INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team EVEREST

Vérification et sécurité du logiciel

Sophia Antipolis - Méditerranée

THEME SYM

Table of contents

1. Team

Research Scientist

Gilles Barthe [Research Director INRIA. Since April 2008, research scientist at the Madrid Institute for Advanced Studies in Software Development Technologies (IMDEA software) now, HdR]
Marieke Huisman [Research scientist INRIA until August 2008, research scientist at Twente University now]
Benjamin Grégoire [Research scientist INRIA]
Tamara Rezk [Research scientist INRIA, since September 2007]

Technical Staff

Sophie Hadjadj [until November 2007]
Anne Pacalet
Colin Riba [until August 2008]

PhD Student

Julien Charles [MESR grant, Teaching Assistant UNSA]
Clément Hurlin
César Kunz [until October 2008]
Jorge-Luis Sacchini [since November 2007]
Santiago Zanella [Microsoft grant]
Sylvain Heraud [since september 2008]

Visiting Scientist

Salvador Cavadini [visiting PhD student from University of San Luis, Argentina, until February 2008]
Gustavo Petri [visiting PhD student from University of San Luis, Argentina, until August 2008]

Administrative Assistant

Nathalie Bellesso
Claire Seneca [replacement until June 2008]

2. Overall Objectives

2.1. Overall Objectives

The Everest project concentrates on increasing reliability and security of mobile and embedded software. This is achieved by developing and applying formal methods and language-based techniques, covering both platform and application level. The project's privileged application domain ranges from trusted personal devices, such as mobile phones and smart cards, to ubiquitous computing.

The project focuses on the following research areas:

- Program verification and Proof Carrying Code;
- Machine-checked semantics;
- Machine-checked provable cryptography;
- Foundations of proof assistants.

3. Scientific Foundations

3.1. Type systems

Types are often considered as one of the great successes of programming language theory, and their use permeates modern programming languages. The widespread use of types is a consequence of three crucial factors:

- *Types are intuitive*. Types are a particularly simple form of assertion. They can be explained to the user without the need to understand precise details about why and how they are used in order to achieve certain effects. For example, Fortran and C use type systems to generate memory layout directives at compile time; yet the users of C can write type-correct programs and understand typing errors without knowing exactly how the type concept is related to memory layout.
- *Types are automatic*. In many cases an untyped program can be enhanced with types automatically by type inference. Of course, adherence of a program to even the simplest policy is an algorithmically undecidable property. Type systems circumvent this obstacle by guaranteeing a safe over-approximation of the desired policy. For example, a branching statement with one unsafe branch will usually be considered unsafe by a type system. This not only restores decidability but contributes to the aforementioned intuitiveness and simplicity of type systems.
- *Types scale up*. Besides their simplicity and the possibility to infer types, type systems allow to reduce the verification of a complex system into simpler verification tasks involving smaller parts of the system. Such a compositional approach is a crucial property for making the verification of large, distributed and reconfigurable systems feasible.

Type systems have long been used in programming languages to enforce basic safety properties of programs. For example, the type system of Java is designed to statically detect certain run-time errors such as the application of a string function to a floating point number, or a call to a method that does not belong to a name space of a given class. In addition, the research community has developed numerous type systems that enforce more advanced safety properties dealing with modularity, concurrency, and aliasing in Java programs.

Type systems are also increasingly being studied as a means to enforce security; in particular, the research community has developed type systems that guarantee secure information flow and resource control policies.

3.2. Program verification

Research on program logics has a long history, dating back to the seminal work on Floyd-Hoare logics and weakest precondition calculi in the late 1960s and early 1970s. Although this line of research has not yet lead to a breakthrough in the application of program verification, there has been steady progress, resulting in tool-supported program logics for realistic programming languages.

There are a number of reasons to adopt program verification techniques based on logic to guarantee the correctness of programs.

- *Logic is expressive*. During its long development, logic has been designed to allow for greater and greater expressiveness, a trend pushed by philosophers and mathematicians. This trend continues with computer scientists developing still more expressiveness in logic to encompass notions of resources and locality. Today a rich collection of well developed and expressive logics exists for describing computational systems.
- *Logic is precise*. While types generally over-approximate program behaviour, logic can be used to provide precise statements about program behaviour. Special conditions can be assumed and exceptional behaviours can be described. Via the use of negation and rich quantifier alternation, it is possible to state nearly arbitrary observations about programs and computational systems.
- *Logic allows analyses to be combined*. Logic provides a common setting into which the declarative content of a typing judgement or other static analyses can be translated. The results of such analyses can then be placed into a common logic so that conclusions about their combinations can be drawn.

Recently, there has been a lot of research into logic-based program verification of Java, which has culminated in the realisation of program verification environments for single-threaded Java.

3.3. Machine-checked semantics and algorithms

We are interested in developing formal, machine-checked semantics of programming languages and of their execution platforms such as virtual machines, run-time environments, Application Programming Interfaces, and of the tools that are used for compiling, verifying, validating programs. In particular, we have strong experience in modelling and verifying execution platforms for smart cards, as well as their main security functions, such as bytecode verifiers and access control mechanisms, and the standard deployment architectures for multi-application smart cards, such as Global Platform.

We are also interested in developing formal, machine-checked security proofs for cryptographic algorithms, using tools from provable cryptography. In particular, in the past we have formalised the Generic Model and Random Oracle Model, and formally verified security bounds for the probability of an attacker breaking the discrete logarithm and related encryption and signing schemes. We now focus on formal proofs for the computational model, which has the advantage that it makes less hypotheses on the attacker. Proofs in the computational model are based on game-playing techniques.

3.4. Software security

Security is not a technology, but a property of a system. For this reason, there are new security requirements associated with each new technology or architecture. While these security requirements are often expressed from the perspective of the users, they must also be translated to concrete objectives that can be enforced by security mechanisms.

For instance, the Java security model assumes that end users trust its run-time environment but not the downloaded code. The concrete objectives include adherence to the typing and policy of the JVM and compliance with the stack inspection mechanism, which are enforced by the Java byte code verifier and the run-time environment.

The ability to derive concrete verification objectives from carefully gathered security requirements is an important step for guaranteeing that a component is secure with respect to a given security policy. Typical requirements include information flow security policies; resource control policies; framework-specific security; and application-specific security.

We give precise mathematical definitions of these requirements, using programming language semantics, and provide means to enforce these requirements using type systems or logic.

3.5. Proof Carrying Code

Proof Carrying Code (PCC) is an innovative security framework in which components come equipped with a certificate which can be used by devices (code consumers in PCC terminology) to verify locally and statically that downloaded components issued by an untrusted third party (code producers in PCC terminology) are correct. In order to realise this view, standard PCC infrastructures build upon several elements: a logic, a verification condition generator, a formal representation of proofs, and a proof checker.

- *A formal logic for specifying and verifying policies*. The specification language is used to express requirements on the incoming component, and the logic is used to verify that the component meets the expected requirements. Standard PCC adopts first-order predicate logic as a formalism to both specify and verify the correctness of components, and focuses on safety properties. Thus, requirements are expressed as pre- and post-conditions stating, respectively, properties to be satisfied by the state before and after a given procedure or function is invoked.

- *A verification condition generator (VCGen)*. The VCGen produces, for each component and safety policy, a set of proof obligations whose provability will be sufficient to ensure that the component respects the safety policy. Standard PCC adopts a VCGen based on programming verification techniques such as Hoare-Floyd logics and weakest precondition calculi, and it requires that components come equipped with extra annotations, *e.g.*, loop invariants that make the generation of verification conditions feasible.
- *A formal representation of proofs (Certificates)*. Certificates provide a formal representation of proofs, and are used to convey to the code consumer easy-to-verify evidence that the code it receives is secure. In Standard PCC, certificates are terms of the lambda calculus, as suggested by the Curry-Howard isomorphism, and routinely used in modern proof assistants such as Coq.
- *A proof checker that validates certificates against specifications*. The objective of a proof checker is to verify that the certificate does indeed establish the proof obligations generated by the VCGen. In Standard PCC, proof checking is reduced to type checking by virtue of the Curry-Howard isomorphism. One very attractive aspect of this approach is that the proof checker, which forms part of the Trusted Computing Base is particularly simple.

We study other variants of Proof Carrying Code that cover a wide range of security properties that escape the scope of certifying compilers, and that need to be established interactively on source code programs.

4. Application Domains

4.1. Smart devices

Smart devices, including new generation smart cards and trusted personal devices typically contain a microprocessor and a memory chip (but with limited computing and storage capabilities). They are often used by commercial and governmental organisations and are expected to play a key role in enforcing trust and confidence in e-payment and e-government. Current applications include bankcards, e-purses, SIM cards in mobile phones, e-IDs, *etc*,

These devices provide solutions for application developers by enabling them to program in high-level languages (several dialects of Java exist for this purpose: Java Card, MIDP, *etc.*), on a common software base (a virtual machine and Application Programming Interfaces, APIs), which isolates their code from specific hardware and operating system libraries. These devices support both the flexibility and the evolution of applications by enabling downloading of executable content onto already deployed devices (so-called post issuance), and by allowing several commercially independent applications to run on a single device. This open character forms their commercial strength, but also creates a technical challenge: reliability, correctness and security become crucial issues, since malicious applications might potentially exploit bugs in the smart device platform, with detrimental effects on security and/or privacy.

4.2. Global computing

A global computer is a *distributed* computational infrastructure that aims at providing a global and uniform access to services. However, global computers consist of large networks of *heterogeneous* devices that differ greatly in their computational infrastructure and in the resources they offer to services. In order to deliver services globally and uniformly, each device in a global computer must therefore be *extensible* with the computational infrastructure, platform or libraries, needed to execute the required services. In that respect, global computers transcend the scope of established computational models such as mobile code, the Grid, or agents, which impose a clear separation between mobile applications and the fixed computational infrastructure upon which they execute.

While global computers may deeply affect our quality of life, security is paramount for them to become pervasive infrastructures in our society, as envisioned in ambient intelligence. Indeed, numerous application domains, including e-government or e-health, involve sensitive data that must be protected from unauthorised parties. In spite of clear risks, provisions to enforce security in global computers remain extremely primitive. Some global computers, for instance in the automotive industry, choose to enforce security by maintaining devices completely under the control of the operator. Other models, building upon the Java security architecture, choose to enforce security via a sandbox model that distinguishes between a fixed trusted computing base and untrusted applications. Unfortunately, these approaches do not embrace the complexity of global computers.

5. New Results

5.1. Program verification and Proof Carrying Code

Participants: Gilles Barthe, Benjamin Grégoire, Marieke Huisman, Julien Charles, Sylvain Heraud, Clément Hurlin, Anne Pacalet.

- In collaboration with D. Gurov and I. Aktug from KTH, Sweden, we have continued our work on adapting our compositional verification techniques [19] to richer program models. In particular, we have reviving tool set, and extending it for exceptions and multithreading, as described [12].
- In earlier work, an algorithm to generate annotations for high-level security properties has been implemented in Jack [20]. In collaboration with A. Tamalet from the University of Nijmegen, Netherlands, we are working on a formalisation of this algorithm, in order to prove the approach correct formally. Given a property described by a finite state machine (FSM) and a Java application, we have defined how the FSM can be captured by appropriate JML (Java Modeling Language) set annotations. For a basic program format, we have defined a general semantics that can be instantiated in different ways, to specify a run-time annotation checking or a monitoring semantics. Further, we have defined an algorithm that transforms a program monitored by an automaton into an annotated program. We prove that the generated annotations correctly capture the monitoring, *i.e.*, if the monitored program reaches an error-state, then one of the generated assertions will be violated. The formal correctness proof is being developed in a theorem prover. As a last case, we still have to find suitable restrictions for try-catch-finally statements, under which the correctness result holds. A next step is to propagate the annotations, and to show that eventually correctness of the annotations can be proven statically.
- The Mobius specific task force, in which the Everest team is one of the active partners, has continued work on the BML Reference Manual. BML, short for Bytecode Modeling Language, is the bytecode cousin of JML [18]. The reference manual specifies the syntax and semantics of the language, the class file format that is used to store the annotations, and it defines the compilation from JML to BML. The reference manual serves as a basis for the tool development of the bytecode subcomponent of the Mobius tool set. A preliminary version of the reference manual is available via http://www-sop.inria.fr/everest/BML.
- Frama-C (http://www.frama-c.cea.fr) is a suite of tools dedicated to the analysis of the source code of software written in C. In collaboration with the CEA, we developed some plugins for that framework : mainly a slicer, but also a PDG (Program Dependences Graph) generator, a sparecode analyzer, data scope visualisation... In 2008, Frama-C first release has been distributed and some real users began to use it : the plugins have been adapted to better fit their needs, and to be more robust to handle real programs. The main improvements are about adding precision and more genericity to the PDG tool to be able to use it for impact analysis, and being able to slice a program starting from some properties. The cases where we have to lose precision because we don't have enough information are also better handled.

- In collaboration with C. Haack from the University of Nijmegen, we have lifted previous work on object-oriented separation logic to multithreaded object-oriented programs. Our extension includes new proof rules to deal with Java's style of concurrency [11], i.e. fork and join, whereas the literature previously focused on a classical parallel operator. We studied how abstraction facilities provided by the literature could be adapted to this new setting. Further, we extended our work to deal with Java's reentrant locks [9]. Finally, we have applied our system to various flavors of Java's Iterator [10].
- Sylvain Heraud did an internship on the development of an optimizing compiler that transforms certificates from an RTL (Register Transfer Language) program into certificates of an optimized RTL program. He developed four optimizations together with their corresponding certificate transformations. This work lead to a reduction in the size of the generated certificates while at the same time making the transformations simpler and more independent from the compiler than in the previous work of Cesar Kunz.
- Sequoia is a language designed to program parallel divide-and-conquer programs over a hierarchical, tree-structured, and explicitly managed memory. We developed a framework to reason about Sequoia programs using abstract interpretation techniques. Also, we adapted previous work on Certificate Translation to this setting, showing that common program optimizations transform provably correct Sequoia programs into provably correct Sequoia programs.
- We continue to work on the Mobius DirectVCGen as well as Bico a component used in the aforementioned tool. The DirectVCGen is a tool that generates verification conditions from annotated Java programs source code and bytecode, and then gives a proof of their equivalence. The tool Bico is used as the background predicates generator for the DirectVCGen. It has been enhanced to manage modular aspects properly. Julien Charles wrote a paper with Joseph Kiniry of UCD Dublin entitled A Lightweight Theorem Prover Interface for Eclipse which was presented at UITP08.

5.2. Machine-checked provable cryptography

Participants: Gilles Barthe, Benjamin Grégoire, Santiago Zanella, Federico Olmedo, Sylvain Heraud.

We have continued the development of our framework built on top of the Coq proof assistant for mechanized reasoning about probabilistic programs. We have worked on the foundations of the framework, and in its application to the formalization of security proofs of a handful of cryptographic systems. In particular, we have formalised the proof of unforgeability of the Full Domain Hash digital signature scheme, and the semantic security of the OAEP padding scheme and the semantics security of Hash ElGamal in the random oracle model and in the standard model.

5.3. Type theory and proof assistants

Participants: Gilles Barthe, Benjamin Grégoire, Jorge-Luis Sacchini, Colin Riba.

- On another line of work, we studied the pattern-matching mechanism of Coq. The benefits of pattern matching in dependently typed settings have been known for years, and have been exploited in programming languages such as Agda and Epigram. We adapted these techniques to Coq. In particular, we propose a new elimination rule that allows the elimination of impossible cases and propagation of inversion constraints. We studied the practical and theoretical consequences of the extended system.
- We also continue our work on type base-termination, we have developed some new result which allows more flexibility in the type annotations. We have proved the strong normalisation of a restricted version of our system CCI-sombrero, and started the proof of the full system.

6. Contracts and Grants with Industry

6.1. Contracts

CEA The CEA (Commissariat à l'Energie Atomique) develops static analysis tools, Frama-C, for C programs based on techniques such as precondition computation using Hoare logic and abstract interpretation. The collaboration (2006-2009) aims to enhance these tools by adding slicing capabilities in order to help managing bigger applications. Building smaller models is especially useful to study applications that can not be fully handled by other analyses. Anne Pacalet and Salvador Cavadini are involved in this collaboration.

INRIA-Microsoft Research Joint Laboratory (Secure Distributed Computations and their Proofs, 2006-2009). Other participants are the Programming Principles and Tools group at Microsoft Research Cambridge and the Moscova team (Rocquencourt). This project intends to design formal tools for programming distributed computations with effective security guarantees. Gilles Barthe, Benjamin Grégoire, Tamara Rezk, and Santiago Zanella are involved in the project, see http://www.msr-inria.inria.fr/.

6.2. National projects

ParSec (Parallelism and Security, 2007 - 2010). Other participants are Mimosa (INRIA Sophia-Antipolis), Moscova (INRIA Rocquencourt), PPS (Paris 7), Lande (INRIA Rennes), see http://moscova.inria.fr/~zappa/projects/parsec/consortium.html.

Scalp (Security of Cryptographic Algorithms with Probabilities, 2007 - 2011). Other participants are Verimag (Grenoble), Plume (ENS Lyon), LRI (Paris 11) and Cedric (CNAM), see http://scalp.gforge.inria.fr/.

6.3. European projects

Mobius Gilles Barthe is the scientific coordinator of the European Integrated Project Mobius (Mobility, Ubiquity and Security), launched under the FET Global Computing Proactive Initiative in the 6th Framework program (2005-2009). The project gathers 12 academic and 4 industrial partners. The goal of this project is to develop the technology for establishing trust and security for the next generation of global computers, using the Proof Carrying Code paradigm. The essential features of the Mobius security architecture will be innovative trust management, static enforcement mechanisms and support for system component downloading, see http://mobius.inria.fr.

Thematic Networks The team participates in the networks **Types** (type theory), (2004-2008), see http://www.cs.chalmers.se/Cs/Research/Logic/Types/ and **Appsem 2** (Applied Semantics, concluded), see http://www.tcs.informatik.uni-muenchen.de/~mhofmann/appsem2/.

AlFA LERNET The team participates in the AlFA LERNET "LER-Language Engineering and Rigorous Software Development" project, a grant contract funded by the European Commission, started in March 2005 until December 2008, in which European and South American universities and research centers participate.

6.4. International projects and collaborations

STIC AmSud, Reseco project: Gilles Barthe is coordinator of the Reseco project within STIC AmSud, a regional program of scientific cooperation between France, Argentina, Brazil, Chile, Peru and Uruguay. The objective of the project is to investigate reliability and security in a computational model where both the platform and applications are dynamic, so that incoming software, built from off-the-shelf components, may be destined to form part of the platform or to execute as a standard application. The partners of the Reseco project are Oasis (INRIA-Sophia Antipolis), Chile University and Diego Portales University in Chile, Republic University in Montevideo, Uruguay and National University of Cordoba (FaMAF) in Argentina.

7. Dissemination

7.1. Conference and workshop attendance, travel

- Tamara Rezk presented her work at the POPL conference in San Francisco, January 2008.
- Gilles Barthe participated at a FET Info Day on overlay computing and communication in Brussels, February 2008.
- Gilles Barthe, Benjamin Grégoire, Colin Riba and Jorge Luis Sacchini attended a Types topical workshop on Dependently Typed Programming in Nottingham, February 2008.
- Gilles Barthe participated at Types, Logics and Semantics for State seminar in Dagstuhl, February 2008.
- Gilles Barthe and Tamara Rezk participated to the LERNET project third meeting in Piriapolis, February 2008.
- Gustavo Petri attended the Lernet Summer School on Language Engineering and Rigourous Software Development, February 2008.
- Colin Riba attended to the Russell workshop on Proof Theory meets Type Theory in Swansea, March 2008.
- Jorge Luis Sacchini participated to the Types conference in Torino, March 2008.
- Gilles Barthe presented his work at the ETAPS conference in Budapest, March 2008.
- Cesar Kunz presented his work at ESOP'08 in the ETAPS conference in Budapest, March 2008.
- Cesar Kunz presented his work at FOAL wokshop in Brussels, April 2008.
- Gustavo Petri presented his work workshop in the ETAPS conference in Budapest, March 2008.
- Gilles Barthe, Benjamin Grégoire, Julien Charles, Clément Hurlin, Jorge-Luis Sacchini and Martin Ochoa participated to the Mobius annual meeting in Munich, June 2008.
- Clément Hurlin presented his work at the ParSec meeting in Rennes, June 2008.
- Clément Hurlin presented his work at the ECOOP'08 conference in the IWACO workshop, in Paphos, July 2008.
- Clément Hurlin attended the Marktoberdorf summer school, August 2008.
- Clément Hurlin presented his work to the SERP conference in Las Vegas, July 2008.
- Clément Hurlin presented his work in an informal workshop in Nijmegen, November 2008.
- Clément Hurlin attented the FIRST fall school in Copenhagen, October 2008.
- Clément Hurlin attended APLAS'08 to present his work in Bangalore, December 2008.
- Santiago Zanella and Jorge-Luis Sacchini participated to the Summer School on Logic and Theorem Proving in Programming Languages in Eugene (Oregon), July 2008.
- Benjamin Grégoire, Santiago Zanella and Sylvain Heraud visited the Madrid Institute for Advanced Studies in Software Development technologies (IMDEA Software), November 2008.
- Colin Riba presented his work at the CIE Conference in Athens, June 2008.
- Colin Riba presented his work at the EACSL Conference in Bertinoro, September 2008.
- Charles Julien presented his work at the TPHOL Conference and the UITP Workshop in Montreal, August 2008.
- Tamara Rezk presented her work at the Instituto Superior Ténico of Lisbon, October 2008.
- Tamara Rezk participated to a STIC AmSud meeting in Lima and a RESECO seminar in Santiago del Chile, November 2008.

- Cément Hurlin started his visit of the Formal Method Group in the University of Twente, September 2008.
- Benjamin Grégoire was invited to present the work of the team on formal verification of cryptography in the 4-th Franco-Japanese Computer Security Workshop in Tokyo, December 2008.

7.2. Leadership within scientific community

- Gilles Barthe is scientific coordinator of:
  - the FET Integrated Project Mobius;
  - the Stic-Amsud project *Reseco*.
- Benjamin Grégoire and Marieke Huisman are task leaders for tasks in the Mobius project.
- Gilles Barthe is a member of the editorial board of the Journal of Automated Reasoning.
- Gilles Barthe was a member of the program committees for TGC 2007 (co-chair), FMSE 2007, FM 2008.
- Marieke Huisman was chair of the program committees FTfJP 2008.
- Benjamin Grégoire organized a meeting with the partners of the SCALP ANR, Sophia Antipolis, April 2007.

7.3. Supervision of Ph.D. projects

- Anne Pacalet is the local supervisor of Salvador Cavadini.
- Gilles Barthe supervises the Ph.D. project of Santiago Zanella (started June 2006).
- Benjamin Grégoire supervises the Ph.D. project of Julien Charles (started September 2005).
- Gilles Barthe and Benjamin Grégoire supervise the Ph.D. project of César Kunz (started October 2005).
- Marieke Huisman supervises the Ph.D. project of Clément Hurlin (started September 2006).
- Benjamin Grégoire and Gilles Barthe supervise the Ph.D. project of Jorge-Luis Sacchini (started November 2007).
- Benjamin Grégoire and Yves Bertot supervise the Ph.D. project of Sylvain Heraud (started September 2008).

7.4. Ph.D. committees

Gilles Barthe was a member of the PhD jury of Daniel Hedin (Goteborg University, Sweden), and of the PhD jury of Yann Regis-Gianas (Paris) and Colin Riba (Nancy).

7.5. Supervision of internships

- Tamara Rezk supervised Damian Nadales.
- Anne Pacalet supervised Sylvain Heraud.
- Tamara Rezk supervised Renato Cherini.
- Gilles Barthe and Benjamin Grégoire co-supervised Federico Olmedo.
- Tamara Rezk supervised Zhengqin Luo.
- Benjamin Grégoire supervised Martin Ochoa.
- Benjamin Grégoire supervised Sylvain Heraud.

7.6. Teaching

Julien Charles taught Preparation for C courses (3rd year, 15 hours) and C courses (3rd year, 45 hours) at Polytechélec, Scheme (2nd year, 16 hours and 3rd year, 8 hours) at Polytec'elec, and RMI (3rd year, 37 hours 45) at IUT.

Benjamin Grégoire and Tamara Rezk taught “Vérification et Sécurtité” in Master 2 at Ecole polytechnique de l’université Nice Sophia-Antipolis.

8. Bibliography

Year Publications

Articles in International Peer-Reviewed Journal

[1] G. BARTHE, B. GRÉGOIRE, C. KUNZ, T. REZK. *Certificate Translation for Optimizing Compilers*, in "ACM Transactions on Programming Languages and Systems", 2008.

[2] D. GUROV, M. HUISMAN, C. SPRENGER. *Compositional Verification of Sequential Programs with Procedures*, in "Information and Computation", team, vol. 206, 2008, p. 840–868.

International Peer-Reviewed Conference/Proceedings

[3] G. BARTHE, B. GRÉGOIRE, C. RIBA. *Type-Based Termination with Sized Products*, in "17th EACSL Annual Conference on Computer Science Logic, 15th-19th September 2008, Bertinoro, Italy", Lecture Notes in Computer Science, Springer, 2008.

[4] G. BARTHE, B. GRÉGOIRE, M. PAVLOVA. *Preservation of Proof Obligations from Java to the Java Virtual Machine*, in "Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings", Lecture Notes in Computer Science, Springer, 2008, p. 83-99.

[5] G. BARTHE, C. KUNZ, D. PICHARDIE, J. SAMBORSKI-FORLESE. *Preservation of proof obligations for hybrid verification methods*, in "SEFM", PUB-IEEE, 2008.

[6] G. BARTHE, C. KUNZ, J. SACCHINI. *Certified Reasoning in Memory Hierarchies*, in "APLAS", G. RAMALINGAM (editor), LNCS, PUB-SV, 2008.

[7] J. O. BLECH, B. GRÉGOIRE. *Certifying Code Generation with Coq*, in "Proceedinfs of the Workshop Compiler Optimization meets Compiler Verification (COCV 2008)", 2008.

[8] D. GUROV, M. HUISMAN, C. SPRENGER. *An Algorithmic Approach to Compositional Verification of Sequential Programs with Procedures: An Overview*, in "Foundations of Interface Technologies (FIT 2008)", 2008.

[9] C. HAACK, M. HUISMAN, C. HURLIN. *Reasoning about Java’s Reentrant Locks*, in "Asian Symposium on Programming Languages and Systems (APLAS), Bangalore, India", Lecture Notes in Computer Science, Springer-Verlag, December 2008.

[10] C. HAACK, C. HURLIN. *Resource Usage Protocols for Iterators*, in "International Workshop on Aliasing, Confinement and Ownership in Object-Oriented Programming (IWACO), Paphos, Cyprus", July 2008.

[11] C. HAACK, C. HURLIN. *Separation Logic Contracts for a Java-like Language with Fork/Join*, in "International Conference on Algebraic Methodology and Software Technology (AMAST), Urbana, Illinois, USA", J. MESEGUER, G. ROSU (editors), Lecture Notes in Computer Science, n^{o} 5140, Springer-Verlag, July 2008, p. 199–215, http://hal.inria.fr/inria-00218114/en/.

[12] M. HUISMAN, I. AKTUG, D. GUROV. *Program Models for Compositional Verification*, in "ICFEM 2008", To appear, 2008.

[13] M. HUISMAN, G. PETRI. *BicolanoMT: a formalization of multi-threaded Java at bytecode level*, in "Bytecode 2008", Electronic Notes in Theoretical Computer Science, 2008.

Scientific Books (or Scientific Book chapters)

[14] G. BARTHE, B. GRÉGOIRE, C. RIBA. *A Tutorial on Type-Based Termination*, LNCS Tutorial Series, Springer, 2008.

Research Reports

[15] J. CHRZASZCZ, M. HUISMAN, A. SCHUBERT, J. KINIRY, M. PAVLOVA, E. POLL. *BML Reference Manual*, Available online, 2008, http://bml.mimuw.edu.pl/.

[16] C. HAACK, M. HUISMAN, C. HURLIN. *Reasoning about Java's Reentrant Locks*, Technical report, n^{o} ICIS-R08014, Radboud University Nijmegen, October 2008.

[17] C. HAACK, C. HURLIN. *Separation Logic Contracts for a Java-like Language with Fork/Join*, Technical report, n^{o} 6430, INRIA Sophia Antipolis - Méditerranée, March 2008, http://hal.inria.fr/inria-00218114/en/.

References in notes

[18] L. BURDY, M. HUISMAN, M. PAVLOVA. *Preliminary Design of BML: A Behavioral Interface Specification Language for Java bytecode*, in "Fundamental Approaches to Software Engineering (FASE 2007)", Lecture Notes in Computer Science, vol. 4422, Springer-Verlag, 2007, p. 215-229.

[19] D. GUROV, M. HUISMAN, C. SPRENGER. *Compositional Verification of Sequential Programs with Procedures*, in "Information and Computation", Conditionally accepted, 2007.

[20] M. PAVLOVA, G. BARTHE, L. BURDY, M. HUISMAN, J.-L. LANET. *Enforcing High-Level Security Properties For Applets*, in "Proceedings of CARDIS'04", P. PARADINAS, J.-J. QUISQUATER (editors), kluwer, 2004, ftp://ftp-sop.inria.fr/everest/publis/P+04cardis.pdf.