# Activity Report 2013

# Team FORMES

## Formal Methods for Embedded Systems

RESEARCH CENTER
**Paris - Rocquencourt**

THEME
**Programs, Verification and Proofs**

# Table of contents

**Team FORMES**

**Keywords:** Simulation, Formal Methods, Proof Theory, Proofs Of Programs, Verification

*Creation of the Team:* 2009 January 01, end of the Team: 2013 December 31.

# 1. Members

**Research Scientists**

Frédéric Blanqui [Inria Researcher, until Aug 2013, HdR]
Vania Joloboff [Inria Team Leader]
Jean-Pierre Jouannaud [Inria Senior Researcher, HdR]
Jean-François Monin [UJF delegation CNRS, until Aug 2013, HdR]

**Faculty Members**

Ming Gu [Tsinghua University Full Professor, China PI]
Fei He [Tsinghua University Assistant Professor]
Jianqi Li [Tsinghua University Assistant Professor]

**PhD Students**

Jiaxiang Liu [Tsinghua and École Polytechnique]
Kim-Quyen Ly [UJF, until Aug 2013]
Xiaomu Shi [UJF, until July 2013]
Qian Wang [Tsinghua and École Polytechnique]

**Administrative Assistant**

Mei Zhang [LIAMA]

**Others**

Shenpeng Wang [Tsinghua University Master Student, Sep. 2012-May 2013]
Antoine Rouquette [Lyon Master Student internship, Sep. 2012-Aug. 2013]

# 2. Overall Objectives

## 2.1. Overall Objectives

FORMES stands for FORmal Methods for Embedded Systems. FORMES is aiming at making research advances towards the development of safe and reliable embedded systems, by exploiting synergies between two different approaches, namely (real time) hardware simulation and formal proofs development.

Embedded systems have become ubiquitous in our everyday life, ranging from simple sensors to complex systems such as mobile phones, network routers, airplane, aerospace and defense apparatus. As embedded devices include increasingly sophisticated hardware and software, the development of combined hardware and software has become a key to economic success.

The development of embedded systems uses hardware with increasing capacities. As embedded devices include increasingly sophisticated hardware running complex functions, the development of software for embedded systems is becoming a critical issue for the industry. There are often stringent time to market and quality requirements for embedded systems manufacturers. Safety and security requirements are satisfied by using strong validation tools and some form of formal methods, accompanied with certification processes such as DO178 or Common Criteria certification. These requirements for quality of service, safety and security imply to have formally proved the required properties of the system before it is deployed.

Within the context described above, the FORMES project aims at addressing the challenges of embedded systems design with a new approach, combining fast hardware simulation techniques with advanced formal methods, in order to formally prove qualitative and quantitative properties of the final system. This approach requires the construction of a simulation environment and tools for the analysis of simulation outputs and proofs of properties of the simulated system. We therefore need to connect simulation tools with code-analyzers and easy-to-use theorem provers for achieving the following tasks:

- Enhance the hardware simulation technologies with new techniques to improve simulation speed, and produce program representations that are adequate for formal analysis and proofs of the simulated programs;
- Connect validation tools that can be used in conjunction with simulation outputs that can be exploited using formal methods;
- Extend and improve the theorem proving technologies and tools to support the application to embedded software simulation.

A main novelty of the project, besides improving the existing technologies and tools, relies in the application itself: to combine simulation technologies with formal methods in order to cut down the development time for embedded software and scale up its reliability. Apart from being a novelty, this combination is also a necessity: proving very large code is unrealistic and will remain so for quite some time; and relying only on simulation for assessing critical properties of embedded systems is unrealistic as well.

We assume that these properties can be localized in critical, but small, parts of the code, or dedicated hardware models. This nevertheless requires scaling up the proof activity by an order of magnitude with respect to the size of codes and the proof development time. We expect that it is realistic to rely on both combined. We plan to rely on formal proofs for assessing properties of small, critical components of the embedded system that can be analyzed independently of the environment. We rely on formal proofs as well for assessing correctness of the elaboration of program representation abstractions. We rely on simulations for testing the whole embedded system, and to formal proofs to verify the completeness of test sets. We rely on formal proofs again for verifying the correct functioning of our tools. Proving properties of these various abstractions requires using a certified, interactive theorem prover.

## 2.2. History

The project FORMES was created by union of three different smaller groups, the origin and interests of which were somewhat different: a group working on simulation of embedded systems at CASIA since march 2007 under the leadership of Vania Joloboff; a second group working on user-assisted theorem proving under the leadership of Jean-Pierre Jouannaud originated from the Inria project-teams LOGICAL at Inria-Saclay-Île-de-France and PROTHEO at Inria-Lorraine; and a group working on model-checking and trustworthy computing at Tsinghua University under the leadership of Gu Ming. The second group moved from France to Beijing in September 2008. A previous 4 weeks visit of Jean-Pierre Jouannaud and Frédéric Blanqui in March 2008 had been used to define the new project FORMES, and prepare its installation at Tsinghua university.

FORMES is the acronym for FORmal Methods for Embedded Systems, and indeed we aim at combining in this project formal methods of very different origins for analyzing embedded systems. We develop a software **SimSoC** for *simulating* embedded systems, but we also develop other techniques and tools in order to analyze and predict their behavior, and that of the software running on such systems. These techniques themselves are of different origin, and are usually developed in different teams around the world. *Verification* techniques based on model checking have been extensively and successfully used in the past to analyze hardware systems. *Decisions procedures*, like SAT, are now common place to analyze specific software applications, such as scheduling. Proof assistants are more and more employed to carry out *formal proofs* of correctness of security protocols and more generally non-trivial pieces of software. One originality of our project is to COMBINE all these techniques in order to achieve our goal: to design methods and tools allowing one to build reliable software, also called *trustworthy computing*. In the next sections, we describe in more details these five areas, and their relationship to FORMES.

## 2.3. Highlights of the Year

The project has released a new version of its **SimSoC** simulation software, as an open source software release 0.8, available from http://gforge.inria.fr/projects/simsoc/

# 3. Research Program

## 3.1. Formal Proofs

Coq [52] is one of the most popular proof assistant, in the academia and in the industry. Based on the Calculus of Inductive Constructions, Coq has three kinds of basic entities: objects are used for computations (data, programs, proofs are objects); types express properties of objects; kinds categorize types by their logical structure. Coq's type checker can decide whether a given object satisfies a given type, and if a given type has a logical structure expressed by a given kind. Because it is possible to (uniformly) define inductive types such as lists, dependent types such as lists-of-length-n, parametric types such as lists-of-something, inductive properties such as $(even\ n)$ for some natural number $n$, etc, writing small specifications in Coq is an easy task. Writing proofs is a harder (non automatable) task that must be done by the user with the help of tactics. Automating proofs when possible is a necessary step for dissemination of these techniques, as is scaling up. These are the problems we are interested in.

Modeling in Coq is not always as easy as argued. In Coq, a powerful, very useful mechanism identifies expressions up to computation. For example, identifying two lists of identical content but respective lengths $m+n$ and $n+m$ is no problem if $m$ and $n$ are given integers, but does not work if $m$ and $n$ are unknowns, since $n+m=m+n$ is a valid theorem of arithmetic which cannot be proved by mere computation. It follows that the statement $reverse(l :: l') = reverse(l') :: reverse(l)$ is not typable, :: standing for appending two lists. This problem that seemingly innocent statements cannot be written in Coq because they do not type-check has been considered a major open problem for years. Blanqui, Jouannaud and Strub have recently introduced a new paradigm named *Coq modulo Theories*, in which computations do not operate only on closed terms (as are $1+2$ and $2+1$) but on open expressions of a decidable theory (as is $n+m=m+n$ in Presburger arithmetic). This work started with the PhD thesis of Pierre-Yves Strub [1] [51]. It addresses three problems at once: decidable goals become solved automatically by a program taken from the shelves; writing specifications and proofs becomes easier and closer to the mathematical practice; assuming that calls to a decision procedure return a *proof certificate* in case of success, the correctness of a Coq proof now results from type checking the proof as well as the various certificates generated along the proof. Trusting Coq becomes incremental, resulting from trusting each certificate checker when added in turn to Coq's kernel. The development of this new paradigm is our first research challenge here.

Scaling up is yet another challenge. Modeling a large, complex software is a hard task which has been addressed within the Coq community in two different ways. By developing a module system for Coq in the OCaml style, which makes it possible to modularize proof developments and hence to develop modular libraries. By developing a methodology for modeling real programs and proving their properties with Coq. This methodology allows to translate a JavaCard (tool Krakatoa) or C (tool FRAMA-C) program into an ML-like program. The correctness of this first step is ensured by proving in Coq verification conditions generated along the translation. The correctness of the ML-like program annotated by the user is then done by Coq via another tool called Why. This methodology and the associated tools are developed by the Inria project PROVAL in association with CEA. Part of our second challenge is to reuse these tools to prove properties at the source code level of programs used in an embedded application. As part of this effort, we are interested in the development of termination tools and automatic provers, in particular an SMT prover which is indeed complementary of our first challenge. The second part of the challenge is to ensure that these properties are still satisfied by the machine code executed on the embedded CPU. Here, we are going to rely on a different technology, certified compilers, and reuse the certified compilers from CLight (a well-chosen subset of C)

[1] The thesis was supported by the "Fondation EADS".

to ARM or PowerPC developed in the COMPCERT Inria project. We will be left with the development of certified compilers from source languages which are frequently used for developing embedded applications into CLight. These languages are either variants of C, or languages for the description of automata with timers in the case of Programmable Logic Controllers.

Our last challenge is to rely on certified tools only. In particular, we decided to certify in Coq all extensions of Coq developed in the project: the core logic of CoqMT (a Calculus of Inductive Constructions incorporating Presburger arithmetic) has been certified with Coq. Of course, Coq itself cannot be reduced to CIC anymore, which makes the certification of the *real logic* of CoqMT a major challenge. The most critical parts of the simulator will also be certified. As for compilers, there are two ways to certify tools: either, the code is proved correct, or it outputs a certificate that can be checked. The second approach demands less man-power, and has the other advantage to be compatible with the use of tools taken from the shelves, provided these tools are open-source since they must be equipped with a mechanism for generating certificates. This is the approach we will favor for the theories to be used in CoqMT, as well as for the SMT prover to be developed. For the simulator **SimSoC** itself, we shall probably combine both approaches.

## 3.2. Rewriting

Rewriting is at the heart of proof systems, since mathematical proofs are made of reasonning steps, expressed by the typing rules of a given proof system, and computational steps, expressed by its rewrite rules. The certification of a proof system involves, in particular, proving three main properties of its rewrite rules: subject reduction (rewriting should preserve types), confluence (computations should be deterministic), and termination (computations must always terminate). The fact that falsity is not provable in a given proof system follows from the previous properties. These meta-theoretical proofs are indeed very complex, depending on both the typing rules and the rewrite rules, and require expertise in both rewriting and type theory. To maintain this combined expertise in FORMES, we carry out theoretical activities in these areas, even if they may sometimes appear remotely connected to the mainstream of our work on the verification of embedded systems.

Indeed, our goal is not only to maintain our expertise, but also to develop certification tools aiming at automating these meta-theoretical proofs. Such tools participate to the so-called POPLmark challenge. Building such tools requires new results allowing to check subject-reduction, confluence and termination of higher-order calculi that are found in proof systems like the Calculus of Inductive Constructions on which Coq is based. Since subject-reduction is usually easy to check and consistency follows from the others, we are mostly interested in confluence and termination here.

Termination is an undecidable property of rewriting, even in its first-order incarnation. There are many (interactive) methods for proving termination of first-order rewrite rules, but a single method for proving termination of higher-order calculi equipped with polymorphic types, the so-called *reducibility candidates* method. Unfortunately, this method is extremely complex. The challenge here is to provide with an easy-to-use method which uses the reducibility candidates for its justification. Our approach is to define an order on terms which allows to reduce the termination property of computations to a comparison between the lefthand and righthand sides of the rewrite rules present in the proof system. Such an order must of course be well-founded, which should be proved thanks to the reducibility candidates method which becomes therefore hidden to the user who needs to carry out the comparisons only.

Our second challende is confluence. There are two approaches here, depending whether confluence can be proved after termination, or must be proved before in case confluence must be used in the termination proof (as is often the case with systems equipped with dependent types). In the first case, we basically know how to proceed, this is described next in the new results section. However, our results do not cover the whole spectrum of typing disciplines as of today. The second case is much more difficult. We have made some progress here too for the simple case of first-order rewriting, thanks to the recent notion of *decreasing diagrams* due to van Oostrom [55]. Decreasing diagrams can be interpreted as a way to carry out confluence proofs in the non-terminating case in a way which mimics how they are carried out in the terminating case. As a consequence, there should not be any difference anymore in the future in the way confluence proofs are carried out. This unified framework has been carried out so far for *abstract rewriting*, that is for binary relations on an abstract set. Our challenge is to extend this unified framework to concrete rewriting, that is rewriting on terms *generated* by rewrite rules. We are still far from this objective, which is a hard, but exciting, research challenge.

## 3.3. Verification

Model checking is an automatic formal verification technique [30]. In order to apply the technique, users have to formally specify desired properties on an abstract model of the system under verification. Model checkers will check whether the abstract model satisfies the given properties. If model checkers are able to prove or disprove the properties on the abstract model, they report the result and terminate. In practice, however, abstract models can be extremely complicated, model checkers may not conclude with reasonable computational resources.

Compositional reasoning is a way to ameliorate the complexity in abstract models [54]. Compositional reasoning tries to prove global properties on abstract models by establishing local properties on their components. If local properties on components are easier to verify, compositional reasoning can improve the capacity of model checking by local reasoning. Experiences however suggest that local reasoning may not suffice to establish global properties. It is rare that a global property can be established without considering their interactions. In assume-guarantee reasoning, model checkers try to verify local properties under a contextual assumption of each component. If contextual assumptions faithfully capture interactions among components, model checkers can conclude the verification of global properties.

Finding contextual assumptions however is difficult and may require clairvoyance. Interestingly, a fully automated technique for computing contextual assumptions was proposed in [33]. The automated technique formalizes the contextual assumption generation problem as a learning problem. If properties and abstract models are formalized as finite automata, then a contextual assumption is nothing but an unknown finite automaton that characterizes the environment. Applying a learning algorithm for finite automata, the automated technique will generate contextual assumptions for assume-guarantee reasoning. Experimental results show that the automated technique can outperform a monolithic and explicit verification algorithm.

The success of the learning-based assume-guarantee reasoning is however not satisfactory. Most verification tools are using implicit algorithms. In fact, implicit representations such as Binary Decision Diagrams can improve the capacity of model checking algorithms in order of magnitude. Early learning-based techniques, on the other hand, are based on the $L^*$ learning algorithm using explicit representations. If a contextual assumption requires hundreds of states, the learning algorithm will take too much time to infer an assumption. Subsequently, early learning-based techniques cannot compete with monolithic implicit verification [32].

We have proposed assume-guarantee reasoning with implicit learning [29]. Our idea is to adopt an implicit representation used in the learning-based framework. Instead of enumerating states of contextual assumptions explicitly, our new technique computes transition relations as an implicit representation of contextual assumptions. Using a learning algorithm for Boolean functions, the new technique can easily compute contextual assumptions with thousands of states. Our preliminary experimental results show that the implicit learning technique can outperform interpolation-based monolithic implicit model checking in several parametrized test cases such as synchronous bus arbiters and the MSI cache coherence protocol.

Learning Boolean functions can also be applied to loop invariant inference [40], [41]. Suppose that a programmer annotates a loop with pre- and post-conditions. We would like to compute a loop invariant to verify that the annotated loop conforms to its specification. Finding loop invariants manually is very tedious. One makes a first guess and then iteratively refines the guess by examining the loop body. This process is in fact very similar to learning an unknown formula. Applying predicate abstraction and decision procedures, a learning algorithm for Boolean functions can infer loop invariants generated by a given set of atomic predicates. Preliminary experimental results show that the learning-based technique is effective for annotated loops extracted from source codes of Linux and SPEC2000 benchmarks.

Although implicit learning techniques have been developed for assume-guarantee reasoning and loop invariant inference successfully, challenges still remain. Currently, the learning algorithm is able to infer Boolean functions over tens of Boolean variables. Contextual assumptions over tens of Boolean variables are not enough. Ideally, one would like to have contextual assumptions over hundreds (even thousands) of Boolean variables. On the other hand, it is known that learning arbitrary Boolean functions is infeasible. The scalability of implicit learning techniques cannot be improved satisfactorily by tuning the learning algorithm alone. Combining implicit learning with abstraction will be essential to improve its scalability.

Our second challenge is to extend learning-based techniques to other computation models. In addition to finite automata, probabilistic automata and timed automata are also widely used to specify abstract models. Their verification problems are much more difficult than those for finite automata. Compositional reasoning thus can improve the capacity of model checkers more significantly. The $L^*$ algorithm has been applied in assume-guarantee reasoning for probabilistic automata [35]. The new technique is unfortunately incomplete. Developing a complete learning-based assume-guarantee reasoning technique for probabilistic automata and timed automata will be very useful to their verification.

Through predicate abstraction, learning Boolean functions can be very useful in program analysis. We have successfully applied algorithmic learning to infer both quantified and quantifier-free loop invariants for annotated loops. Applying algorithmic learning to static analysis or program testing will be our last challenge. In the context of program analysis, scalability of the learning algorithm is less of an issue. Formulas over tens of atomic predicates usually suffice to characterize relation among program variables. On the other hand, learning algorithms require oracles to answer queries or generate samples. Designing such oracles necessarily requires information extracted from program texts. How to extract information will be essential to applying algorithmic learning in static analysis or program testing.

## 3.4. Decision Procedures

Decision procedures are of utmost importance for us, since they are at the heart of theorem proving and verification. Research in decision procedures started several decades ago, and are now commonly used both in the academia and industry. A decision procedure [42] is an algorithm which returns a correct yes/no answer to a given input decision problem. Many real-world problems can be reduced to the decision problems, making this technique very practical. For example, Intel and AMD are developing solvers for their circuit verification tools, while Microsoft is developing decision procedures for their code analysis tools.

Mathematical logic is the appropriate tool to formulate a decision problem. Most decision problems are formulated as a decidable fragment of a first-order logic interpreted in some specific domain. One such easy and popular fragment is propositional (or Boolean) logic, to which corresponding decision procedure is called SAT. Representing real problems in SAT often results in awkward encodings that destroy the logical structure of the original problem.

A very popular, effective recent trend is Satisfiability Modulo Theories (SMT) [53], a general technique to solve decision problems formulated as propositional formulas operating on atoms in a given background theory, for example linear real arithmetic. Existing approaches for solving SMT problems can be classified into two categories: *lazy* method [49], and *eager* method [50]. The eager method encodes an SMT problem into an equi-satisfiable SAT problem, while the lazy method employs different theory solvers for each theory and coordinates them appropriately. The eager method does allow the user to express her problem in a natural way, but does not exploit its logical structure to speed up the computation. The lazy approach is more appealing, and has prompted much interest in algorithms for the various background theories important in practice.

Our SMT solver aCiNO is based on the lazy approach. So far, it provides with two (popular) theories only: linear real arithmetic (LRA) and uninterpreted functions (UF). For efficiency consideration, the solver is implemented in an incremental way. It also invokes an online SAT solver, which is now a modified DPLL procedure, so that recovery from conflicts is possible. Our challenge here is twofold: first, to add other theories of interest for the project, we are currently working on fragments of the theory of arrays [44], [26]. The theory of arrays is important because of its use for expressing loop invariants in programs with arrays, but its full first-order theory is undecidable. We are also interested in the theory of bit vectors, very much used for hardware verification.

Theory solvers implement state-of-the-art algorithms, but their sophistication makes their correct implementation a delicate task. Moreover, SMT solvers themselves employ a quite complex machinery, making them error prone as well [2]. We therefore strongly believe that decision procedures, and SMT provers, should come along with a formal assessment of their correctness. As usual, there are two ways: ensure the correctness of an arbitrary output by proving the code, or deliver for each input a certificate ensuring the correctness of the corresponding output when the checker says so. Developing concise certificates together with efficient certificate checkers for the various decision procedures of interest and their combination with SMT is yet another challenge which is at the heart of the project FORMES.

## 3.5. Simulation

The development of complex embedded systems platforms requires putting together many hardware components, processor cores, application specific co-processors, bus architectures, peripherals, etc. The hardware platform of a project is seldom entirely new. In fact, in most cases, 80 percent of the hardware components are re-used from previous projects or simply are COTS (Commercial Off-The-Shelf) components. There is no need to simulate in great detail these already proven components, whereas there is a need to run fast simulation of the software using these components.

These requirements call for an integrated, modular simulation environment where already proven components can be simulated quickly, (possibly including real hardware in the loop), new components under design can be tested more thoroughly, and the software can be tested on the complete platform with reasonable speed.

Modularity and fast prototyping also have become important aspects of simulation frameworks, for investigating alternative designs with easier re-use and integration of third party components.

The project aims at developing such a rapid prototyping, modular simulation platform, combining new hardware components modeling, verification techniques, fast software simulation for proven components, capable of running the real embedded software application without any change.

To fully simulate a complete hardware platform, one must simulate the processors, the co-processors, together with the peripherals such as network controllers, graphics controllers, USB controllers, etc. A commonly used solution is the combination of some ISS (Instruction Set Simulator) connected to a Hardware Description Language (HDL) simulator which can be implemented by software or by using a FPGA [43] simulator. These solutions tend to present slow iteration design cycles and implementing the FPGA means the hardware has already been designed at low level, which comes normally late in the project and become very costly when using large FPGA platforms. Others have implemented a co-simulation environment, using two separate technologies, typically one using a HDL and another one using an ISS [36], [38], [48]. Some communication and synchronization must be designed and maintained between the two using some inter-process communication (IPC), which slows down the process.

The idea we pursue is to combine hardware modeling and fast simulation into a fully integrated, software based (not using FPGA) simulation environment, which uses a single simulation loop thanks to Transaction Level Modeling (TLM) [28], [19] combined with a new ISS technology designed specifically to fit within the TLM environment.

The most challenging way to enhance simulation speed is to simulate the processors. Processor simulation is achieved with Instruction Set Simulation (ISS). There are several alternatives to achieve such simulation. In *interpretive simulation*, each instruction of the target program is fetched from memory, decoded, and executed. This method is flexible and easy to implement, but the simulation speed is slow as it wastes a lot of time in decoding. Interpretive simulation is used in Simplescalar [27]. Another technique to implement a fast ISS is

[2]It took almost 20 years to have a correct implementation of a correct version of Shostak's algorithm for combining decision procedures, which can be seen as an ancestor of SMT.

*dynamic translation* [31], [47], [34] which has been favored by many implementors [45], [34], [46], [47] in the past decade.

With dynamic translation, the binary target instructions are fetched from memory at run-time, like in interpretive simulation. They are decoded on the first execution and the simulator translates these instructions into another representation which is stored into a cache. On further execution of the same instructions, the translated cached version is used. Dynamic translation introduces a translation time phase as part of the overall simulation time. But as the resulting cached code is re-used, the translation time is amortized over time. If the code is modified during run-time, the simulator must invalidate the cached representation. Dynamic translation provides much faster simulation while keeping the advantage of interpretive simulation as it supports the simulation of programs that have either dynamic loading or self-modifying code.

There are many ways of translating binary code into cached data, which each come at a price, with different trade-offs between the translation time and the obtained speed up on cache execution. Also, simulation speed-ups usually don't come for free: most of time there is a trade-off between accuracy and speed.

There are two well known variants of the dynamic translation technology: the target code is translated either directly into machine code for the simulation host, or into an intermediate representation, independent from the host machine, that makes it possible to execute the code with faster speed. Both have pros and cons.

Processor simulation is also achieved in Virtual Machines such as QEMU [23] and GXEMUL [37] that emulate to a large extent the behavior of a particular hardware platform. The technique used in QEMU is a form of dynamic translation. The target code is translated directly into machine code using some pre-determined code patterns that have been pre-compiled with the C compiler. Both QEMU and GXEMUL include many device models of open-source C code, but this code is hard to reuse. The functions that emulate device accesses do not have the same profile. The scheduling process of the parallel hardware entities is not specified well enough to guarantee the compatibility between several emulators or re-usability of third-party models using the standards from the electronics industry (e.g. IEEE 1666).

A challenge in the development of high performance simulators is to maintain simultaneously fast speed and simulation accuracy. In the FORMES project, we expect to develop a dynamic translation technology satisfying the following additional objectives:

- provide different levels of translation with different degrees of accuracy so that users can choose between accurate and slow (for debugging) or less accurate but fast simulation.
- to take advantage of multi-processor simulation hosts to parallelize the simulation;
- to define intermediate representations of programs that optimize the simulation speed and possibly provide a more convenient format for studying properties of the simulated programs.

Another objective of the FORMES simulation is to extract information from the simulated applications to prove properties. Running a simulation is exercising a test case. In most cases, if a test is failing, a bug has been found. One can use model checking tools to generate tests that can be run on the simulator to check whether the test fails or not on the real application. It is also a goal of FORMES simulation activity to use such formal methods tools to detect bugs, either by generating tests, or by using formal methods tools to analyze the results of simulation sessions.

## 3.6. Trustworthy Software

Since the early days of software development, computer scientists have been interested in designing methods for improving software quality. Formal methods based on model checking, correctness proofs, common criteria certification, all address this issue in their own way. None of these methods, however, considers the trustworthiness of a given software system as a system-level property, requiring to grasp a given software within its environment of execution.

The major challenge we want to address here is to provide a framework in which to formalize the notion of trustworthiness, to evaluate the trustworthiness of a given software, and if necessary improve it.

To make trustworthiness a fruitful concept, our vision is to formalize it via a hierarchy of observability and controllability degrees: the more the software is observable and controllable, the more its behaviors can be trusted by users. On the other hand, users from different application domains have different expectations from the software they use. For example, aerospace embedded software should be safety-critical while e-commerce software should be insensitive to attacks. As a result, trustworthiness should be domain-specific.

A main challenge is the evaluation of trustworthiness. We believe that users should be responsible for describing the level of trustworthiness they need, in the form of formal requirements that the software should satisfy. A major issue is to come up with some predefined levels of trustworthiness for the major applicative areas. Another is to use stepwise refinement techniques to achieve the appropriate level of trustworthiness. These levels would then drive the design and implementation of a software system: the objective would be to design a model with enough details (observability) to make it possible to check all requirements of that level.

The other challenge is the effective integration of results obtained from different verification methods. There are many verification techniques, like simulation, testing, model checking and theorem proving. These methods may operate on different models of the software to be then executed, while trustworthiness should measure our trust in the real software running in its real execution environment. There are also monitoring and analysis techniques to capture the characteristics of actual executions of the system. Integrating all the analysis in order to decide the trustworthiness level of a software is quite a hard task.

# 4. Application Domains

## 4.1. Proof of Programs

In many life critical application such as nuclear power or transportation, formal proofs of programs are required, and theorem provers provide an essential tool in that area.

## 4.2. Simulation

Simulation is relevant to most areas where complex embedded systems are used, not only to the semiconductor industry for System-on-Chip modeling, but also to any application where a complex hardware platform must be assembled to run the application software. It has applications for example in industry automation, digital TV, telecommunications and transportation.

## 4.3. Certified Compilation for Embedded systems

Many frameworks have been designed in order to make the design and the development of embedded systems more rigourous and secure on the basis of some formal model. All these frameworks implicitly assume the *reliability of the translation* to executable code, in order to guarantee the verified properties in the design level are preserved in the implementation. In other words, they rely on a claim saying that the compilers from high level model description to the implementation will not introduce undesired behaviors or errors in silence. The only safe way to satisfy such a claim is to certify correctness of the compilers, that is, to prove that the code they produce has exactly the semantics of the source code or model.

## 4.4. Distributed Systems

Many embedded systems run in a distributed environment. Distributed systems raise extremely challenging issues, both for the design and the implementation, because decisions can be made only from a local knowledge, which is imperfect due to communication time and unreliability of transmissions.

## 4.5. Security

The convergence between embedded technologies and the Internet offers many opportunities to malicious people for breaking the privacy of consumers or of organisations. Using cryptography is not enough for ensuring the protection of data, because of possible flaws in protocols and interfaces, providing opportunities for many well-known attacks. This area is therefore an important target of formal methods.

# 5. Software and Platforms

## 5.1. CoLoR

**Participants:** Frédéric Blanqui, Kim-Quyen Ly.

CoLoR is a Coq library on rewriting theory and termination of more than 83,000 lines of code [4]. It provides definitions and theorems for:

- Mathematical structures: relations, (ordered) semi-rings.
- Data structures: lists, vectors, polynomials with multiple variables, finite multisets, matrices, finite graphs.
- Term structures: strings, algebraic terms with symbols of fixed arity, algebraic terms with varyadic symbols, pure and simply typed $\lambda$-terms.
- Transformation techniques: conversion from strings to algebraic terms, conversion from algebraic to varyadic terms, arguments filtering, rule elimination, dependency pairs, dependency graph decomposition, semantic labelling.
- Termination criteria: polynomial interpretations, multiset ordering, lexicographic ordering, first and higher order recursive path ordering, matrix interpretations.

CoLoR is distributed under the CeCILL license. It is currently developed by Frédéric Blanqui and Kim-Quyen Ly, but various people participated to its development since 2006.

## 5.2. HOT

**Participant:** Frédéric Blanqui.

HOT is an automated termination prover for higher-order rewrite systems based on the notion of computability closure and size annotation [24]. It won the 2012 competition in the category “higher-order rewriting union beta”. The sources are not public.

## 5.3. Moca

**Participant:** Frédéric Blanqui.

Moca is a construction functions generator for OCaml data types with invariants.

It allows the high-level definition and automatic management of complex invariants for data types. In addition, it provides the automatic generation of maximally shared values, independently or in conjunction with the declared invariants.

A relational data type is a concrete data type that declares invariants or relations that are verified by its constructors. For each relational data type definition, Moca compiles a set of construction functions that implements the declared relations.

Moca supports two kinds of relations:

- predefined algebraic relations (such as associativity or commutativity of a binary constructor),
- user-defined rewrite rules that map some pattern of constructors and variables to some arbitrary users defined expression.

The properties that user-defined rules should satisfy (completeness, termination, and confluence of the resulting term rewriting system) must be verified by a programmer’s proof before compilation. For the predefined relations, Moca generates construction functions that allow each equivalence class to be uniquely represented by their canonical value.

Moca is distributed under QPL. It is developed by Frédéric Blanqui, Pierre Weis (EPI Pomdapi) and Richard Bonichon (CEA).

## 5.4. Rainbow

**Participants:** Frédéric Blanqui, Kim-Quyen Ly.

Rainbow is a tool for automatically verifying the correctness of termination certificates expressed in the CPF XML format as used in the termination competition. Termination certificates are currently translated and checked in Coq by using the CoLoR library. But a new standalone version is under development using Coq extraction mechanism (PhD subject of Kim-Quyen Ly).

Rainbow is distributed under the CeCILL license. It is currently developed by Frédéric Blanqui and Kim-Quyen Ly. See the web site for more information.

## 5.5. CoqMT

**Participants:** Qian Wang [correspondant], Jean-Pierre Jouannaud.

The proof-assistant Coq is based on a complex type theory, which resulted from various extensions of the Calculus of Constructions studied independently from each other. With the collaboration of Bruno Barras, we decided to address the challenge of proving the real type theory underlying Coq, and even, indeed, of its recent extension CoqMT developed in FORMES by Pierre-Yves Strub. To this end, we have studied formally the theory CoqMTU, which extends the pure Calculus of Constructions by inductive types, a predicative hierarchy of universes, and a decidable theory T for some first-order inductive types. Recently, we were able to announce the complete certification of CoqMTU in Coq augmented with appropriate intuitionistic set-theoretic axioms in order to fight Gödel's incompleteness theorem~[16]. As a consequence, Coq and CoqMTU are the first proof assistants, of which consistency (relative to intuitionistic set theory IZF augmented with the afore-mentioned axioms) is formally entirely proved (in Coq). While previous formal proofs for Coq and other proof assistants all assumed strong normalization, the present one *proves* strong normalization thanks to the new notion of *strongly-normalizing* model introduced by Bruno Barras. While consistency is done already, decidability of type-checking in CoqMTU remains to be done. This is a straightforward consequence for Coq, but a non-trivial task for CoqMTU because of the interaction between inductive types and the first-order theory T. It should however be done by the summer of 2014. We consider this work as a major scientific achievement of the team.

## 5.6. SimSoC

**Participants:** Vania Joloboff [correspondant], Antoine Rouquette, Shenpeng Wang.

**SimSoC** is an infrastructure to run simulation models which comes along with a library of simulation models. **SimSoC** allows its users to experiment various system architectures, study hardware/software partition, and develop embedded software in a co-design environment before the hardware is ready to be used. **SimSoC** aims at providing high performance, yet accurate simulation, and provide tools to evaluate performance and functional or non functional properties of the simulated system.

**SimSoC** is based on SystemC standard and uses Transaction Level Modeling for interactions between the simulation models. The current version is based on the open source libraries from the OSCI Consortium: SystemC version 2.3 and TLM 2.0.1 [39], [21]. Hardware components are modeled as TLM models, and since TLM is itself based on SystemC, the simulation is driven by the SystemC kernel. We use standard, unmodified, SystemC, hence the simulator has a single simulation loop.

The third open source version of **SimSoC**, release 0.8.0, has been released in September 2013. It contains a full simulator for ARM (V5 and V6) and PowerPC both running at an average speed of about 100 Millions instructions per second in, and a deprecated simulator for the MIPS architecture. **SimSoC** is distributed under LGPL on Inria Gforge web site.

## 5.7. SimSoC-Cert

**Participants:** Frédéric Blanqui, Vania Joloboff, Jean-François Monin [correspondant], Xiaomu Shi.

Simulators such as **SimSoC** make it possible to reduce development time and development cost, allowing for the software engineers to run fast iterative cycles without requiring a hardware development board. Then a critical issue is: *does the simulator actually simulate the real hardware?*

Considering only one module in **SimSoC**, namely the ARM simulator, it somehow encodes the 1138 pages of the ARM reference manual in C++. The whole simulator, which simulates ARM and PowerPC architecture, includes about 60,000 lines of manually coded C++ code. Then, mistakes in the hand written code are unavoidable and difficult to find due to the complexity. From the experiments performed on **SimSoC**, bugs bringing a wrong behavior were observed from time to time but it was hard to reveal where they were. Using intensive tests can cover most of the instructions, but still left some untested rare cases of instructions, which lead to potential problems.

Therefore, a better approach is required to gain confidence in the correctness of the simulator. Our proposal has been to certify the ARM CPU simulator from **SimSoC** using formal methods. We aimed at proving a significant part of the correctness of **SimSoC** in order to support the claim that the implementation of the simulator and the real hardware system will exhibit the same behavior.

In addition, we developed tools that can automatically generate in various C the core simulator, including the decoding functions and the instruction set of the ARMv6 architecture manual [18] (implemented by the ARM11 processor family). The input of SimSoC-Cert is the ARMv6 architecture manual itself.

In order to get the required flexibility and accuracy, we wanted to experiment a direct approach based on a general proof assistant such as Coq. Fortunately, an operational semantics formalized in Coq of a large enough subset of the C language is available from the **CompCert** project. We then decided to base our correctness proofs on this technology. Up to our knowledge, this is the first development of formal correctness proofs based on operational semantics, at least at this scale.

Based on this, we first developed *simlight* (8000 generated lines of C, plus 1500 hand-written lines of C), a simulator for ARMv6 programs using no peripheral and no coprocessor. Next, we developed *simlight2*, a fast ARMv6 simulator integrated inside a SystemC/TLM module, now part of SimSoC v0.8.

We can also generate similar programs for SH4 [20] but this is still experimental (work done by Frédéric Tuong in 2011).

Finally, we proved that the C code for simulating ARM instructions in Simlight is correct with respect to the Coq model.

# 6. New Results

## 6.1. Type and rewriting theory

**Participants:** Frédéric Blanqui, Jean-Pierre Jouannaud, Jianqi Li, Qian Wang.

Qian Wang and Bruno Barras have proved the strong normalization property of CoqMTU in presence of strong elimination, a major step towars the full certification of CoqMTU [16].

Jouannaud and Li have developped a new framework, Normal Abstract Rewriting Systems (NARS), that captures all known Church-Rosser results in presence of a termination assumption allowing to reduce equality of terms to a simpler equality on their normal forms. This result applies to the paticular case of higher-order rewriting for which it solved long-standing open problems [10].

Jouannaud and Liu have continued their investigation of Church-Rosser properties of non-terminating rewrite systems [10], showing recently first, that many results found in the litterature could be captured, and generalized, by using van Oostrom's decreasing diagram technique (accepted at Symposium on Algebraic Specifications, Kanazawa, Japan, April 2014). The next step, which has been recently completed, is a powerful result generalizing Knuth and Bendix confluence test to non terminating rewrite system (submitted).

Frédéric Blanqui, Jean-Pierre Jouannaud and Albert Rubio (Technical University of Catalonia) have developed a method aiming at carrying out termination proof for higher-order calculi. CPO appears to be the ultimate improvement of the higher-order recursive path ordering (HORPO) [25] in the sense that this definition captures the essence of computability arguments *à la* Tait and Girard, therefore explaining the name of the improved ordering. It has been shown that CPO allows to consider higher-order rewrite rules in a simple type discipline with inductive types, that most of the guards present in the recursive calls of its core definition cannot be relaxed in any natural way without losing well-foundedness, and that the precedence on function symbols cannot be made more liberal anymore. This result is submitted to journal, and has been concurrently generalized to higher-order calculi with dependent types by Jouannaud and Li (submitted).

Frédéric Blanqui worked on the formalization in the Coq proof assistant of various definitions of the notion of $\alpha$-equivalence on pure $\lambda$-terms. In particular, he formalized and formally proved equivalent the definitions of Church (1932), Curry and Feys (1958), Krivine (1993), and Gabbay and Pitts (1999). This work is freely available from the CoLoR library released on December 13th.

Frédéric Blanqui worked with John Steinberger (Tsinghua University) on the formal verification in Coq of proofs of theorems on coset arrays and non-negative integer linear combinations.

## 6.2. Automated theorem proving

**Participant:** Kim-Quyen Ly.

Kim-Quyen Ly extended her formally-proved (in Coq) automated termination-certificate (for first-order rewrite systems) verifier Rainbow for dealing with certificates using arguments filtering [22] and other termination techniques.

## 6.3. Simulation

**Participants:** Vania Joloboff, Antoine Rouquette, Shenpeng Wang.

There exists very fast Loosely Timed simulators such as **SimSoC** that can run the application software to validate its functionality and possibly test real time software using timers. But such simulators do not provide good enough timings to evaluate the software performance. The idea of "Approximately Timed" simulation is to provide a fast simulation that can be used by software developers, and yet provide performance estimate. The goal of approximately timed simulation is to provide estimates that are within a small margin error from the real hardware performance, but at a simulation speed that is an order of magnitude faster than a cycle accurate one.

Modern processors have complex architectures. They can execute a certain number of instructions per clock cycle. There are however several cases where the instruction flow is disrupted, introducing delays in the computation. In order to make an Approximately Timed simulator, our idea is to simulate enough of the processes causing the delays, not simulating the exact hardware processes of the caches and pipe line and I/Os, but using a model with wich the delays can be computed with a reasonable approximation while maintaining fast simulation. Delays may also be related to bus arbitration and interconnect access. These delays are beyond the scope of our work, but can be captured by TLM (timed) transactions. In our work, we are considering only the processor model and we rely upon TLM interface to the interconnect for peripheral access to provides us with timing delays.

We have started to investigate a new approach to provide a fast Approximately Timed ISS, that does not simulate fully the hardware, yet provides good precision estimates, and does not use stastistical methods. Our approach consists in developing a higher abstraction model of the processor (than the CA models) that still executes instructions using fast SystemC/TLM code, but in parallel maintains some architecture state to measure the delays introduces by cache misses and pipe line stalls, although the pipe line is not really simulated.

## 6.4. Certification of a Simulator

**Participants:** Vania Joloboff, Jean-François Monin, Xiaomu Shi.

We have developed a correctness proof of a part of the hardware simulator **SimSoC**. This is not only an attempt to certify a simulator, but also a new experiment on the certification of non-trivial programs written in C. We have provided a formalized representation of the ARM instruction set and addressing modes in Coq. We also constructed a Coq representation of the ARM simulator in C, using the abstract syntax defined in **CompCert**.

From these two Coq representations, we have developed Coq proofs to prove the correctness of the C code, using the operational semantics of C provided by **CompCert**.

During this work, we have also improved the technology available in Coq for performing *inversions*, a kind of proof steps which heavily occurs in this context.

All of this work has been described in Xiaomu SHI PhD thesis dissertation, presented at University of Grenoble in July 2013, and at ITP 2013 conference[15].

# 7. Partnerships and Cooperations

## 7.1. National Initiatives

### *7.1.1. Tsinghua Grant*

contract: Tsinghua National Laboratory for Information Science and Technology, Cross-discipline Foundation grant 2011-9

title: An Intensional Logical Framework and Its Implementation

Participants: Jean-Pierre Jouannaud, Jianqi Li

duration: 2011 - 2012

Amount: 100,000 RMB

### *7.1.2. NSFC Grant*

contract: National Science Foundation of China grant 61272002

title: The meta-theories of higher-order rewriting and their proof automation: toward the next generation theorem prover

PIs: Jean-Pierre Jouannaud, Jianqi Li

duration: 2013-2016

Amount: 600,000 RMB

## 7.2. International Initiatives

### *7.2.1. Inria International Partners*

#### *7.2.1.1. Declared Inria International Partners*

The FORMES project has been held since the beginning at Tsinghua University, Beijing, China. Tsinghua University is a founding member of LIAMA laboratory.

#### *7.2.1.2. Informal International Partners*

The FORMES project has also collaborated with:

- Pr John Koo at Shenzhen Institute of Advanced Technology, until August 2013.
- the Institute of Software of the Chinese Academy of Science where Frédéric Blanqui has been kindly hosted between July 2012 and August 2013.

### *7.2.2. Inria International Labs*

FORMES is one of the LIAMA projects.

### *7.2.3. Participation In other International Programs*

LIAMA is a member of the AURA network: Association of Units of Research in Asia.

## 7.3. International Research Visitors

### *7.3.1. Visits of International Scientists*

FORMES project member Jean-Pierre Jouannaud organized jointly with Pr Ming Gu the LIAMA-Tsinghua Software Day, where the following scientists reported on their research:

- Pr Edmund Clarke, from Carnegie Mellon.
- Erik Hagersten from University of Uppsala.
- Marc Pouzet from University Pierre et Marie Curie.

#### *7.3.1.1. Internships*

- *Jiaxiang Liu*
  - Subject: Diagramatic Confluence,
  - Date: from Jul 2013 to Dec 2013,
  - Institution: Ecole Polytechnique
- *Antoine Rouquette*
  - Subject: Upgrade of SimSoC simulator,
  - Date: from September 2012 to August 2013,
  - Institution: Shenzhen Institutes of Advanced Technology
- *Shenpeng Wang*
  - Subject: Approximately Timed Simulation of PowerPC e200z,
  - Date: from March 2012 to May 2013,
  - Institutions: Tsinghua University and Shenzhen Institutes of Advanced Technology

# 8. Dissemination

## 8.1. Scientific Animation

Frédéric Blanqui was member of the Steering Committee of the International Conference on Rewriting Techniques and Applications (RTA) for 3 years until June 2013.

Frédéric Blanqui was invited to present his work on "the formalization of $\lambda$-calculus and Tait-Girard's notion of computability" at the 3rd Workshop on Proof Theory and Rewriting (PR), March 2013, Kanazawa, Japan.

Vania Joloboff has organized a LIAMA Open Day in Shanghai in May 2013, in collaboration with East China Normal University.

## 8.2. Teaching - Supervision - Juries

### *8.2.1. Teaching*

Frédéric Blanqui organized a 7-days school at the Institute of Applied Mechanics and Informatics (IAMA) of the Vietnamese Academy of Sciences and Technology (VAST) at Ho Chi Minh City, Vietnam, from 12 to 19 March 2013. The mornings were dedicated to theoretical lectures introducing basic notions in mathematics and logic for the analysis of computer programs. The afternoons were practical sessions introducing the OCaml programming language and the Coq proof assistant. Lecture notes are given in [17].

Vania Joloboff has taught simulation seminars at Shenzhen Institutes of Advanced Technology.

Licence: Jean-François Monin, Introduction to Interactive Proof of Software, 50 hours, L3, Tsinghua University, China
This course is expected to attract students in the FORMES group via the local PhD program; already one of them (2009) is currently a PhD student of Jean-Pierre Jouannaud, another (2010) in is the PhD track with Gu Ming and 2 others (2010) work with Jean-François Monin and Vania Joloboff.

Doctorate: Jean-François Monin (organizer and teacher), Coq Summer School, 30 hours, Tsinghua University, China

### 8.2.2. Supervision

PhD : Xiaomu Shi, “Certification of an Instruction Set Simulator”, University of Grenoble, July 2013, [14] Jean-François Monin, Vania Joloboff.

PhD in progress: Kim-Quyen Ly, automated formal verification of termination certificates, October 2011, Frédéric Blanqui

PhD in progress : Jiaxiang Liu, Testing Confluence via Critical Pairs, 2012, École Polytechnique, Jean-Pierre Jouannaud

PhD in progress: Qian Wang, CoqMTU: a secure combination of the Calculus of Construction, inductive types, universes and built-in equality, 2011, École Polytechnique, Jean-Pierre Jouannaud

### 8.2.3. Juries

Frédéric Blanqui has been in the jury of Zhiwu Xu for his PhD on “Parametric Polymorphism for XML Processing Languages” (directors: Giuseppe Castagna and Haiming Chen).

Frédéric Blanqui refered the habilitation thesis of René Thiemann (Innsbrück University) on “A Formalization of Termination Techniques in Isabelle/HOL”.

Jean-François Monin has been in the jury of Xiaomu Shi (see above).

Vania Joloboff has been in the jury of Xiaomu Shi (see above).

# 9. Bibliography

## Major publications by the team in recent years

[1] B. BARRAS, J.-P. JOUANNAUD, P.-Y. STRUB, Q. WANG. *CoqMTU: a higher-order type theory with a predicative hierarchy of universes parametrized by a decidable first-order theory*, in "Twenty-Sixth Annual IEEE Symposium on "Logic in Computer Science" - LICS 2011", Toronto, Canada, 2011, This research is sponsored by NSFC Program (No.91018015) and 973 Program (No.2010CB328003) of China, http://hal.inria.fr/inria-00583136

[2] F. BLANQUI. *Definitions by rewriting in the Calculus of Constructions*, in "Mathematical Structures in Computer Science", 2005, vol. 15, n$^{o}$ 1, pp. 37-92 [*DOI :* 10.1017/S0960129504004426], http://hal.inria.fr/inria-00105648/en/

[3] F. BLANQUI, C. HELMSTETTER, V. JOLOBOFF, J.-F. MONIN, X. SHI. *Designing a CPU model: from a pseudo-formal document to fast code*, in "3rd Workshop on: Rapid Simulation and Performance Evaluation: Methods and Tools", Grèce Heraklion, 2011, Best paper award, http://hal.inria.fr/inria-00546228/en/

[4] F. BLANQUI, A. KOPROWSKI. *CoLoR: a Coq library on well-founded rewrite relations and its application to the automated verification of termination certificates*, in "Mathematical Structures in Computer Science", 2011, vol. 21, n$^{o}$ 4, pp. 827-859, http://hal.inria.fr/inria-00543157/en/

[5] F. BLANQUI, J.-P. JOUANNAUD, P.-Y. STRUB. *From formal proofs to mathematical proofs: a safe, incremental way for building in first-order decision procedures*, in "5th IFIP International Conference on Theoretical Computer Science - TCS 2008", Milan Italie, IFIP, 2008, vol. 273 [*DOI :* 10.1007/978-0-387-09680-3_24], http://hal.inria.fr/inria-00275382/en/

[6] B. BÉRARD, L. FRIBOURG, F. KLAY, J.-F. MONIN. *A compared study of two correctness proofs for the standardized algorithm of ABR conformance*, in "Formal Methods in System Design", january 2003

[7] B. DELSART, V. JOLOBOFF, E. PAIRE. *JCOD: A Lightweight Modular Compilation Technology for Embedded Java*, in "Second International Conference on Embedded Software", Lecture Notes in Computer Science, Springer-Verlag, 2002, vol. 2491, pp. 197–212, ISBN 3-540-44307-X

[8] F. HE, X. SONG, M. GU, J. SUN. *Heuristic-Guided Abstraction Refinement*, in "Computer Journal", May 2009, vol. 52, n$^{o}$ 3, pp. 280-287

[9] J.-P. JOUANNAUD, J.-Q. LI. *Church-Rosser Properties of Normal Rewriting*, in "Computer Science Logic", Fontainebleau, France, P. CÉGIELSKY, A. DURAND (editors), LIPIcs, Dagstuhl Publishing, September 2012, vol. 16, pp. 350-365 [*DOI :* 10.4230/LIPICS.CSL.2012.I], http://hal.inria.fr/hal-00730271

[10] J.-P. JOUANNAUD, J. LIU. *From diagrammatic confluence to modularity*, in "Theor. Comput. Sci.", 2012, vol. 464, pp. 20-34

[11] J.-P. JOUANNAUD, A. RUBIO. *Polymorphic Higher-Order Recursive Path Orderings*, in "Journal of the ACM", 2007, vol. 54, n$^{o}$ 1, pp. 1-48

[12] J.-P. JOUANNAUD, V. VAN OOSTROM. *Diagrammatic Confluence and Completion*, in "International Conference in Automata, Languages and Programming", Grèce Rhodes, W. THOMAS (editor), Springer Berlin/Heidelberg, 2009, vol. 2, http://hal.inria.fr/inria-00436070/en/

[13] X. SHI, J.-F. MONIN, F. TUONG, F. BLANQUI. *First Steps towards the Certification of an ARM Simulator Using Compcert*, in "Certified Proofs and Programs - First International Conference", Kenting, Taiwan, J.-P. JOUANNAUD, Z. SHAO (editors), LNCS, Springer, December 7-9 2011, vol. 7086, pp. 346-361

## Publications of the year

### Doctoral Dissertations and Habilitation Theses

[14] X. SHI. , *Certification of an Instruction Set Simulator*, Université de Grenoble, July 2013, http://hal.inria.fr/tel-00937524

### International Conferences with Proceedings

[15] J.-F. MONIN, X. SHI. *Handcrafted Inversions Made Operational on Operational Semantics*, in "ITP 2013 - 4th International Conference Interactive Theorem Proving", Rennes, France, S. BLAZY, C. PAULIN-MOHRING, D. PICHARDIE (editors), LNCS - Lecture Notes in Computer Science, Springer, July 2013, vol. 7998, pp. 338-353 [*DOI :* 10.1007/978-3-642-39634-2_25], http://hal.inria.fr/hal-00937168

[16] Q. WANG, B. BARRAS. *Semantics of Intensional Type Theory extended with Decidable Equational Theories*, in "Computer Science Logic 2013", Dagstuhl, Germany, S. R. D. ROCCA (editor), Leibniz International

Proceedings in Informatics (LIPIcs), Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, August 2013, vol. 23, pp. 653–667 [*DOI :* 10.4230/LIPICS.CSL.2013.653], http://hal.inria.fr/hal-00937197

### Other Publications

[17] F. BLANQUI. , *Elements of mathematics and logic for computer program analysis*, March 2013, 37 p. , http://hal.inria.fr/cel-00934160

## References in notes

[18] , *ARM Architecture Reference Manual DDI 0100I*, ARM, 2005

[19] F. GHENASSIA (editor). , *Transaction-Level Modeling with SystemC. TLM Concepts and Applications for Embedded Systems*, Springer, June 2005, ISBN 0-387-26232-6

[20] , *Software Manual, Renesas 32-Bit RISC Microcomputer SuperHTM RISC engine Family*, Renesas, 2006

[21] , *OSCI SystemC TLM 2.0.1*, Open SystemC Initiative, 2009, http://www.systemc.org/

[22] T. ARTS, J. GIESL. *Termination of Term Rewriting Using Dependency Pairs*, in "Theoretical Computer Science", 2000, vol. 236, pp. 133-178

[23] F. BELLARD. *QEMU, A Fast And Portable Dynamic Translator*, in "USENIX Annual Technical Conference", Philadelphia, PA, USA, 2005

[24] F. BLANQUI. , *Terminaison des systèmes de réécriture d'ordre supérieur basée sur la notion de clôture de calculabilité*, Université Paris-Diderot - Paris VII, July 2012, HDR, http://tel.archives-ouvertes.fr/tel-00724233

[25] F. BLANQUI, J.-P. JOUANNAUD, A. RUBIO. *The Computability Path Ordering: the End of a Quest*, in "Proceedings of the 22nd International Conference on Computer Science Logic, Lecture Notes in Computer Science 5213", 2008, Invited paper

[26] A. R. BRADLEY, Z. MANNA, H. B. SIPMA. *What's decidable about arrays*, in "VMCAI '06", E. A. EMERSON, K. S. NAMJOSHI (editors), LNCS, Springer, 2006, vol. 3855, pp. 427–442

[27] D. BURGER, T. M. AUSTIN. *The SimpleScalar tool set, version 2.0*, in "SIGARCH Comput. Archit. News", 1997, vol. 25, n$^{o}$ 3, pp. 13–25, http://doi.acm.org/10.1145/268806.268810

[28] L. CAI, D. GAJSKI. *Transaction level modeling: an overview*, in "CODES+ISSS '03: Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis", New York, NY, USA, ACM Press, 2003, pp. 19–24, http://doi.acm.org/10.1145/944645.944651

[29] Y.-F. CHEN, E. CLARKE, A. FARZAN, M.-H. TSAI, Y.-K. TSAY, B.-Y. WANG. *Automated Assume-Guarantee Reasoning through Implicit Learning*, in "Computer Aided Verification", Royaume-Uni Edinburgh, 2010, http://hal.inria.fr/inria-00496949/en/

[30] E. CLARKE, O. GRUMBERG, D. A. PELED. , *Model Checking*, The MIT PressCambridge, Massachusetts, 1999

[31] B. CMELIK, D. KEPPEL. *Shade: a fast instruction-set simulator for execution profiling*, in "SIGMETRICS Perform. Eval. Rev.", 1994, vol. 22, n$^{o}$ 1, pp. 128–137, http://doi.acm.org/10.1145/183019.183032

[32] J. M. COBLEIGH, G. S. AVRUNIN, L. A. CLARKE. *Breaking Up is Hard to do: An Evaluation of Automated Assume-Guarantee Reasoning*, in "ACM Trans. Software Engineering Methodology", 2008, vol. 17, n$^{o}$ 2

[33] J. M. COBLEIGH, D. GIANNAKOPOULOU, C. S. PĂSĂREANU. *Learning Assumptions for Compositional Verification*, in "TACAS", H. GARAVEL, J. HATCLIFF (editors), Lecture Notes in Computer Science, Springer Verlag, 2003, vol. 2619, pp. 331–346

[34] J. D'ERRICO, W. QIN. *Constructing portable compiled instruction-set simulators: an ADL-driven approach*, in "DATE '06: Proceedings of the conference on Design, automation and test in Europe", 3001 Leuven, Belgium, Belgium, European Design and Automation Association, 2006, pp. 112–117

[35] L. FENG, M. KWIATKOWSKA, D. PARKER. *Compositional Verification of Probabilistic Systems using Learning*, in "QEST", G. CIARDO, R. SEGAL (editors), IEEE CS Press, 2010

[36] F. FUMMI, G. PERBELLINI, M. LOGHI, M. PONCINO. *ISS-centric modular HW/SW co-simulation*, in "ACM Great Lakes Symposium on VLSI", 2006, pp. 31-36

[37] A. GAVARE. , *GXemul Documentation*, 2007, http://gxemul.sourceforge.net/gxemul-stable/doc/index.html

[38] P. GERIN, S. YOO, G. NICOLESCU, A. A. JERRAYA. *Scalable and flexible cosimulation of SoC designs with heterogeneous multi-processor target architectures*, in "ASP-DAC '01: Asia South Pacific Design Automation Conference", ACM, 2001, pp. 63–68

[39] IEEE. , *IEEE Standard 1666 - SystemC Language Reference Manual*, IEEE, 2006

[40] Y. JUNG, S. KONG, B.-Y. WANG, K. YI. *Deriving Invariants by Algorithmic Learning, Decision Procedures, and Predicate Abstraction*, in "Verification, Model Checking, and Abstract Interpretation", Espagne Madrid, 2010, http://hal.inria.fr/inria-00517257/en/

[41] S. KONG, Y. JUNG, C. DAVID, B.-Y. WANG, K. YI. *Automatically Inferring Quantified Loop Invariants by Algorithmic Learning from Simple Templates*, in "ASIAN Symposium on Programming Languages and Systems", Chine Shanghai, K. UEDA (editor), 2010, http://hal.inria.fr/inria-00515166/en/

[42] D. KROENING, O. STRICHMAN. , *Decision Procedures: An Algorithmic Point of View*, Springer, 2008, ISBN-10: 3540741046

[43] M. MEERWEIN, C. BAUMGARTNER, T. WIEJA, W. GLAUERT. *Embedded systems verification with FGPA-enhanced in-circuit emulator*, in "ISSS '00: Proceedings of the 13th international symposium on System synthesis", Washington, DC, USA, IEEE Computer Society, 2000, pp. 143–148, http://doi.acm.org/10.1145/501790.501821

[44] G. NELSON. , *Techniques for program verification*, Stanford UniversityStanford, CA, USA, 1980

[45] A. NOHL, G. BRAUN, O. SCHLIEBUSCH, R. LEUPERS, H. MEYR, A. HOFFMANN. *A universal technique for fast and flexible instruction-set architecture simulation*, in "DAC '02: Proceedings of the 39th conference

on Design automation", New York, NY, USA, ACM, 2002, pp. 22–27, http://doi.acm.org/10.1145/513918.513927

[46] M. PONCINO, J. ZHU. *DynamoSim: a trace-based dynamically compiled instruction set simulator*, in "ICCAD '04: Proceedings of the 2004 IEEE/ACM International conference on Computer-aided design", Washington, DC, USA, IEEE Computer Society, 2004, pp. 131–136, http://dx.doi.org/10.1109/ICCAD.2004.1382557

[47] M. RESHADI, P. MISHRA, N. DUTT. *Instruction set compiled simulation: a technique for fast and flexible instruction set simulation*, in "DAC '03: Proceedings of the 40th conference on Design automation", New York, NY, USA, ACM, 2003, pp. 758–763, http://doi.acm.org/10.1145/775832.776026

[48] P. SCHAUMONT, D. CHING, I. VERBAUWHEDE. *An interactive codesign environment for domain-specific coprocessors*, in "ACM Trans. Des. Autom. Electron. Syst.", 2006, vol. 11, n$^{o}$ 1, pp. 70–87, http://doi.acm.org/10.1145/1124713.1124719

[49] R. SEBASTIANI. *Lazy satisfiability modulo theories*, in "Journal on Satisfiability, Boolean Modeling and Computation", 2007, vol. 3, n$^{o}$ 3-4, pp. 141–224

[50] H. SHEINI, K. SAKALLAH. *From propositional satisfiability to satisfiability modulo theories*, in "Theory and Applications of Satisfiability Testing-SAT 2006", 2006, pp. 1–9

[51] P.-Y. STRUB. , *Type Theory and Decision Procedures*, École Polytechnique, July 2008

[52] COQ. D. TEAM. , *The Coq Reference Manual, Version 8.2*, Inria Rocquencourt, France, 2008, http://coq.inria.fr/

[53] L. DE MOURA, B. DUTERTRE, N. SHANKAR. *A tutorial on satisfiability modulo theories*, in "CAV'07: Proceedings of the 19th international conference on Computer aided verification", Berlin, Heidelberg, Springer-Verlag, 2007, pp. 20–36

[54] W.-P. DE ROEVER, F. DE BOER, U. HANNEMAN, J. HOOMAN, Y. LAKHNECH, M. POEL, J. ZWIERS. , *Concurrency Verification: Introduction to Compositional and Noncompositional Methods*, Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 2001, n$^{o}$ 54

[55] V. VAN OOSTROM. *Confluence by Decreasing Diagrams*, in "RTA", A. VORONKOV (editor), Lecture Notes in Computer Science, Springer, 2008, vol. 5117, pp. 306-320