INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team Lande

Logiciel : ANalyse et DEveloppement

Rennes - Bretagne-Atlantique

THEME SYM

Activity Report

2008

Table of contents

1. Team **1**
2. Overall Objectives **1**
3. Scientific Foundations **2**
3.1. Static program analysis 2
3.1.1. Static program analysis 2
3.1.2. The structure of a static analysis 3
3.1.3. Certified static analysis 3
3.2. Program testing 3
3.3. Reachability analysis over term rewriting systems 5
4. Software **5**
4.1. A Coq library of lattices 5
4.2. NIT: Null-ability Inference Tool 6
4.3. JavaLib: parsing Java class files into OCaml 6
4.4. Micromega: a reflexive Coq tactic for arithmetic 6
4.5. Timbuk: a tree automata library 6
4.6. SPAN: Security Protocol ANimator for AVISPA 7
4.7. Constraint-Based Testing of critical C programs 8
5. New Results **8**
5.1. Access control model for interactive devices 8
5.2. Non-Null annotation inference 8
5.3. A Calculational Approach to Control-Flow Analysis by Abstract Interpretation 9
5.4. Static analysis and Proof Carrying Code 9
5.5. Tree Automata Completion Checker 10
5.6. Improving formal specifications of cryptographic protocols 10
5.7. Quantitative static analysis with linear operators over dioids 11
5.8. Constraint reasoning in Path-oriented Random Testing 12
6. Contracts and Grants with Industry **12**
6.1. The RNTL CAT project 12
6.2. The MEFORSE collaborative research contract with France Télécom R&D 12
6.3. European FET-Integrated project MOBIUS 13
6.4. The ANR SETIN RAVAJ 13
6.5. The ANR SETIN PARSEC 14
6.6. ANR SESUR 2007 CAVERN 14
6.7. The COST Action IC0701 14
7. Dissemination **15**
7.1. Conferences: program committees, organization, invitations 15
7.2. PhD theses defended 15
7.3. PhD and Habilitation committees 15
7.4. Teaching: university courses and summer schools 15
7.5. Administrative responsibilities 16
8. Bibliography **16**

1. Team

Research Scientist
Thomas Jensen [CNRS, DR, HdR]
Frédéric Besson [Inria, CR]
Arnaud Gotlieb [Inria, CR]
David Pichardie [Inria, CR]
Olivier Heen

Faculty Member
Thomas Genet [Delegation from sept. 2007 to dec. 2009]
David Cachera [On leave from ENS Cachan, at CNRS until 2008/08/31]

Technical Staff
Nicolas Barré [Ingénieur associé]

PhD Student
Matthieu Petit [Région Bretagne grant]
Tristan Denmat [MENRT grant]
Tiphaine Turpin [MENRT grant]
Pascal Sotin [BDI CNRS-DGA]
Florence Charreteur [MENRT grant]
Benoît Boyer [University Rennes 1 grant]
Laurent Hubert [BDI CNRS-Région Bretagne]
Mickael Delahaye [CEA grant]
Nadjib Lazaar [MENRT grant]
Arnaud Jobin [MENRT grant]

Post-Doctoral Fellow
Jan Midtgaard [INRIA]
Frédéric Dabrowski [INRIA]
Pierre Rousseau [INRIA]

Administrative Assistant
Lydie Mabil [Inria, TR]

2. Overall Objectives

2.1. Project overview

The Lande project is concerned with formal methods for constructing and validating software. Our focus is on providing methods with a solid formal basis (in the form of a precise semantics for the programming language used and a formal logic for specifying properties of programs) in order to provide firm guarantees as to their correctness. In addition, it is important that the provided methods are highly automated so as to be usable by non-experts in formal methods.

The project's foundational activities are concerned with the semantics-based analysis of the behaviour of a given program. These activities draw on techniques from static and dynamic program analysis, testing and automated theorem proving. In terms of **static program analysis**, our foundational studies concern the specification of analyses by inference systems, the classification of analyses with respect to precision using abstract interpretation and reachability analysis for software specified as a term rewriting system. Particular analyses such as pointer analysis for C and control flow analysis for Java and Java Card have been developed. For the implementation of these and other analyses, we are improving and analysing existing iterative techniques based on constraint-solving and rewriting of tree automata. Concerning the **testing** of software, we have in particular investigated how flow analysis of programs based on constraint solving can help in the process of generating test cases from programs and from specifications. More speculatively, a long-term goal is to integrate the techniques of proving and testing into a common framework for approximating program behaviour. **Proof assistants** are used in the project to increase confidence in the verification analyses that are being developed.

An important application domain for these techniques is that of **software security**. Our activity in the area of **programming language security** has lead to the definition of a framework for defining and verifying security properties based on a combination of static program analysis and model checking. This framework has been applied to software for the Java 2 security architecture, to multi-application Java Card smart cards, to Java applets for mobile telephones and to cryptographic protocols. This has lead to methods for examining the access control and usage of resources on Java-enabled devices and to a tool for analyzing and simulating cryptographic protocols.

Lande is a joint project with the CNRS, the University of Rennes 1 and Insa Rennes.

3. Scientific Foundations

3.1. Static program analysis

Keywords: *abstract interpretation*, *optimising compilers*, *semantics*, *static program analysis*.

Abstract interpretation Abstract interpretation is a framework for relating different semantic interpretations of a program. Its most prominent use is in the correctness proofs of static program analyses, when these are defined as a non-standard semantic interpretation of a language in a domain of abstract program properties

Fixpoint iteration The result of a static analysis is often given implicitly as the solution of a system of equations $\{\overline{x} = f_i(\overline{x})\}_{i=1}^{n}$ where the f_i are monotone functions over a partial order. The Knaster-Tarski Fixpoint Theorem suggests an iterative algorithm for computing a solution as the limit of the ascending chain $f^n(\perp)$ where $\perp$ is the least element of the partial order.

3.1.1. Static program analysis

[39], [49] cover a variety of methods for obtaining information about the run-time behaviour of a program without actually running it. It is this latter restriction that distinguishes static analysis from its dynamic counterparts (such as debugging or profiling) which are concerned with monitoring the execution of the program. It is common to impose a further requirement *viz.*, that an analysis is decidable, in order to use it in program-processing tools such as compilers without jeopardizing their termination behaviour.

Static analysis has so far 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. 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 invariants of the computation trace done by a program. This is for example the case in "reaching definitions" analysis that aims at determining what definitions (in the shape of assignment statements) are always valid at a given program point.

- Alias analysis is another data flow analysis that finds out which variables in a program addresses the same memory location. This information is significant *e.g.*, when trying to recover unused memory statically ("compile-time garbage collection").
- Strictness analysis for lazy functional languages is aimed at detecting when the lazy call-by-need parameter passing strategy can be replaced with the more efficient call-by value strategy. This transformation is safe if the function is strict, that is, if calling the function with a diverging argument always leads to a diverging computation. This is *e.g.*, the case when the function is guaranteed to use this parameter; strictness analysis serve to discover when this is the case.
- Dependency analysis determines those parts of a program whose execution can influence the value of a particular expression in a program. This information is used in *program slicing*, a technique that permits to extract the part of a program that can be at the origin of an error, and to ignore the rest. Dependency information can also be used for determining when two instructions are independent, and hence can be executed in any order, or in parallel. Finally, dependency analysis plays an important role in software security where it forms the core of most information flow analyses.
- 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.

3.1.2. The structure of a static analysis

A static analysis can often be viewed as being split into two phases. The first phase performs an *abstract interpretation* of the program producing a system of equations or constraints whose solution represents the information of the program found by the analysis. The second phase consists in finding such a solution. The first phase involves a formal definition of the abstract domain *i.e.*, the set of properties that the analysis can detect, a technique for extracting a set of equations describing the solution from a program, and a proof of semantic correctness showing that a solution to the system is indeed valid information about the program's behaviour. The second phase can apply a variety of techniques from symbolic calculations and iterative algorithms to find the solution. An important point to observe is that the resolution phase is decoupled from the analysis and that the same resolution technique can be combined with different abstract interpretations.

3.1.3. Certified static analysis

In spite of the nice mathematical theory of program analysis (notably abstract interpretation) and the solid algorithmic techniques available one problematic issue persists, *viz.*, the *gap* between the analysis that is proved correct on paper and the analyser that actually runs on the machine. While this gap might be small for toy languages, it becomes important when it comes to real-life languages for which the implementation and maintenance of program analysis tools become a software engineering task.

A *certified static analysis* is an analysis whose implementation has been formally proved correct using a proof assistant. Such analysis can be developed in a proof assistant like Coq [30] by programming the analyser inside the assistant and formally proving its correctness. The Coq extraction mechanism then allows for extracting a Caml implementation of the analyser. The feasibility of this approach has been demonstrated in [6].

3.2. Program testing

Keywords: *Test data*, *Testing criterion*, *Testing hypothesis*, *integration testing*, *oracle*, *structural and functional testing*, *unit testing*.

Test data A test datum is a complete valuation of all the input variables of a program.

Test set A test set is a non-empty finite set of test data.

Testing criterion A testing criterion defines finite subsets of the input domain of a program. Testing criteria can be viewed as testing objectives.

Successful test set A test set is successful when the execution of the program with all the test data of a test set has given expected results

A reliable criterion A testing criterion is reliable iff it only selects either successful test set or unsuccessful test set.

A valid criterion A testing criterion is valid iff it produces unsuccessful test set as soon as there exists at least one test datum on which the program gives an incorrect result.

Ideal test set A test set is ideal iff it is unsuccessful or if it is successful then the program is correct over its input domain.

Fundamental theorem of structural testing A reliable and valid criterion selects only ideal test sets.

Program Testing involves several distinct tasks such as test data generation, program execution, outcome checking, non-regression test data selection, etc. Most of them are based on heuristics or empirical choices and current tools lack help for the testers. One of the main goal of the research undertaken by the Lande Project in this field consists in finding ways to automate some parts of the testing process. Current works focus on automatic test data generation and automatic oracle checking. Difficulties include test criteria formalization, automatic source code analysis, low-level specification modeling and automatic constraint solving. The benefits that are expected from these works include a better and deeper understanding of the testing process and the design of automated testing tools.

Program testing requires to select test data from the input domain, to execute the program with the selected test data and finally to check the correctness of the computed outcome. One of the main challenge consists in finding techniques that allow the testers to automate this process. They face two problems that are referenced as the *test data selection problem* and *the oracle problem*.

Test data selection is usually done with respect to a given structural or functional testing criterion. Structural testing (or white-box testing) relies on program analysis to find automatically a test set that guarantees the coverage of some testing objectives whereas functional testing (or black-box testing) is based on software specifications to generate the test data. These techniques both require a formal description to be given as input : the source code of programs in the case of structural testing ; the formal specification of programs in the case of functional testing. For example, the structural criterion *all_statements* requires that every statement of the program would be executed at least once during the testing process. Unfortunately, automatically generating a test set that entirely cover a given criterion is in general a formally undecidable task. Hence, it becomes necessary to compromise between completeness and automatization in order to set up practical automatic testing methods. This problem is called the *test data selection problem*.

Outcome checking is usually done with the help of a procedure called an oracle that computes a verdict of the testing process. The verdict may be either pass or fail. The former case corresponds to a situation where the computed outcome is equal to the expected one whereas the latter case demonstrates the existence of a fault within the program. Most of the techniques that tend to automate the generation of input test data consider that a complete and correct oracle is available. Unfortunately, this situation is far from reality and it is well known that most programs do not have such an oracle. Testing these programs is then an actual challenge. This is called *the oracle problem*.

Partial solutions to these problems have to be found in order to set up practical testing procedures. Structural testing and functional testing techniques are based on a fundamental hypothesis, known as the *uniformity hypothesis*. It says that selecting a single element from a proper subdomain of the input domain suffices to explore all the elements of this subdomain. These techniques have also in common to focus on the generation of input values and propositions have been made to automate this generation. They fall into two main categories :

- Deterministic methods aim at selecting a priori the test data in accordance with the given criterion. These methods can be either symbolic or execution-based. These methods include symbolic evaluation, constraint-based test data generation, etc.
- Probabilistic methods aim at generating a test set according to a probability distribution on the input domain. They are based on actual executions of the program. They include random and statistical testing, dynamic-method of test data generation, etc.

The main goal of the Lande project in this area consists in designing automated tools able to test complex imperative and sequential programs. An interesting technical synergy comes from the combination of several techniques including program analysis and constraint solving to handle difficult problems of Program Testing.

3.3. Reachability analysis over term rewriting systems

Keywords: *Term rewriting systems*, *reachability analysis*, *tree automata*.

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).

In rewriting, the problem of reachability is well-known: given a term rewriting system $\mathcal{R}$ and two ground terms s and t, t is $\mathcal{R}$-reachable from s if s can be finitely rewritten into t by $\mathcal{R}$, which is formally denoted by $s \rightarrow_{\mathcal{R}}^{*} t$. On the opposite, t is $\mathcal{R}$-unreachable from s if s cannot be finitely rewritten into t by $\mathcal{R}$, denoted by $s \not\rightarrow_{\mathcal{R}}^{*} t$.

Depending on the computing system modelled using rewriting, a deduction system, a function, some parallel processes or state transition systems, reachability (and unreachability) permit 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.

We are interested in proving (as automatically as possible) reachability or unreachability on term rewriting systems for verification and automated deduction purposes. The reachability problem is known to be decidable for terminating term rewriting systems. However, in automated deduction and in verification, systems considered in practice are rarely terminating and, even when they are, automatically proving their termination is difficult. On the other hand, reachability is known to be decidable on several syntactic classes of term rewriting systems (not necessarily terminating nor confluent). On those classes, the technique used to prove reachability is rather different and is based on the computation of the set $\mathcal{R}^*(E)$ of $\mathcal{R}$-reachable terms of an initial set of terms E. For those classes, $\mathcal{R}^*(E)$ is a regular tree language and can thus be represented using a *tree automaton*. Tree automata offer a finite way to represent infinite (regular) sets of reachable terms when a non terminating term rewriting system is under concern.

For the negative case, i.e. proving that $s \not\rightarrow_{\mathcal{R}}^{*} t$, we already have some results based on the over-approximation of the set of reachable terms [40], [41]. Now, we focus on a more general approach dealing with the positive and negative case at the same time. We propose a common, simple and efficient algorithm [7] for computing exactly known decidable regular classes for $\mathcal{R}^*(E)$ as well as to construct some approximation when it is not regular. This algorithm is essentially a *completion* of a *tree automata*, thus taking advantage of an algorithm similar to the Knuth-Bendix [46] *completion* in order not to restrict to a specific syntactic class of term rewriting systems and *tree automata* in order to deal efficiently with infinite sets of reachable terms produced by non-terminating term rewriting systems.

4. Software

4.1. A Coq library of lattices

Keywords: *Constructive logic*, *Coq modules*, *Lattices*.

Participant: David Pichardie.

We have developed a library of Coq modules for implementing lattices, the fundamental data structure of most static analysers. The motivation for this library was the development and extraction of certified static analysis in the Coq proof assistant—see Section 3.1. Using the abstract interpretation methodology, static analyses are specified as least solution of system of equations (inequations) on lattice structures. The library of Coq modules allows to construct complex and efficient lattices by combination of functors and base lattices. The lattice signature possesses a parameter which ensure termination of a generic fixed-point solver. The delicate problem of termination of fixpoint iterations is hence dealt with once and for all when building a lattice as a combination of the different lattice functors.

This library is currently freely available at http://www.irisa.fr/lande/pichardie/lattice/ under GPL license. It has been presented during the international FICS conference [25].

4.2. NIT: Null-ability Inference Tool

Keywords: *Java*, *Null pointer exceptions*, *Static analysis*.

Participant: Laurent Hubert.

The Null-ability Inference Tool is a tool to find suitable annotations for fields, method parameters and return values. It works at the bytecode level (on .class files or .jar files) so it can be used on programs where the source is not available. While this can look strange for a programmer, this tool can also be used by other static analyses to improve their precision. This software is distributed under the GNU General Public License.

4.3. JavaLib: parsing Java class files into OCaml

Keywords: *Java*, *Parsing*.

Participants: Laurent Hubert, Tiphaine Turpin.

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. It is distributed under the GNU General Public License.

4.4. Micromega: a reflexive Coq tactic for arithmetic

Keywords: *Farkas Lemma*, *Positivstellensatz*, *reflexive proof*, *result certification*.

Participant: Frédéric Besson.

Micromega is a reflexive Coq tactic able to decide quantifier free fragments of integer arithmetic [33]. We have implemented, and proved correct, a certificate checker for proofs obtained by Farkas lemma (linear arithmetic) and the *Positivstellensatz* (non-linear arithmetic). For the specific case of integer linear arithmetic, the certificate checker also accepts *cutting plane proofs* and proofs obtained from *branch-and-bound* algorithms.

A nice feature of the micromega checker is that it is built upon the existing Coq ring tactic [44]. Another nice feature is that the certificate generators are off-the-shelf algorithms that do not need to be trusted.

Micromega is now a Coq "contribution" included in the forthcoming Coq 8.2.

4.5. Timbuk: a tree automata library

Keywords: *Tree automata*, *approximations*, *term rewriting systems*.

Participant: Thomas Genet.

Timbuk [41] 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 in particular):

- 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.

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 in version 2.1. This version contains several optimisations and utilities. The completion algorithm complexity has been optimised for better performance in space and time. Timbuk now provides two ways to achieve completion: a dynamic version which permits to compute approximation step by step and a static version which pre-compiles matching and approximation in order to enhance speed of completion. Timbuk 2.1 also provides a graphical interface called *Tabi* for browsing tree automata and figure out more easily what are the recognized language, as well as *Taml* an Ocaml toplevel with basic functions on tree automata. Timbuk 2.1 has been used for a case study done with Thomson-Multimedia for cryptographic protocol verification.

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) [52], [51]. Pierre-Cyrille Heam, Yohan Boichut and Olga Kouchnarenko of the Cassis Inria project use Timbuk as a verification back-end [45] for AVISPA [32]. AVISPA is a powerful 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.

4.6. SPAN: Security Protocol ANimator for AVISPA

Keywords: *Cryptographic protocols*, *HLPSL specifications*, *Message Sequence Charts*, *execution trace*, *protocol animator*.

Participant: Thomas Genet.

AVISPA is now a commonly used verification tool for cryptographic protocols [32]. It is composed of four verification tools: ATSE, OFMC, SATMC and TA4SP. A protocol designer interacts with the tool by specifying a security problem (i.e. a protocol paired with a security property that the protocol is expected to achieve) in the High-Level Protocol Specification Language (HLPSL for short [38]). The HLPSL is an expressive, modular, role-based, formal language that is used to specify control-flow patterns, data-structures, alternative intruder models and complex security properties, as well as different cryptographic primitives and their algebraic properties. These features make HLPSL well suited for specifying modern, industrial-scale protocols.

In order to help protocol designers in designing and debugging HLPSL specifications, we have developed SPAN [43], a tool for animating them, i.e. interactively producing Message Sequence Charts (MSC for short) which can be seen as an “Alice & Bob” trace from an HLPSL specification. Starting from such an HLPSL specification, SPAN helps to build one possible MSC corresponding to that specification. This tool can represent one or more sessions of the protocol in parallel according to the information given in the HLPSL specification. Then, MSCs are produced interactively with the user. SPAN’s *intruder mode* makes it possible to interactively build attacks. This is of great interest when automatic verification tools do not produce the desired attack. SPAN also includes the possibility to check the values, at every moment, of the variables of each principal: the user chooses the variables of each roles he wants to monitor. The tool can save an execution trace corresponding to the execution of the protocol supervised by the user, and it is possible to reload it.

SPAN has been developed with Yann Glouche and Erwan Houssay and is registered at the APP with number IDDN.FR.001.25013.000.S.P.2007.000.10600. SPAN is distributed under the Gnu Library General Public License and freely available at http://www.irisa.fr/lande/genet/span/ in source format and as windows, linux and Mac OS binaries.

In 2008, we are now at version 1.5 that includes the ability to produce an interactive MSC trace from attacks found by the OFMC or ATSE tools. In 2007, there were more than 600 downloads of this software. In 2008, there were more than 1500. Recently, some experiments on the formalisation and verification of a protocol being developed at Thomson have been published in [22].

4.7. Constraint-Based Testing of critical C programs

Keywords: *SSA*, *constraint propagation*, *dynamic linear relaxations*, *search-based test data generation*.

Participant: Arnaud Gotlieb [contact point].

Euclide is software testing tool that features three main applications: structural test data generation, counter-example generation and partial program proving for critical C programs. The core algorithm of the tool takes as input a C program and a point to reach somewhere in the code. As a result, it outcomes either a test datum that reaches the selected point, or an "unreachable" indication showing that the selected point is unreachable. Optionally, the tool takes as input additional safety properties that can be given under the form of pre/post conditions or assertions directly written in the code. In this case, Euclide can either prove that these properties or assertions are verified according to an error-free semantics of the language or find a counter-example when there is one. As these problems are undecidable in the general case, Euclide only provides a semi-correct procedure (when it terminates, it provides the right answer) for them. Hopefully, by restricting the subset of C that the tool can handle (no dynamic memory allocation, no recursion) these non-termination problems remain infrequent in practice. In addition, Euclide implements several procedures that combine atomic calls to the core algorithm. For example, by selecting appropriate points to reach in the source code, the tool can generate a complete test suite able to cover the all_statements or the all_decisions criteria.

5. New Results

5.1. Access control model for interactive devices

Keywords: *access control*, *resource analysis*, *security model*.

Participants: Frédéric Besson, Thomas Jensen, David Pichardie.

The Lande groups continues its investigation of access control mechanisms by studying the security policy of mobile devices [14]. We have designed 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. This work extends a previous model proposed in [34].

5.2. Non-Null annotation inference

Keywords: *Java*, *Null pointer exceptions*, *Static analysis*.

Participants: Laurent Hubert, Thomas Jensen, David Pichardie.

A common source of exceptional program behaviour is the dereferencing of null references (also called null pointers), resulting in segmentation faults in C or null pointer exceptions in Java. Even if such exceptions are caught, the presence of exception handlers creates an additional amount of potential branching which in turn implies that: 1) fewer optimizations are possible and 2) verification is more difficult (bigger certification conditions, implicit flow in information flow verification, etc.). Furthermore, the Java virtual machine is obliged to perform run-time checks for non-nullness of references when executing a number of its bytecode instructions, thereby incurring a performance penalty. For all these reasons, a static program analysis which can guarantee before execution of the program that certain references will definitely be non-null is useful.

We propose a nullness static analysis [24] that automatically infers non-null annotations for local variables, method signatures and fields. The analysis has been mechanically proved sound in the Coq proof assistant. The Null-ability Inference Tool presented at PASTE'08 [23] is based on this work.

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

Keywords: *Abstract interpretation*, *Control Flow Analysis*.

Participants: Thomas Jensen, Jan Midtgaard.

Control-flow analysis (CFA) is a fundamental static analysis on which many other analyses rely. As such it has been the focus of researchers throughout the past two decades. Surprisingly, very few formulate CFA within the classical abstract interpretation methodology. Such a formulation of CFA is advantageous in that it is constructive: Rather than proving CFA safe a priori, CFA is induced by systematically composing and calculating with Galois connections. Unfortunately it has remained an open problem of how to exploit Galois connections and widenings for CFA since its formulation by Nielson and Nielson [50]. This work [27] represents a preliminary answer.

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

5.4. Static analysis and Proof Carrying Code

Keywords: *Proof-Carrying Code*, *abstract interpretation*, *proof-assistant*, *result certification*.

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

Proof-Carrying Code (PCC) is a technique for downloading mobile code on a host machine while ensuring that the code adheres to the host's safety policy. In the past, we have demonstrated how to use certified abstract interpretations (see Section 3.1.3) as the foundation for PCC architectures. For the approach to be practical, a number of issues have to tackled:

- the invariants inferred by abstract interpreters have to be kept small;
- checking the validity of these invariants has to be fast.

Abstract interpretation-based proof carrying code uses post-fixpoints of abstract interpretations to witness that a program respects a safety policy. Some witnesses carry more information than needed and are therefore unnecessarily large and costly to verify [35]. As a case study, we propose an enhanced lightweight Java bytecode verifier able to check interface types with negligible extra cost [18]. The standard way of checking interfaces is to encode the type of an interface by a conjunction of types. We present a fixpoint pruning algorithm able to remove conjunction types while preserving typability thus allowing to reuse the standard verification algorithm.

The goal of the european Mobius project is to develop a PCC architecture to secure global computers that consist of Java-enabled mobile devices. In [16], we have presented the consumer side of the Mobius PCC infrastructure, for which we have developed formally certified, executable checkers. We consider wholesale Proof Carrying Code scenarios, in which a trusted authority verifies the certificate before cryptographically signing the application. We also discuss retail Proof Carrying Code, where the verification is performed on the consumer device.

PCC rely on hybrid methods that combine static analyses and verification condition generation. While preliminary verification operate on source programs, it is often preferable to achieve guarantees about executable code. We have showed [17] that, for a hybrid verification method based on numerical static analysis and verification condition generation, compilation preserves proof obligations and therefore it is possible to transfer evidence from source to compiled programs. Our result relies on the preservation of the solutions of analysis by compilation; this is achieved by relying on a bytecode analysis that performs symbolic execution of stack expressions in order to overcome the loss of precision incurred by performing static analyses on compiled (rather than source) code. Finally, we show that hybrid verification methods are sound by proving that every program provable by hybrid methods is also provable (at a higher cost) by standard methods.

5.5. Tree Automata Completion Checker

Keywords: *Certification*, *Reachability Analysis*, *Term Rewriting*, *Tree Automata*.

Participants: Benoît Boyer, Thomas Genet, Thomas Jensen.

As shown in section 3.3, provided that a program or system can be represented by a term rewriting system, the (un)reachability of some program states can be proven using a tree automata completion algorithm. In this setting, completion computes a tree automaton recognising the set of all possible reachable terms. On the one side, we are interested in designing an efficient completion implementation so as to perform analysis on real-size problems. For example, the completion algorithm has been implemented in Tom (http://tom.loria.fr) and permitted to obtain completion times 100 times faster than the original implementation [15]. However, to obtain such results, it is necessary to achieve low level optimisations on the implementation that may corrupt the result, i.e. the completed tree automaton.

Hence, on the other side, we are also interested in designing 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 [19]. This checker is now used to certify all analysis results produced by regular completion as well as the optimised version of [15].

5.6. Improving formal specifications of cryptographic protocols

Keywords: *AVISPA*, *Cryptographic protocols*, *SPAN*, *formal specification*, *formal verification*.

Participants: Thomas Genet, Olivier Heen.

The verification of cryptographic protocols has greatly improved these last years. Automated tools such as AVISPA [31] provide real help in finding and characterizing attacks. The counterpart is that such tool require a formal specification of the protocol, using an appropriate language such as HLPSL. Since HLPSL is a very expressive language, this stage is complicated and error-prone before a correct specification is eventually obtained. The verification tools of AVISPA are not designed to detect such specification errors. In particular, a protocol whose HLPSL text is wrong may not be fully executable. Usually, a protocol that cannot be executed cannot be attacked by an intruder since not all the messages can be exchanged. Thus, a bugged HLPSL specification may be declared as *safe* by AVISPA tools. Hence, as long as it contains typo-like errors, the verification of a HLPSL specification is pointless.

In order to help designers during the formalisation of their cryptographic protocols, we have developed the SPAN tool [43] (see 4.6). SPAN can be seen as a companion tool for AVISPA which interactively produce Message Sequence Charts (MSC) from the formal HLPSL specification. We used AVISPA and SPAN in order to formalise and verify some industrial protocols designed by Thomson R&D. During this work, it appears that animation of HLPSL specifications can also be of great interest during the design of the protocol itself. Instead of short-lived, laborious and intricate drawings on a black board, designing protocol specification using HLPSL and SPAN reveals to be very efficient. Using HLPSL and animation at early stages of development is a convenient way of explaining and justifying the choices made during the protocol development. Furthermore, it also permits to rapidly consider and play with different environment assumptions or different execution of the protocol.

On one of the protocol under development at Thomson R&D, we found a flaw using AVISPA and SPAN. Again the conjoint use of AVISPA for verification and SPAN for explanation revealed to be useful. Experiments on the formalisation and verification of this protocol under development have been published in [22]. One the one side, automatic verification of AVISPA was crucial since the protocol and the attack were too complex to be found by manual analysis. In fact, the attack needs two different protocol sessions, four distinct agents and thirteen messages. On the other side, because of the complexity of the attack, animation using SPAN was necessary to figure out if *it was a real attack* and not an artifact of an ambiguous specification. Then, it helped in discussing with Thomson R&D so as to understand if this attack was realistic on an industrial point of view and, finally, how critical it was. Some other specification and verification experiments on a protocol devoted to ensuring anonymity within trust communities has been published in [26].

Conjointly to the development of SPAN and the verification of real-size protocols, we also made several efforts to promote cryptographic protocol verification. The first effort consists in a popularisation article explaining what is a cryptographic protocol and what are the properties to be shown. This is explained on a general public protocol: the basic credit card payment protocol with a smartcard [28]. On the other side, we also published an article in the french security journal MISC (Multi-system & Internet Security Cookbook) [29] to show that the verification tools for cryptographic protocols, such as AVISPA and SPAN, are now ready to be used in the industry.

5.7. Quantitative static analysis with linear operators over dioids

Keywords: *Dioids*, *Long-Run Cost*, *Resource Analysis*, *Static analysis*.

Participants: David Cachera, Thomas Jensen, Arnaud Jobin, Pascal Sotin.

We have defined a static analysis technique for modeling and approximating the long-run resource usage of programs. We take as starting point a standard small-step operational semantics expressed as a transition relation between states extended with costs associated to each transition. The set of costs is supposed to have two operations for composing costs: a "product" operator that combines the costs along an execution path, and a "sum" operator that combines costs coming from different paths. These operators will a structure of dioid to the set of costs. The sum operator induces a partial order on costs that will serve as a basis for approximating costs.

From such a rule-based semantics, there is a straightforward way to obtain a transition matrix, which entries represent the cost of passing from one state of the program to another. This expresses the semantics of a program as a linear operator on the moduloid of vectors of costs indexed over states.

We are interested in analysing programs with cyclic behaviour (such as reactive systems) in which the asymptotic average cost along cycles, rather than the global cost of the entire execution, is of interest. We define the notion of long-run cost for a program which provides an over-approximation of the average cost per transition of long traces. This notion corresponds to the maximum average of costs accumulated along a cycle of the program semantics and is computed from the traces of the successive iterates of the cost matrix. We have encapsulated all the properties necessary for defining such a long-run cost into the notion of a cost dioid, namely complete idempotent dioids equipped with a n-th root operator.

The quantitative operational semantics operates on state spaces that may be large or even infinite so the computation of quantitative semantic models, like their qualitative counterparts, is usually not tractable. Hence, it is necessary to develop techniques for abstracting this semantics, in order to return an approximation of the program costs that is feasible to compute. In line with the semantic machinery used to model programs, abstractions are also defined as linear operators from the moduloid over the concrete state space into the moduloid over the abstract one. Given such an abstraction over the semantic domains, we then have to abstract the transition matrix of the program itself into a matrix of reduced size. We give a sufficient condition for an abstraction of the semantics to be correct, *i.e.* to give an over-approximation of the real cost, and show how an abstract semantics that is correct by construction can be derived from the concrete one. The long-run cost of a program is thus safely approximated by an abstract long-run cost, with respect to the order relation induced by the summation operator of the dioid [20].

The framework proposed here covers a number of different costs related to resource usage (time and memory) of programs. To demonstrate the generality of the framework, we consider the less common (compared to time and space) analysis of cache behaviour and the number of cache misses in programs. We illustrate the notions of quantitative semantics, abstraction and long-run cost on a program written in a simple, intermediate bytecode language (inspired by Java Card) onto which we impose a particular cache model.

5.8. Constraint reasoning in Path-oriented Random Testing

Keywords: *constraint solving*, *path testing*, *random testing*, *uniform distribution*.

Participants: Matthieu Petit, Arnaud Gotlieb.

Path-oriented Random Testing (PRT) aims at generating a uniformly spread out sequence of random test data that activate a single control flow path within an imperative program. The main challenge of PRT is to build efficiently such a test suite in order to minimize the number of rejects (test data that activate another control flow path). In [21], we addressed this problem with an original divide-and-conquer approach based on constraint reasoning over finite domains, a well-recognized Constraint Programming technique. Our approach derives path conditions by using backward symbolic execution and computes an approximation of their associated subdomain by using constraint propagation and constraint refutation. We implemented our approach and got experimental results that show the practical interest of PRT based on constraint reasoning. In particular, we got a two-order magnitude CPU time improvement over a standard Random Testing approach when building a uniform test data generator for the longest path of a C implementation of the TCAS (Traffic Collision Avoidance System).

6. Contracts and Grants with Industry

6.1. The RNTL CAT project

Keywords: *C programming language*, *Static program analysis*, *pointer analysis*.

Participants: Arnaud Gotlieb, Thomas Jensen, Tristan Denmat.

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

6.2. The MEFORSE collaborative research contract with France Télécom R&D

Keywords: *Java on mobile devices J2ME*, *cryptographic protocols*, *static analysis*.

Participants: Frédéric Besson, Thomas Genet, Thomas Jensen.

Since 2004, the Lande project has a formalized collaboration with the France Télécom R&D team TAL/VVT based in Lannion. The collaboration is concerned with the modeling and analysis of software for telecommunication, in particular cryptographic protocols and Java (J2ME) applets written using the profile dedicated to mobile devices. The collaboration has so far lead to a list of features to verify on Java-enabled mobile telephones in order to ensure their security. We are notably interested in validating properties pertaining to the proper use of resources (eg. sending of SMS messages) for which we have developed a static analysis that allows to assert that a given applet will not use an unbounded amount of resources.

In another strand of the collaboration we analyse cryptographic protocols by over-approximating the protocol's and intruder's behavior. In general, the over-approximation is computable, whereas the exact behavior is not. To prove that there is no possible attack on the protocol we show that there is no attack on the over-approximation of its behavior. This leaves the problem of false positives: if the approximation contains an attack, it is not possible to say if it is a real attack or if it is due to the over-approximation. We thus work on attack reconstruction from the over-approximation of protocol's and intruder's behavior in order to discriminate between real and false attacks. We have already proposed a first algorithm which have been implemented and tested under the Timbuk library.

6.3. European FET-Integrated project MOBIUS

Keywords: *Java*, *Proof Carrying Code*, *Security*, *smart phones*, *static analysis*.
Participants: Thomas Jensen, Frédéric Besson, David Pichardie, Tiphaine Turpin.

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

In this project, we participate in the specification of security requirements and resource policies to be studied throughout the project. We are working at generating small and easy to check PCC certificates for resource-aware static analyses, see 5.4. We also aim at running PCC checkers on resource-constrained mobile devices.

6.4. The ANR SETIN RAVAJ

Keywords: *Application certification*, *Java*, *Reachability Analysis*, *Term Rewriting*.
Participants: Nicolas Barré, Benoît Boyer, Thomas Genet, Thomas Jensen.

The RAVAJ ANR (http://www.irisa.fr/lande/genet/RAVAJ/) started on january 2007, for 3 years. RAVAJ means "Rewriting and Approximation for the Verification of Java Applications". Thomas Genet is the coordinator of this project that concerns partners from Loria (Nancy), LIFC (Besançon) and IRISA (Rennes). The goal of this project is to propose a general purpose verification technique based based on approximations and reachability analysis over term rewriting systems. To tackle this goal, the tree automata completion method has to be refined in two different ways. First, though the Timbuk tool is efficient enough to verify cryptographic protocols, it is not the case for more complex software systems. In that direction, we aim at using some results obtained in rewriting [48] 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 [47] [53] [42]. On the other hand, we will automatically generate approximation equations from the property to be proved, using [36] [37], and also provide an automatic approximation refinement methodology adapted to the equational approximation framework.

6.5. The ANR SETIN PARSEC

Keywords: *Application certification*, *Java*, *Reachability Analysis*, *Term Rewriting*.

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

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

6.6. ANR SESUR 2007 CAVERN

Keywords: *Abstraction*, *Constraint-based Testing*, *Constraints*, *Program Verification*.

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

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

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

6.7. The COST Action IC0701

Keywords: *Object-oriented programs*, *Program verification*.

Participants: Thomas Jensen, David Pichardie.

COST Action IC0701 is an European scientific cooperation. The Action aims to develop verification technology with the power to ensure dependability of object-oriented programs on industrial scale. The action is composed of 15 countries.

7. Dissemination

7.1. Conferences: program committees, organization, invitations

Thomas Jensen was invited to give a talk on “ Static Analysis for Extended Byte Code Verification” at the 2nd International Workshop on Proof Carrying Code (PCC’08), Carnegie Mellon University Pittsburgh, Pennsylvania, USA, June 2008.

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

David Cachera was on the program committee of the ACM SAC’08 Software Verification Track.

7.2. PhD theses defended

Tristan Denmat, *Contraintes et abstractions pour la génération automatique de données de test*, PhD Université Rennes 1, on June 5th, 2008 [10].

Matthieu Petit, *Test statistique structurel par résolution de contraintes de choix probabiliste*, PhD Université Rennes 1, on July 4th, 2008 [11].

Pascal Sotin, *Quantitative Aspects of Program Analysis*, PhD Université Rennes 1, on December 5th, 2008 [12].

Tiphaine Turpin, *Pruning program invariants, Élagage d’invariants de programmes*, PhD Université Rennes 1, on December 15th, 2008 [13].

7.3. PhD and Habilitation committees

Frédéric Besson was external examiner on the PhD thesis of Sebastian Pavel at École des Mines de Nantes.

David Pichardie was external examiner on the PhD thesis of Dorina Ghindici at the University of Lille.

7.4. Teaching: university courses and summer schools

David Cachera teaches theoretical computer science at École Normale Supérieure de Cachan.

Thomas Genet teaches Cryptographic Protocols for M2 level (5th university year).

Arnaud Gotlieb teaches structural testing at M2 level in collaboration with Thierry Jéron (VERTECS project) and Sophie Pinchinat (S4 project) in the VTS module. He is principal teacher and responsible of the 5INFO module “Testing embedded software” at the 5th year of Insa Rennes. He also teaches “Structural Testing” at the Ecole des Mines de Nantes at the Master level. He was invited to give a lecture at the European TAROT Summer school on “Constraint-Based Testing”.

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

David Pichardie also teaches theoretical computer science at École Normale Supérieure de Cachan and formal methods for software engineering (the B method) at the 4th year of Insa Rennes in collaboration with Mireille Ducassé.

Thomas Genet gave a lecture on “Cryptographic protocols: principles, attacks and verification tools” at the summer school “École Jeune Chercheurs en Programmation” (Rennes, may 2008).

Olivier Heen gave a lecture on “Evaluating Internet Risk” at Telecom ParisTech (Paris, september 2008). He also co-animated a seminar on “Multiple Identities” for the The Media & Networks cluster (Rennes, october 2008).

Thomas Jensen is scientific leader of the *École Jeunes Chercheurs en Programmation*, an annual summer school for graduate students on programming languages and verification, organized under the auspices of the CNRS GdR ALP. This year's event was organised by Mamoun Filali at IRIT and took place at Luchon and Toulouse. The school attracted 40 participants for two weeks during June.

7.5. Administrative responsibilities

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

8. Bibliography

Major publications by the team in recent years

[1] A. BANERJEE, T. JENSEN. *Control-flow analysis with rank-2 intersection types*, in "Mathematical Structures in Computer Science", vol. 13, n^{o} 1, 2003, p. 87–124.

[2] F. BESSON, T. DE GRENIER DE LATOUR, T. JENSEN. *Interfaces for stack inspection*, in "Journal of Functional Programming", vol. 15, n^{o} 2, 2005, p. 179–217.

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

[4] F. BESSON, T. JENSEN, D. LE MÉTAYER, T. THORN. *Model ckecking security properties of control flow graphs*, in "Journal of Computer Security", vol. 9, 2001, p. 217–250.

[5] B. BOTELLA, A. GOTLIEB, C. MICHEL. *Symbolic execution of floating-point computations*, in "The Software Testing, Verification and Reliability journal", vol. 16, n^{o} 2, June 2006, p. 97–121.

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

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

[8] T. GENET, F. KLAY. *Rewriting for Cryptographic Protocol Verification*, in "Proc. of the 17th International Conference on Automated Deduction", LNAI, vol. 1831, Springer-Verlag, 2000, p. 271 – 290.

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

Year Publications

Doctoral Dissertations and Habilitation Theses

[10] T. DENMAT. *Contraintes et abstractions pour la génération automatique de données de test*, Ph. D. Thesis, Université Rennes 1, June 2008.

[11] M. PETIT. *Test statistique structurel par résolution de contraintes de choix probabiliste*, Ph. D. Thesis, Université Rennes 1, July 2008.

[12] P. SOTIN. *Quantitative Aspects of Program Analysis*, Ph. D. Thesis, Université Rennes 1, December 2008.

[13] T. TURPIN. *Pruning program invariants, Élagage d'invariants de programmes*, Ph. D. Thesis, Université Rennes 1, December 2008.

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", To appear, 2009.

International Peer-Reviewed Conference/Proceedings

[15] E. BALLAND, Y. BOICHUT, T. GENET, P.-E. MOREAU. *Towards an Efficient Implementation of Tree Automata Completion*, in "Algebraic Methodology and Software Technology, 12th International Conference, AMAST 2008", Lectures Notes in Computer Science, vol. 5140, Springer-Verlag, 2008, p. 67-82.

[16] G. BARTHE, P. CRÉGUT, B. GRÉGOIRE, T. JENSEN, D. PICHARDIE. *The MOBIUS Proof Carrying Code infrastructure*, in "Proc. of the 6th International Symposium on Formal Methods for Components and Objects (FMCO'07)", Lecture Notes in Computer Science, To appear, Springer-Verlag, 2008.

[17] G. BARTHE, C. KUNZ, D. PICHARDIE, J. S. FORLESE. *Preservation of Proof Obligations for Hybrid Certificates*, in "Proc. of the 6th IEEE International Conferences on Software Engineering and Formal Methods (SEFM'08)", To appear, IEEE Computer Society, 2008.

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

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

[20] D. CACHERA, T. JENSEN, A. JOBIN, P. SOTIN. *Long-Run Cost Analysis by Approximation of Linear Operators over Dioids*, in "Algebraic Methodology and Software Technology, 12th International Conference, AMAST 2008", Lectures Notes in Computer Science, vol. 5140, Springer-Verlag, 2008, p. 122-138.

[21] A. GOTLIEB, M. PETIT. *Constraint reasonning in Path-oriented Random Testing*, in "32nd Annual IEEE International Computer Software and Applications Conference (COMPSAC'08), Turku, Finland", 4 pages, Jul. 2008.

[22] O. HEEN, T. GENET, S. GELLER, N. PRIGENT. *An industrial and academical joint experiment on automated verification of a security protocol*, in "IFIP MWNS'08 Workshop", 2008.

[23] 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)", To appear, ACM, 2008.

[24] L. HUBERT, T. JENSEN, D. PICHARDIE. *Semantic foundations and inference of non-null annotations*, in "Proc. of the 10th International Conference on Formal Methods for Open Object-based Distributed Systems (FMOODS'08)", Lecture Notes in Computer Science, vol. 5051, Springer-Verlag, 2008, p. 132-149.

[25] D. PICHARDIE. *Building certified static analysers by modular construction of well-founded lattices*, in "Proc. of the 1st International Conference on Foundations of Informatics, Computing and Software (FICS'08)", Electronic Notes in Theoretical Computer Science, vol. 212, 2008, p. 225-239.

National Peer-Reviewed Conference/Proceedings

[26] O. HEEN, G. GUETTE, T. GENET. *Anonymity within trust communities*, in "SAR-SSI 2008, 3rd conference on security in network architectures and information systems", N. CUPPENS-BOULAHIA, P. OWEZARSKI (editors), 2008, p. 183–195.

[27] J. MIDTGAARD, T. JENSEN. *A Calculational Approach to Control-Flow Analysis by Abstract Interpretation*, in "Proc. of the 15th Static Aanalysi Symposium", LNCS, vol. 5079, Springer Verlag, 2008, p. 347-362.

Scientific Popularization

[28] T. GENET. *Le protocole cryptographique de paiement par carte bancaire*, Interstices, Février 2008, http://interstices.info/jcms/c_33835/le-protocole-cryptographique-de-paiement-par-carte-bancaire.

[29] O. HEEN, T. GENET, E. HOUSSAY. *Votre protocole est-il vérifié?*, vol. 39, Multi-system & Internet Security Cookbook, Diamond, Septembre 2008.

References in notes

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

[31] 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. *AVISPA – a tool for Automated Validation of Internet Security Protocols*, http://www.avispa-project.org.

[32] 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 "17th International Conference on Computer Aided Verification, CAV'2005, Edinburgh, Scotland", K. ETESSAMI, S. RAJAMANI (editors), Lecture Notes in Computer Science, vol. 3576, Springer, 2005, p. 281-285.

[33] F. BESSON. *Fast Reflexive Arithmetic Tactics the linear case and beyond*, in "Types for Proofs and Programs (TYPES'06)", LNCS, vol. 4502, Springer-Verlag, 2007, p. 48–62.

[34] F. BESSON, G. DUFAY, T. JENSEN. *A Formal Model of Access Control for Mobile Interactive Devices*, in "11th European Symposium On Research In Computer Security (ESORICS'06)", Lecture Notes in Computer Science, vol. 4189, Springer, 2006.

[35] F. BESSON, T. JENSEN, T. TURPIN. *Small Witnesses for Abstract Interpretation-Based proofs*, in "Proc. of 16th European Symposium on Programming (ESOP'07)", LNCS, vol. 4421, Springer, 2007, p. 268–283.

[36] 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.

[37] Y. BOICHUT, P.-C. HÉAM, O. KOUCHNARENKO. *Automatic Verification of Security Protocols Using Approximations*, in revision for Journal of Automated Reasoning, Research Report, n^{o} RR 5727, INRIA, 2005, http://hal.inria.fr/inria-00070291/fr/.

[38] Y. CHEVALIER, L. COMPAGNA, J. CUELLAR, P. HANKES DRIELSMA, J. MANTOVANI, S. MÖDERSHEIM, L. VIGNERON. *A High Level Protocol Specification Language for Industrial Security-Sensitive Protocols*, in "Proceedings of Workshop on Specification and Automated Processing of Security Requirements (SAPS), Linz, Austria", vol. 180, Oesterreichische Computer Gesellschaft (Austrian Computer Society), 2004.

[39] P. COUSOT, R. COUSOT. *Abstract Interpretation: A unified lattice model for static analysis of programs by construction of approximations of fixpoints*, in "Proc. of 4th ACM Symposium on Principles of Programming Languages", ACM Press, New York, 1977, p. 238–252.

[40] T. GENET. *Decidable Approximations of Sets of Descendants and Sets of Normal forms*, in "Proc. 9th International Conference on Rewriting Techniques and Applications", LNCS, vol. 1379, Springer-Verlag, 1998, p. 151–165.

[41] T. GENET, V. VIET TRIEM TONG. *Reachability Analysis of Term Rewriting Systems with Timbuk*, in "Proc. of the 8th International Conference on Logic for Programming, Artificial Intelligence and Reasoning", LNAI, vol. 2250, Springer-Verlag, 2001, p. 691–702.

[42] T. GENET, V. VIET TRIEM TONG. *Proving Negative Conjectures on Equational Theories using Induction and Abstract Interpretation*, Technical report, n^{o} RR-4576, INRIA, 2002, http://hal.inria.fr/inria-00072012.

[43] Y. GLOUCHE, T. GENET. *SPAN – A Security Protocol ANimator for AVISPA – User Manual*, 2006, http://www.irisa.fr/lande/genet/span/, IRISA / Université de Rennes 1.

[44] B. GRÉGOIRE, A. MAHBOUBI. *Proving Equalities in a Commutative Ring Done Right in Coq*, in "Proc. of the 18th Int. Conference on Theorem Proving in Higher Order Logics", 2005, p. 98-113.

[45] P.-C. HÉAM, Y. BOICHUT, O. KOUCHNARENKO, F. OEHL. *Improvements on the Genet and Klay Technique to Automatically Verify Security Protocols*, in "Proc. of AVIS", 2004.

[46] D. E. KNUTH, P. B. BENDIX. *Simple word problems in universal algebras*, in "Computational Problems in Abstract Algebra, Oxford", J. LEECH (editor), Pergamon Press, 1970, p. 263–297.

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

[48] P.-E. MOREAU, C. RINGEISSEN, M. VITTEK. *A Pattern Matching Compiler for Multiple Target Languages*, in "12th Conference on Compiler Construction, Warsaw (Poland)", G. HEDIN (editor), LNCS, vol. 2622, Springer-Verlag, May 2003, p. 61–76.

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

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

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

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

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