INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team celtique

Software certification with semantic analysis

Rennes - Bretagne-Atlantique

Theme : Programs, Verification and Proofs

Activity Report

2010

Table of contents

1. Team

Research Scientists
Thomas Jensen [Team leader, DR, CNRS, HdR]
Frédéric Besson [Inria, CR]
Arnaud Gotlieb [Inria, CR]
David Pichardie [Inria, CR]

Faculty Members
Sandrine Blazy [University of Rennes 1, HdR]
Thomas Genet [University of Rennes 1, HdR]
David Cachera [Ens Cachan, HdR]

Technical Staff
Tiphaine Turpin [INRIA]
Nada Benduro [Software Engineer, until 07/04/2010]
Vincent Monfort [Software Engineer, from 15/10/2009]

PhD Students
Benoît Boyer [ATER University Rennes 1]
Florence Charreteur [ATER INSA Rennes]
Valérie Murat [MENRT grant, joint PhD with S4 team]
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]
André Oliveira Maronèze [MENRT grant, from 01/09/2010]

Post-Doctoral Fellows
Florent Kirchner [INRIA]
Matthieu Carlier [ANR U3CAT project]

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

CELTIQUE has developed a new type system system for Java to track dangerous uses of initialised objects. This contribution makes part of the Javasec project, commissioned by the national information security agency (ANSSI), in order to propose extensions to enhance the security of a Java virtual machine. The work provides a practical solution, develops a supporting type system, and formally proves its soundness in Coq (for a simplified formal subset of Java). The work was presented at this year's European Symposium on Research in Computer Security. Its take part of the phd work of Laurent Hubert that will defend this thesis at the end of the year.

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 [77].
- 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 [53], 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** [50], [82]. 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 [51]. 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) [41] is a scalable algorithm for constructing an initial call-graph of the program. Tip and Palsberg [89] 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 [80], [81] 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.,* [63], [62] have explored the algorithmic design space of contexts for benchmarks of significant size. Latter on, Milanova *et. al.,* [71] 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 [67] have extended the empiric evaluation of object-sensitivity using a BDD implementation allowing to cope with benchmarks otherwise out-of-scope. Besson and Jensen [45] proposed to use DATALOG in order to specify context-sensitive analyses. Whaley and Lam [90] 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 [68] and the race analysis of Naik, Aiken [73] and Whaley [74] both heavily rely on the precision of a control-flow analysis.

Control-flow analysis allows to statically prove the absence of certain run-time errors such as "message not understood" or cast exceptions. Yet it does not tackle the problem of "null pointers". Fahnrich and Leino [57] propose a type-system for checking that after object creation fields are non-null. Hubert, Jensen and Pichardie have formalised the type-system and derived a type-inference algorithm computing the most precise typing [66]. The proposed technique has been implemented in a tool called NIT [65]. Null pointer detection is also done by bug-detection tools such as FindBugs [65]. 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 [64], [40], linear logic approaches to implicit computational complexity [42], cost model for Java byte code [35] based on size relation inference, and WCET computation by abstract interpretation based loop bound interval analysis techniques [54].

We have proposed an original approach for designing static analyses computing program costs: inspired from a probabilistic approach [83], 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 [49]. An example of cache miss analysis has been developed within this framework [87].

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 [84], [56]. 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 [61].

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 [86], [44]. 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 [14]. **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) [75] is a technique to download mobile code on a host machine while ensuring that the code adheres to a specified security policy. The key idea is that the code producer sends the code along with a proof (in a suitably chosen logic) that the code is secure. Upon reception of the code and before executing it, the consumer submits the proof to a proof checker for the logic. Our project focus on two components of the PCC architecture: the proof checker and the proof generator.

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

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

In practice, the *proof checker* is a complex tool which relies on a complex Verification Condition Generator (VCG). VCGs for real programming languages and security policies are large and non-trivial programs. For example, the VCG of the Touchstone verifier represents several thousand lines of C code, and the authors observed that "there were errors in that code that escaped the thorough testing of the infrastructure" [76]. Many solutions have been proposed to reduce the size of the trusted computing base. In the *foundational proof carrying code* of Appel and Felty [38], [37], the code producer gives a direct proof that, in some "foundational" higher-order logic, the code respects a given security policy. Wildmoser and Nipkow [92], [91]. prove the soundness of a *weakest precondition* calculus for a reasonable subset of the Java bytecode. Necula and Schneck [76] 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 [85], 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.* [36] 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 [34] 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 [58], [59]. We defined a simple and efficient algorithm [55] 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 [43].

4. Software

4.1. Sawja: Static Analysis Workshop for Java Applications

Participants: 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.

4.2. Timbuk: a tree automata library

Participants: Thomas Genet, Benoît Boyer.

Timbuk [59] 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 [3].

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 5.7.

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) [79], [78]. Pierre-Cyrille Heam, Yohan Boichut and Olga Kouchnarenko of the Cassis Inria project use Timbuk as a verification back-end [46] for AVISPA [39]. 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.

4.3. Euclide: a constraint-based testing platform for critical C programs

Participant: Arnaud Gotlieb.

Euclide is an open source prototype tool that can help testing and verifying critical C programs. The prototype takes a C program as input, optionally annotated with assertions or post-conditions, and generates input test data that can reach specified locations within the code. Additionally, it can either prove that the assertions or post-conditions are verified, or proposes counter-examples to these properties. The core of the tool includes a powerful constraint solver based on constraint propagation, integer linear relaxations and labelling, that was built specifically for this purpose. Euclide is mainly developed in Prolog and is accessible online through a web interface . Euclide has been registered at the APP with number IDDN.FR.001.250011.000.S.P.2009.000.10600. This software is distributed under the CECILL-C licence. A. Gotlieb received the best poster award at the Annual National Days of the GDR-GPL for a presentation of the Euclide tool. In the context of the CAVERN project, we recently developed a constraint solver dedicated to modular integer computations [25]. This solver should be integrated soon within the Euclide platform.

5. New Results

5.1. Static Analysis of Object-Oriented Languages

Participants: Frédéric Besson, Delphine Demange, Laurent Hubert, Thomas Jensen, Vincent Monfort, David Pichardie.

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

5.1.1. Secure Object Initialization

The initialization of an information system is usually a critical phase where essential defense mechanisms are being installed and a coherent state is being set up. In object-oriented software, granting access to partially initialized objects is a delicate operation that should be avoided. We propose a modular type system to formally specify the initialization policy of libraries or programs and a type checker to statically check at load time that all loaded classes respect the policy. This allows to prove the absence of bugs which have allowed some famous privilege escalations in Java. Our experimental results show that our safe default policy allows to prove 91% of classes of `java.lang`, `java.security` and `javax.security` safe without any annotation and by adding 57 simple annotations we proved all classes but four safe. The type system and its soundness theorem have been formalized and machine checked using Coq [27].

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 [24], 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. Security of the Java Platform

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. We provide in [31] 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.

5.1.4. Access control model for interactive devices

We have designed [14] a security model for programming applications in which the access control to resources can employ user interaction to obtain the necessary permissions. Our work is inspired by and improves on the current Java security architecture used in Java-enabled mobile smart phones. We consider access control permissions with multiplicities in order to allow to use a permission a certain number of times and reduce the number of user interactions. To support our security model, a static analysis is enforcing, at load-time, that resources are accessed correctly.

5.1.5. Sawja: Static Analysis Workshop for Java Applications

We describe in [26] the Sawja library: a static analysis workshop fully compliant with Java 6 which provides OCaml modules for efficiently manipulating Java bytecode programs. We present the main features of the library, including i) efficient functional data-structures for representing a program with implicit sharing and lazy parsing, ii) an intermediate stack-less representation, and iii) fast computation and manipulation of complete programs. We provide experimental evaluations of the different features with respect to time, memory and precision.

5.2. Certified Static Analysis and Compilation

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

5.2.1. Certified Abstract Interpretation

Proving the correctness of an analyzer is based on semantic properties, and becomes difficult to ensure when complex analysis techniques are involved. In [20] 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.

5.2.2. Certified Polyhedral Analysis

In [17] we develop a certified checker in Coq that is able to certify the results of a polyhedral array-bound analysis for an imperative, stack-oriented bytecode language with procedures, arrays and global variables. The checker uses, in addition to the analysis result, certificates which at the same time improve efficiency and make correctness proofs much easier. In particular, our result certifier avoids complex polyhedral computations such as convex hulls and is using easily checkable inclusion certificates based on Farkas lemma. Benchmarks demonstrate that our approach is effective and produces certificates that can be efficiently checked not only by an extracted Caml checker but also directly in Coq.

5.2.3. Certified Generation of Linear Arithmetic Proofs

In [32], we show how to generate checkable certificate for linear arithmetic using an inexact inexact LP solver. Off-the-shelf linear programming (LP) solvers trade soundness for speed: for efficiency, the arithmetic is not exact rational arithmetic but floating-point arithmetic. As a side-effect the results come without any formal guarantee and cannot be directly used for deciding linear arithmetic. In this work we explain how to design a sound procedure for linear arithmetic built upon an inexact floating-point LP solver. Our approach relies on linear programming duality to instruct a black-box off-the-shelf LP solver to output, when the problem is not satisfiable, an untrusted proof certificate. We present a heuristic post- processing of the certificate which accommodates for certain numeric inaccuracies. Upon success it returns a provably correct proof witness that can be independently checked. Our preliminary results are promis- ing. For a benchmark suite extracted from SMT verification problems the floating-point LP solver returns a result for which proof witnesses are successfully and efficiently generated. The proof witnesses are used by our Certified Polyhedral Analysis.

5.2.4. *Certified compilation*

Iterated Register Coalescing (IRC) is a widely used heuristic for performing register allocation via graph coloring. In [18], we present a formal verification of the whole IRC algorithm, that can be used as a reference for IRC. The automatic extraction of our IRC algorithm yields a program with competitive performance. This work has been integrated into the CompCert verified compiler.

In 2010, Airbus evaluated the CompCert compiler and tested it on critical flight control software. A WCET (Worst-Case Execution Time) analysis was performed by Airbus to estimate the performance of the generated code. The results were very encouraging. A promising way to improve these results is to give extra information to the WCET analysis.

Since the recent beginning of André Oliveira Maronèze Ph.D. thesis's, and in cooperation with Isabelle Puaut (ALF project team), we are designing an annotation language dedicated to WCET properties of C programs that will be integrated in the CompCert compiler. We are also studying how to generate some of these properties from the CompCert compiler and how to compile them.

5.3. Quantitative Aspects of Static Analysis

Participants: David Cachera, Thomas Jensen, Arnaud Jobin.

We have developed our linear model of cost computations based on dioid theory, in order to show the deep connections between this model and the classical interpretation approach. The main difficulties come from the fact that abstraction has to take two distinct notions of order into account: the order on costs and the order on states [19]. A detailed paper collecting our results on this approach, including a new case study on power consumption estimation has been published in [15].

5.4. Automatic test data generation for Java Bytecode programs

Participants: Florence Charreteur, Arnaud Gotlieb.

In [22], we introduce a constraint-based reasoning approach to automatically generate test input for Java bytecode programs. Our goal-oriented method aims at building an input state of the Java Virtual Machine (JVM) that can drive program execution towards a given location within the bytecode. An innovative aspect of the method is the definition of a constraint model for each bytecode that allows backward exploration of the bytecode program, and permits to solve complex constraints over the memory shape (e.g., `p == p.next` enforces the creation of a cyclic data structure referenced by `p`). We implemented this constraint-based approach in a prototype tool called JAUT, that can generate input states for programs written in a subset of JVM including integers and references, dynamic-allocated structures, objects inheritance and polymorphism by virtual method call, conditional and backward jumps. Experimental results show that JAUT can generate test input for executing locations not reached by other state-of-the-art code-based test input generators such as jCUTE, JTEST and Pex.

5.5. Path infeasibility generalization in dynamic symbolic execution

Participants: Mickael Delahaye, Arnaud Gotlieb.

Recent code-based test input generators based on *dynamic symbolic execution* increase path coverage by solving path condition with a constraint or an SMT solver. When the solver considers path condition produced from an infeasible path, it tries to show unsatisfiability, which is a useless time-consuming process. In [23], we propose a new method that takes opportunity of the detection of a single infeasible path to generalize to a (possibly infinite) family of infeasible paths, which will not have to be considered in further path conditions solving. The method exploits non-intrusive constraint-based explanations, a technique developed in Constraint Programming to explain unsatisfiability. Experimental results obtained with our prototype tool IPEG show that, whatever is the underlying constraint solving procedure (IC, Colibri and the SMT solver Z3), this approach can save considerable computational time. This is a joint work with Bernard Botella from CEA.

5.6. Automatic testing of constraint programs

Participants: Nadjib Lazaar, Arnaud Gotlieb.

The success of several constraint-based modeling languages such as OPL, ZINC, or COMET, appeals for better software engineering practices, particularly in the testing phase. In [30], [29], we introduce a testing framework enabling automated test case generation for constraint programming. We propose a general framework of constraint program development which supposes that a first declarative and simple constraint model is available from the problem specifications analysis. Then, this model is refined using classical techniques such as constraint reformulation, surrogate and global constraint addition, or symmetry-breaking to form an improved constraint model that must be thoroughly tested before being used to address real-sized problems. We think that most of the faults are introduced in this refinement step and propose a process which takes the first declarative model as an oracle for detecting non-conformities. We derive practical test purposes from this process to generate automatically test data that exhibit non-conformities. We implemented this approach in a new tool called CPTEST that was used to automatically detect non-conformities on two classical benchmark programs, namely the Golomb rulers and the car-sequencing problem. This is a joint work with Yahia Lebbah from University of Oran.

5.7. Static analysis based on rewriting and tree automata

Participants: Thomas Genet, Benoît Boyer.

5.7.1. 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 [60] and more recently by José Meseguer, Miguel Palomino and Narciso Martí-Oliet [69]. With regards to this last work, the interest of our approach is that it imposes fewer restriction on the equations used to define approximations. Our only syntactical constraint is that equations have to be linear though [69] 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 [7], 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.

5.7.2. CounterVerification of Temporal Properties on Tree Automata

We extended this static analysis framework based on term rewriting systems and tree automata with Counterexample Example Guided Automatic Refinement (CEGAR [52]). The refinement of approximations on tree automata has already been investigated in [48], where semantics of programs is encoded using tree transducers. With Axel Legay (S4 team) and Yohan Boichut (LIFO), we defined a CEGAR approach of completion with automatic approximation refinement, where semantics is encoded using term rewriting systems [33]. We chose to stick to term rewriting systems because it permits a more straightforward encoding of program semantics than tree transducers. Furthermore, our completion based CEGAR avoids a lot of forward and backward computations that are necessary in [48]. This approach is currently being implemented in Timbuk 4.2.

6. Contracts and Grants with Industry

6.1. Contracts with Industry

6.1.1. 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.

7. Other Grants and Activities

7.1. Regional Initiatives

7.1.1. The CERTLOGS project

Participants: Thomas Genet, Thomas Jensen, David Pichardie, Vincent Monfort, Florent Kirchner.

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.

7.2. National Initiatives

7.2.1. ANR DECERT project

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

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).

7.2.2. The ANR SETIN RAVAJ

Participants: 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 [72] 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 [70] [88] [60]. On the other hand, we will automatically generate approximation equations from the property to be proved, using [46] [47], and also provide an automatic approximation refinement methodology adapted to the equational approximation framework.

7.2.3. The ANR SETIN PARSEC

Participants: 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).

7.2.4. The JAVASEC project

Participants: Thomas Jensen, David Pichardie.

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.

7.2.5. 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.

7.2.6. 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 INRIA Rennes - 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.

The CAVERN project will last until december 2011.

7.3. European Initiatives

7.3.1. 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.

8. Dissemination

8.1. Animation of the scientific community

Thomas Jensen is member of the executive bureau of the French network GDR GPL on software engineering and formal methods in programming. Arnaud Gotlieb is co-president of the MTV2 project of GDR-GPL.

Arnaud Gotlieb served in the program committees of several international conferences, including IEEE ICST'10, TAP'10 and QSIC'10. He also co-organized the CSTVA'10 workshop, a satellite event of ICST'10. For the organization of the future editions of CSTVA, he received a support from Microsoft Research under the banner of the Verified Software Initiative.

David Pichardie served as program chair of the BYTECODE workshop (ETAPS 2010 satellite event) and in the program committees of ITP 2010, VERIFY 2010 and IFM 2010. He also gave an invited talk in the workshop NSAD 2010 (SAS 2010 satellite workshop).

Thomas Jensen was on the program committee for FOPARA (Foundational and practical aspects of Resource Analysis) workshop and the TGC (Trustworthy Global Computing) conference.

8.2. PhD and habilitation theses defended

Florence Charreteur defended her PhD thesis, entitled “Modélisation par contraintes de programmes en bytecode Java pour la génération automatique de tests” on March 9 [12].

David Cachera defended his Habilitation thesis “Analyses statiques : certifier et quantifier” on August 30 [11].

Laurent Hubert defended his PhD thesis, entitled “Foundations and Implementation of a Tool Bench for Static Analysis of Java Bytecode Programs” on December 17 [13].

Benoît Boyer defended his PhD thesis, entitled “Réécriture d’automates certifiée pour la vérification de modèle” on December 13 [10].

8.3. Teaching

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

David Pichardie also taught algorithmics 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 Cachera teaches logics, computability, algorithmics and formal languages at École normale supérieure de Cachan, and semantics of programing languages at Master 1 level at University of Rennes.

Arnaud Gotlieb is responsible of two teaching master-level modules at Insa Rennes: “Compilation” at the 4th year level and “Validation, Verification and Test” at the 5 year level. He also taught software testing at the Ecole des Mines de Nantes at the 5 year level. He was invited to give a conference at the Master 2 Alma of University of Nantes.

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).

Thomas Genet gave a lecture on “Cryptographic protocols: principles, attacks and verification tools” at the summer school “École Jeune Chercheurs en Programmation” (Rennes, may 2010).

Sandrine Blazy taught 2 modules for M2 level (Software Vulnerabiblities, Software testing). She also taught formal methods for program proof at M1 level.

Sandrine Blazy gave a lecture on certified compilation at the 2nd Asian-Pacific Summer School (Beijing, China, August 2010).

8.4. PhD and Habilitation committees

Arnaud Gotlieb participated as a an examiner to the PhD committee of Nicolas Berger at University of Nantes and Sergio Segura from University of Seville.

Thomas Jensen participated as rapporteur in the Habilitation committe of Frederic Prost at the University of Grenoble, and as raporteur in the PhD thesis committee of Manuel Garnacho, also University of Grenoble.

Thomas Jensen was president of the PhD thesis committee of Brice Morin and the Habilitation committee of Benoit Baudry, both at the University of Rennes.

Sandrine Blazy was a member of the PhD committee of Benoît Robillard at CNAM Paris.

8.5. 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 a member of the IRISA council as well as a member of the ISTIC council of Rennes 1 University. Sandrine Blazy participated to a recruitment committee of Rennes 1 University.

David Cachera is a member of INRIA Rennes council.

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.

David Pichardie is co-responsible of the Component Based Embedded Software Track of the second year of the research Master in Computer Science at Université de Rennes 1.

Arnaud Gotlieb participated to the recruitment committees of Insa Rennes and IUT Orsay.

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 "Special Issue on Applied Semantics of 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, Special Issue: TAIC-PART 2007 and MUTATION 2007.

[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] 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.

Publications of the year

Doctoral Dissertations and Habilitation Theses

[10] B. BOYER. *Réécriture d'automates certifiée pour la vérification de modèle*, Université Européenne de Bretagne, Déc 2010.

[11] D. CACHERA. *Analyses statiques : certifier et quantifier*, École normale supérieure de Cachan, 2010, Habilitation à Diriger des Recherches.

[12] F. CHARRETEUR. *Modélisation par contraintes de programmes en bytecode Java pour la génération automatique de tests*, Université Européenne de Bretagne, Mar 2010, http://hal.inria.fr/tel-00497785.

[13] L. HUBERT. *Foundations and Implementation of a Tool Bench for Static Analysis of Java Bytecode Programs*, Université de Rennes 1, December 2010.

Articles in International Peer-Reviewed Journal

[14] 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.

[15] 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.

[16] 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.

International Peer-Reviewed Conference/Proceedings

[17] F. BESSON, T. JENSEN, D. PICHARDIE, T. TURPIN. *Certified Result Checking for Polyhedral Analysis of Bytecode Programs*, in "Proc. of the 5th International Symposium on Trustworthy Global Computing (TGC 2010)", Lecture Notes in Computer Science, Springer-Verlag, 2010, To appear, http://hal.inria.fr/inria-00537816.

[18] S. BLAZY, B. ROBILLARD, A. W. APPEL. *Formal Verification of Coalescing Graph-Coloring Register Allocation*, in "19th European Symposium on Programming (ESOP)", Chypre Paphos, Springer, Mar 2010, vol. 6012, p. 145-164, http://hal.inria.fr/inria-00477689.

[19] D. CACHERA, A. JOBIN. *Injecting Abstract Interpretations into Linear Cost Models*, in "8th Workshop on Quantitative Aspects of Programming Languages (QAPL)", Paphos, EPTCS, 2010.

[20] D. CACHERA, D. PICHARDIE. *A Certified Denotational Abstract Interpreter*, in "Proc. of International Conference on Interactive Theorem Proving (ITP-10)", Lecture Notes in Computer Science, Springer-Verlag, 2010, vol. 6172, p. 9-24, http://hal.inria.fr/inria-00537810.

[21] M. CARLIER, C. DUBOIS, A. GOTLIEB. *Constraint Reasonning in FOCALTEST*, in "5rd International Conference on Software and Data Technologies (ICSOFT'10)", Athens, Greece, Jul. 2010.

[22] F. CHARRETEUR, A. GOTLIEB. *Constraint-Based Test Input Generation for Java Bytecode*, in "Proc. of the 21st IEEE Int. Symp. on Softw. Reliability Engineering (ISSRE'10)", San Jose, CA, USA, Nov. 2010.

[23] M. DELAHAYE, B. BOTELLA, A. GOTLIEB. *Explanation-based generalization of infeasible path*, in "3rd IEEE International Conference on Software Testing, Validation and Verification (ICST'10)", Paris, France, Apr. 2010.

[24] D. DEMANGE, T. JENSEN, D. PICHARDIE. *A Provably Correct Stackless Intermediate Representation for Java Bytecode*, in "8th Asian Symposium on Programming Languages and Systems (APLAS)", Lecture Notes in Computer Science, Springer-Verlag, 2010, vol. 6461, http://hal.inria.fr/inria-00537815.

[25] A. GOTLIEB, M. LECONTE, B. MARRE. *Constraint Solving on Modular Integers*, in "Proc. of the 9th Int. Workshop on Constraint Modelling and Reformulation (ModRef'10), co-located with CP'2010", St Andrews, Scotland, Sept. 2010.

[26] L. HUBERT, N. BARRÉ, F. BESSON, D. DEMANGE, T. JENSEN, V. MONFORT, D. PICHARDIE, T. TURPIN. *Sawja: Static Analysis Workshop for Java*, in "1st International Conference on Formal Verification of Object-Oriented Software (FoVeOOS)", Lecture Notes in Computer Science, Springer-Verlag, 2010, http://hal.inria.fr/inria-00504047.

[27] 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.

[28] H. KIRCHNER, K. FLORENT, C. KIRCHNER. *Constraint Based Strategies*, in "18th International Workshop on Functional and Constraint Logic Programming - WFLP 2009", Brésil Brasilia, Springer Berlin / Heidelberg, 2010, vol. 5979, p. 13-26, http://hal.inria.fr/inria-00494531.

[29] N. LAZAAR, A. GOTLIEB, Y. LEBBAH. *Fault Localization in Constraint Programs*, in "22th Int. Conf. on Tools with Artificial Intelligence (ICTAI'2010)", Arras, France, Oct. 2010.

[30] N. LAZAAR, A. GOTLIEB, Y. LEBBAH. *On Testing Constraint Programs*, in "16th Int. Conf. on Principles and Practices of Constraint Programming (CP'2010)", St Andrews, Scotland, Sept. 2010.

National Peer-Reviewed Conference/Proceedings

[31] G. HIET, F. GUIHÉRY, G. GUIHEUX, D. PICHARDIE, C. BRUNETTE. *Sécurité de la plate-forme d'exécution Java : limites et propositions d'améliorations*, in "Proc. of Symposium sur la sécurité des technologies de l'information et des communications (SSTIC 2010)", 2010, http://hal.inria.fr/inria-00537820.

Workshops without Proceedings

[32] F. BESSON. *On using an inexact floating-point LP solver for deciding linear arithmetic in an SMT solver*, in "8th International Workshop on Satisfiability Modulo Theories", 2010, http://hal.inria.fr/inria-00517308.

Research Reports

[33] Y. BOICHUT, B. BOYER, T. GENET, A. LEGAY. *Fast Equational Abstraction Refinement for Regular Tree Model Checking*, INRIA, Jul 2010, http://hal.inria.fr/inria-00501487.

References in notes

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

[35] 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.

[36] 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.

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

[38] 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.

[39] 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, Springer, 2005, vol. 3576, p. 281-285.

[40] 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.

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

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

[43] 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.

[44] 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.

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

[46] 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.

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

[48] A. BOUAJJANI, P. HABERMEHL, A. ROGALEWICZ, T. VOJNAR. *Abstract Regular Tree Model Checking*, in "ENTCS", 2006, vol. 149, n^{o} 1, p. 37-48.

[49] 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.

[50] 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.

[51] 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.

[52] E. M. CLARKE. *Counterexample-Guided Abstraction Refinement*, in "TIME", IEEE Computer Society, 2003, 7.

[53] 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.

[54] 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.

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

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

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

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

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

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

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

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

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

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

[65] 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.

[66] 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.

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

[68] 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.

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

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

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

[72] 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, Springer, May 2003, vol. 2622, p. 61-76, http://www.loria.fr/~moreau/Papers/MoreauRV-CC2003.ps.gz.

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

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

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

[76] 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.

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

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

[79] 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.

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

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

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

[83] 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.

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

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

[86] 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.

[87] 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.

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

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

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

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

[92] 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.