IN PARTNERSHIP WITH:
CNRS

Université Rennes 1

Ecole normale supérieure de Cachan

Activity Report 2011

Project-Team CELTIQUE

Software certification with semantic analysis

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

RESEARCH CENTER
Rennes - Bretagne-Atlantique

THEME
Programs, Verification and Proofs

Table of contents

1. Members **1**
2. Overall Objectives **1**
2.1. Project overview 1
2.2. Highlights of the Year 2
3. Scientific Foundations **2**
3.1. Static program analysis 2
3.1.1. Static analysis of Java 3
3.1.2. Quantitative aspects of static analysis 3
3.1.3. Semantic analysis for test case generation 4
3.2. Software certification 4
3.2.1. Process-oriented software certification 5
3.2.2. Semantic software certificates 5
3.2.3. Certified static analysis 6
4. Software **7**
4.1. Javalib 7
4.2. SAWJA 7
4.3. Timbuk 7
5. New Results **7**
5.1. Control-Flow Analysis by Abstract Interpretation 7
5.2. Modular SMT Proofs for Fast Reflexive Checking inside Coq 8
5.3. Secure the Clones: Static Enforcement of Policies for Secure Object Copying 8
5.4. Fault localization and correction in Constraint Programs 9
5.5. Floating-point constraint solving 9
5.6. Fast inference of polynomial invariants 9
6. Contracts and Grants with Industry **9**
6.1. ANR DECERT project 9
6.2. ANR CAVERN project 10
6.3. ANR PiCoq project 10
6.4. ANR U3CAT project 10
6.5. The FRAE ASCERT project 10
7. Partnerships and Cooperations **11**
7.1. Regional Initiatives 11
7.2. European Initiatives 11
7.2.1. The COST Action IC0701 11
7.2.2. The Valves consortium 11
7.3. International Initiatives 11
8. Dissemination **12**
8.1. Animation of the scientific community 12
8.2. Teaching 12
9. Bibliography **13**

Project-Team CELTIQUE

Keywords: Programming Languages, Object Oriented Programming, Formal Methods, Proofs Of Programs, Security, Interactive Theorem Proving

Beginning of the Team: 01/07/2009.

1. Members

Research Scientists

Thomas Jensen [Team leader, DR, INRIA, HdR]
Frédéric Besson [Inria, CR]
Arnaud Gotlieb [Inria, CR, HdR]
David Pichardie [Inria, CR]
Alan Schmitt [Inria, CR, since September 2011, HdR]

Faculty Members

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

Engineers

Matthieu Carlier [post-doc]
Florent Kirchner [post-doc, until 07/04/2011]
Vincent Monfort [Software Engineer]
Pierre Vittet [Software Engineer, from 15/11/2011]
Ronan Saillard [Software Engineer, from 1/12/2011]

PhD Students

Valérie Murat [MENRT grant, joint PhD with Distribcom team]
Yann Salmon [MENRT grant]
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 Maroneze [MENRT grant, from 01/09/2010]
Stéphanie Riaud [INRIA grant, from 01/09/2011]

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

Sandrine Blazy received the 2011 La Recherche award in Information Sciences for her contributions to the CompCert verified C compiler, together with Zaynah Dargaye, Xavier Leroy and Jean-Baptiste Tristan.

3. Scientific Foundations

3.1. Static program analysis

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

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

- Data-flow analysis as it is used in optimising compilers for imperative languages. The properties can either be approximations of the values of an expression ("the value of variable x is greater than 0" or x is equal to y at this point in the program") or more intensional information about program behaviour such as "this variable is not used before being re-defined" in the classical "dead-variable" analysis [60].
- 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 [40], 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** [38], [63]. 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 [39]. 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) [31] is a scalable algorithm for constructing an initial call-graph of the program. Tip and Palsberg [69] 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 [61], [62] 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.,* [49], [48] have explored the algorithmic design space of contexts for benchmarks of significant size. Latter on, Milanova *et. al.,* [55] 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 [53] have extended the empiric evaluation of object-sensitivity using a BDD implementation allowing to cope with benchmarks otherwise out-of-scope. Besson and Jensen [35] proposed to use DATALOG in order to specify context-sensitive analyses. Whaley and Lam [70] 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 [54] and the race analysis of Naik, Aiken [56] and Whaley [57] 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 [44] 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 [52]. The proposed technique has been implemented in a tool called NIT [51]. Null pointer detection is also done by bug-detection tools such as FindBugs [51]. The main difference is that the approach of findbugs is neither sound nor complete but effective in practice.

3.1.2. Quantitative aspects of static analysis

Static analyses yield qualitative results, in the sense that they compute a safe over-approximation of the concrete semantics of a program, w.r.t. an order provided by the abstract domain structure. Quantitative aspects

of static analysis are two-sided: on one hand, one may want to express and verify (compute) quantitative properties of programs that are not captured by usual semantics, such as time, memory, or energy consumption; on the other hand, there is a deep interest in quantifying the precision of an analysis, in order to tune the balance between complexity of the analysis and accuracy of its result.

The term of quantitative analysis is often related to probabilistic models for abstract computation devices such as timed automata or process algebras. In the field of programming languages which is more specifically addressed by the Celtique project, several approaches have been proposed for quantifying resource usage: a non-exhaustive list includes memory usage analysis based on specific type systems [50], [30], linear logic approaches to implicit computational complexity [32], cost model for Java byte code [26] based on size relation inference, and WCET computation by abstract interpretation based loop bound interval analysis techniques [41].

We have proposed an original approach for designing static analyses computing program costs: inspired from a probabilistic approach [64], 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 [37]. An example of cache miss analysis has been developed within this framework [68].

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 [65], [43]. 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 [47].

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 [67], [34]. Another kind of problems concern the way that computational resources are being accessed and used, in order to ensure that a given access policy is being implemented correctly and that a given application does not consume more resources that it has been allocated. Members of the Celtique team have proposed a verification technique that can check the proper use of resources of Java applications running on mobile telephones [36]. **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) [58] 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" [59]. Many solutions have been proposed to reduce the size of the trusted computing base. In the *foundational*

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

proof carrying code of Appel and Felty [29], [28], the code producer gives a direct proof that, in some "foundational" higher-order logic, the code respects a given security policy. Wildmoser and Nipkow [72], [71]. prove the soundness of a *weakest precondition* calculus for a reasonable subset of the Java bytecode. Necula and Schneck [59] 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 [66], 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.* [27] 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 [25] 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 [45], [46]. We defined a simple and efficient algorithm [42] 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 [33].

4. Software

4.1. Javalib

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

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

- Version: 2.2
- Programming language: Ocaml

4.2. SAWJA

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

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

Moreover, Sawja provides some stackless intermediate representations of code, called JBir and A3Bir. The transformation algorithm, common to these representations, has been formalized and proved to be semantics-preserving.
See also the web page http://sawja.inria.fr/.

- Version: 1.2
- Programming language: Ocaml

4.3. Timbuk

Participant: Thomas Genet [correspondant].

Timbuk is a library of OCAML functions for manipulating tree automata. More precisely, Timbuk deals with finite bottom-up tree automata (deterministic or not). This library provides the classical operations over tree automata (intersection, union, complement, emptiness decision) as well as exact or approximated sets of terms reachable by a given term rewriting system. This last operation can be certified using a checker extracted from a Coq specification.

- Version: 3.1
- Programming language: Ocaml

5. New Results

5.1. Control-Flow Analysis by Abstract Interpretation

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

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

This work was presented at the Japanese Shonan workshop on Verification of higher-order functional programs in September 2011. A journal article is accepted to appear in Information and Computation.

5.2. Modular SMT Proofs for Fast Reflexive Checking inside Coq

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

Satisfiability Modulo Theory (SMT) solvers are efficient automatic provers for combination of theories. Those solvers have proved very successful in program verification because they discharge automatically and efficiently challenging verification conditions. SMT solvers are therefore *d*e facto part of the Trusted Computing Base of many program verification methodologies. A consequence is that a soundness bug in a SMT solver can make the whole program verification process unsound.

To tackle this problem, we propose a new methodology for exchanging unsatisfiability proofs between an untrusted SMT solver and a sceptical proof assistant with computation capabilities like Coq. We advocate modular SMT proofs that separate boolean reasoning and theory reasoning; and structure the communication between theories using Nelson-Oppen combination scheme.

We present the design and implementation of a Coq reflexive verifier that is modular and allows for fine-tuned theory-specific verifiers. The current verifier is able to verify proofs for quantifier-free formulae mixing linear arithmetic and uninterpreted functions. Our proof generation scheme benefits from the efficiency of state-of-the-art SMT solvers while being independent from a specific SMT solver proof format. Our only requirement for the SMT solver is the ability to extract unsat cores and generate boolean models. In practice, unsat cores are relatively small and their proof is obtained with a modest overhead by our proof-producing prover. We present experiments assessing the feasibility of the approach for benchmarks obtained from the SMT competition.

This work has been presented at the CPP conference [15] and the international PxTP workshop [21], [20].

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

Participants: Thomas Jensen, Florent Kirchner, David Pichardie.

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

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

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

The mechanism has been implemented and experimentally evaluated on clone methods from several Java libraries. The work as been presented at ESOP [18] this year and is under reviewing for a journal special issue.

5.4. Fault localization and correction in Constraint Programs

Participants: Nadjib Lazaar, Arnaud Gotlieb.

Nowadays, constraint programs are written in high-level modelling languages. Their verification is currently based on trace analysis techniques but does not integrate systematic testing techniques. In this work, we developped a Testing framework for catching the peculiarities of constraint program development, throughout the notions of conformity relations, fault localization and correction.

Within the context of the Nadjib Lazaar's PhD (defense on 5 Dec. 2011), we explored in 2011 the testing of constraint programs written in OPL and the development of trace-based fault localization and correction techniques [19]. Lazaar's tool called CPTEST showed impressive experimental results on four hard problems of the CP Community, leading to a publication (in progress) in the Constraints Journal.

5.5. Floating-point constraint solving

Participants: Matthieu Carlier, Arnaud Gotlieb.

Programs including floating-point computations are known to be hard-to-test. Generating test inputs for those programs requires solving constraints over floating-points computations, which led us to the development of specific constraint filtering techniques. In this work, we extended the Marre and Michel property regarding the use of internal floating-point representation to increase the filtering capabilities of addition to the case of multiplication/division. We came up with an optimized implementation of FPSE (our current FP constraint solver) that was able to deal with large C programs that include (non-linear) floating-point computations. We already got a first publication of this work [17].

5.6. Fast inference of polynomial invariants

Participants: David Cachera, Thomas Jensen, Arnaud Jobin, Florent Kirchner.

The problem of automatically inferring polynomial (non-linear) invariants of programs is still a major challenge in program verification. We have proposed an abstract interpretation based method to compute polynomial invariants for imperative programs. Our analysis is a backward propagation approach that computes preconditions for equalities like $g = 0$ to hold at the end of execution. Properties are expressed using ideals, a structure that satisfies the descending chain condition, enabling fixpoints computations to terminate without use of a widening operator. In the general case, termination would be characterized using ideal membership tests and Gröbner bases computations. In order to optimize computational complexity, we propose a specialized analysis dealing with inductive invariants which ensures fast termination of fixpoints computations. The optimized procedure has been shown by experiments to work well in practice, and to be two orders of magnitude faster than state of the art analyzers [23].

6. Contracts and Grants with Industry

6.1. 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 Systerel, CEA List and INRIA teams Mosel, Cassis, Marelle, Proval and Celtique (coordinator).

6.2. ANR CAVERN project

Participants: Arnaud Gotlieb, Matthieu Carlier.

The CAVERN project (ANR, 2007-2011) gathers national research teams to study the capabilities of Constraint Programming for Program Verification (http://cavern.inria.fr/). This year, we focussed on WP4 on floating-point computations and got new results with the CeP team of University of Nice Sophia-Antipolis. The overall results of the project will be presented at the annual 2012 ANR meeting in Lyon.

6.3. ANR PiCoq project

Participant: Alan Schmitt.

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

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

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

6.4. 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.

6.5. 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. Partnerships and Cooperations

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. European Initiatives

7.2.1. 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.

7.2.2. The Valves consortium

This year, we built the VALVES (Variability Testing of Highly-Variable Systems) European proposal gathering University of Sevilla, University of Namur, University of Uppsala, Isotrol, Thales and INRIA Rennes (Arnaud Gotlieb being the coordinator of the proposal). The proposal was submitted to the FP7 program (Call 7, challenge 3.3) and got well evaluated but not enough to be funded this year. From this, we got a support of the Brittany Region to organize a physical meeting during Fall 2011 and prepare a new submission. This meeting was held in the Paris INRIA's offices, the 18th November 2011.

7.3. International Initiatives

7.3.1. INRIA International Partners

Since three years, we have developed a long-term collaboration with Yahia Lebbah, from University of Oran, Algeria. This collaboration has been fruitful with several publications, the last one being [19] and the INRIA International programme support DGRI. This fund permitted us to visit each other's group in 2011 with the 1-month visit of N. Lazaar to the University of Oran and the 1-week visit of Y. Lebbah to INRIA Rennes in Dec. 2011.

8. Dissemination

8.1. Animation of the scientific community

David Pichardie served in the program committees of JFLA 2011, BYTECODE 2011, PxTP 2011, PSATTT 2011 and FoVeOOS 2011. Arnaud Gotlieb served in the PC of the International conferences QSIC'11 and TAP'11, and the VAST'11 workshop. He co-organized the CSTVA'11 satellite workshop of the ICST'11 conference that was held in Berlin in March. He will be hold the workshop chair of ICST'11 next year. A. Schmitt is a member of the steering committee of the Journées Françaises des Langages Applicatifs (JFLA). David Cachera served on the program committee of FOPARA 2011. Sandrine Blazy served on the program committee of the ITP 2011. Thomas Jensen served on the program committee of FoVeOOS 2011 and on the FPS 2011 conference.

Sandrine Blazy and Thomas Jensen organize a seminar devoted to security and formal methods. The seminar is funded by DGA-MI. It takes place at INRIA twice a month. It is open to the public and attended by researchers and engineers.

8.2. Teaching

Licence :

Thomas Genet, Programmation fonctionnelle, 44h, L3, Rennes 1, France

David Cachera, Logique et calculabilité, 36h, L3, ENS Cachan Bretagne, France

David Cachera, Algorithmique avancée, 18h, L3, ENS Cachan Bretagne, France

David Cachera, Langages formels, 24h, L3, ENS Cachan Bretagne, France

Sandrine Blazy, Programmation fonctionnelle, 20h, L3, Rennes 1, France

Master :

Alan Schmitt, Méthodes Formelles pour le développement de logiciels sûrs, 36h, M1, Rennes 1, France

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

Thomas Genet, Bases de la cryptographie, 18h, M2, Rennes 1, France

Thomas Genet, Analyse et conception objet, 48h, M1, Rennes 1, France

Thomas Genet, Validation et vérification formelle, 38h, M1, Rennes 1, France

David Cachera, Sémantique des langages de programmation, 36h, M1, Rennes 1, France

David Cachera, Préparation à l'agrégation, 60h, M2, ENS Cachan Bretagne, France

Sandrine Blazy, Méthodes Formelles pour le développement de logiciels sûrs, 48h, M1, Rennes 1, France

Sandrine Blazy, Conception de logiciels surs, 40h, M2, Rennes1, France.

Sandrine Blazy, Évaluation des vulnérabilités des logiciels, 21h, M2, Rennes 1, France.

Sandrine Blazy, Veille technologique, 19h, M2, Rennes 1, France

Thomas Jensen, Program Analysis and Semantics, 20h, M2, Rennes 1, France

Thomas Jensen, Software Security, 20h, M2, Rennes 1, France.

PhD & HdR :

HdR : Arnaud Gotlieb, Contributions to Constraint-Based Testing, Université de Rennes, 12 Décembre 2011

PhD : Mickael Delahaye, Généralisation de chemins infaisables pour l'exécution symbolique dynamique, Université de Rennes, 26 Octobre 2011, Arnaud Gotlieb et Thomas Jensen

PhD : Nadjib Lazaar, Méthodologie et outil de test, de localisation de fautes et de correction automatique des programmes contraintes, Université de Rennes, 5 Décembre 2011, Arnaud Gotlieb and Thomas Jensen

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

PhD in progress: Yann Salmon, Optimized rewriting proof search using approximations and tree automata, octobre 2011, Thomas Genet

PhD in progress: Arnaud Jobin, Dioïdes et idéaux de polynômes en analyse statique, ENS Cachan, septembre 2008, soutenance prévue en janvier 2012, David Cachera and Thomas Jensen

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

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

PhD in progress: Pierre-Emmanuel Cornilleau, PCC certificates for static analysis, octobre 2009, Thomas Jensen and Frédéric Besson

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

PhD in progress: Delphine Demange, Certified Intermediate Representations, octobre 2009, Thomas Jensen and David Pichardie.

9. Bibliography

Major publications by the team in recent years

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

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

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

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

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

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

[7] T. GENET, V. RUSU. *Equational Approximations for Tree Automata Completion*, in "Journal of Symbolic Computation", 2010, vol. 45(5):574-597, May 2010, n^{o} 5, p. 574-597, http://hal.inria.fr/inria-00495405.

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

[9] L. HUBERT, T. JENSEN, V. MONFORT, D. PICHARDIE. *Enforcing Secure Object Initialization in Java*, in "15th European Symposium on Research in Computer Security (ESORICS)", Lecture Notes in Computer Science, Springer, 2010, vol. 6345, p. 101-115, http://hal.inria.fr/inria-00503953.

Publications of the year

Doctoral Dissertations and Habilitation Theses

[10] M. DELAHAYE. *Généralisation de chemins infaisables pour l'exécution symbolique dynamique*, University of Rennes 1, 2011.

[11] A. GOTLIEB. *Contributions to Constraint-Based Testing*, University of Rennes 1, 2011, Habilitation à Diriger des Recherches.

[12] N. LAZAAR. *Méthodologie et outil de test, de localisation de fautes et de correction automatique des programmes contraintes*, University of Rennes 1, 2011.

Articles in International Peer-Reviewed Journal

[13] J. MIDTGAARD, T. JENSEN. *Control-Flow Analysis by Abstract Interpretation*, in "Information and Computation", 2012.

International Conferences with Proceedings

[14] R. BEDIN FRANCA, S. BLAZY, D. FAVRE-FELIX, X. LEROY, M. PANTEL, J. SOUYRIS. *Formally verified optimizing compilation in ACG-based flight control software*, in "ERTS2 congress", 2012, to appear.

[15] F. BESSON, P.-E. CORNILLEAU, D. PICHARDIE. *Modular SMT Proofs for Fast Reflexive Checking inside Coq*, in "First International Conference on Certified Programs and Proofs", Kenting, Taïwan, Province De Chine, Lecture Notes in Computer Science, Springer-Verlag, 2011, vol. 7086, p. 151-166, http://hal.inria.fr/hal-00646960/en.

[16] Y. BOICHUT, T.-B.-H. DAO, V. MURAT. *Characterizing Conclusive Approximations by Logical Formulae*, in "Reachability Problems 2011", Gênes, Italy, G. DELZANNO, I. POTAPOV (editors), 2011, vol. LNCS 6945, http://hal.inria.fr/inria-00606100/en.

[17] M. CARLIER, A. GOTLIEB. *Filtering by ULP Maximum*, in "Proc. of the IEEE Int. Conf. on Tools for Artificial Intelligence (ICTAI'11)", Nov. 2011, Short paper, 4 pages.

[18] T. JENSEN, F. KIRCHNER, D. PICHARDIE. *Secure the Clones: Static Enforcement of Policies for Secure Object Copying*, in "Proc. of 20th European Symposium on Programming (ESOP 2011)", Lecture Notes in Computer Science, Springer-Verlag, 2011, vol. 6602, p. 317-337.

[19] N. LAZAAR, A. GOTLIEB, Y. LEBBAH. *A framework for the automatic correction of Constraint Programs*, in "4th IEEE International Conference on Software Testing, Validation and Verification (ICST'11)", Berlin, Germany, Mar. 2011.

Conferences without Proceedings

[20] F. BESSON, P.-E. CORNILLEAU, D. PICHARDIE. *A Nelson-Oppen based Proof System using Theory Specific Proof Systems*, in "Workshop on Proof eXchange for Theorem Proving (PxTP)", 2011.

[21] F. BESSON, P. FONTAINE, L. THÉRY. *A Flexible Proof Format for SMT: a Proposal*, in "Workshop on Proof eXchange for Theorem Proving (PxTP)", 2011.

Research Reports

[22] G. BARTHE, D. DEMANGE, D. PICHARDIE. *A formally verified SSA-based middle-end compiler*, INRIA, October 2011, http://hal.inria.fr/inria-00634702/en.

[23] D. CACHERA, T. JENSEN, A. JOBIN, F. KIRCHNER. *Fast inference of polynomial invariants for imperative programs*, INRIA, May 2011, n^{o} RR-7627, http://hal.inria.fr/inria-00595783/en.

[24] N. LAZAAR, N. ARIBI, A. GOTLIEB, Y. LEBBAH. *Negation for Free!*, INRIA, October 2011, n^{o} RR-7749, http://hal.inria.fr/inria-00629657/en.

References in notes

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

[26] 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.

[27] 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.

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

[29] 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.

[30] 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.

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

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

[33] 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.

[34] 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.

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

[36] 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° 6, p. 971-998, http://hal.inria.fr/inria-00537821.

[37] 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.

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

[39] 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.

[40] 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.

[41] 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.

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

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

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

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

[46] 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.

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

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

[49] 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.

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

[51] 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.

[52] 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.

[53] 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.

[54] 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.

[55] 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.

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

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

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

[59] 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.

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

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

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

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

[64] 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.

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

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

[67] 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.

[68] 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.

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

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

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

[72] 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.