Activity Report 2011

Team FORMES

FORmal Methods for Embedded Systems

RESEARCH CENTER
Paris - Rocquencourt

THEME
Programs, Verification and Proofs

Table of contents

Team FORMES

Keywords: Interactive Theorem Proving, Formal Methods, Safety

FORMES [1] *is one of the projects of the LIAMA consortium*[2]*. It is funded by CNRS, INRIA and Tsinghua University*[3]*, and located at Tsinghua University, Beijing, China. It was created on September 2008 by extending with formal methods Vania Joloboff's DeviceWare project on system-on-chip simulation started in 2007.*

1. Members

Research Scientists

Frédéric Blanqui [CR1 INRIA]
Ming Gu [Tsinghua professor, HdR]
Fei He [Tsinghua assistant professor]
Vania Joloboff [DR INRIA]
Jean-Pierre Jouannaud [DR INRIA and Tsinghua software chair, team leader, HdR]
Jean-François Monin [DR CNRS, HdR]

PhD Students

Hui Kong [Tsinghua]
Jiaxiang Liu [Tsinghua since September 1]
Kim-Quyen Ly [UJF Grenoble]
Xiaomu Shi [UJF Grenoble]
Hai Wan [Tsinghua until January 31]
Qian Wang [Tsinghua and École Polytechnique]
Rui Wang [Tsinghua]
Liangze Yin [Tsinghua]
Lianyi Zhang [Tsinghua]
Min Zhou [Tsinghua]
Litian Xiao [Tsinghua]

Post-Doctoral Fellows

Jianqi Li [Tsinghua]
Guillaume Merle [INRIA]
Hai Wan [Tsinghua since February 1]
Sidi Ould Biha [INRIA until July 31]
Hehua Zhang [Tsinghua]

Visiting Scientist

Bow-Yaw Wang [INRIA visiting professor and Tsinghua invited professor until July 31]

Administrative Assistants

Lin Cui [Tsinghua, part time]
Mei Zhang [LIAMA, part time]

Others

Meixian Chen [Master Shanghai Jiaotong]
Xiaowei Gao [Master Tsinghua]
Sen Guo [Master Guangxi until February 28]
Yu Jiang [Master Tsinghua]
William Kilque [Master CPE Lyon until September 4]

---

[1] *http://formes.asia*
[2] *http://liama.ia.ac.cn*
[3] *http://www.tsinghua.edu.cn*

Jiaxiang Liu [Master Tsinghua until August 31]
Mengqi Liu [Master Tsinghua since September 1]
Shengpeng Liu [Master Tsinghua since September 1]
Wenrui Meng [Master Tsinghua]
Lifan Su [Master Tsinghua since September 1]
Xia Wu [Master Tsinghua since September 1]
Huiying Luo [Master Tsinghua until June 30]
Peng Shan [Master Guangxi until February 28]
Frédéric Tuong [Master Paris 7 until October 10]
Shenpeng Wang [Master Tsinghua]
Yuhui Wang [Master Tsinghua until July 31]
Yang Yu [Master Guangxi]
Xuke Zhang [Master Tsinghua]
Zuyu Zhang [Master Harbin]
Xinlei Zhou [Master Beihang]
Lei Zhu [Master Tsinghua]

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 plan to rely on formal proofs as well for assessing correctness of the elaboration of program representation abstractions from object code. We plan to rely on simulations for testing the whole embedded system, and to formal proofs to verify the completeness of test sets. We finally plan to 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.

3. Scientific Foundations

3.1. Historical context

The project FORMES was created in September 2008, by union of three different smaller groups which origin and interests 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.

3.2. 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 [63] 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 [48], [50], [69]. 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 named SimSoC, which uses a single simulation loop thanks to Transaction Level Modeling (TLM) [38], [30] 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 [37]. Another technique to implement a fast ISS is *dynamic translation* [41], [68], [45] which has been favored by many [66], [45], [67], [68] 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 [34] and GXEMUL [49] 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.

3.3. Formal proofs

Coq [44] 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[4] [72]. 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.

[4]The thesis was supported by the "Fondation EADS"

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[5]) or C (tool FRAMA-C[6]) 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[7]. 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) to ARM or PowerPC developed in the COMPCERT INRIA project[8]. 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.

Some of these challenges require expertise in both rewriting and type theory. To maintain this combined expertise in FORMES, we also 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. First and higher-order rewriting deal with relations on sets (abstract rewriting), term algebras (first-order rewriting), and binding algebras (higher-order rewriting), which are generated by a (usually finite) set of pairs. Important problems are few: termination (also called strong normalization) is the property of non-existence of infinite computations; confluence is the property that rewriting computations, although non-deterministic, return a unique result, hence define functions; Subject reduction is the property that computations preserve types. Since the third is usually easy to check, we are mostly interested in confluence and termination.

3.4. Verification

Model checking is an automatic formal verification technique [40]. 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.

[5] http://why.lri.fr
[6] http://frama-c.com
[7] http://why.lri.fr
[8] http://compcert.inria.fr

Compositional reasoning is a way to ameliorate the complexity in abstract models [77]. 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 [43]. 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 several order of magnitudes. 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 [42].

Recently, we propose assume-guarantee reasoning with implicit learning [39]. 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 [56], [57]. 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. Recently, the L^* algorithm is applied in assume-guarantee reasoning for probabilistic automata [47]. 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.

3.5. 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 [58] 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. On such, easy and popular fragment, is propositional (or Boolean) logic, 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) [76], 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 [70], and *eager* method [71]. 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 [64], [36]. 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 which sophistication makes their correct implementation a delicate task. Moreover, SMT solvers themselves employ a quite complex machinery, making them error prone as well[9] 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.

[9] 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.

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. Application domains

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.

5. Software

5.1. aCiNO

Participants: Fei He [correspondant], Min Zhou.

aCiNO is an SMT (Satisfiability Modulo Theory) solver based on a Nelson-Oppen [65] architecture, and written in C++. Currently, two popular theories are considered: linear real arithmetic (LRA) and uninterpreted functions (UF). A lazy approach is used for solving SMT problem. For efficiency consideration, the solver is implemented in an incremental way. It also invokes an online SAT solver, which is now a modified MiniSAT, so that recovery from conflict is possible.

5.2. CoLoR and Rainbow

Participants: Frédéric Blanqui [correspondant], Kim-Quyen Ly, Sidi Ould Biha.

CoLoR is a Coq [44] library on rewriting theory and termination of nearly 70,000 lines of code [11]. it provides definitions and theorems for:

- Mathematical structures: relations, (ordered) semi-rings.
- Data structures: lists, vectors, polynomials with multiple variables, finite multisets, matrices.
- Term structures: strings, algebraic terms with symbols of fixed arity, algebraic terms with varyadic symbols, 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.

Rainbow is a tool for automatically certifying termination certificates expressed in the CPF XML format [29] used in the termination competition on termination [32]. Termination certificates are translated and checked in Coq by using the CoLoR library.

CoLoR and Rainbow are distributed under the CeCILL license on http://color.inria.fr/. Various people participated to its development (see the website for more information).

5.3. EDOLA

Participants: Hehua Zhang [correspondant], Ming Gu, Hui Kong, Yu Jiang.

Joint work with Jiaguang Sun (Tsinghua University, China).

EDOLA [26] is an integrated tool for domain-specific modeling and verification of PLC applications [74]. It is based on a domain-specific modeling language to describe system models. It supports both model checking and automatic theorem proving techniques for verification. The goal of this tool is to possess both the usability in domain modeling, the reusability in its architecture and the capability of automatic verification.

For the moment, we have developed a prototype of the EDOLA language, which can easily describe the features of PLC applications like the scan cycle mechanism, the pattern of environment model, time constraints and five property patterns. TLA+ [59] was chosen as the intermediate language to implement the automatic verification of EDOLA models. A prototype of EDOLA has also been developed, which comes along with an editor to help writing EDOLA models. To automatically verify properties on EDOLA models, it provides the interface for both a model checker TLC [59] and a first-order theorem prover SPASS [75].

5.4. Moca

Participant: Frédéric Blanqui [correspondant].

Joint work with Pierre Weis (INRIA Rocquencourt) and Richard Bonichon (CEA).

Moca is a construction functions generator for OCaml [60] 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 user's define 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 on http://moca.inria.fr/.

5.5. SimSoC

Participant: Vania Joloboff [correspondant].

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 of SimSoC is based on the open source libraries from the OSCI Consortium: SystemC version 2.2 and TLM 2.0.1 [54], [33]. 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 (version 2.2), hence the simulator has a single simulation loop.

The second open source version of SimSoC, SimSoC v0.7.1, has been released in November 2010. It contains a full simulator for ARM V5 and PowerPC both running at an average speed of about 80 Millions instructions per second in, and a simulator for the MIPS architecture with an average speed of 20 Mips in mode DT1. It represents about 70,000 lines of source code and includes:

- Instruction Set Simulators. The ARM Version 5 architecture has been implemented with DT0, DT1, DT2 mode. The ARM and PowerPC 600 architecture with DT0 and DT1 mode. For both architectures, complete simulation models of the processor and MMU are provided, making it possible to run operating systems of the simulated platform. MIPS architecture in DT0 mode is under development.
- A dynamic translator from binary programs to an internal representation. For the ARM architecture a compiler has been developed that generates the C++ translated code (for DT2), using parametrized specialization options.
- Peripheral models including a serial line controller, a flash memory controller, an interrupt controller.
- A utility to generate permanent storage for flash memory simulation; a compiler tool to generate instruction binary decoder.
- Examples illustrating the use of the library and infrastructure.

SimSoC is distributed under LGPL on https://gforge.inria.fr/projects/simsoc.

5.6. SimSoC-Cert

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

SimSoC-Cert is a set of tools that can automatically generate in various target languages (Coq and C) the decoding functions and the state transition functions of each instruction and addressing mode of the ARMv6 architecture manual [28] (implemented by the ARM11 processor family) but the Thumb and coprocessor instructions. The input of SimSoC-Cert is the ARMv6 architecture manual itself.

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 v0.7.

We can also generate similar programs for SH4 [31] but this is still under test.

6. New Results

6.1. Simulation

6.1.1. Simulation of vector architecture

Participants: Vania Joloboff, Yang Yu.

Many architectures including PowerPC and ARM now have *vectorized* instructions, that is, instructions that can execute on several data items in parallel (e.g 8 simultaneous additions) on specific vector data.

We have implemented the ALTIVEC extension of the PowerPC to support the vector instructions.

6.1.2. Native translation using LLVM

Participants: Vania Joloboff, Xinlei Zhou, Zuyu Zhang.

We have started to implement a new technique of dynamic translation. This new method consists in decompiling the binary object code into an abstract representation and recompiling it to native host code.

The decompilation of the program amounts to reconstructing the simulated program Control Flow Graph using an intermediate representation. We have chosen LLVM (Low Level Virtual Machine), defined by University of Illinois, and now widely adopted in many projects, as our representation format. Using LLVM allows us to directly use the LLVM Intel code generator.

The SimSoC binary decoder has been modified to identify basic blocks (blocks of sequential instructions ending with a branch instruction). After instructions have been grouped into basic blocks, they are translated into an LLVM representation and finally the LLVM compiler is called to generate native code.

A first version of this technique has been implemented for both the ARM and Power Architecture. We have reach a considerable speed improvement in the generated code, with the execution speed multiplied by factor of 2 to 8. However the translation time from binary to LLVM and from LLVM to native code is significant (translation speed is roughly 1000 instructions per second). Consequently the overall speed is improved by only a factor of 20 to 50 percent when the simulation are relatively short test programs [20].

In order to reach still higher simulation speed we need to use a more sophisticated analysis of the control flow graph. The idea is to do an edge profiling analysis of the basic blocks in order to identify larger blocks. This work is under development.

Another idea is to use multi-processor hosts machine to parallelize translation from LLVM to native code. This is also under investigation.

6.1.3. Trace Analysis

Participants: Guillaume Merle, Vania Joloboff.

Simulation sessions produce huge trace files, sometimes now in hundreds of gigabytes, that are hard to analyze with a quick response time. This comes down to two sub-problems:

- The trace file size. Trace files are huge because they include lots of information. But when looking for a specific problem, one does not need all of this information. To search one given defect, one may ignore a large amount of the data in the trace file. One would like the trace file to contain only relevant information to the concerned problem.
- The expressive power of the language to analyze the trace, and its usability. If the language is limited to expression search, it is easy to use but hard to construct sophisticated formulas. If the language used is Linear Temporal Logic (LTL), there is a lot of expressive power but many engineers are unable to write a LTL formula and to maintain it over time.

We would like to build a trace analysis tool that includes a language which allows expression of time-related formulas but is simple to formulate expressions. When this language is compiled, ideally the compiler is smart enough to identify independent formulae, the search of which can be parallelized, and it is also smart enough to generate "filter scripts".

When compiling one trace language input file, it would generate, from one input file, N filter scripts and N analyzers. Then during the simulation, the huge raw trace file is actually split into N smaller trace files, each relevant to one problem only, filtering out all unnecessary data. Hence trace files sizes would be considerably reduced.

We have started to design a trace language and a compiler, and extended the SimSoC simulator to support generation of trace files with a filter.

A first version of the trace language compiler has been coded in OCAML.

In the current version under development, the filters are not generated but coded manually, and filters are not parallelized.

6.1.4. Generation of simulators from vendor specification

Participants: Frédéric Blanqui, Vania Joloboff, Jean-François Monin, Xiaomu Shi, Frédéric Tuong.

Starting last year, we undertook the task of generating automatically an instruction set simulator (ISS) from the vendor specification in a PDF file. In order to generate the C code of the simulator, it is assumed such vendor specification contains at least some formal definitions of the instruction set that can be analyzed. It is the case to a wide extent for the ARM, the PowerPC and the SH architectures.

The process of generating the simulator consists of 4 major steps, first eliminating from the PDF file irrelevant information, next construct from the relevant data an abstract syntax representation of the instruction set, then to generate the C code of the simulator, using some additional data provided manually to complete the vendor specification.

This work was completed last year for the ARM architecture with the documentation form ARM corporation [35]. This year, we did similar work for the SH architecture from specification from RENESAS corporation.

We have indeed generated a simulator for the SH4 architecture [31], which has not been fully tested yet.

However, this works has proved that the abstract syntax we have defined is powerful enough to describe two different architectures with significant differences in the way they are described by the vendor.

6.1.5. First steps towards the certification of an ARM simulator

Participants: Frédéric Blanqui, Jean-François Monin, Xiaomu Shi, Frédéric Tuong.

The simulation of Systems-on-Chip (SoC) is nowadays a hot topic because, beyond providing many debugging facilities, it allows the development of dedicated software before the hardware is available. Low-consumption CPUs such as ARM play a central role in SoC. However, the effectiveness of simulation depends on the faithfulness of the simulator. To this effect, in [24], we propose here to prove significant parts of such a simulator, SimSoC. Basically, on one hand, we develop a Coq formal model of the ARM architecture while on the other hand, we consider a version of the simulator including components written in Compcert-C [61]. Then we prove that the simulation of ARM operations, according to Compcert-C formal semantics, conforms to the expected formal model of ARM. Size issues are partly dealt with using automatic generation of significant parts of the Coq model and of SimSoC from the official textual definition of ARM. However, this is still a long-term project. We report here the current stage of our efforts and discuss in particular the use of Compcert-C in this framework.

6.2. Type and rewriting theory

6.2.1. A type theory for Coq

Participants: Jean-Pierre Jouannaud, Qian Wang.

In this joint work with Bruno Barras and Pierre-Yves Strub [17], we describe an abstract model of CoqMT [73] called CoqMTU, which puts together the Calculus of Inductive Constructions, decidable first-order theories, and an infinite hierarchy of universes which are all predicative but the first impredicative universe of propositions. We have shown its consistency, strong normalization and decidability of type checking in presence of weak elimination (and absence of strong elimination). An important feature of this work is that the first-order theory is abstract, characterized by the three natural axioms that (i) it is non-degenerated (its models have at least two elements), (ii) constructors are free, and (iii) defined symbols are completely defined. On the theoretical side, this allows us to give an abstract elimination principle for such non-canonical theories. On the practical side, this justifies the implementation of CoqMT in which decidable theories can be dynamically dowloaded. It should be noticed that these proofs are done in Coq, except for the strong normalization part. Qian Wang is now continuing this work at Ecole Polytechnique with Bruno Barras and Pierre-Yves Strub, the target being strong normalization.

6.2.2. *Confluence by decreasing diagrams*

Participants: Jean-Pierre Jouannaud, Huiying Luo, Jiaxiang Liu.

Invented by Vincent Van Oostrom, decreasing diagrams capture both kinds of diagrams arising from Newmann's Lemma and Hindley's Lemma: they indeed allow to reduce all known confluence methods to critical pairs computations, and a search of decreasing diagrams for them all, where decreasingness is measured by a well-founded order on proof steps.

In [55], we give a new simple proof of Van Oostrom's main theorem, and extend the method of decreasing diagrams to rewrite relations on a term algebra. We prove that the union of a terminating left-linear systems, and a non-terminating linear system is confluent provided the various critical pairs existing in in their combination have decreasing diagrams (with respect to some order built from the respective orders of both systems).

During this year, we have further simplified and generalized these results in order to get rid of the left-linearity assumption for the first system, and of the right-linearity assumption for the second. This yields a true generalization of the well-known Knuth-Bendix-Huet confluence result for terminating systems, and at the same time of various critical-pair based results found in the literature for non-terminating systems.

6.2.3. *Confluence of normal rewriting*

Participants: Jean-Pierre Jouannaud, Jianqi Li.

Confluence results for first-order and higher-order rewriting differ in many ways: by the rewriting relation used, and by the strong normalization assumption made. We believe that these differences hide the strong similarities of these (and other) kinds of rewriting.

In this work, we introduce a new notion of rewriting, *normal rewriting*, which aims at capturing all known results reducing confluence to critical (and extension) pair computations in presence of some termination assumption.

We achieve this goal in the following way. First, we consider theories made of a set R of *rules*, a set S of *simplifiers*, and a set E of *equations*. Rewriting operates on terms in S modulo E normal forms, and uses $S \cup E$-pattern matching for firing the rules in R, before to normalize the result with respect to S modulo E. Termination is assumed for the union of S modulo E and R modulo $S \cup E$. Second, we introduce relations on an abstract set of terms, and an abstract, well-founded set of positions, and reduce the Church-Rosser property of abstract normal rewriting to abstract notions of critical pairs and extensions. We can then apply this result to first-order rewriting, as well as to various forms of higher-order rewriting. These results capture plain rewriting ($S \cup E = \varnothing$), Stickel's rewriting modulo ($S = \varnothing$), Nipkow's higher-order reswriting (S is made of beta-reduction and eta-expansion, and E is alpha-conversion), and allow to describe new forms of first and higher-order rewriting relations.

6.2.4. *Argument filterings and usable rules in higher-order rewrite systems*

Participant: Frédéric Blanqui.

Joint work with Keiichirou Kusakari and Sho Suzuki from Nagoya University, Japan.

The static dependency pair method is a method for proving the termination of higher-order rewrite systems *à la* Nipkow [62]. It combines the dependency pair method introduced for first-order rewrite systems with the notion of strong computability introduced for typed lambda-calculi [52]. Argument filterings and usable rules are two important methods of the dependency pair framework used by current state-of-the-art first-order automated termination provers [51], [53]. In [12], we extend the class of higher-order systems on which the static dependency pair method can be applied. Then, we extend argument filterings and usable rules to higher-order rewriting, hence providing the basis for a powerful automated termination prover for higher-order rewrite systems.

6.3. Decision procedures

6.3.1. A certificate framework for DPLL(T)

Participants: Min Zhou, Fei He, Bow-Yaw Wang, Wenrui Meng.

Satisfiability Modulo Theories (SMT) techniques are widely used nowadays. SMT solvers are used to decide the satisfiability of first-order formulas. When an SMT solver is invoked, it is important to ensure correctness of the result. For this purpose, we proposed a certificate framework based on DPLL(T), including generation of certificates and verification of certificates. Some properties are discussed and proved theoretically. The certificate is easy to generate because it only needs minor modification to the existing SMT solvers. Experiment results show that the overhead for certificates generation is only 10%. Moreover, verifying the certificate requires few memory and time, which outperforms other approaches.

6.3.2. Automated verification of termination certificates

Participants: Frédéric Blanqui, Kim-Quyen Ly, Sidi Ould Biha.

The research community on rewriting developed a grammar for termination certificates called CPF [29] (given by a XML Schema file). Our goal is to develop a safe, modular and efficient termination certificate verifier based on the formal library of mathematical results on termination called CoLoR that has been developed for the proof assistant Coq [11].

Because the CPF format is regularly modified and extended with new features, it is useful to have a tool that can automatically generate data structures, parsers and pretty-printers for that format. Hence, we developed a first version of such a tool in OCaml.

Once we got a representation of termination certificates in Coq, we could start defining a boolean function checking the correctness of a certificate, and formally prove its correctness. For the moment, we only considered the case of polynomial interpretations on integers. The proof is almost finished. To do so, we had to modify some of the CoLoR files to be able to use its results (transformation of modules into records that are first-class objects). The use of dependent types in CoLoR makes also definitions and proofs much more difficult.

6.3.3. Proving computational geometry algorithms in TLA+2

Participants: Hui Kong, Hehua Zhang, Ming Gu.

Geometric algorithms are widely used in many scientific fields like computer vision, computer graphics. To guarantee the correctness of these algorithms, it is important to apply formal method to them. In this work, we propose an approach to proving the correctness of geometric algorithms [22]. The main contribution is that a set of proof decomposition rules is proposed which can help improve the automation of the proof of geometric algorithms. We choose TLA+2, a structural specification and proof language, as our experiment environment. The case study on a classical convex hull algorithm shows the usability of the method.

6.4. Compositional verification

6.4.1. BDD-based assume-guarantee reasoning through implicit learning

Participants: Fei He, Bow-Yaw Wang, Lei Zhu.

We present a purely BDD-based assume-guarantee reasoning technique to improve the scalability of symbolic model checking. The new technique adopts a BDD learning algorithm to generate BDD's as contextual assumptions. A new witness analysis algorithm is proposed to exploit the multitude of traces returned by symbolic model checkers. Using the classification tree-based BDD learning algorithm to generate contextual assumptions, we compare assume-guarantee reasoning with monolithic symbolic model checking. The new technique always infers smaller contextual assumptions than contexts in our experiments.

6.4.2. Predicate generation for learning-based loop invariant inference

Participant: Bow-Yaw Wang.

We address the predicate generation problem in the context of loop invariant inference. Motivated by the interpolation-based abstraction refinement technique, we apply the interpolation theorem to synthesize predicates implicitly implied by program texts. Our technique is able to improve the effectiveness and efficiency of the learning-based loop invariant inference algorithm in [21]. Experiments excerpted from Linux, SPEC2000, and Tar source codes are reported.

This is a joint work with Yungbum Jung, Wonchan Lee, and Kwangkuen Yi of Seoul National University, South Korea.

6.4.3. Thread-modular model checking with iterative refinement

Participants: Wenrui Meng, Fei He, Bow-Yaw Wang.

Thread-modular analysis is an incomplete compositional technique for verifying concurrent systems. The heuristic works rather well when there is limited interaction among system components. In this project, we develop a refinement algorithm that makes thread-modular model checking complete. Our algorithm refines abstract reachable states by exposing local information through auxiliary variables. The experiments show that our complete thread-modular model checking can outperform other complete compositional reasoning techniques.

6.5. Specification and verification of TLA+ and PLC systems

6.5.1. Formal semantics of PLC programming languages

Participants: Sidi Ould Biha, Litian Xiao, Ming Gu.

We formalized a semantics of the Instruction List (IL) language, one of the five programing languages defined in the IEC 61131-3 standard for PLC programing [23]. This semantics support a significant subset of the IL language that includes on-delay timers. This semantics was used in a join work to with Jan Olaf Blech from Fortiss (Germany) to prove some safety properties for a real industrial example of PLC program [18].

A second widely used language for programming PLC is the graphical language Ladder Diagrams (LD). We defined a formal semantics of LD in the proof assistant Coq. Based on this semantics and the IL one, we developed a translation function from LD to IL. We also proved a semantic preservation property for this translation function. We have now a certified compilation function from the graphical language LD to IL. This work opens the way for the development of a certified compilation chain for PLC. A journal paper about this work and others aspects of PLC certification is under reviewing.

In [16], [15], we study the definition of denotational semantics on PLC program language, which is convenient to PLC programs modeling and model checking. The purpose of the work is the correctness verification on PLC programs by formal methods. Based on the extended λ-calculus definition, this work has defined the configuration of PLC program architecture, denotational semantics of PLC programs and functions of denotational semantics. It is the basis of model checking and theorem proving.

6.5.2. Formalization and verification of PLCs

Participants: Hai Wan, Litian Xiao, Ming Gu.

PLCs are widely used in embedded systems. Timers play a pivotal role in PLC real-time applications. The formalization of timers is of great importance. In [13], we present a formalization of PLC timers in the theorem proving system Coq, in which the behaviors of timers are characterized by a set of axioms at an abstract level. The authors discuss how to model timers at a proper and sound abstract level. PLC programs with timers are modeled. As a case study, a quiz machine problem with a timer is investigated. This work demonstrates the complexity of formal timer modeling.

In [25], we modeled kernel data type and basic statements and and the denotational semantics of PLC program in Coq. It has given the correctness proof of PLC program based on theorem proving, i.e. based on semantics function the relationship of configuration between the before codes execution and the after is proved. The main purpose is to prove whether a PLC program satisfies certain nature within a scan period.

6.5.3. Synthesis of PLC programs

Participants: Rui Wang, Ming Gu.

PLCs are complex cyber-physical systems which are widely used in industry. In [14], we present a robust approach to design and implement PLC-based embedded systems. Timed automata are used to model the controller and its environment. We validate the design model with resort to model checking techniques. We propose an algorithm to generate PLC code from timed automata and implement this algorithm with a prototype tool. This method can condense the developing process and guarantee the correctness of PLC programs. A case study demonstrates the effectiveness of the method.

6.5.4. Domain-driven probabilistic analysis of PLCs

Participants: Hehua Zhang, Yu Jiang, Ming Gu.

Programmable Logic Controllers are widely used in industry. Reliable PLCs are vital to many critical applications. We present a novel symbolic approach for analysis of PLC systems [27]. The main components of the approach consists of: (1) calculating the uncertainty characterization of the PLC systems, (2) abstracting the PLC system as a Hidden Markov Model, (3) solving the Hidden Markov Model using domain knowledge, (4) integrating the solved Hidden Markov Model and the uncertainty characterization to form an integrated (regular) Markov Model, and (5) harnessing probabilistic model checking to analyze properties on the resultant Markov Model. The framework provides expected performance measures of the PLC systems by automated analytical means without expensive simulations. Case studies on an industrial automated system are performed to demonstrate the effectiveness of our approach.

6.5.5. Edola: a domain modeling and verification language for PLCs

Participants: Hehua Zhang, Ming Gu.

Formal modeling and verification of PLC systems become paramount in engineering applications. The work presents a novel PLC domain-specific modeling language Edola [26]. Important characteristics of PLC embedded systems, such as reactivity, scan cycling, real-time and property patterns, are embodied in the language design. Formal verification methods, such as model checking and automatic theorem proving, are supported in Edola modeling. The TLA+ specification language constitutes an intermediate language layer between Edola and the verification tools, enhancing a large degree of reusability. A prototype IDE for Edola and its seamless integration of a model checker TLC and an automatic theorem prover Spass are implemented. A case study illustrates and validates the applicability of the language.

6.6. Distributed algorithms

6.6.1. Formal model and proofs for Netlog protocols

Participants: Meixian Chen, Jean-François Monin.

Joint work with Yuxin Deng (Jiaotong University, Shanghai) and Stéphane Grumbach (LIAMA/Netquest).

Netlog is a language designed and implemented in the Netquest project for describing protocols. Netlog has a precise semantics, provides a high level of abstraction thanks to its Datalog flavor and benefits from an efficient implementation. This makes it a very interesting target language for proofs of protocols. Netlog comes with two possible semantics: a synchronous semantics, better suited to tightly coupled parallel systems and an asynchronous semantics, better suited to distributed systems.

We designed a formal model of Netlog in Coq, where the two possible semantics are derived from common basic blocks. In a fully certified framework, a formal proof of the Netlog engine (running on each node) would be required. We don't attack this part at the moment: we assume that the implementation respects the general properties stated in our model and focus on the issues raised by the distributed model of computation provided by Netlog.

As a proof of concept, we applied in 2010 this framework to an algorithm constructing a Breadth-First Search Spanning Tree (BFS) in a distributed system [46]. This work has been slightly improved this year and published in [19].

Moreover, we generalized the model in order to take the removal of datalog facts into account, and started to use this feature for more complicated protocols. In main one under study is Prim's algorithm (publication under submission), and we target next GHS, which still resists to palatable proof techniques.

7. Contracts and Grants with Industry

7.1. Schneider Electric

The goal of this project contracted with Schneider Electric China is to develop a full system simulator for a System-on-Chip used by Schneider Electric in their automation product line.

7.2. Orange IT Labs

The goal of this project is to complete the PowerPC simulator and compare its performance with another simulator used internally by Orange IT Labs.

8. Partnerships and Cooperations

8.1. National Initiatives

- FORMES is part of the working group LTP on Languages, Types and Proofs of the GDR GPL[10], the French research network on software engineering.
- FORMES is part of the working group LAC on Logic, Algebra and Calculus of the GDR IM[11], the French research network on mathematics and computer science.

8.2. International Initiatives

8.2.1. Visits of International Scientists

8.2.1.1. Long-term visitors

- Jean-Jacques Lévy (INRIA, France), director of the MSR-INRIA Joint Center, visited FORMES from September 26 to November 18, gave lectures on reductions and causality.
- Pierre-Louis Curien (PPS, CNRS and University Paris 7) visited FORMES in April and May, and co-organized a working group on rewriting theory and algebra.

[10] http://gdr-gpl.cnrs.fr/
[11] http://www.gdr-im.fr/

- Joseph Sifakis (VERIMAG, France) visited FORMES in March and October and participated to various working groups.

8.2.1.2. Short-term visitors

- Zhang Min (JAIST, Japan) gave a talk on December 20 on algebraic-based verification of a dynamic software updating system.
- Vladimir Voevodsky (IAS Princeton, USA), Fields Medal 2002, gave a talk on December 12 on univalent semantics of constructive type theories.
- Jianhua Gao (ISCAS, China) gave a talk on November 25 on the clausal presentation of theories in deduction modulo.
- Iddo Tzameret (ITCS, Tsinghua University) gave a talk on November 18 on short propositional refutations for dense random 3-CNF formulas.
- Eric Madelaine (INRIA, France) gave a talk on November 11 at Shenzhen SIAT on specification, model generation and verification of distributed applications.
- Jean-Raymond Abrial (ETH, Switzerland) gave a talk on September 9 on modeling, refining and proving with Event-B.
- Graham Steel (LSV, ENS Cachan, France) gave lectures on the security of APIs at Tsinghua University and Nokia from August 22 to August 25.
- Thomas Anberree (Nottingham University at Ningbo, China) gave a talk on June 22 on definable quotients in type theory.
- Hsu-Chun Yen (National Taiwan University) gave a talk on May 20 on two-way transducers and parametrized machines.
- Lijun Zhang (Denmark Technical University) gave a talk on May 13 on ODEs in probabilistic model checking.
- Flemming Nielson (Denmark Technical University) gave a talk on May 13 on model checking as static analysis of modal logic.
- Christian Urban (TU Munich, Germany) gave a talk on April 29 on verifying a regular expression matcher and formal language theory.
- Zhaohui Luo (University of London, UK) visited FORMES in April and gave lectures on type theory from April 13 to April 19.
- On April 11, for the 1st Tsinghua Software Day organized by the FORMES team, we had the following talks: A journey into the semantics of programming languages, by Pierre-Louis Curien; type theory and its application, by Zhaohui Luo; advances towards the formal proof of the classification of finite groups, by Georges Gonthier; from boolean to quantitative theories of software, by Tom Henzinger.
- Joseph Sifakis (VERIMAG, France) gave a talk on March 10 on a vision for computer science: the system perspective.

8.2.2. Participation In International Programs

- SIVES[12] is a French-Chinese ANR-NSFC project for 2009-2011 between INRIA FORMES, Tsinghua University and ST Microelectronics on the development of a "SImulation and Verification based platform for Embedded Systems" (coordinated by Frédéric Blanqui on the French side and Ming Gu on the Chinese side).
- *Logical Frameworks* is a grant from the National Science Foundation of China obtained by Jean-Pierre Jouannaud and Jianqi Li to sustain their work on the subject.

[12] http://formes.asia/cms/sives

9. Dissemination

9.1. Animation of the scientific community

- FORMES organizes a weekly seminar which is a major local forum in the area of formal methods, with a steady participation of colleagues who come from the other nearby research institutions, CASIA, ISCAS and Peking University, to attend the presentations. All seminars are announced on our website, as well as the other relevant local seminars or events, in particular those taking place at ISCAS.
- Jean-Pierre Jouannaud and Zhong Shao (Yale University) have initiated a new conference, the 1st international conference on Certified Programs and Proofs (CPP'11), held on December 7-9 at Kenting, Taiwan. The local organization is done by Tyng-Ruey Chang (Academia Sinica), Yih-Kuen Tsay (NTU) and Bow-Yaw Wang (INRIA and Academia Sinica).
- Vania Joloboff co-organized with Pr John Koo the first Shenzhen International Summer School on Embedded Systems Design, held at Shenzhen SIAT from July 4-8.
- FORMES organized on April 11-12 the 1st Tsinghua Software Day and Tsinghua Student Day on the occasion of Tsinghua's 100 years anniversary with talks by Pierre-Louis Curien, Zhaohui Luo, Georges Gonthier and Tom Henzinger.
- Frédéric Blanqui is member of the Steering Committee of the International Conference on Rewriting Techniques and Applications (RTA) from July 2010 to July 2013.
- Frédéric Blanqui was a PC member of the 22nd International Conference on Rewriting Techniques and Applications (RTA'11), 30 May - 1st June, Novisad, Serbia.
- Jean-Pierre Jouannaud is a member of the LICS organizing committee.
- Jean-Pierre Jouannaud is a member of the editorial board of the International Journal of Software and Informatics (IJSI).
- Jean-Pierre Jouannaud is a guest co-editor of JACM (selection of 3 papers from LICS 2010), and a co-guest editor of LMCS (selection of papers from LICS 2010).
- Jean-Pierre Jouannaud is a member of the advisory committee of Academia Sinica, Taipei, Taiwan.
- Jean-Pierre Jouannaud is PC co-chair of CPP 2011, 7-9 December 2011, Kenting, Taiwan.
- Jean-Pierre Jouannaud participated to the STIC-Asie meeting in Guangdong in June, and the AURA meeting in Hanoi, Vietnam, in November, where he gave talks.

9.2. Teaching

- Vania Joloboff taught a class at Tsinghua University on SystemC and Transaction Level Modeling.
- Vania Joloboff taught a session at Shenzhen International Summer School on Embedded Systems Design.
- Jean-François Monin taught a module on Coq entitled *Introduction to Interactive Proofs of Software* at Tsinghua School Software, 3rd year undergraduate, but also followed by 4th year students, a master student of Beihang and 2 master students of University of Beijing, department of mathematics; volume: 35 hours.

10. Bibliography

Major publications by the team in recent years

[1] F. BLANQUI. *Definitions by rewriting in the Calculus of Constructions*, in "Mathematical Structures in Computer Science", 2005, vol. 15, n^{o} 1, p. 37-92, Journal version of LICS'01 [*DOI :* 10.1017/S0960129504004426], http://hal.inria.fr/inria-00105648/en/.

[2] 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, 2010, Best paper award, http://hal.inria.fr/inria-00546228/en/.

[3] 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, p. 827-859, http://hal.inria.fr/inria-00543157/en/.

[4] 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/.

[5] 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.

[6] 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, p. 197–212, ISBN 3-540-44307-X.

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

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

[9] 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", Madrid, Spain, January 2010, http://hal.inria.fr/inria-00517257/en/.

[10] Y.-K. TSAY, B.-Y. WANG. *Automated Compositional Reasoning of Intuitionistically Closed Regular Properties*, in "International Journal on Foundation of Computer Science", 2009, vol. 20, n^{o} 4, p. 747-762.

Publications of the year

Articles in International Peer-Reviewed Journal

[11] 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, p. 827-859 [*DOI :* 10.1017/S0960129511000120], http://hal.inria.fr/inria-00543157/en.

[12] S. SUZUKI, K. KUSAKARI, F. BLANQUI. *Argument filterings and usable rules in higher-order rewrite systems*, in "IPSJ Transactions on Programming", March 2011, vol. 4, n^{o} 2, p. 1-12, http://hal.inria.fr/inria-00555008/en.

[13] H. WAN, C. GANG, X. SONG, M. GU. *Formalisation and verification of programmable logic controllers timers in Coq*, in "IET Software", February 2011, http://hal.inria.fr/inria-00612410/en.

[14] R. WANG, X. SONG, J. ZHU, M. GU. *Formal modeling and synthesis of programmable logic controllers*, in "Computers in Industry", January 2011, http://hal.inria.fr/inria-00612411/en.

[15] L. XIAO, M. GU, J. SUN. *The Denotational Semantics Definition of PLC Programs Based on Extended λ-Calculus*, in "Communications in Computer and Information Science", July 2011, vol. 176(II), n° 40-46, http://hal.inria.fr/inria-00612409/en.

Articles in National Peer-Reviewed Journal

[16] L. XIAO, M. GU, J. SUN. *A Formal Definition Method of Denotational Semantics and Functions for PLC Program Language*, in "Journal of Central South University (in Chinese)", July 2011, http://hal.inria.fr/inria-00612407/en.

International Conferences with Proceedings

[17] 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, http://hal.inria.fr/inria-00583136/en.

[18] J. O. BLECH, S. OULD BIHA. *Verification of PLC Properties Based on Formal Semantics in Coq*, in "International Conference on Software Engineering and Formal Methods, SEFM 2011", Montevideo, Uruguay, June 2011, http://hal.inria.fr/inria-00601907/en.

[19] Y. DENG, S. GRUMBACH, J.-F. MONIN. *A Framework for Verifying Data-Centric Protocols*, in "DisCoTec 2011 - 6th International Federated Conferences on Formal Techniques for Distributed Systems", Reykjavik, Iceland, R. BRUNI, J. DINGEL (editors), Lecture Notes in Computer Science, Springer, December 2011, vol. 6722, p. 106-120 [*DOI :* 10.1007/978-3-642-21461-5_7], http://hal.inria.fr/hal-00647802/en.

[20] V. JOLOBOFF, X. ZHOU, C. HELMSTETTER, X. GAO. *Fast Instruction Set Simulation Using LLVM-based Dynamic Translation*, in "International MultiConference of Engineers and Computer Scientists 2011", Hong Kong, China, Springer, July 2011, vol. 2188, p. 212-216, http://hal.inria.fr/hal-00646947/en.

[21] Y. JUNG, W. LEE, B.-Y. WANG, K. YI. *Predicate Generation for Learning-Based Quantifier-Free Loop Invariant Inference*, in "TACAS 2011 - Seventeenth International Conference on Tools and Algorithms for the Construction and Analysis of Systems", Saarbruecken, Germany, Lecture Notes in Computer Science, Springer, March 2011, vol. 6605, p. 205-219 [*DOI :* 10.1007/978-3-642-19835-9], http://hal.inria.fr/hal-00648946/en.

[22] H. KONG, H. ZHANG, X. SONG, M. GU, J. SUN. *Proving Computational Geometry Algorithms in TLA+2*, in "5th IEEE International Conference on Theoretical Aspects of Software Engineering(TASE 2011)", Xi'an, China, August 2011, http://hal.inria.fr/inria-00612413/en.

[23] S. OULD BIHA. *A formal semantics of PLC programs in Coq*, in "IEEE Computer Software and Applications, COMPSAC'11", Munich, Germany, July 2011, http://hal.inria.fr/inria-00601906/en.

[24] X. SHI, J.-F. MONIN, F. TUONG, F. BLANQUI. *First steps towards the certification of an ARM simulator using Compcert*, in "First International Conference on Certified Programs and Proofs", Hengchun, Taiwan, Province Of China, December 2011, http://hal.inria.fr/inria-00624833/en.

[25] L. XIAO, M. GU, J. SUN. *The Verification of PLC Program Based on Interactive Theorem Proving Tool COQ*, in "4th IEEE International Conference on Computer Science and Information Technology(ICCSIT2011)", Chengdu, China, June 2011, http://hal.inria.fr/inria-00612408/en.

[26] H. ZHANG, M. GU, X. SONG. *Edola: A Domain Modeling and Verification Language for PLC Systems*, in "The Sixth International Conference on Software Engineering (ICSEA 2011)", Barcelona, Spain, October 2011, http://hal.inria.fr/inria-00612416/en.

[27] H. ZHANG, Y. JIANG, H. WILLIAM N.N., X. SONG, M. GU. *Domain-driven Probabilistic Analysis of Programmable Logic Controllers*, in "13th International Conference on Formal Engineering Methods(ICFEM 2011)", Durham, United Kingdom, October 2011, http://hal.inria.fr/inria-00612414/en.

References in notes

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

[29] *Certification Problem Format*, http://cl-informatik.uibk.ac.at/software/cpf/.

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

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

[32] *Termination Competition*, http://termination-portal.org/wiki/Termination_Competition.

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

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

[35] 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, http://hal.inria.fr/inria-00546228/en/.

[36] 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, p. 427–442.

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

[38] 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, p. 19–24, http://doi.acm.org/10.1145/944645.944651.

[39] 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/.

[40] E. CLARKE, O. GRUMBERG, D. A. PELED. *Model Checking*, The MIT Press, Cambridge, Massachusetts, 1999.

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

[42] 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° 2.

[43] 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, p. 331–346.

[44] COQ DEVELOPMENT TEAM. *The Coq Reference Manual, Version 8.2*, INRIA Rocquencourt, France, 2008, http://coq.inria.fr/.

[45] 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, p. 112–117.

[46] Y. DENG, S. GRUMBACH, J.-F. MONIN. *Towards Verifying Declarative Netlog Protocols with Coq*, 2010, http://hal.inria.fr/inria-00506093/en/.

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

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

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

[50] 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, p. 63–68.

[51] J. GIESL, R. THIEMANN, P. SCHNEIDER-KAMP, S. FALKE. *Mechanizing and Improving Dependency Pairs*, in "Journal of Automated Reasoning", 2006, vol. 37, n° 3, p. 155-203.

[52] J.-Y. GIRARD, Y. LAFONT, P. TAYLOR. *Proofs and Types*, Cambridge University Press, 1988.

[53] N. HIROKAWA, A. MIDDELDORP. *Tyrolean Termination Tool: Techniques and Features*, in "Information and Computation", 2007, vol. 205, n° 4, p. 474-511.

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

[55] 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/.

[56] 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/.

[57] 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/.

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

[59] L. LAMPORT. *Specifying Systems, The TLA+ Language and Tools for Hardware and Software Engineers*, Addison-Wesley, 2002.

[60] X. LEROY, D. DOLIGEZ, J. GARRIGUE, D. RÉMY, J. VOUILLON. *The Objective Caml system release 3.11, Documentation and user's manual*, INRIA, France, 2008, http://caml.inria.fr/.

[61] X. LEROY. *A formally verified compiler back-end*, in "Journal of Automated Reasoning", 2009, vol. 43, n^{o} 4, p. 363-446.

[62] R. MAYR, T. NIPKOW. *Higher-Order Rewrite Systems and their Confluence*, in "Theoretical Computer Science", 1998, vol. 192, n^{o} 2, p. 3-29.

[63] 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, p. 143–148, http://doi.acm.org/10.1145/501790.501821.

[64] G. NELSON. *Techniques for program verification*, Stanford University, Stanford, CA, USA, 1980.

[65] G. NELSON, D. C. OPPEN. *Simplification by cooperating decision procedures*, in "ACM Trans. Program. Lang. Syst.", 1979, vol. 1, n^{o} 2, p. 245–257.

[66] 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, p. 22–27, http://doi.acm.org/10.1145/513918.513927.

[67] 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, p. 131–136, http://dx.doi.org/10.1109/ICCAD.2004.1382557.

[68] 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, p. 758–763, http://doi.acm.org/10.1145/775832.776026.

[69] 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, p. 70–87, http://doi.acm.org/10.1145/1124713.1124719.

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

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

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

[73] P.-Y. STRUB. *Coq Modulo Theory*, in "19th EACSL Annual Conference on Computer Science Logic", Tchèque, République Brno, A. DAWAR, H. VEITH (editors), Springer, 2010, vol. 6247, p. 529–543, http://hal.inria.fr/inria-00497404/en/.

[74] TECHNICAL COMMITTEE NO.65. *IEC 1131 - Programmable Controllers*, International Electrotechnical Commission, 1997.

[75] C. WEIDENBACH, D. DIMOVA, A. FIETZKE, R. KUMAR, M. SUDA, P. WISCHNEWSKI. *SPASS Version 3.5*, in "Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings", R. A. SCHMIDT (editor), Lecture Notes in Computer Science, Springer Verlag, 2009, p. 140-145.

[76] 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, p. 20–36.

[77] 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.