INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team Lande

Logiciel : ANalyse et DEveloppement

Rennes

THEME SYM

Table of contents

1. Team

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

Administrative assistant
Lydie Letort [Administrative assistant, TR Inria]

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

Université Rennes 1 Personnel
Thomas Genet [Assistant Professor]

ENS Cachan Personnel
David Cachera [Assistant Professor]

PhD students
Stéphane Hong Tuan-Ha [Inria-Region Bretagne]
Gurvan Le Guernic [MENRT grant]
Matthieu Petit [Region Bretagne grant]
Tristan Denmat [MENRT grant]
Luka Leroux [Inria-France Télécom R&D]
Pascal Sotin [BDI CNRS-DGA]

Research engineers
Yann Glouche [Contract with the FNADT]
Laurent Hubert [RNTL Castles]

Post-doctoral researchers
Guillaume Dufay [Post-doc Inria, until June 2006]
Yohan Boichut [Post-doc University Rennes 1, from September 2006]

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

3.1.1. Static program analysis

[26], [31] 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 [23] by programming the analyser inside the assistant and formally proving its correctness. The Coq extraction mechanism then allows for extracting a Caml implementation of the analyser. The feasibility of this approach has been demonstrated in [5].

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 $s\neg \rightarrow_{\mathcal{R}}^{*} t$.

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 $s\neg \rightarrow_{\mathcal{R}}^{*} t$, we already have some results based on the over-approximation of the set of reachable terms [27], [28]. 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 [6] 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 [30] *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. Timbuk: a tree automata library

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

Participant: Thomas Genet.

Timbuk [28] 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:

- boolean operations: intersection, union, complement,
- emptiness decision, inclusion decision,
- cleaning, renaming,
- determinisation,
- transition normalisation,
- 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 also 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) [33], [32]. Pierre-Cyrille Heam, Yohan Boichut and Olga Kouchnarenko of the Cassis Inria project use Timbuk as a verification back-end for verification of protocols [29] defined in high level protocol specification format.

4.3. Protocol Animator

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

Participants: Thomas Genet, Yann Glouche.

AVISPA is now a commonly used verification tool for cryptographic protocols [24]. 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 [25]). 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 debugging HLPSL specifications, we have developed 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. This tool can be run from the local graphical interface. Starting from such an HLPSL specification, the protocol animator helps to build one possible MSC corresponding to that specification. The animator can represent one or more sessions of the protocol in parallel according to the information given in HLPSL specification. Then, MSCs are produced interactively with the user. The protocol animator 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. The MSC can be exported in postscript format or PDF format. A short description of the tool has been published [17].

4.4. Automatic functional test case generation

Keywords: *Constraint Handling Rules*, *Functional testing*, *Jakarta Specification Language*, *automatic test case generation*.

Participants: Sandrine Gouraud, Arnaud Gotlieb [contact point].

JSL2CHR converts a formal model written in JSL in a set of CHRs that can be interpreted to automatically generate test case for testing an implementation. Test generation is performed w.r.t. rule coverage testing objectives.

JSL2CHR is a multi-plateform utilitary that is currently experimentated by Silicomp/AQL and Oberthur Card Systems to generate test cases for testing an implementation of the Java Card Virtual Machine.

5. New Results

5.1. Access control model for interactive devices

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

Participants: Frédéric Besson, Guillaume Dufay, Thomas Jensen.

The Lande groups continues its investigation of access control mechanisms by studying the security policy of mobile devices [13]. 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.

5.2. From certified abstract interpretation to proof-carrying code

Keywords: *Program analysis*, *Proof-Carrying Code*, *constraint solving*, *constructive logic*, *lattices*, *theorem proving*.

Participants: Frédéric Besson, Thomas Jensen, David Pichardie.

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. We have shown [4][14], [8] how certified static analyses (see Section 3.1) can be used to build a PCC architecture where the code producer can produce program certificates automatically. Code consumers use proof checkers derived from certified analysers to check certificates. Proof checkers carry their own correctness proofs and accepting a new proof checker amounts to type checking the checker in Coq. Certificates take the form of strategies for reconstructing a fixpoint and are kept small due to techniques for fixpoint compression [15]. The PCC architecture has been implemented and evaluated experimentally on a byte code language for which we have designed an interval analysis that allows to generate certificates ascertaining that no array-out-of-bounds accesses will occur.

In our work on non-interference (see 5.9) we have followed a similar approach to extract a certified non-interference type checker.

5.3. Certificates checkers for arithmetics invariants

Keywords: *arithmetics invariants*, *certificates*, *reflexive tactics*.

Participant: Frédéric Besson.

When proof obligations fall in decidable logic fragments, users of proof-assistants expect automation. However, despite the availability of decision procedures, automation does not come for free. The reason is that decision procedures do not generate proof terms. We have shown how to design efficient and lightweight reflexive tactics for a hierarchy of quantifier-free fragments of integer arithmetics [12]. The tactics can cope with a wide class of linear and non-linear goals. For each logic fragment, off-the-shelf algorithms generate *certificates* of infeasibility that are then validated by straightforward reflexive checkers proved correct inside the proof-assistant. The approach has been prototyped using the Coq proof-assistant. Preliminary experiments are promising as the tactics run fast and produce small proof terms.

This work has been motivated by our PCC architecture based on certified abstract interpretation (see Section 5.2). The efficient certified checkers defined here are key components of the architecture that will be used to efficiently check polyhedral invariants inferred by abstract interpretation.

In our worl about on-interference (see

5.4. Resource-oriented analysis

Keywords: *Memory usage*, *quantitative program analysis*.

Participants: David Cachera, Stéphane Hong Tuan Ha, Thomas Jensen, Pascal Sotin.

A program's resources usage (time, memory or energy) is a valuable information about that program, even more when these resources are limited, as in the case of embedded software. There exist numerous methods for estimating these consumptions, going from monitoring executions to the exact computation of the complexity of the program, passing by techniques for determining Worst Case Execution Time.

We have proposed a technique inspired by Di Piero and Wicklicky's Quantitative Abstract Interpretation for computing quantitative non-functional properties of a program's behaviour. In contrast to usual static analyses that use structures like cpos and fixpoint computations, these approaches use vector spaces and linear operators as semantic universes.

We have shown that it is possible to define a quantitative semantics, and to abstract its result in order to get and over-approximation of the cost—a measure of the ressource usage—of a program [22]. In particular, we are able to define and compute a notion of so-called long run cost, that represents a maximal average cost during execution.

This approach has been applied to a case study, which aims at estimating the number of cache misses in a computation described by a Java byte code program. The analysis is based on a quantitative semantics for a subset of the Java language for Java Card, where we model the data cache impact on the execution cost. We then give ways to abstract this semantics, to allow for an effective computation of an over-approximation of the real costs.

Finally, we have studied the use of aspect-oriented programming for resource management with the aim of improving resource availability. In this approach, aspects specify time limits or orderings in the allocation of resources and can be seen as the specification of an availability policy. The approach relies on timed automata to specify services and aspects. This allows us to implement weaving as an automata product and to use model-checking tools to verify that aspects enforce the required availability properties [9] (joint work with Pascal Fradet from Inria Rhône-Alpes).

5.5. Trace reconstruction for rewriting approximations

Participants: Yohan Boichut, Thomas Genet.

Term Rewriting Systems are now commonly used as a modeling language for programs or systems. On those rewriting based models, reachability analysis, i.e. proving or disproving that a given term is reachable from a set of input terms, provides an efficient verification technique. For disproving reachability (i.e. proving non reachability of a term) on non terminating and non confluent rewriting models, Knuth- Bendix completion and other usual rewriting techniques do not apply. Using the tree automaton completion technique developed in the Lande project, it has been shown that the non reachability of a term t can be shown by computing an overapproximation of the set of reachable terms and then prove that t is not in the approximation. However, when the term t is in the approximation, nothing can be said. In [16] we refine this approach and propose a method taking advantage of the approximation to compute a rewriting path to the reachable term when it exists, i.e. produce a counter-example. The algorithm has been prototyped in the Timbuk tool. Experiments with this prototype show the interest of such an approach w.r.t. verification of rewriting models.

5.6. Path-oriented random testing

Keywords: *Path testing*, *Random testing*.

Participants: Arnaud Gotlieb, Matthieu Petit.

Test campaigns usually require only a restricted subset of paths in a program to be thoroughly tested. As random testing (RT) offers interesting fault-detection capacities at low cost, we face the problem of building a sequence of random test data that execute only a subset of paths in a program. We address this problem with an original technique based on backward symbolic execution and constraint propagation to generate random test data based on an *uniform* distribution. Our approach derives path conditions and computes an over-approximation of their associated subdomain to find such a uniform sequence. The challenging problem consists in building *efficiently* a path-oriented random test data generator by minimizing the number of rejects within the generated random sequence. Our first experimental results, conducted over a few academic examples, clearly showed a dramatic improvement of our approach over classical random testing [19].

5.7. A semi-empirical model of test quality in symmetric testing

Keywords: *Random Testing*, *Symmetric Testing*, *Testing Java Card APIs*.

Participant: Arnaud Gotlieb.

In the smart card quality assurance field, Software Testing is the privileged way of increasing the confidence level in the implementation correctness. When testing Java Card application programming interfaces (APIs), the tester has to deal with the classical oracle problem, i.e. to find a way to evaluate the correctness of the computed output. We have conducted an experiment in testing methods of the Oberthur Card Systems Cosmo 32 RSA Java Card APIs by using the Symmetric Testing paradigm. This paradigm exploits user-defined symmetry properties of Java methods as test oracles. We propose an experimental environment that combines random testing and symmetry checking for (on-card) cross testing of several API methods. We developed a semi-empirical model (a model fed by experimental data) to help deciding when to stop testing and to assess test quality [18].

5.8. Testing of Java Card software

Keywords: *CHR*, *Java Card*, *structural testing*, *symmetric testing*.

Participant: Arnaud Gotlieb.

Automated functional testing consists in deriving test cases from the specification model of a program to detect faults within an implementation. We have investigated using Constraint Handling Rules (CHRs) to automate the test cases generation process of functional testing. In the context of the CASTLES project, our case study is a formal model of the Java Card Virtual Machine (JCVM) written in a sub-language of the Coq proof assistant. We have defined an automated translation from this formal model into CHRs and have proposed to generate test cases for each bytecode definition of the JCVM [20]. Using CHRs allows to faithfully model the formally specified operational semantics of the JCVM. The approach has been implemented in Eclipse Prolog and a full set of test cases have been generated for testing the JCVM.

5.9. Dynamic and static information flow analysis

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

Participants: Gurvan Le Guernic, Thomas Jensen, David Pichardie.

A standard way of formalising confidentiality is via the notions of information flow and non-interference. In collaboration with A. Banerjee and D. Schmidt at Kansas State University we have developed an information flow monitoring mechanism for sequential programs. The monitor executes a program on standard data that are tagged with labels indicating their security level. The originality of the approach is that we formalize the monitoring mechanism as a big-step operational semantics that integrates a static information flow analysis to gather information flow properties of non-executed branches of the program. This essentially shows how to mix static and dynamic non-interference analysis. Using the information flow monitoring mechanism, it is then possible to partition the set of all executions in two sets. The first one contains executions which *are safe* and the other one contains executions which *may be unsafe*. Based on this information, we show that, by resetting the value of some output variables, it is possible to alter the behavior of executions belonging to the second set in order to ensure the confidentiality of secret data [21].

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 [11] 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; an additional benefit of the formalization is that we have extracted from our proof a certified lightweight bytecode verifier for information flow. Our work provides, to our best knowledge, the first sound and implemented information flow type system for such an expressive fragment of the JVM.

6. Contracts and Grants with Industry

6.1. The CASTLES RNTL project

Keywords: *Java Card*, *testing and certified analysis*.

Participants: Arnaud Gotlieb, Sandrine Gouraud, Thomas Jensen, Gerardo Schneider.

This project aims at defining an automated environment for the certification of a platform Java Card and the applets that are intended to be executed on it. This environment will be based on abstraction tools, automated proofs checkers, static analysis and software testing techniques. The originality of the proposed approach comes from the integration of these distinct tools into a single framework able to deal with the specification step, the development and the validation steps. Parts of this environment already exist and this project will alleviate the two remaining difficulties :

- the formal verification of the Java Card platform requires a major effort due to the absence of automated verification techniques and mechanisms that allow modular and reusable proofs ;
- the applet validation process is based on complex static analysis that must be justified with regards to the certification process.

The choice of Java Card, which is the open smart card reference language, maximizes the industrial benefits and provides a realistic application domain. However, the proposed solutions are not limited to the context of Java Card and could be eventually adapted to other smart card platforms for small secured objects and mobile codes, such as Java or .NET.

The CASTLES project is funded by the French Ministry of Research as part of the RNTL funding scheme. It is a 3 years project and 4 partners are involved : the Inria Everest project from Inria Sophia-Antipolis, the Lande project, Oberthur Card Systems and Alliance Qualité Logicielle. The works to do include :

- Package “SP0: Requirements in certification“ provides a precise analysis of needs in certification of the Java Card platform.
- Package “SP1: Java Card platform Validation” is concerned with techniques and tools for the validation of the Java Card platform will be defined. In particular, formal proof checkers and automatic test data generator will be proposed—see 5.8.
- Package “SP2: Static analysis of security properties”. This package targets the certification of Java Card applets. It is based on the definition of formal security analysis for confidentiality and disponibility—for results see 5.4.

6.2. The RNTL CAT project

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

Participants: Arnaud Gotlieb, Thomas Jensen.

The RNTL CAT project aims at developping techniques and tools for analysing critical C programs. In this project, we concentrate on developping dynamic analyses for C programs that contain pointer-based computations and dynamically allocated structures. The work is based on our previous results on analysing and testing C programs that manipulate dynamic memory structures [10]. The other members of the project are the CEA LIST laboratory (project leader), Proval (Inria Futurs), Lande (Inria Rennes), France Télécom R&D, Dassault-Aviation, Siemens VDO and Airbus Industries.

6.3. Static analysis of Java Midlets with rewriting and reachability analysis

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

Participants: Thomas Jensen, Thomas Genet, Luka Leroux.

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.

6.4. 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, Luka Leroux.

The Lande project has initiated 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.

6.5. European FET-Integrated project MOBIUS

Keywords: *Java*, *Proof Carrying Code*, *Security*, *smart phones*, *static analysis*.

Participants: Thomas Jensen, Frédéric Besson, Tiphaine Turpin, Guillaume Dufay.

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 also develop resource-aware static analyses to enforce the proposed resource policies, see 5.1.

7. Other Grants and Activities

7.1. The ACI Sécurité Informatique project DISPO

Keywords: *Availability*, *aspects*, *software components*.

Participants: Thomas Jensen, Stéphane Hong Tuan-Ha.

The DISPO project, coordinated by Lande, is concerned with specifying, verifying and enforcing security policies governing the *availability* of services offered by software components. Contrary to the other two kinds of security properties (integrity and confidentiality), there are only few attempts on formalising availability policies. Furthermore, the results obtained for availability has so far not been connected with the software engineering process that manufactures the components. The aim of the project is therefore to develop suitable specification formalisms together with formal methods for program verification and transformation techniques taking advantage of modern program structuring techniques such as component-based and aspect-oriented software development.

The project is composed of three sub-activities:

- Developing a formalism for specifying availability properties. This formalism is based on temporal and deontic logics. We are paying particular attention to the problem of verifying the *coherence* of policies specified in this formalism.
- We use a combination of static and dynamic techniques for enforcing a particular availability property on a particular software component. In the purely static view, properties are enforced by a combination of static program analysis and model checking, in which the code of a component is abstracted into a finite or finitary model on which behavioral properties can be model-checked. When such verifications fail (either because the property does not hold or because the analysis is incapable of proving it), we will employ the technique of aspect-oriented programming to transform the program so as to satisfy a given property.
- At the architectural level, the project aims at developing a component model equipped with a notion of availability interface, describing not only static but also dynamic properties of the component. The challenge here is to define suitable composition operators that allow to reason at the level of interfaces. More speculatively, we intend to investigate how the notion of aspects apply at the architecture level, and in particular how to specify aspects of components in such a way that properties can be enforced at component assembly time.

The project consortium consists of four partners: École des Mines de Nantes (the OBASCO project), Irisa (Rennes), IRIT (Toulouse) and ENST-Bretagne.

7.2. The ACI Sécurité Informatique project V3F

Keywords: *Validation*, *Verification*, *constraint solving*, *floating-point numbers computations*.

Participant: Arnaud Gotlieb.

Computations with floating-point numbers are a major source of failures of critical software systems. It is well known that the result of the evaluation of an arithmetic expression over the floats may be very different from the exact value of this expression over the real numbers. Formal specifications languages, model-checking techniques, and testing are currently the main techniques used to improve the reliability of critical systems. A significant effort over the past year was directed towards development and optimization of these techniques, but few works has been done to tackle applications with floating-point numbers. A correct handling of a floating point representation of the real numbers is very difficult because of the extremely poor mathematical properties of floating-point numbers; moreover the results of computations with floating-point numbers may depends on the hardware, even if the processor complies with the IEEE 754 norm.

The goal of this project is to provide tools to support the verification and validation process of programs with floating-point numbers. More precisely, project V3F will investigate techniques to check that a program satisfies the calculations hypothesis on the real numbers that have been done during the modelling step. The underlying technology will be based on constraint programming. Constraints solving techniques have been successfully used during the last years for automatic test data generation, model-checking and static analysis. However in all these applications, the domains of the constraints were restricted either to finite subsets of the integers, rational numbers or intervals of real numbers. Hence, the investigation of solving techniques for constraint systems over floating-point numbers is an essential issue for handling problems over the floats.

So, the expected results of project V3F are a clean design of constraint solving techniques over floating-point number, and a deep study of the capabilities of these techniques in the software validation and verification process. More precisely, we will develop an open and generic prototype of a constraint solver over the floats. We will also pay a special attention on the integration of floats into various formal notations (e.g., B, Lustre, UML/OCL) to allow an effective use of the constraint solver in formal model verification, automatic test data generation (functional and structural) and static analysis.

The V3F project is funded by the French National Science Fund. It is a 3 years project and 4 partners are involved : the Inria Cassis project from LORIA, the Inria Coprin project from RU of Sophia-Antipolis, the Inria Lande and Vertecs projects from Irisa and the LSL group of CEA.

7.3. The ACI Sécurité Informatique project SATIN

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

Participant: 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.

8. Dissemination

8.1. Conferences: program committees, organization, invitations

Thomas Jensen was invited to give a talk on “Certificates for resource usage on mobile telephones” at the 2nd IEEE International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (Isola 2006), Cyprus November 2006.

8.2. PhD theses defended

There were no PhD theses defended in the project in 2006.

8.3. PhD and Habilitation committees

Thomas Jensen was president of the PhD thesis jury of Yohan Boichut at the Univeristy of Franche-Comté at Besançon and Thomas Chatain at the University of Rennes. He was external examiner on the PhD thesis of S. Devane at IIT Bombay and on the thesis of G. Bobeff at the Ecole des Mines de Nantes.

8.4. Teaching: university courses and summer schools

David Cachera teaches theoretical computer science at École Normale Supérieure de Cachan. He also teaches formal methods and proof assistants to M2 students.

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 Yves Le Traon (TRISKELL project). He also teaches at the 5th year of Insa Rennes in collaboration with Mireille Ducassé.

Thomas Jensen teaches semantics, type systems and static analysis at Master 2 level in collaboration with Bertrand Jeannet (VERTECS project).

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 the Mamouon Filali at IRIT and took place at Luchon and Toulouse. The school attracted 40 participants for two weeks during June.

8.5. Administrative responsibilities

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

Thomas Jensen was chairman of the hiring committee for junior researchers at Inria Rennes.

9. 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. L. MÉTAYER, T. THORN. *Model ckecking security properties of control flow graphs*, in "Journal of Computer Security", vol. 9, 2001, p. 217–250.

[4] 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.

[5] 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.

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

[7] 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.

Year Publications

Articles in refereed journals and book chapters

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

[9] P. FRADET, STÉPHANE. HONG TUAN HA. *Systèmes de gestion de ressources et aspects de disponibilité*, in "L'objet", vol. 12, n^{o} 2-3, 2006, p. 183–210.

[10] A. GOTLIEB, T. DENMAT, B. BOTELLA. *Goal-oriented test data generation for pointer programs*, in "Information and Software Technology", Accepted for publication - Jul. 2006, 2006.

Publications in Conferences and Workshops

[11] 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, to appear, Springer-Verlag, 2007.

[12] F. BESSON. *Fast Reflexive Arithmetic Tactics the linear case and beyond*, in "Types for Proofs and Programs (TYPES'06)", Lecture Notes in Computer Science, to appear, Springer-Verlag, 2006.

[13] 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)", Springer LNCS vol. 4189, 2006.

[14] F. BESSON, T. JENSEN, D. PICHARDIE. *A PCC Architecture based on Certified Abstract Interpretation*, in "Proc. of 1st International Workshop on Emerging Applications of Abstract Interpretation (EAAI'06)", ENTCS, Springer-Verlag, 2006.

[15] F. BESSON, T. JENSEN, T. TURPIN. *Small witnesses for abstract interpretation based proofs*, in "Proc. of 16th European Symposium on Programming (ESOP'07)", Lecture Notes in Computer Science, to appear, Springer-Verlag, 2007.

[16] Y. BOICHUT, T. GENET. *Feasible Trace Reconstruction for Rewriting Approximations.*, in "Rewriting Techniques and Applications (RTA'06)", Springer LNCS vol. 4098, 2006, p. 123-135.

[17] Y. GLOUCHE, T. GENET, O. HEEN, O. COURTAY. *A Security Protocol Animator Tool for AVISPA*, in "ARTIST-2 workshop on security of embedded systems, Pisa (Italy)", 2006.

[18] A. GOTLIEB, P. BERNARD. *A Semi-empirical Model of Test Quality in Symmetric Testing: Application to Testing Java Card APIs*, in "Sixth International Conference on Quality Software (QSIC'06), Beijing, China", Oct. 2006.

[19] A. GOTLIEB, M. PETIT. *Path-oriented Random Testing*, in "Proceedings of the 1st ACM Int. Workshop on Random Testing (RT'06), Portland, Maine", July 2006.

[20] S. GOURAUD, A. GOTLIEB. *Using CHRs to generate test cases for the JCVM*, in "Eighth International Symposium on Practical Aspects of Declarative Languages, PADL 06, Charleston, South Carolina", Lecture Notes in Computer Science, vol. 3819, January 2006.

[21] G. LE GUERNIC, A. BANERJEE, T. JENSEN, D. SCHMIDT. *Automaton-based Confidentiality Monitoring*, in "11th Annual Asian Computing Science Conference (ASIAN'06), Tokyo, Japan", Lecture Notes in Computer Science, to appear, December 2006.

[22] 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.

References in notes

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

[24] A. ARMANDO, ET AL. *The AVISPA Tool for the automated validation of internet security protocols and applications*, in "17th CAV Conf., Scotland", K. ETESSAMI, S. RAJAMANI (editors). , LNCS, 3576, vol. 3576, Springer, 2005, p. 281-285.

[25] Y. CHEVALIER, ET AL. *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.

[26] 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.

[27] 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.

[28] 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.

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

[30] 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.

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

[32] 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.

[33] 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.