretation. Our starting point is a transition system semantics defined as an abstract machine for a small functional language in continuation-passing style. We obtain a Galois connection for abstracting the machine states by composing Galois connections, most notable an independent-attribute Galois connection on machine states and a Galois connection induced by a closure operator associated with a constituent-parts relation on environments. We calculate abstract transfer functions by applying the state abstraction to the collecting semantics, resulting in a novel characterization of a standard demand-driven control-flow analysis – namely 0-CFA.

5.4.2. CPA beats oo-CFA

There is a generic framework for defining context-sensitive control-flow analyses. Various notions of contexts have been proposed allowing to trade time for speed. We have formally established a conjecture of Grove *et al.*, [56] stating that Agesen's Cartesian Product Algorithm (CPA) is strictly more precise than oo-CFA [15]. This result holds despite the fact that (contrary to CPA) computing oo-CFA would require an infinite number of contexts. For the sake of the proof we define a core object-oriented language and prove correct a generic control-flow analysis. This generic analysis is then instantiated using the CPA and oo-CFA contexts. The proof consists in showing that the concrete states approximated by CPA are a subset of those computed by oo-CFA.

5.4.3. BDD-based computation of control-flow analyses

DATALOG and BDDS have been proposed to compute the results of context-sensitive control-flow analyses [41], [85]. We are working on lifting the expressiveness restrictions imposed by DATALOG while retaining the efficiency of BDDs. To reach this goal, we are developing a theory for computing the least-fixpoint semantics of PROLOG programs using BDD operations [29]. Over DATALOG, PROLOG has the advantage of providing first-order terms thus allowing for a more natural specification of control-flow analyses. The implementation and evaluation of a prototype based on this theory is in progress.

5.5. Constraint-based test case generation for critical C programs

Participants: Arnaud Gotlieb, Benjamin Cama, Nada Bendouro.

Euclide is a new Constraint-Based Testing tool for verifying safety-critical C programs. By using a mixture of symbolic and numerical analyses (namely static single assignment form, constraint propagation, integer linear relaxation and search-based test data generation), it addresses three distinct applications in a single framework: structural test data generation, counter-example generation and partial program proving. The main capabilities of the tool were presented in [21] and its usage for verifying safety properties for a well-known critical C component of the TCAS (Traffic Collision Avoidance System) was presented in [13]. The tool lies on theoretical foundations that were partially presented in [22].

6. Contracts and Grants with Industry

6.1. The JAVASEC project

Participants: Thomas Jensen, David Pichardie, Frédéric Dabrowski, Christian Brunette.

The Java programming language has been put forward as a language with strong security and several aspects of the language are definite improvements over languages such as C and C++. However, the security architecture is complex and it is not straightforward for a Java developer to identify the security risks that a particular piece of code may imply. The French National Information Security Agency (*Agence Nationale de la Sécurité de Systèmes Informatiques (ANSSI)*) commissioned the JAVASEC project with the double aim of providing secure programming guidelines to Java developers and to build a security-enhanced Java virtual machine whose security can be evaluated and certified according to industrial standards and that can serve as a secure platform for executing Java applications. The results have been an in-depth analysis of Java, its security architecture, its language features relevant to security and the pertinence of formal methods for enhancing the security of Java applications. This analysis has lead to a "Secure Java development guide", that provides a series of guidelines for what to do an not to do when developing security-critical applications in Java. As a complement to the guidelines, we have identified a series of program properties that can be verified by static analysis of Java byte code in order to improve further the security checks offered by the Java byte code verifier.

The project is conducted in collaboration with two Rennes located SMEs: Silicom and Amossys.

6.2. The FRAE ASCERT project

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

The ASCERT project (2009–20012) 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.

6.3. ANR DECERT project

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

The DECERT project (2009–2011) is funded by the call Domaines Emergents 2008, a program of the Agence Nationale de la Recherche.

The objective of the DECERT project is 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 Systeral, CEA List and INRIA teams Mosel, Cassis, Marelle, Proval and Celtique (coordinator).

6.4. The CERTLOGS project

Participants: Thomas Genet, Thomas Jensen, David Pichardie.

The CERTLOGS project (2009–20012) is funded by the CREATE action of the *Région Bretagne*. The objective of this project is to develop new kinds of program certificates and innovating certifying verification techniques using static analysis as the fundamental tool and combine this with techniques coming from probabilistic algorithms and cryptography.

6.5. The RNTL CAT project

Participants: Arnaud Gotlieb, Thomas Jensen.

The RNTL CAT project (2006–2009) aims at developing techniques and tools for analysing critical C programs. In this project, we focus on exploring the capabilities of constraint techniques to address the verification of C programs that manipulates complex computations (non-linear operators) and pointers. The other members of the project are the CEA LIST laboratory (project leader), Proval (Inria Futurs), France Télécom R&D, Dassault-Aviation, Siemens VDO and Airbus Industries.

6.6. The ANR U3CAT project

Participants: Sandrine Blazy, Matthieu Carlier, Arnaud Gotlieb, David Pichardie.

The ANR U3CAT project (2009–2012) is built upon the results of the RNTL CAT project, which delivered the Frama-C platform for the analysis of C programs and the ACSL assertion language. The ANR U3CAT project focuses on providing a unified interface that would allow to perform several analyses on a same code and to study how these analyses can cooperate in order to prove properties that culd not have been established by one single technique. The other members of the project are the CEA LIST laboratory (project leader), Proval (Inria Futurs), Gallium (Inria Paris-Rocquencourt), Cedric (CNAM), Atos Origin, CS, Dassault-Aviation, Sagem Defense and Airbus Industries.

6.7. European FET-Integrated project MOBIUS

Participants: Thomas Jensen, Frédéric Besson, David Pichardie, Tiphaine Turpin.

Mobius (IST-15905) is an Integrated Project launched under the FET Global Computing Proactive Initiative. The project has started on September 1st 2005 for 48 months and involves 16 partners. The goal of this project is to develop the technology for establishing trust and security for mobile devices using the Proof Carrying Code (PCC) paradigm. Proof Carrying Code is a technique for downloading mobile code on a host machine while ensuring that the code adheres to the host's security policy. The basic idea is that the code producer sends the code with a formal proof that the code is secure. Upon reception of the code, the receiver uses a simple and fast proof validator to check, with certainty, that the proof is valid and hence the untrusted code is safe to execute.

In this project, we participate in the specification of security requirements and resource policies to be studied throughout the project. We have contributed with techniques for generating small and easy to check PCC certificates for resource-aware static analyses. One of the major achievement of the project has been to run PCC checkers on resource-constrained mobile devices.

6.8. The ANR SETIN RAVAJ

Participants: Nicolas Barré, Benoît Boyer, Thomas Genet, Thomas Jensen.

The RAVAJ ANR (http://www.irisa.fr/lande/genet/RAVAJ/) started on january 2007, for 3 years. RAVAJ means "Rewriting and Approximation for the Verification of Java Applications". Thomas Genet is the coordinator of this project that concerns partners from LORIA (Nancy), LIFC (Besançon) and IRISA (Rennes). The goal of this project is to propose a general purpose verification technique based based on approximations and reachability analysis over term rewriting systems. To tackle this goal, the tree automata completion method has to be refined in two different ways. First, though the Timbuk tool is efficient enough to verify cryptographic protocols, it is not the case for more complex software systems. In that direction, we aim at using some results obtained in rewriting [66] to bring the efficiency of our tool closer to what has been obtained in the model-checking domain. Second, automation of approximation has to be enhanced. At present, the approximation automaton construction is guided by a set of approximation rules very close to the tree automata formalism and given by the user of the tool. On the one hand, we plan to replace approximation rules, which are difficult to define by a human, by approximation equations which are more natural. Approximation equations define equivalence classes of terms equal modulo the approximation as in [64] [83] [54]. On the other hand, we will automatically generate approximation equations from the property to be proved, using [42] [43], and also provide an automatic approximation refinement methodology adapted to the equational approximation framework.

6.9. The ANR SETIN PARSEC

Participants: Frédéric Besson, Frédéric Dabrowski, Thomas Jensen, David Pichardie.

The ParSec project (2007–2010) intends to study concurrent programming techniques for new computing architectures like multicore processors or multiprocessor machines, focusing on the security issues that arise in multi-threaded systems. In this project the CELTIQUE team focuses on static analysis of multi-threaded Java programs and specially on data race checkers. The other members of the project are INRIA Sophia-Antipolis, INRIA Rocquencourt and PPS (Université Paris 7).

6.10. ANR SESUR 2007 CAVERN

Participants: Arnaud Gotlieb, Florence Charreteur.

The CAVERN project (Constraints and Abstractions for program VERificatioN) (aims to enhance the potential of Constraint Programming for the automated verification of imperative programs. The classic approach consists in building a constraint system representating the objective to meet. Constraint solving is currently delegated to "generic" constraint propagation based solvers developed for other applications (combinatorial optimization, planning, etc.). The originality of the project lies in the design of abstraction-based constraint solver dedicated to the automated testing of imperative programs. In Static Analysis, the last few years have seen the development of powerful techniques over various abstract domains (polyhedra, congruence, octagons, etc.) and this project aims to explore results obtained in this area to develop constraint solvers with improved deductive capabilities. The main scientific outcome of the project will be a profound understanding of the benefit of using abstraction techniques in constraint solvers for the automated testing of imperative programs.

The CAVERN project includes four partners involved in the development of constraint-based testing tools:

- the CELTIQUE team of IRISA in Rennes (CELTIQUE) - coordinator
- the "Constraints and Proofs" team from CNRS I3S laboratory in Sophia-Antipolis(CeP)
- the CEA-LIST laboratory in Saclay (CEA)
- the ILOG Company in Gentilly (ILOG)

In addition, the project will include a foreign associate partner: Andy King from the University of Kent.

Concretely, the CAVERN project partners will study the integration of selected abstractions in their own constraint libraries, as currently used in their testing tools, in order to improve the treatment of loops, memory accesses (references and dynamic structures) and floating-point computations. Dealing efficiently with these constructs will allow us to scale-up constraint-based testing techniques for imperative programs. This should open the way to more automated testing processes which will facilitate software dependability assessment.

6.11. The COST Action IC0701

Participants: Thomas Jensen, David Pichardie.

COST Action IC0701 is a European scientific cooperation. The Action aims at developing verification technology with the power to ensure dependability of object-oriented programs on industrial scale. The action is composed of 15 countries. The COST action has been a forum for presenting our results concerning the data race analysis and our proposal for an intermediate language into which Java byte code can be transformed in order to faciliate the static analysis of byte code programs.

7. Dissemination

7.1. Conferences: program committees, organization, invitations

Thomas Jensen gave an invited talk on “From stack maps to software certificates” at the 2009 International Byte code workshop at ETAPS.

Thomas Jensen was co-chair of the program committee for the 2009 Proof-carrying Code workshop. He served on the program committee of the 35th Int. Conf. on Curr. Trends in Theory and Practice of Computer Science (Sofsem). Foundations track. 2009, the ACM SIGPLAN 2009 Workshop on Partial Evaluation and Program Manipulation (PEPM ’09), the 14th IEEE International Conference on Engineering of Complex Computer Systems, 2009 and the workshop on Foundational and Practical Aspects of Resource Analysis (FOPARA’09).

Olivier Heen was on the program committee of CESAR 2008, ACM-WISTP 2008, SSTIC 2008 and SAR-SSI 2008. He is also co-organiser of the DIWALL seminar.

David Pichardie was on the program committee of the BYTECODE’09 international workshop and the PCC’09 international workshop.

7.2. PhD and habilitation theses defended

Thomas Genet defended his Habilitation thesis "Reachability Analysis of Rewriting for Software Verification" on November 30 [9].

7.3. PhD and Habilitation committees

David Pichardie was external reviewer on the PhD thesis of Miguel Gomez-Zamalloa at the Complutense University of Madrid (spanish students need two positive reviews from two European Non-Spanish researchers to start the standard Phd defense process).

Thomas Jensen served as external reviewer on the HdR of Etienne Payet, U. de la Réunion, and on the PhD theses of Y. Zhang, Technical U. Denmark, and Jean-Baptiste Tristan (U. Paris 7). He was president of the jury for the PhD thesis of César Kunz (Mines ParisTech).

Arnaud Gotlieb was external examiner on the PhD thesis of Matthieu Carlier at ENSIIE Evry.

7.4. Teaching: university courses and summer schools

Sandrine Blazy taught two 32-hour lectures at the Master 2 level (on reliable software and on software vulnerabilities).

David Cachera teaches theoretical computer science (in charge of 4 modules) at École Normale Supérieure de Cachan.

Thomas Genet teaches Cryptographic Protocols and their verification for M2 level (5th university year). He also teaches formal methods for software verification and model driven design at M1 level (4th university year).

Arnaud Gotlieb teaches code-based testing at M2 level in collaboration with Thierry Jéron (VERTECS project) and Sophie Pinchinat (S4 project) in the VTS module. He is principal teacher and responsible of the 5INFO module "Software Testing" at the 5th year of Insa Rennes. He also teaches "Code-based Testing" at the Ecole des Mines de Nantes at the Master level.

Thomas Jensen and David Pichardie taught semantics, type systems and abstract interpretation at Master 2 level.

David Pichardie also taught theoretical computer science at École Normale Supérieure de Cachan and formal methods for software engineering (the B method) at the 4th year of Insa Rennes in collaboration with Mireille Ducassé. David Pichardie gave a four hours lecture on certified static analysis at the 9th International School on Foundations of Security Analysis and Design (Bertinoro, Italy, September 2009).

Thomas Genet gave a lecture on "Cryptographic protocols: principles, attacks and verification tools" at the summer school "École Jeune Chercheurs en Programmation" (Rennes, may 2009).

Thomas Jensen is scientific leader of the *École Jeunes Chercheurs en Programmation*, an annual summer school for graduate students on programming languages and verification, organized under the auspices of the CNRS GdR ALP. This year's event was organised by Vlad Rusu and took place in Dinard and Rennes.

7.5. Administrative responsibilities

Thomas Jensen is *délégué scientifique* for the INRIA centre in Rennes and president of the joint Scientific Committee (*comité des projets*) between Irisa and Inria Rennes Bretagne Atlantique. Through this duty he is member of the INRIA evaluation board.

Sandrine Blazy is in charge of a graduate curriculum (M2 level) at Université de Rennes 1. dedicated to information system security. Thomas Genet is in charge of the first year of the Master in Computer Science at Université de Rennes 1.

Thomas Jensen is member of the executive bureau of the French network GDR GPL on software engineering and formal methods in programming.

8. 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 "Special Issue on Applied Semantics of Theoretical Computer Science", vol. 364, n^{o} 3, 2006, 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, vol. 5142, Springer-Verlag, 2008, 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, vol. 5195, Springer-Verlag, 2008, p. 347–362.

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

[5] F. CHARRETEUR, B. BOTELLA, A. GOTLIEB. *Modelling dynamic memory management in Constraint-Based Testing*, in "The Journal of Systems and Software", vol. 82, n^{o} 11, Nov. 2009, p. 1755–1766, Special Issue: TAIC-PART 2007 and MUTATION 2007.

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

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

[8] A. GOTLIEB. *EUCLIDE: A Constraint-Based Testing platform for critical C programs*, in "2th International Conference on Software Testing, Validation and Verification (ICST'09), Denver, CO", Apr. 2009.

Year Publications

Doctoral Dissertations and Habilitation Theses

[9] T. GENET. *Reachability Analysis of Rewriting for Software Verification*, Université de Rennes 1, 2009, Habilitation à Diriger des Recherches.

Articles in International Peer-Reviewed Journal

[10] F. BESSON, T. JENSEN, G. DUFAY, D. PICHARDIE. *Verifying Resource Access Control on Mobile Interactive Devices*, in "Journal of Computer Security", 2010, To appear.

[11] S. BLAZY, X. LEROY. *Mechanized semantics for the Clight subset of the C language*, in "Journal of Automated Reasoning", vol. 43, n^{o} 3, 2009, p. 263-288.

[12] T. GENET, V. RUSU. *Equational Tree Automata Completion*, in "Journal of Symbolic Computation", 2010, To Appear.

[13] A. GOTLIEB. *TCAS software verification using Constraint Programming*, in "The Knowledge Engineering Review", 2009, Under revision.

Articles in National Peer-Reviewed Journal

[14] D. CACHERA, D. PICHARDIE. *Programmation d'un interpréteur abstrait certifié en logique constructive*, in "Technique et Science Informatiques (TSI)", 2010, To appear.

International Peer-Reviewed Conference/Proceedings

[15] F. BESSON. *CPA beats oo-CFA*, in "Proceedings of the 11th International Workshop on Formal Techniques for Java-like Programs", ACM, 2009, p. 1–6.

[16] S. BLAZY, B. ROBILLARD, A. APPEL. *Formal Verification of Coalescing Graph-Coloring Register Allocation*, in "Proceedings of the 19th European Symposiumon Programming (ESOP 2010)", Lecture Notes in Computer Science, vol. 6012, Springer-Verlag, 2010, 20 pages, to appear US .

[17] S. BLAZY, B. ROBILLARD. *Live-range Unsplitting for Faster Optimal Coalescing*, in "Proceedings of the ACM SIGPLAN/SIGBED 2009 conference on Languages, Compilers, and Tools for Embedded Systems (LCTES 2009)", ACM, 2009, p. 70–79, http://doi.acm.org/10.1145/1542452.1542462.

[18] B. BOYER, T. GENET. *Verifying Temporal Regular properties of Abstractions of Term Rewriting Systems*, in "Proc. of RULE'09", EPTCS, 2010, To Appear.

[19] D. CACHERA, D. PICHARDIE. *Comparing Techniques for Certified Static Analysis*, in "Proc. of the 1st NASA Formal Methods Symposium (NFM'09)", NASA Ames Research Center, 2009, p. 111-115.

[20] F. DABROWSKI, D. PICHARDIE. *A Certified Data Race Analysis for a Java-like Language*, in "Proc. of 22nd International Conference on Theorem Proving in Higher Order Logics (TPHOLs'09)", Lecture Notes in Computer Science, vol. 5674, Springer-Verlag, 2009, p. 212-227.

[21] A. GOTLIEB. *EUCLIDE: A Constraint-Based Testing platform for critical C programs*, in "2th International Conference on Software Testing, Validation and Verification (ICST'09), Denver, CO", Apr. 2009.

[22] A. GOTLIEB, M. PETIT. *Towards a Theory for Testing Non-terminating Programs*, in "33nd Annual IEEE International Computer Software and Applications Conference (COMPSAC'09), Seattle, USA", Jul. 2009, 6 pages.

[23] G. GUETTE, O. HEEN. *A TPM-based Architecture for Improved Security and Anonymity in Vehicular Ad hoc Networks*, in "In International Vehicular Networking Conference (IEEE VNC 2009)", 2009.

[24] O. HEEN, G. GUETTE, T. GENET. *On the Unobservability of a Trust Relation in Mobile Ad Hoc Networks*, in "WISTP 2009 3rd edition", LNCS, vol. 5746, Springer, 2009.

[25] L. HUBERT, D. PICHARDIE. *Soundly Handling Static Fields: Issues, Semantics and Analysis*, in "Proc. of the 4th International Workshop on Bytecode Semantics, Verification, Analysis and Transformation (BYTECODE'09)", Electronic Notes in Theoretical Computer Science, vol. 253, n^{o} 5, 2009, p. 15–30.

[26] J. MIDTGAARD, T. JENSEN. *Control-flow analysis of function calls and returns by abstract interpretation*, in "Proceedings of the 14th ACM international conference on Functional programming", ACM, 2009, p. 287–298 DK .

Scientific Books (or Scientific Book chapters)

[27] F. BESSON, D. CACHERA, T. JENSEN, D. PICHARDIE. *Certified Static Analysis by Abstract Interpretation*, in "Foundations of Security Analysis and Design V, FOSAD 2007/2008/2009 Tutorial Lectures", Lecture Notes in Computer Science, vol. 5705, Springer-Verlag, 2009, p. 223-257.

Research Reports

[28] D. DEMANGE, T. JENSEN, D. PICHARDIE. *A Provably Correct Stackless Intermediate Representation For Java Bytecode*, n^{o} RR-7021, INRIA, 2009, http://hal.inria.fr/inria-00414099/en/, Research Report.

[29] T. TURPIN, F. BESSON, T. JENSEN. *Computing the Least Fix-point Semantics of Logic Programs Using BDDs*, n^{o} 7107, INRIA, 2009, Research Report.

References in notes

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

[31] 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.

[32] 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.

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

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

[35] A. ARMANDO, D. BASIN, Y. BOICHUT, Y. CHEVALIER, L. COMPAGNA, J. CUELLAR, P. HANKES DRIELSMA, P.-C. HÉAM, O. KOUCHNARENKO, J. MANTOVANI, S. MÖDERSHEIM, D. VON OHEIMB, M. RUSINOWITCH, J. SANTOS SANTIAGO, M. TURUANI, L. VIGANÒ, L. VIGNERON. *The AVISPA Tool for the automated validation of internet security protocols and applications*, in "CAV'2005", LNCS, vol. 3576, Springer, 2005, p. 281-285.

[36] 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.

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

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

[39] 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, vol. 5140, Springer-Verlag, 2008, p. 67-82.

[40] 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, vol. 4421, Springer-Verlag, 2007, p. 125-140.

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

[42] Y. BOICHUT, P.-C. HÉAM, O. KOUCHNARENKO. *Automatic Approximation for the Verification of Cryptographic Protocols*, in "Proc. AVIS'2004, joint to ETAPS'04, Barcelona (Spain)", 2004.

[43] Y. BOICHUT, P.-C. HÉAM, O. KOUCHNARENKO. *Automatic Verification of Security Protocols Using Approximations*, n^{o} RR 5727, INRIA, 2005, Research Report.

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

[45] 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, vol. 5140, Springer-Verlag, 2008, p. 122-138.

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

[47] 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.

[48] 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.

[49] 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.

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

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

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

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

[54] T. GENET, V. VIET TRIEM TONG. *Proving Negative Conjectures on Equational Theories using Induction and Abstract Interpretation*, n^{o} RR-4576, INRIA, 2002, Technical report.

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

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

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

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

[59] 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.

[60] 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, vol. 5051, Springer-Verlag, 2008, p. 132-149.

[61] 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.", vol. 18, n^{o} 1, 2008.

[62] 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.

[63] J. MESEGUER, M. PALOMINO, N. MARTÍ-OLIET. *Equational abstractions*, in "TCS", vol. 403, n^{o} 2-3, 2008, p. 239-264.

[64] J. MESEGUER, M. PALOMINO, N. MARTÍ-OLIET. *Equational Abstractions*, in "Proc. 19th CADE Conf., Miami Beach (Fl., USA)", LNCS, vol. 2741, Springer, 2003, p. 2-16.

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

[66] P.-E. MOREAU, C. RINGEISSEN, M. VITTEK. *A Pattern Matching Compiler for Multiple Target Languages*, in "12th Conference on Compiler Construction, Warsaw (Poland)", G. HEDIN (editor), LNCS, vol. 2622, Springer, May 2003, p. 61-76, http://www.loria.fr/~moreau/Papers/MoreauRV-CC2003.ps.gz.

[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. 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] H. R. NIELSON, F. NIELSON. *Infinitary Control Flow Analysis: a Collecting Semantics for Closure Analysis*, in "Proc. of the 24th ACM Symposium on Principles of Programming Language", ACM Press, 1997, p. 332-345.

[73] F. OEHL, G. CÉCÉ, O. KOUCHNARENKO, D. SINCLAIR. *Automatic Approximation for the Verification of Cryptographic Protocols*, in "Proc. of FASE'03", LNCS, vol. 2629, Springer, 2003, p. 34-48.

[74] F. OEHL, D. SINCLAIR. *Combining two approaches for the formal verification of cryptographic protocols*, in "Proceedings of ICLP Workshop on Specification, Analysis and Validation for Emerging technologies in computational logic", 2001.

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

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

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

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

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

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

[81] A. SABELFELD, A. C. MYERS. *Language-based Information-Flow Security*, in "IEEE Journal on Selected Areas in Communication", vol. 21, n^{o} 1, January 2003, p. 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, vol. 164, Elsevier, 2006, p. 153-167.

[83] T. TAKAI. *A Verification Technique Using Term Rewriting Systems and Abstract Interpretation*, in "Proc. 15th RTA Conf., Aachen (Germany)", LNCS, vol. 3091, Springer, 2004, p. 119-133.

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

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

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

[87] 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.