INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team Lande

Logiciel : ANalyse et DEveloppement

Rennes - Bretagne Atlantique

THEME SYM

Table of contents

1. Team

Head of project-team
Thomas Jensen [CNRS, DR, HdR]

Administrative assistant
Lydie Letort [Inria, TR]
Laetitia Rihet [Inria, TR]

Research scientists
Frédéric Besson [Inria, CR]
Arnaud Gotlieb [Inria, CR]
David Pichardie [Inria, CR]

Assistant Professors
Thomas Genet [University of Rennes 1]
David Cachera [ENS Cachan]

PhD students
Gurvan Le Guernic [ATER, until September 2007]
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]

Post-doctoral fellows
Yohan Boichut [University Rennes 1, until August 2007]
Jan Midtgaard [INRIA]
Frédéric Dabrowski [INRIA]
Pierre Rousseau [INRIA]

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.

2.2. Highlights of the year

Constraint and abstraction The combination of constraint programming and abstraction techniques exhibit high potential for program verification. This year has brought the first results of this combination in the area of verifying non-linear invariants over C and Java programs [19], [20]. Also starting this year is the ANR SESUR CAVERN project (coordinated by Arnaud Gotlieb from the Lande team) which aims at studying the combination of these techniques further.

Micromega Micromega is an efficient reflexive Coq tactic for linear and non-linear arithmetic. Certificates generated by (untrusted) external provers are verified by an optimised checker proved correct inside the Coq proof-assistant. For linear goals, Micromega outperforms existing tactics. For non-linear goals, Micromega provides automatic support where none was available. The theoretical foundations of Micromega are Farkas lemma and the *Positivstellensatz*. (See Section 4.2).

Rewriting and static analysis We have succeeded in bringing together the dedicated verification technique based on rewriting and tree automata with usual static program analysis. Reconciling the two gives more flexibility to static analysis by making it more independent of the target language and of the used abstractions. This technique has been experimented for the fast prototyping of class analysis on Java byte code programs [17]. (See section 5.1).

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 $f^n(\bot)$ where $\bot$ is the least element of the partial order.

3.1.1. Static program analysis

[33], [44] 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 [27] 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 [35], [36]. 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 [41] *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/these/ under GPL license.

4.2. 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 [14]. 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 [39]. Another nice feature is that the certificate generators are off-the-shelf algorithms that do not need to be trusted.

This piece of software is a key component of our PCC architecture based on certified abstract interpretation (see Section 3.1.3). The theory behind Micromega is currently used to efficiently check polyhedral invariants inferred by abstract interpretation 5.4.

Micromega is available at http://www.irisa.fr/lande/fbesson/.

4.3. Timbuk: a tree automata library

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

Participant: Thomas Genet.

Timbuk [36] 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.

A version 2.1 of *Timbuk* is now available. This new 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) [46], [45]. Pierre-Cyrille Heam, Yohan Boichut and Olga Kouchnarenko of the Cassis Inria project use Timbuk as a verification back-end [40] for AVISPA [29]. AVISPA is a powerful tool for verifying cryptographic protocols defined in high level protocol specification format.

4.4. 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 [29]. 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 [32]). 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 [38], 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. This

software has been developed in contact with Thomson R&D in order to be as close as possible with industrial expectations. Some of the results of this collaboration have been published in [16].

SPAN has been developed with Yann Glouche 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 2007, there were more than 600 downloads of this software.

4.5. Symbolic Evaluation of Floating-Point computations

Keywords: *IEEE 754*, *Symbolic execution*, *floating-point computations*.

Participant: Arnaud Gotlieb [contact point].

FPSE is a symbolic evaluation tool dedicated to ANSI C floating-point computations that conform IEEE 754. The tool takes as input an expression of the path conditions resulting from the selection of a path within a C function. It computes either an over-approximation of the domain associated to the path or a test data that activates the selected path. It handles restricted forms of C floating-point computations including the fourth standard operators (+,-,x,%) and integer-floats conversions, the single and double format, zeros and infinities. The tool is available on demand and its results are accessible through a simple C API.

5. New Results

5.1. Rewriting for fast prototyping of static analysers

Keywords: *Java*, *Reachability Analysis*, *Static Analysis*, *Term Rewriting*, *Tree Automata*.

Participants: Yohan Boichut, Thomas Genet, Thomas Jensen.

Usually, for defining a static analyser, one has to choose a formal semantics of a programming language, then choose an abstract domain in order to approximate the variables, stacks, heaps, etc. and then to develop an *analyser* using both. However, this approach lacks flexibility since for any evolution either of the semantics or of the abstract domain, one has to redevelop the analyser. This is a problem when the objective is either to prototype or to maintain the static analyser.

In this work, we show how to construct static analysers using tree automata and rewriting techniques. Starting from a term rewriting system representing the operational semantics of the target programming language and given a program to analyse, we automatically construct an over-approximation of the set of reachable terms [34], i.e. of the program states that can be reached. The approach enables fast prototyping of static analysers because, on the one hand, modifying the semantics simply amounts to changing the set of rewrite rules defining the operational semantics. On the other hand, changing the abstract domain can be performed by changing so-called *normalisation rules*. Another salient feature of this approach is that the approximation is correct by construction and hence does not require an explicit correctness proof. In [17], we presented this framework and instantiate it on a real test case, namely Java. We encode the Java Virtual Machine (JVM for short) operational semantics and Java bytecode programs into TRS and construct over-approximations of JVM states.

With regards to rewriting, the contribution of this work is dual. First, we propose an encoding of a significant part of Java into *left-linear, unconditionnal* TRS. The second contribution is to have scaled up the theoretical construction of tree automata completion to the verification of Java bytecode programs. With respect to static analysis, the contribution of this work is to show that regular approximations can be used as a foundational mechanism for ensuring, by construction, safety of static analysers.

5.2. Improving formal specifications of cryptographic protocols

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

Participants: Yohan Boichut, Thomas Genet.

The verification of cryptographic protocols has greatly improved these last years. Automated tools such as AVISPA [28] 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 [38] (see 4.4). SPAN can be seen as a companion tool for AVISPA which interactively produce Message Sequence Charts (MSC) from the formal HLPSL specification. In [16], we have shown how visualisation eases the communication between the protocol designers and the formalisation group: drawing of typical execution diagrams, visualisation of protocol termination, understanding of interleaved sessions, detection of unwanted side effects, etc. We also shown how visualisation and simulation of an intruder helps in finding attacks that are not automatically detected by tools.

We used AVISPA and SPAN conjointly with Olivier Heen 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. Once again the conjoint use of AVISPA for verification and SPAN for explanation revealed to be useful. 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.

5.3. Dynamic and static information flow analysis

Keywords: *Information flow*, *dependency*, *dynamic analysis*, *machine checked proofs*, *security levels*.

Participants: Gurvan Le Guernic, David Pichardie.

A standard way of formalising confidentiality is via the notions of information flow and non-interference. We have contributed to this research area through two distinct approachs: monitoring and type systems.

During his Phd work [11] , Gurvan Le Guernic has developed an information flow monitoring mechanism for sequential programs. A monitor enforcing noninterference is more complex than standard execution monitors. The technique proposed here is based on the combination of dynamic and static information flow analyses. The practicality of such an approach is demonstrated by the development of a monitor for concurrent programs including synchronization commands [21]. Based on these techniques, we have also proposed an information flow testing mechanism [22]. This mechanism is sound from the point of view of noninterference. It is based on standard testing techniques and on a combination of dynamic and static analyses.

On the purelly static approach, much previous work on type systems for non-interference has focused on calculi or high-level programming languages, and existing type systems for low-level languages typically omit objects, exceptions, and method calls, and/or do not prove formally the soundness of the type system. In collaboration with G. Barthe and T. Rezk at INRIA Sophia Antipolis we have developed [13] an information flow type system for a sequential JVM-like language that includes classes, objects, arrays, exceptions and method calls, and proved that it guarantees non-interference. For increased confidence, we have formalized the proof in the proof assistant Coq. Our work provides, to our best knowledge, the first sound and implemented information flow type system for such an expressive fragment of the JVM.

5.4. Certified result certification of abstract interpretations

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.

We have developed a comprehensive approach to deal with these issues.

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. We propose techniques for reducing the size of such certificates [15]. For distributive analyses, we provide a constructive proof of existence of a smallest witness. For non-distributive analyes we propose a technique for pruning a witness and keeping only the minimal information needed to prove a given safety property.

The PCC architecture must be equipped with certified algorithms able to ensure that an invariant (in the form of an element of an abstract domain) is actually a post-fixpoint of the abstract interpreter. To alleviate the burden of certifying complicated and costly decision procedures, we advocate a result certification approach. As complexity theory shows, NP-complete decision procedures admit polynomial result checkers that are guided by a suitable certificate. For a wide class of numeric abstract domain, we provide simple and fast certificate checkers for which certificates generators are off-the-shelf algorithms [14].

The previous improvements are included in our latest experiment [25]: a certified result checker for a relational polyhedra-based abstract interpretation. The analysis is defined for an imperative, stack-oriented byte code language with procedures, arrays and global variables. The analysis has automatic inference of loop invariants and method pre-/post-conditions. Invariants, which can be large, can be specialized for proving a safety policy using pruning techniques [15] which reduce their size. The result of the analysis can be checked efficiently by annotating the program with parts of the invariant together with certificates of polyhedral inclusions, which allow to avoid certain complex polyhedral computation such as the convex hull of two polyhedra. Small, easily checkable inclusion certificates are obtained using Farkas lemma for proving the absence of solutions to systems of linear inequalities [14]. The resulting checker is sufficiently simple to be entirely certified within the Coq proof assistant.

5.5. Quantitative static analysis with linear operators over dioids

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

Participants: David Cachera, Thomas Jensen, 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, where each transition is labelled by a cost taken in a complete idempotent dioid. From such a rule-based semantics, there is a straightforward way to obtain a transition matrix, in which a matrix entry represents 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 the set of states.

Using these mathematical structures allows for defining two notions of costs for a program. On one hand, the notion of global cost corresponds to the maximum cost of an execution from an input state to an output state. On the other hand, the notion of long-run cost corresponds to the maximum average of costs accumulated along a cycle of the semantics graph, and thus to an asymptotic average behaviour of the program.

The framework proposed here covers a number of different costs related to resource usage (time and memory) of programs. As an application example, we have chosen a slightly unusual cost model pertaining to the analysis of cache behaviour and the number of cache misses in programs. In a preliminary work, we have illustrated the notions of quantitative semantics and the associated costs on programs written in a simple, intermediate bytecode language (inspired by Java Card) onto which we impose a particular cache model [47].

The quantitative operational semantics operates on finite, but potentially large state spaces so quantitative semantic models are usually not tractable. Hence, it is necessary to develop techniques for abstracting this semantics. In line with the semantic machinery used to model programs, abstractions are also defined as linear operators from the set of concrete vectors into the set of abstract ones. 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 [26].

5.6. Uniform selection of feasible paths through stochastic constraint solving

Keywords: *Probabilistic constraint solving*, *statistical testing*, *uniform selection of feasible paths*.

Participants: Matthieu Petit, Arnaud Gotlieb.

Uniform selection of feasible paths is a difficult problem that arise in statistical testing. In this work, we proposed using stochastic constraint solving techniques to build a model of imperative programs on which uniform selection of feasible paths was possible. This required to improve the implementation of current probabilistic choice operators by defining and implementing new filtering algorithms [23]. A stochastic constraint library called PCC(FD) for SICStus Prolog was built and experimentated. We got interesting results by using it to select uniformly feasible paths from simples Java Card examples [24]. We plan to address more complicated examples such as the SUN's e-purse implementation.

5.7. Constraint-Based Testing

Keywords: *Constraint-based automatic test data generation*, *constraint propagation*, *linear relaxation*.

Participants: Florence Charreteur, Tristan Denmat, Arnaud Gotlieb.

Constraint-Based Testing (CBT) is the process of generating test cases against a testing objective by using constraint solving techniques. In CBT, testing objectives are given under the form of properties to be satisfied by program's input/output. Whenever the program or the properties contain disjunctions or multiplications between variables, CBT faces the problem of solving non-linear constraint systems. Currently, existing CBT tools tackle this problem by exploiting a finite-domains constraint solver. But, solving a non-linear constraint system over finite domains is NP_hard and CBT tools fail to handle properly most properties to be tested. In this work, we introduced a CBT approach where a finite domain constraint solver is enhanced by Dynamic Linear Relaxations (DLRs). DLRs are based on linear abstractions derived during the constraint solving process. They dramatically increase the solving capabilities of the solver in the presence of non-linear

constraints without compromising the completeness or soundness of the overall CBT process. We implemented DLRs within the CBT tool TAUPO that generates test data for programs written in C and got initial results over a few (academic) C programs [20]. Extending this approach to handle loops that depend on input variables led us to combine narrowing techniques within constraint combinators by defining a new constraint language inspired from the imperative programming style [19]. We also addressed the problem of Constraint-Based Testing of pointer programs [12], [18].

This activity is done in collaboration with Mireille Ducassé from the IRISA LIS team.

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. Static analysis of Java Midlets with rewriting and reachability analysis

Keywords: *Java MIDP*, *Reachability Analysis*, *Term Rewriting*.

Participants: Yohan Boichut, Thomas Genet, Laurent Hubert, Thomas Jensen.

This contract with France Telecom R & D started on October 1, 2004, for 3 years. The objective is to prove security properties on Java Midlets to be downloaded and executed on a cell phone. The first goal is to extract the flow graph of the graphical interface (i.e. flow graph between the screens) of the midlet directly from the bytecode. Then, the security property in question is *e.g.*, whether critical methods of the Java MIDP library are called under some restrictions. A typical property to prove is that every sent message has been authorized by the user, i.e. every call to the message sending JAVA MIDP method should be done after a screen where the user has to confirm that he agrees with the sending of a message.

Since the specification of the JAVA MIDP library is constantly evolving, it is impossible to define once for all a static analyser able to deal with the analysis described above. We thus aim at taking advantage of the reachability analysis technique over term rewriting systems (see module 3.3) to define an analyser that can easily evolve with MIDP. The idea is to specify the semantics of the Java virtual machine, the semantics of MIDP and of a given midlet using term rewriting systems and then to use the automatic approximation of term rewriting systems to over-approximate the flow graph of the midlet. Then, for making the analyser evolve with MIDP it should be enough to adapt the term rewriting system describing the semantics of MIDP. The main challenges of this project are, first to define the Java MIDP semantics using term rewriting in a simple and usable way, second to express the usual static analysis achieved on Java by means of approximations on term rewriting systems and last to produce a complete analyser efficient enough to prove properties on real midlets.

6.3. 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.

6.4. 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.

6.5. The ANR SETIN RAVAJ

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

Participants: Benoît Boyer, Thomas Genet, Thomas Jensen.

The RAVAJ ANR (http://www.irisa.fr/lande/genet/RAVAJ/) started on january 2007, for 3 years. RAVAJ means "Rewriting and Approximation for the Verification of Java Applications". Thomas Genet is the coordinator of this project that concerns partners from Loria (Nancy), LIFC (Besançon) and IRISA (Rennes). The goal of this project is to propose a general purpose verification technique based based on approximations and reachability analysis over term rewriting systems. To tackle this goal, the tree automata completion method has to be refined in two different ways. First, though the Timbuk tool is efficient enough to verify cryptographic protocols, it is not the case for more complex software systems. In that direction, we aim at using some results obtained in rewriting [43] 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 [42] [48] [37]. On the other hand, we will automatically generate approximation equations from the property to be proved, using [30] [31], and also provide an automatic approximation refinement methodology adapted to the equational approximation framework.

6.6. 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 focus 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).

6.7. The ACI Sécurité Informatique project SATIN

Keywords: *cryptographic protocols*, *security protocols*, *verification*.

Participants: Yohan Boichut, Thomas Genet.

The SATIN ACI (http://lifc.univ-fcomte.fr/ heampc/SATIN/) started on July 2004, for 3 years. SATIN means Security Analysis for Trusted Infrastructures and Network protocols. This project gathers some academic (Loria, LIFO, LIFC, Irisa) and industrial researchers (CEA-DAM and France Telecom R&D). The main goal of the project is to bring academic tools closer to industrial expectations. Indeed, there are more and more academic tools, based on process algebras and on rewriting techniques dedicated to protocol verification. Furthermore, several interesting security analysis results have recently been obtained in these directions, but some fundamental issues remain before these results can be applied to practical problems of industrial type: more efficient decision procedures and closer approximation results, taking into account the incidence of time, modeling imperfect cryptographic primitives and the notion of denial of service suitability to consider attacks based on them.

In this ACI, Timbuk is used as one of the verification back-end. For instance, researchers of LIFC use Timbuk to prove security properties on protocol specification in HLPSL format (High Level Protocol Specification Language). HLPSL has been designed in the European Project AVISS. During the ACI SATIN, Lande is going to strengthen several aspects of the Timbuk tool. First of all we plan to refine the system so that it can produce more precise trace information when it discovers an attack on a protocol. Then, we aim at implementing the completion algorithm for conditional term rewriting systems so that it can handle more detailed protocol models and more specific protocols behaviors, like conditional protocols. Finally, since the tree automata built with Timbuk, representing the set of reachable terms, is used to prove properties on critical programs – security protocols – we would like to certify this result. We propose to build with Coq a checker proving that the tree automaton, produced by Timbuk, is complete w.r.t. reachable terms.

7. Dissemination

7.1. Conferences: program committees, organization, invitations

Thomas Jensen was invited to give a talk on “Certificates for resource usage on mobile telephones” at the 7th International Workshop on Issues in the Theory of Security (WITS’07), Braga, Portugal, March 2007.

Thomas Genet was invited to give a talk on “Rewriting and Reachability for Software Security” at the 2nd workshop on Security and Rewriting Techniques (SecReT 2007), Paris July 2007.

Thomas Genet was invited to give a talk on “Cryptographic protocol verification: mutual benefits from academic and industrial viewpoints” at 14^{th} “Rencontres INRIA - Industrie”, INRIA Rocquencourt, October 2007.

David Pichardie was on the program committee of NORDSEC’07 (Nordic Workshop on Secure IT-systems).

Frédéric Besson was on the program committee of the ETAPS Workshop BYTECODE’07.

Arnaud Gotlieb served in the program committee of the int. conf. QSIC'07, the second workshop on Random Testing (RT'06) and the first workshop on oracles (STEV'07).

7.2. PhD theses defended

Gurvan Le Guernic, *Confidentiality Enforcement Using Dynamic Information Flow Analyses*, PhD Université Rennes 1 and Kansas State University, on September 27th, 2007 [11].

Stéphane Hong Tuan Ha, *Programmation par aspects et tissage de propriétés - Application à l'ordonnancement et à la disponibilité.* PhD Université Rennes 1, on January 30th, 2007 [10].

7.3. PhD and Habilitation committees

Thomas Jensen was external examiner (*rapporteur*) on the PhD thesis of Yann Hodique at the University of Lille and at the *habilitation* of Mario Sudholt at the Ecole des Mines de Nantes.

Thomas Genet was external examiner at the *habilitation* of Vlad Rusu at the University of Rennes I.

Arnaud Gotlieb was examiner on the PhD thesis of Patricia Mouy at CEA-LIST.

7.4. Teaching: university courses and summer schools

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

Thomas Genet teaches formal methods for software engineering (with "B") and Cryptographic Protocols for M1 level (4th university year). He also teaches formal methods and proof assistants to M2 students in collaboration with Vlad Rusu (VERTECS project).

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.

David Pichardie gave a lecture on certified programming with the Coq proof assistant and initiation on abstract interpretation at the summer school "École Jeune Chercheurs en Programmation" (Rennes, june 2007). He also gave a lecture on Proof Carrying Code at the TYPES summer school (Bertinoro, Italy, August 2007) and a tutorial about Secure Information Flow during the Mobius tutorial at ETAPS'07 (Braga, Portugal).

Thomas Genet gave a lecture on "Proofs for program analysis: rewriting for verification" at International School on Rewriting (Nancy, july 2007).

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 president of the Scientific Committee (*comité des projets*) at Irisa.

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.

[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] S. HONG TUAN HA. *Programmation par aspects et tissage de propriétés - Application à l'ordonnancement et à la disponibilité.*, Ph. D. Thesis, Université Rennes 1, January 2007.

[11] G. LE GUERNIC. *Confidentiality Enforcement Using Dynamic Information Flow Analyses*, Ph. D. Thesis, Université Rennes 1 and Kansas State University, September 2007.

Articles in refereed journals and book chapters

[12] 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.

Publications in Conferences and Workshops

[13] G. BARTHE, D. PICHARDIE, T. REZK. *A Certified Lightweight Non-Interference Java Bytecode Verifier*, in "Proc. of 16th European Symposium on Programming (ESOP'07)", Lecture Notes in Computer Science, vol. 4421, Springer-Verlag, 2007, p. 125-140.

[14] 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.

[15] 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.

[16] Y. BOICHUT, T. GENET, Y. GLOUCHE, O. HEEN. *Using Animation to Improve Formal Specifications of Security Protocols*, in "Joint conference SAR-SSI", 2007.

[17] Y. BOICHUT, T. GENET, T. JENSEN, L. LEROUX. *Rewriting Approximations for Fast Prototyping of Static Analyzers*, in "RTA", LNCS, vol. 4533, Springer Verlag, 2007, p. 48-62.

[18] F. CHARRETEUR, B. BOTELLA, A. GOTLIEB. *Modelling dynamic memory management in Constraint-Based Testing*, in "TAIC-PART (Testing: Academic and Industrial Conference), Windsor, UK", Sep. 2007, p. 111–120.

[19] T. DENMAT, A. GOTLIEB, M. DUCASSE. *An Abstract Interpretation Based Combinator for Modeling While Loops in Constraint Programming*, in "Proceedings of Principles and Practices of Constraint Programming (CP'07), Providence, USA", Springer Verlag, LNCS 4741, Sep. 2007, p. 241–255.

[20] T. DENMAT, A. GOTLIEB, M. DUCASSE. *Improving Constraint-Based Testing with Dynamic Linear Relaxations*, in "18th IEEE International Symposium on Software Reliability Engineering (ISSRE' 2007), Trollhättan, Sweden", Nov. 2007.

[21] G. LE GUERNIC. *Automaton-based Confidentiality Monitoring of Concurrent Programs*, in "Proceedings of the 20th IEEE Computer Security Foundations Symposium (CSFS20)", IEEE Computer Society, July 6–8 2007.

[22] G. LE GUERNIC. *Information Flow Testing*, in "Proceedings of the Annual Asian Computing Science Conference", Lecture Notes in Computer Science, To appear, December 9–11 2007.

[23] M. PETIT, A. GOTLIEB. *Boosting Probabilistic Choice Operators*, in "Proceedings of Principles and Practices of Constraint Programming, Providence, USA", Springer Verlag, LNCS 4741, September 2007, p. 559–573.

[24] M. PETIT, A. GOTLIEB. *Uniform Selection of Feasible Paths as a Stochastic Constraint Problem*, in "Proceedings of International Conference on Quality Software (QSIC'07), Portland, USA", IEEE, October 2007.

Internal Reports

[25] F. BESSON, T. JENSEN, D. PICHARDIE, T. TURPIN. *Result certification for relational program analysis*, Research Report, n^{o} 6333, Inria, September 2007, http://hal.inria.fr/inria-00166930/.

[26] D. CACHERA, T. JENSEN, P. SOTIN. *Long-Run Cost Analysis by Approximation of Linear Operators over Dioids*, Research Report, n° 6338, INRIA, 10 2007, https://hal.inria.fr/inria-00182338.

References in notes

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

[28] 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.

[29] 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.

[30] 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.

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

[32] 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.

[33] 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.

[34] G. FEUILLADE, T. GENET, V. VIET TRIEM TONG. *Reachability Analysis over Term Rewriting Systems*, in "Journal of Automated Reasoning", vol. 33 (3-4), 2004, p. 341-383, http://www.irisa.fr/lande/genet/publications.html.

[35] 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.

[36] 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.

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

[38] 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.

[39] 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.

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

[41] 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.

[42] 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.

[43] 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.

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

[45] 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.

[46] 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.

[47] P. SOTIN, D. CACHERA, T. JENSEN. *Quantitative Static Analysis over semirings: analysing cache behaviour for Java Card*, in "4th International Workshop on Quantitative Aspects of Programming Languages (QAPL 2006)", Electronic Notes in Theoretical Computer Science, vol. 164, Elsevier, 2006, p. 153-167.

[48] 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.