INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team FORMES

FOrmal Methods for Embedded Systems

Paris - Rocquencourt

Theme : Programs, Verification and Proofs

Table of contents

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

1. Team

Research Scientists

Frédéric Blanqui [CR1 INRIA]
Ming Gu [Tsinghua professor]
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]

Technical Staff

Hui Xiao [INRIA until October 8]

PhD Students

Hui Kong [Tsinghua]
Li Li [Tsinghua until July 31]
Kim-Quyen Ly [UJF Grenoble since October 19]
Xiaomu Shi [UJF Grenoble]
Hai Wan [Tsinghua]
Qian Wang [Tsinghua]
Rui Wang [Tsinghua]
Liangze Yin [Tsinghua]
Lianyi Zhang [Tsinghua]
Min Zhou [Tsinghua]

Post-Doctoral Fellows

Claude Helmstetter [INRIA until December 17]
Jianqi Li [Tsinghua]
Guillaume Merle [INRIA since September 15]
Sidi Ould Biha [INRIA since March 1]
Pierre-Yves Strub [INRIA until September 30]
Hehua Zhang [Tsinghua since April 1]

Visiting Scientist

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

Administrative Assistants

Lin Cui [Tsinghua, part time]
Lin Tang [LIAMA part time until June 15]
Mei Zhang [LIAMA part time since June 15]

Others

Meixian Chen [Master Shanghai Jiatong since December 5]
Xiaowei Gao [Master Tsinghua since September 1]
Sen Guo [Master Guangxi from January 2010 to May 2011]
Yu Jiang [Master Tsinghua since September 1]
William Kilque [Master CPE Lyon from September 2010 to Sep 2011]
Jiaxiang Liu [Master Tsinghua since September 1]

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

Liu Liu [Master Tsinghua until July 31]
Huiying Luo [Master Tsinghua]
Kim-Quyen Ly [Master Bordeaux I from May 18 to October 18]
Peng Shan [Master Guangxi from January 2010 to May 2011]
Frédéric Tuong [Master Paris 7 since November 12]
Yuhui Wang [Master Tsinghua]
Yang Yu [Master Guangxi since December]
Xuke Zhang [Master Tsinghua since September 1]
Xianpeng Zhao [Master Tsinghua until July 22]
Xinlei Zhou [Master Beihang since July 1]
He Zhu [Master Tsinghua until July 22]
Jianzhong Zhu [Master Tsinghua until July 22]
Lei Zhu [Master Tsinghua since September 1]
Zuyu Zhang [Master Harbin since December]
Zhen Zhu [Master Tsinghua until July 22]

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.

2.2. Highlights of the period

- The first open source version of the SimSoC[4] simulator has been release in March 2010.
- Since October 2010, SimSoC contains an ARM processor simulator that has been generated automatically from the ARM documentation (see Section 6.1.2).
- CoqMT[5], a new version of Coq allowing dynamic loading of decision procedures returning certificates, has been released in January 2010.

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-Ile-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 cary 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*.

[4] http://gforge.inria.fr/projects/simsoc
[5] http://strub.nu/research/coqmt/

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 [84] 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 [66], [68], [92]. 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) [53], [38] 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 [52]. Another technique to implement a fast ISS is *dynamic translation* [57], [91], [61] which has been favored by many [89], [61], [90], [91] 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 [44] and GXEMUL [67] 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 peformance 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 [60] 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 mecanism 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[6] [96]. It addresses three problems at

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

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[7]) or C (tool FRAMA-C[8]) 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[9]. 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 wemll-chosen subset of C) to ARM or PowerPC developed in the COMPCERT INRIA project[10]. 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 encorporating Presburger arithmetic) has been certified with Coq. Of course, Coq itself cannot be reduced to CCI 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 [56]. In order to apply the technique, users have to formally specify desired properties on an abstract model of the system under verification. Model

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

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 [102]. 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 [59]. 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 [58].

Recently, we propose assume-guarantee reasoning with implicit learning [18]. 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 [22], [23]. 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 [64]. 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 [76] 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 ackward encodings that destroy the logical structure of the original problem.

A very popular, effective recent trend is Satisfiability Modulo Theories (SMT) [101], 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 [93], and *eager* method [94]. 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 spped 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 [86], [51]. 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[11] We thefore strongly believe that decision procedures, and SMT provers, should come along

[11] 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.

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. Devolopping 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.

3.6. 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 trustworthyness, 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 trustworthyness 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 trustworthyness for the major applicative areas. Another is to use stepwise refinement techniques to achieve the appropriate level of trustworthyness. 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 trustworthness 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 analyses 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 [87] 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. aCiNO is currently under testing. We plan a first release of aCiNO on January 2011, and a website shortly later. Participation to the SMT-COMP competition may occur already in 2011.

5.2. CoLoR and Rainbow

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

CoLoR is a Coq [60] library on rewriting theory and termination of nearly 70,000 lines of code [13]. 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 [37] used in the termination competition on termination [39]. Termination certificates are translated and checked in Coq by using the CoLoR library.

Rainbow won the 2007 and 2008 competitions and was 2nd in the 2009 competition of certified termination provers [39]. It did not take part in the 2010 competition organized only 6 months after the 2009 competition.

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. CoqMT

Participant: Pierre-Yves Strub [correspondant].

CoqMT is an evoluation of the Coq proof assistant allowing to dynamically load decision procedures for first-order theories in the conversion checker of the Coq kernel. Users decide which Coq symbols are handled by the decision procedures through the use of mapping primitives. Having dynamic loading and mapping facilities allows users to write their own decision procedures or take anyone from the shelves and use them in Coq without any additional modification of the Coq source code. Because proofs in CoqMT may differ substantially from proofs in Coq, backwards compatibility with previous versions of Coq cannot be ensured.

At this moment, CoqMT comes with a predefined decision procedure for integer linear arithmetic which generates small certificates (unlike previously existing procedures).

CoqMT, the decision procedure for integer linear arithmetic and the theory of dependent lists, are accessible via a GIT repository accessible from http://strub.nu/research/coqmt/.

5.4. EDOLA-PLC

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

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

EDOLA-PLC is an integrated tool for domain-specific modeling and verification of PLC applications. It is based on a domain-specific modeling language, EDOLA, 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+ [77] was chosen as the intermediate language to implement the automatic verification of EDOLA models. A prototype of EDOLA-PLC 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 [77] and a first-order theorem prover SPASS [99].

5.5. Moca

Participant: Frédéric Blanqui [correspondant].

Joint work with Pierre Weis (INRIA Rocquencourt).

Moca is a construction functions generator for OCaml [80] 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:

- predifined 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 predifined 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/.

5.6. SimSoC

Participants: Claude Helmstetter, Vania Joloboff [correspondant], Pierre-Yves Strub, Hui Xiao.

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 [73], [40]. 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 first open source version of SimSoC, SimSoC v0.6.1, has been released in March 2010. It contains a full simulator for ARM V5 running at an average speed of 80 Millions instructions per second in mode DT2, and a simulator for the Power PC 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 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 illustratating the use of the library and infrastructure.

In early November, we released a new version: SimSoC v0.7. This new version includes models of two ARM-based boards, an accurate floating-point simulator, new transport layers for network simulation, and an ARMv6 ISS.

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

5.7. SimSoC-Cert

Participants: Frédéric Blanqui, Claude Helmstetter, 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 [36] (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.

6. New Results

6.1. Simulation

6.1.1. SimSoC

Participants: Claude Helmstetter, Vania Joloboff [correspondant], Pierre-Yves Strub, Hui Xiao, Sen Guo, William Kilque, Peng Shan, Xinlei Zhou.

We have released several versions of SimSoC:

- version 0.6 in March. It was the first public release of SimSoC
- version 0.7 in early November. This version provides two additional boards, new debuggers, new Ethernet transport layers, and a new floating point simulator.
- version 0.7.1 on December 10th. This version adds the LLVM-based dynamic translation for ARMv6 and PowerPC, and fix a few bugs.

We have done the following work that is included in the last open-source:

- A floating point unit simulator has been developed. This floating point simulator is an accurate simulator of the target architecture, although most of the software is architecture independent. This module is using the MPFR library from INRIA [65]. Using MPFR function calls with appropriate parameters, a specific floating point architecture can be emulated exactly. This work is presented in [31].
- The PowerPC ISS was improved to support DT2 mode dynamic translation, reaching a speed of 75 Mips.
- Reusing the techniques we used for ARM and PowerPC, the MIPS 32 bits simulator has been improved with DT1 mode and DT2 mode. The MIPS simulator now runs at 67 Mips.
- We have consolidated the debug framework so that the GDB debugger, or any GDB protocol based remote debugger, can now be fully used on SimSoC for both PowerPC and ARM.
- A Linux user mode simulation module has been added. This user mode makes it possible to run application programs compiled for Linux directly on the simulator (without booting a simulated Linux).
- We provide a SoC simulator for an existing ARM based SoC for which the technical documentation is publicly available, namely the Texas Instruments AM1707 chip. Although it is not a complete chip simulator, we have implemented a significant subset of this chip so that Linux kernel can boot and run with network support.

Beyond the above developments, we have started new work to improve simulation speed, with the goal to develop a new translation mode named DT3, and explored several ideas. We have investigated the LLVM software from University of Illinois [79] to consider its integration within SimSoC. We have analyzed the LLVM software, written some experimental code and concluded we could benefit from this approach:

- A possibility is to optimize the simulator code. A simulation typically has much dynamic information that is not known at compile time of the simulation code. However, after the simulation has started, a lot of configuration information has become static. Therefore it is possible to explore recompiling the simulation module, using constant propagation techniques to propagate the newly discovered static information. An experiment made by compiling some benchmarks with `llvm-g++` and such lazy optimizing has been done.
- Another idea is to selectively compile to native code the most frequently executed sequence of simulated instructions. We have developed a first prototype of this DT3 mode, working both for ARMv6 and PowerPC. This first version has been included in the release 0.7.1. On our testbed, the DT3 mode is currently about 20% faster than the DT2 mode, but we have still a lot of optimizations to implement. An article presenting this first version has been submitted to a Chinese conference.

We are also studying an approach to parallelize simulation.

6.1.2. Automatic generation of an ARMv6 simulator

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

SimSoC code uses complex features of the C++ language and of the SystemC library. Moreover, achieving high simulation speeds requires complex optimizations, such as dynamic translation.

This complexity is problematic, because beyond speed, *faithfulness* is required: all instructions have to be simulated exactly as described in the hardware specification. There is a strong need to strengthen the confidence that simulations results provide are indeed based on a model which is faithful to this specification. Intensive tests are a first answer. For instance, as SimSoC is able to run a Linux kernel on top of a simulated ARM, we know that many situations are covered. However it turned out, through further experiments, that it was not sufficient: wrong behaviors coming from rare instructions were observed after several months.

We therefore decided to *certify* the simulator (hence the name SimSoC-Cert) using formal methods based on Coq, to prove that it conforms to the expected behavior. Currently, we focus on the certification of the CPU part of the ARMv6 architecture [36], and not ARMVv5 as for SimSoC itself.

A first issue is whether to certify an existing code or produce a new, certified code. A second is to be able to upgrade the code from ARMv6 to ARMvx, when a new architecture is released by ARM. Our answer to both has been to automatically generate a specification of ARMv6 and the corresponding code *directly from the reference manual*. Then, a Coq specification of the ARMv6 can be generated, or a code satisfying the Coq specification. Using an automatic generator avoids the kind of errors that would have been introduced by manual translation.

We have completed the formalization in Coq of the ARMv6 specification. The most tedious part is automatically generated, using a parser of the reference manual and a Coq generator. The OCaml code of the generator is 3.3 KLOC. It generates 3.2 KLOC of Coq code. Additionally, there are 6 KLOC of Coq code (among which 2 KLOC are written by hand and 4 KLOC are reused from CompCert[12] [81]).

Using then this ARMv6 formalization, we have developed two CPU simulators:

- *simlight*, which is a stand-alone simulator for the ARMv6 instruction set. This simulator is written in C (8 KLOC generated and 1.5 KLOC hand-written), and can simulate ARMv6 programs as long as they do not access any peripherals (excepted the physical memory) nor coprocessors. This simulator is used as an intermediate stage in the certification of SimSoC.
- *simlight2*, which is an optimized simulator for the ARMv6 instruction set. This simulator can be either wrapped in a stand-alone simulator like *simlight(1)*, or integrated in a SystemC/TLM module. This SystemC/TLM module has been added to SimSoC, and will replace the old hand-written ARM CPU simulator. Many optimizations are applied during generation, and so the generated simulator (74 KLOC generated and 4 KLOC hand-written) is as fast as a hand-written one.

A poster describing the generation of *simlight2* has been presented at APLAS, and an article has been accepted [17].

Taking again advantage of our reference manual reader, we also developed a massive test generator, which generates tests for the *simlight* and *simlight2* decoders. One serious bug and two minor bugs have been found by running the generated tests.

6.1.3. Network simulation

Participants: Vania Joloboff, Pierre-Yves Strub.

As more and more devices are becoming 'network-aware' and a set of connected devices is viewed in applications as a coherent sub-system with some properties that can be guaranteed, such as a set of power control devices in a given set of rooms. In order to validate such systems, one must perform networked test and simulate both the network and the network controller hardware.

To validate the application software of networked embedded systems, it is necessary to simulate the network hardware. As the goal is to test embedded application software, it is required to simulate the hardware interface so the software drivers can be run. Conversely the simulated hardware must receive application data packets from remote devices in the specified standard from the (real or simulated) physical layer. That type of network simulation amounts to defining an infrastructure such that each simulated platform can communicate with the application software driver and with the other simulators using the specified standard. Ideally a parametrized generic infrastructure should be defined such that a particular network controller simulator could be derived from the generic infrastructure by instantiating parameters. Testing a new application with a new network controller model should not require to redo everything from scratch. There is an issue of re-usability of the simulation components, which means the parametrization must be carefully designed. A second class of issues to be resolved in this area is the control of the simulation. To simulate particular application conditions, one must be able to control from a central test point what every system does, in order to execute test scenarios

[12] http://compcert.inria.fr

where each device performs in a particular way. This implies a protocol to control each individual simulation on the network.

We are looking forward in this part of the SimSoC project setting up a networked virtual prototyping tool to validate networked embedded systems. To our knowledge, some academic experiments have been done to simulate networked devices on multi-processing machines, but no virtual prototyping tool has been developed for simulating network hardware and test such network hardware over a network cluster in *controlled conditions*, for example with hardware fault injections at specific timings.

Our goal is to create a "Cluster in the Loop Simulator" to simulate embedded systems hardware (including network/telecom hardware) using Cloud computing to run and test embedded application (software) on the networked simulators.

We have completely redesigned the SimSoC network simulation part to that effect. The previous framework used a fixed centralized TCP-based protocol for simulating the Ethernet network. It has been redesigned to allow for easily changing this underlying transport protocol. Moreover, new transport protocols, based on UDP multicasting and packet injections, have been implemented, leading to a mean latency - on the simulated network - grossly divided by 4.

To measure performance enhancements of our new network simulator, we have developed a full system simulation of the Texas Instruments SoC AM1707, which includes an Ethernet 100 Mbits hardware controller. We have developed a SystemC/TLM model of this controller and a few necessary peripherals.

The Linux operating systems boots and runs on this simulated SoC. Using the simulated network controller, one can mount NFS remote file systems and work with acceptable speed. We have been able to boot a GNU/Linux operating system whose root files system is located on a network files server, and to entirely recompile the GNU `coreutils` using a tools chain located on this network files server within a few hours of simulation.

6.2. Type and rewriting theory

6.2.1. A type theory for Coq

Participants: Jean-Pierre Jouannaud, Pierre-Yves Strub, Qian Wang.

A main earlier achievement of the team is the Calculus of Presburger Inductive Constructions [48], [96] which allows to have an extensional equality for Presburger arithmetic or other inductive types instead of an intensional equality, as is the rule since the very early days of Martin-Löf's type theory who first recognized that type checking becomes undecidable in presence of an extensional equality at all types. Recent extensions of the calculus are described below.

In [49], we describe a modification of the Calculus of Inductive Constructions allowing the use of decision procedures in the computation mechanism, which improves over our previous work. This procedure can actually use equations extracted from the proof context, a mechanism which raises challenging technical difficulties. In [27], [26], we give a new definition of the calculus without most of the restrictions made in our previous work, and proved its core logic in Coq. This development has been the basis of CoqMT, our new version of Coq. As a paradigmatic example, we developed the basic theory of dependent lists with CoqMT. Compared with the same development for non-dependent lists, very few modifications were necessary to carry out the proofs.

We have now started a new important generalization of the previous work, in order to abstract the properties of conversion needed for the first-order theory in CoqMT, and to include a predicative hierarchy of universes which is needed for some applications and brings us closer to the actual implementation. We have progressed when inductive types are restricted to weak elimination, and expect to have it written by the end of the year (the mechanism for extracting equations from the proof context is not considered here). The case of strong elimination appears to be doable if we are only interested in showing consistency of the calculus, but very challenging for showing strong normalization and decidability of type checking. Some recent progress in this direction made by Bruno Barras might help.

We plan a few more generalizations aiming at a better understanding of the mechanism for extracting equations from the proof context: the ability to consider polymorphic first-order theories, the extraction of equations from pattern matching definitions, and the extraction of equations from quantified equations. We believe that all these should follow from the current framework, requiring technical work but no new conceptual breakthrough.

6.2.2. Confluence by decreasing diagrams

Participants: Jean-Pierre Jouannaud, Huiying Luo.

This work has been carried out in part with Vincent Van Oostrom, from the university of Utrecht.

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 [75], 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).

Our first goal now is to further simplify and generalize 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 would yield 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. Good progress has been made along this path.

Our second goal is to develop a Coq library in order to search for and certify confluence proofs. We are waiting here for the release of the open source Coq library for confluence certification by Middledorp and his team in order to avoid duplicating efforts.

6.2.3. Confluence of conditional higher-order rewriting

Participant: Frédéric Blanqui.

Joint work with Claude Kirchner (INRIA Bordeaux) and Colin Riba (ENS Lyon).

The confluence of untyped lambda-calculus with unconditional rewriting is now well understood. In [12], we investigate the confluence of lambda-calculus with conditional rewriting and provide general results in two directions.

First, when conditional rules are algebraic. This extends results of Müller [85] and Dougherty [62] for unconditional rewriting. Two cases are considered, whether beta-reduction is allowed or not in the evaluation of conditions. Moreover, Dougherty's result is improved from the assumption of strongly normalizing beta-reduction to weakly normalizing beta-reduction. We also provide examples showing that outside these conditions, modularity of confluence is difficult to achieve.

Second, we go beyond the algebraic framework and get new confluence results using a restricted notion of orthogonality that takes advantage of the conditional part of rewrite rules.

6.2.4. Confluence and termination of parametrized rewriting

Participant: Jean-Pierre Jouannaud.

Joint work with Benjamin Monate (CEA, Laboratory LIST, Saclay).

In [21], we are interested in proofs of properties of infinite families of specifications, like the family of dihedral groups of order n for some natural number n, or the family of a multi-core hardware with n cores for some natural number n. So far, we have described a confluence test when these families can be presented by parametrized words over a finite alphabet of parametrized size, as in the case of the example of dihedral groups.

Technically, we introduce parametrized rewrite systems for describing infinite families of finite string rewrite systems depending upon non-negative integer parameters, as well as ways to reason uniformly over these families. Unlike previous work, the vocabulary on which a rewrite system in the family is built depends itself on the integer parameters. Rewriting makes use of a toolkit for parametrized words which allows to describe a rewrite step made independently by all systems in an infinite family by a single, effective parametrized rewrite step. The main result is a confluence test for all systems in a family at once, based on a critical pair lemma classically based on computing (finitely many) overlaps between left-hand sides of parametrized rules and then checking for their joinability. Although parametrized rewriting is systematically non-terminating, we describe a termination test which succeeds if all systems in the family terminate.

6.2.5. Higher-order computability path ordering for polymorphic terms

Participants: Jean-Pierre Jouannaud, Jianqi Li.

The problem of showing termination (also called strong normalization in this context) of higher-order higher-type calculi such as the calculus of inductive constructions is well-know to be a very difficult question. The major step was done by Girard in his thesis with the notion of reducibility candidates, which are sets of terms enjoying appropriate closure properties so as to show strong normalization by induction on the structure of terms.

In a previous work with Albert Rubio from the Technical University of Catalogna [74], we introduced the higher-order recursive path ordering, which was further improved as the computability path ordering in collaboration with Frederic Blanqui and Albert Rubio [47]. An important, still elusive property of this ordering is that it somehow makes explicit the computational content of Girard's method, therefore allowing to reduce strong normalization proofs to ordering comparisons. This work was carried out for simply typed lambda calculi.

Our project is to lift the definition (and well-foundedness proof) of the computability path ordering to higher-type calculi. So far, we have considered the case of Girard's system F (a fully polymorphic lambda calculus) for which we have preliminary results for which a weak form of redex creation only is allowed [24], [20].

6.2.6. Automated verification of termination certificates

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

Joint work with Adam Koprowski (MLstate).

Termination is an important property of programs; notably required for programs formulated in proof assistants. It is a very active subject of research in the Turing-complete formalism of term rewriting systems, where many methods and tools have been developed over the years to address this problem. Ensuring reliability of those tools is therefore an important issue.

In [13], Frédéric Blanqui and Adam Koprowski present the library CoLoR that formalizes important results of the theory of well-founded (rewrite) relations in the proof assistant Coq, and their application to the automated verification of termination certificates, as produced by termination tools.

Sidi Ould Biha formalized in Coq the subterm criterion and the notion of usable rules used in the dependency pairs framework [69], [72]. In contrast with the formalization of previous criteria, these ones require classical logic and the axiom of choice. A similar work but in the proof assistant Isabelle/HOL is described in [95]. It would therefore be interesting to compare the two formalizations.

Kim-Quyen Ly extended the library on integer polynomials in order to be able to use polynomials on any (ordered) ring. She also formalized other criteria for (strict) monotony in order to allow the use of polynomials with negative coefficients [88].

6.2.7. Size-based termination

Participant: Frédéric Blanqui.

In recent years, several authors devised size-based termination criteria for ML-like function definitions (lambda-calculi with inductive types and case analysis) that can handle non-structural recursive calls [100], [42], [43]. These criteria use a notion of size related to the semantics of inductive types. Roughly speaking, for data types like lists or trees, the size of a normal term is the height of its tree representation. Typing judgments are then extended to derive informations about the size of terms. Hence, termination can be ensured by using a combination of type checking and constraint solving on size annotations. In [11], we extend these works to rewriting-based function definitions, dependent types and more general notions of size. We therefore provide a powerful termination criterion for the combination of rewriting and beta-reduction in the Calculus of Constructions.

This works detail and extend in important ways the conference papers [45], [46]. In particular, we show how to use other notions of size and allow a much larger class of rule left-hand sides in successor-based systems.

6.2.8. *Higher-order dependency pairs*

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 [82]. It combines the dependency pair method introduced for first-order rewrite systems with the notion of strong computability introduced for typed lambda-calculi [70]. 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 [69], [72]. In [34], 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.

6.2.9. *Small inversions in Coq*

Participant: Jean-François Monin.

In [25], we showed how inductive hypotheses can be manually inverted with small proof terms, using just dependent elimination with a diagonal predicate. The technique is “pure” (it works without any auxiliary type). It can also be used to discriminate, in some sense, the constructors of an inductive type of sort Prop in Coq.

6.2.10. *Constructive description scheme*

Participant: Jean-François Monin.

The coq standard library contains a proof of the constructive description schema, which infers the sigma-existence (i.e., Set-existence) of a witness to a predicate from the regular existence (i.e., Prop-existence). One requires that the underlying set is countable and that the predicate is decidable. A much shorter proof than before is now contributed and distributed since the last release of Coq (8.3, ConstructiveEpsilon.v).

6.3. Decision procedures

6.3.1. *Array theory of bounded elements*

Participants: Ming Gu, Fei He, Bow-Yaw Wang, Min Zhou.

Array theory is essential to program verification. In SMT solvers, arrays are formalized by Equality and Uninterpreted Functions (EUF) [97]. Arrays in EUF have an infinite number of unbounded elements, making extensional equality of two arrays in EUF undecidable. Using counter automata, it has been shown that the fragment with a single alternation of quantifiers is decidable [71], [50]. This fragment is however far too restrictive for practical applications.

In [33], we investigate a first-order array theory of bounded elements (UABE). This fragment is of course very meaningful since its corresponds to a limitation of the physical world, and very useful, since it allows arbitrary nesting of quantifiers. In UABE, arrays contain bounded elements, that is elements of a finite set, such as machine real numbers in simple precision. By reducing to weak second-order logic with one successor (WS1S), we show that the proposed array theory is decidable. Finally, we show that the extension to unbounded elements is undecidable. Moreover, allowing linear arithmetic expressions in indices makes array theory of bounded elements unsolvable as well.

6.3.2. Efficient interpolation prover

Participants: Bow-Yaw Wang, Liangze Yin.

For complete SAT-based model checking, the interpolation-based algorithm can be more efficient than loop-free induction [83]. Given two inconsistent Boolean formulas, their interpolant can be generated by a resolution proof of inconsistency. The size of resolution proofs however is exponential. A naïve implementation of the interpolation-based model checking algorithm is very likely to perform poorly in both time and space.

In this project, we implement an efficient interpolation-based model checking algorithm in NuSMV [54]. A BDD-like data structure to represent interpolants is developed. The new data structure allows us to perform reduction and simplification on the fly. Preliminary experimental results show that our implementation is comparable to the BDD-based algorithm.

6.3.3. Certification of SAT solvers

Participants: Jean-Pierre Jouannaud, Pierre-Yves Strub [correspondant], Lianyi Zhang.

To ensure trust in the result of SAT-solvers, one has to prove the correction of the implementation inside a proof checker, or to modify this solver such that it produces a witness which can be checked by an external tool.

This work allows the certification of satisfiability (easy) as well as unsatisfiability (complex) proofs given by a set of regular input resolution proofs (or trace) as provided by the zChaff SAT-solver.

This development includes: i) a formalization of the resolution algorithm in Coq, ii) a formalization of the zChaff trace checker algorithm in Coq, and iii) a tool written in a mix of C++/OCaml taking a zChaff trace and generating a Coq file representing the initial SAT problem, along with a proof of the unsatisfiability of this problem. Checking zChaff traces is then reduced to compiling the file produced in the last step.

The development is done in Coq. It applies with minor modifications to other solvers, in particular to PicoSAT (for which the clauses in the output trace must be sorted). It is described in [16].

6.4. Learning in verification

6.4.1. Compositional abstraction refinement for timed systems

Participants: Ming Gu, Fei He, He Zhu.

Joint work with William N. N. Hung (Synopsys Inc., USA) and Xiaoyu Song (Portland State University, USA).

Model checking suffers from the state explosion problem. Compositional abstraction and abstraction refinement have been investigated in many areas to address this problem. This paper considers the compositional model checking for timed systems. We present an automated approach which combines compositional abstraction and counter-example guided abstraction refinement (CEGAR) [19]. Given a timed system, the proposed approach exploits the semantics of timed automata to procure its abstraction. Our approach is conservative. Hence, any safety property which holds on the abstraction is guaranteed to hold on the concrete model. In the case of a spurious counter-example, our proposed approach refines and strengthens the abstraction in a component-wise method. We implemented our method with the model checking tool Uppaal. Experimental results show promising improvements.

6.4.2. Automated assume-guarantee reasoning through implicit learning

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

Joint work with Yu-Fang Chen (Academia Sinica, Taiwan), Edmund M. Clarke (CMU, USA), Azadeh Farzan (University of Toronto, Canada), Ming-Hsien Tsai (National Taiwan University, Taiwan), and Yih-Kuen Tsay (National Taiwan University, Taiwan).

In recent years, a learning-based technique for assumption generation has attracted lots of attention in assume-guarantee reasoning [59]. The authors apply a learning algorithm for finite automata in order to infer an unknown assumption. The idea is theoretically elegant, but it does not appear to be scalable. It is not clear whether automated assume-guarantee reasoning can actually outperform monolithic verification [58].

In [18], we propose a purely implicit solution to the contextual assumption generation problem in assume-guarantee reasoning. Instead of improving the L^* algorithm – a learning algorithm for finite automata, our algorithm computes implicit representations of contextual assumptions by the CDNF algorithm – a learning algorithm for Boolean functions. We report three parametrized test cases where our solution outperforms the monolithic interpolation-based Model Checking algorithm.

There are, of course, many learning algorithms for Boolean functions. It is of utmost importance to identify the most effective learning algorithm in practice.

In [15], we compare two learning algorithms for generating contextual assumptions in automated assume-guarantee reasoning. The CDNF algorithm implicitly represents contextual assumptions by a conjunction of DNF formulas, while the OBDD learning algorithm uses ordered binary decision diagrams as its representation. Using these learning algorithms, the performance of assume-guarantee reasoning is compared with monolithic interpolation-based Model Checking in parametrized hardware test cases.

6.4.3. Inferring loop invariants by algorithmic learning

Participant: Bow-Yaw Wang.

Joint work with Soonho Kong (Seoul National University, Korea), Yungbum Jung (Seoul National University, Korea), and Kwangkeun Yi (Seoul National University, Korea).

Given a loop annotated with pre- and post-conditions, loop invariants are often used to prove that the annotated loop conforms to its specification. Finding loop invariants, however, is very tedious. Traditional fixpoint-based loop invariant generation techniques essentially concretize least fixed points in the abstract domain. Predicate abstraction and fixed point computation are the bottlenecks of the traditional approach.

By combining algorithmic learning, decision procedures, and predicate abstraction, we present an automated technique for finding loop invariants in propositional formulas [22]. Given invariant approximations derived from pre- and post-conditions, our new technique exploits the flexibility in invariants by a simple randomized mechanism. The proposed technique is able to generate invariants for some Linux device drivers and SPEC2000 benchmarks in our experiments.

Joint work with Soonho Kong (Seoul National University, Korea), Yungbum Jung (Seoul National University, Korea), Cristina David (National University of Singapore, Singapore), and Kwangkeun Yi (Seoul National University, Korea).

Algorithmic learning is applied to loop invariant generation lately [22]. Exploiting the precondition and post-condition from users, a learning algorithm is able to infer quantifier-free loop invariants for the annotated loop. The learning-based technique however is not applicable to loops with arrays. Since loop invariants for array manipulation often require quantification, the work [22] is too restrictive for such loops.

By combining algorithmic learning, decision procedures, predicate abstraction, and simple templates, we present an automated technique for finding quantified loop invariants in [23]. Our technique can find arbitrary first-order invariants (modulo a fixed set of atomic propositions and an underlying SMT solver) in the form of the given template and exploits the flexibility in invariants by a simple randomized mechanism. The proposed technique is able to find quantified invariants for loops from the Linux source, as well as for the benchmark

code used in the previous works. Our contribution is a simpler technique than the previous works yet with a reasonable derivation power.

6.5. Specification and verification of TLA+ and PLC systems

PLC stands for Programmable Logic Controller [98]. PLCs are widely used in the industry. TLA (Temporal Logic of Actions) is a logic for specifying and reasoning about concurrent and reactive systems [78]. It is the basis for TLA+, a complete specification language [77].

6.5.1. Specifying PLC systems with TLA+: a case study

Participants: Ming Gu, Hehua Zhang.

Joint work with Stephan Merz (INRIA Nancy, France).

We report on a method formally specifying and verifying PLCs in the specification language TLA+ [14]. The specification framework is generic. It separates the description of the environment from that of the controller itself and its structure is consistent with the scan cycle mechanism used by PLCs. Specifications can be parametrized with the number of replicated components. We have validated our approach on a concrete case study, a controller for fire fighting equipment in a ship dock, and report on the results obtained for this case study.

6.5.2. A refinement-based validation method for PLCs

Participants: Ming Gu, Hai Wan.

Joint work with Xiaoyu Song (Portland State University, USA).

Timers play a pivotal role in PLC real-time embedded system applications. This work addresses the formal validation of PLC systems with timers in the theorem proving system Coq. The timer behavior is characterized formally. In [29], a refinement validation methodology is presented in terms of an abstract model and a concrete model. The refinement is calibrated by a mapping relation. The soundness of the methodology is shown in the proving system. An illustrative case study demonstrates the effectiveness of the approach.

6.5.3. Specification, verification, and validation of PLCs in Coq

Participants: Ming Gu, Hai Wan.

Joint work with Xiaoyu Song (Portland State University, USA).

Timers play a pivotal role in PLC real-time embedded system applications. This work addresses the formal validation of PLC systems with timers in the theorem proving system Coq. The timer behavior is characterized formally. In [29], a refinement validation methodology is presented in terms of an abstract model and a concrete model. The refinement is calibrated by a mapping relation. The soundness of the methodology is shown in the proving system. An illustrative case study demonstrates the effectiveness of the approach.

In [30], we present a novel method to specify and verify PLC software systems with the theorem proving system Coq. Dependent inductive data types are harnessed to represent the component specifications. Modular and parametrized specification and verification are proposed. An illustrative example demonstrates the effectiveness of the method.

6.5.4. Property-preserving refinements of timed automata for PLCs

Participant: Rui Wang.

Property preservation refinement ensures that the properties of the original model still hold after refinement.

In this on-going resarch, we first design two kinds of refinement operators for timed automata. Then, invariance, absence, exist, and respond patterns for PLCs are modeled within the framework of timed automata, and we investigate the conditions that preserve these properties after refinement.

6.5.5. Counterexample guided predicate abstraction refinement

Participants: Ming Gu, Li Li.

Joint work with Jianmin Wang (Tsinghua University, China).

Abstraction is a necessary step in model checking. When abstract models contain spurious behaviors, they have to be refined to remove such behaviors. Counterexample guided abstraction refinement method (CEGAR) is a well-known technique that automatically refines abstraction in software verification [55]. In [10], we investigate improvements of this method.

First, we present an effective predicate abstraction macanism for program verification. We present a novel method to compute the abstract states, which narrows down their number. Second, an algorithm for property checking is given based on weighted-graphs, which aims at finding a shortest counter-example as early as possible. Third, we present a effective method for detecting linear equalities which can be used in CEGAR. Fourth, we describe an efficient method to constructing interpolants for a quantifier-free first-order logic formula. All these results are then used to verify PLC programs with timers within their environment in CEGAR.

6.5.6. Formal proof of a machine-closed theorem in Coq

Participants: Ming Gu, Hai Wan.

Joint work with Xiaoyu Song (Portland State University, USA).

The notion of machine-closedness plays an important role in system specification [41]. A system specification consists of two parts: a safety property and a liveness property. A specificationis machine-closed if the liveness property does not constrain the safety property. It is commonly agreed that machine-closedness should be considered as a sanity check. It can be shown that TLA+ specifications made of a transition system and a possibly countably infinite set of fairness constraints are machine-closed.

We present a formal proof of the machine-closedness theorem for TLA+ in the theorem proving system Coq. A shallow embedding is employed for the proof which is independent of a concrete syntax. Fundamental concepts needed to state the machine-closedness theorem are addressed in the proof development. A useful proof pattern of constructing a trace with desired properties is defined, and a number of Coq reusable libraries are developed. The proof scripts can be downloaded from http://formes.asia/people/haiwan/.

6.5.7. Specifying time-sensitive systems with TLA+

Participants: Ming Gu, Hehua Zhang.

Joint work with Xiaoyu Song (Portland State University, USA).

With the wide use of real-time systems, embedded systems and pervasive computing in everyday applications, time-sensitive systems (i.e. systems whose behavior is influenced by the passing of time) attract many people's attention. To formally analyze time-sensitive systems, it is necessary to represent time in the formalisms.

We present a method to ameliorate the usability of TLA+ in specifying and verifying time-sensitive systems [32]. A real-time module RealTimeNew is introduced to encapsulate the definitions of commonly used time patterns. The basic patterns specify the time duration of an action or the time interval between actions. Advanced time patterns like delay, deadline, and timeout are further defined based on the basic ones. We then present a general framework to differentiate the temporal characterizations from system functionality with time constraints. The temporal specification is concise and provably as a refinement of its corresponding functional description without time.

6.5.8. Embedding TLA+ in Coq

Participants: Ming Gu, Hai Wan, Yuhui Wang.

TLA+ plays an important role in our modeling and verification framework. It is an intermediate language that connects the specification language EDOLA, model checking tools and the theorem proving system Coq. It is an ongoing work and started three months ago. The preliminary result is that, we have built a tool that can translate a single TLA+ module into Coq, but only a selected set of TLA+ operators is supported.

6.5.9. Translation-based verification of colored Petri net models

Participants: Ming Gu, Hehua Zhang, Xianpeng Zhao.

Verification techniques of colored Petri net (CPN) models include occurrence graphs, invariant analysis, deduction-based method, etc. However, these techniques have limitations. Theorem proving is a supplement for them. We provide a novel translation-based method to formally prove properties on a colored Petri net. In our method, TLA+ is chosen as the intermediate language. We give semantic translation rules from a CPN to a TLA+ model. The formal proof of a CPN is then turned to the proof of a TLA+ model, which can be easily done with the existent TLA+ proving rules and the automatic theorem provers. The translation from CPN to TLA+, fills the gap between CPN and the logic-based proving. A resource allocator example is used to explain the method and validate these rules. We also show that how the refinement relationship of two CPN models can be proved with our method.

6.5.10. Formal semantics of PLC programs

Participants: Frédéric Blanqui, Sidi Ould Biha.

The definition of a formal semantics of PLC programing language is an important step towards the certification of PLC programs. We defined an operational semantics of the Instruction List language for PLC that covers a large subset of the standard. It includes one type of timers used in PLC (On delay timers) and support the cyclic behavior of PLCs. We formalized this semantics in the proof assistant Coq. We used it to proof some properties about simple PLC programs widely used in car parks. In collaboration with Jan Olaf Blech from Fortiss (Germany), we are planing to use this semantics to prove some safety properties on an real case industrial example of PLC. This work was presented in a poster at the APLAS 2010 conference.

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 this framework to an algorithm constructing a Breadth-First Search Spanning Tree (BFS) in a distributed system [35]. The main point was to provide a methodology for proving that a property of global configurations is achieved from local moves.

6.6.2. Modeling and verification of services managements

Participants: Liu Liu, Hai Wan.

Joint work with Zoé Drey (INRIA Bordeaux, France).

Various forms of pervasive computing environments are being deployed in an increasing number of areas including hospitals, homes and military settings. Entities in this environment provide rich functionalities (i.e. services). How to organize these heterogeneous and distributed entities to deliver user-defined services is challenging. Pantagruel is an approach to integrate a taxonomic description of a pervasive computing environment into a visual programming language [63]. A taxonomy describes the relevant entities of a given pervasive computing area and serves as a parameter to a sensor-controller-actuator development paradigm. The orchestration of area-specific entities is supported by high-level constructs, customized with respect to taxonomic information. Pantagruel is also a language that describes and manages services. Furthermore, Pantagruel can be viewed as a high level service contract between the service designer and the program implementer. The work [28] presents a formalization of Pantagruel, both its syntax and semantics. Four kinds of static properties are stated based on the formalization. Predicate abstraction based algorithms are designed to verify the properties.

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. Other Grants and Activities

8.1. International Initiatives

- SIVES[13] 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).
- CCCBIP is a proposal coordinated by Jean-François Monin for building a certified compilation chain for BIP[14]. It was submitted for a new INRIA ARC for 2011-2012 involving FORMES, Tsinghua University, the Institute of Software of the Chinese Academy of Science, and VERIMAG.
- FORMES organized the 2nd Asian-Pacific Summer School on Formal Methods[15] late August 2010. The school attracted 55 participants (mostly from China).
- FORMES organized jointly with Peking University the ARTIST European Network of Excellence on Embedded Systems Design its 2010 Summer School[16] in China, July 18-23.
- 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.

[13] http://formes.asia/cms/sives
[14] http://www-verimag.imag.fr/Rigorous-Design-of-Component-Based.html?lang=en
[15] http://formes.asia/cms/coqschool/2010
[16] http://www.artist-embedded.org/artist/Overview,2082.html

8.2. National Initiatives

- FORMES is part of the working group LTP on Languages, Types and Proofs of the GDR GPL[17], the French research network on software engineering.
- FORMES is part of the working group LAC on Logic, Algebra and Calculus of the GDR IM[18], the French research network on mathematics and computer science.

9. Dissemination

9.1. Visits

- Vania Joloboff was invited to give presentations at Northeastern University in Shenyang, Harbin Engineering University in Harbin, Tsinghua University Campus in Shenzhen, and the Shenzhen Institute of Advanced Technology.
- Vania Joloboff visited CEA Saclay and LIP6 laboratory in April to discuss their respective simulation projects and technologies. He also visited ST Microelectronics in May and September and organized a meeting with ST Microelectronics, Schneider Electric, CEA, TIMA and Verimag laboratory to consider potential collaboration on national or international projects.
- Vania Joloboff had several meetings with EADS representatives in China and France (visit in August) to discuss potential collaboration with LIAMA.
- Jean-François Monin visited Jiaotong University BASICS from March 21 to March 24 and from June 13 to 18.
- Jean-Pierre Jouannaud was invited by the European pavilion at the international exposition in Shanghai (Chinese-European research collaborations).
- Jean-Pierre Jouannaud was invited to give a seminar talk at Tsinghua University, in the seminar of the computer science laboratory.
- Jean-Pierre Jouannaud was keynote speaker at CSCM 2010 in Shanghai.
- Jean-François Monin visited AIST and NII lab at Tokyo on April 27 and 28, and gave a talk “towards proving Netlog programs” at AIST.
- Jean-François Monin visited Academia Sinica at Taipei on December 21, and gave a talk “Proving Netlog programs”.
- Jean-François Monin visited VERIMAG and UJF in January, July and November 2010 to discuss about future collaboration on BIP and the candidature of UJF to the LIAMA consortium.
- Hui Xiao visited Nanjing University to discuss collaboration on a simulation project.

9.2. Committees

- Frédéric Blanqui is a member of the Steering Committee of the International Conference on Rewriting Techniques and Applications (RTA) from July 2010 to July 2013.
- Frédéric Blanqui was PC member of the 12th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP’10) and of the 5th International Workshop on Higher-Order Rewriting (HOR’10).
- Vania Joloboff was PhD jury for Marius Gligor “Fast Simulation Strategies and Adaptive DVFS Algorithms for Low Power MPSoCs”, University of Grenoble, September 2010.

[17] http://gdr-gpl.cnrs.fr/
[18] http://www.gdr-im.fr/

- Jean-Pierre Jouannaud was PC chair of the 25th IEEE Symposium on Logic in Computer Science (LICS'10).
- Jean-Pierre Jouannaud was PC member of DCM 2009 and DCM 2010.
- Jean-Pierre Jouannaud chaired the committee for the Gödel prize 2010 and the committee for the Kleene Prize 2010, and was a member of the committee for the LICS Test of Time Award 2010.
- 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-François Monin was PhD jury for Serguei Lenglet "Bisimulations dans les calculs avec passivation", University of Grenoble, January 2010.
- Jean-François Monin was PhD jury and rapporteur for Johannes Kanig "Specification and Proof of Higher-Order Programs", University of Paris-Sud, November 2010.
- Jean-François Monin was HDR jury for Xavier Urbain "Preuve automatique : techniques, outils et certification", University of Paris-Sud, November 2010.
- Bow-Yaw Wang was a PC co-chair of the 4th IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE'10).

9.3. Internships

- Frédéric Blanqui supervised the 5-months internship of Kim-Quyen Ly on the formalization of rational polynomial interpretations in Coq.
- Vania Joloboff supervised Chinese Master students Peng Shan and Sen Guo since January, and Xinlei Zhou since end of June. He is also supervising since early September French student William Kilque from France CPE engineering school doing "année de césure".

9.4. Teaching

- Frédéric Blanqui, Jianqi Li, Jean-François Monin, Sidi Ould Biha, Xiaomu Shi, Pierre-Yves Strub, Lianyi Zhang, Qian Wang participated to the classes in the 2nd Asian-Pacific Summer School on Formal Methods.
- Frédéric Blanqui gave a lecture on type checking and type inference and a lecture on termination of β-reduction at the 2nd Asian-Pacific Summer School on Formal Methods.
- Fei He and Bow-Yaw Wang gave a graduate course *Formal Verification for Software Systems* at Tsinghua University (16 lectures, 2 hours each) from March 2010 to June 2010.
- Jean-Pierre Jouannaud gave a lecture on modelization with automata at the school of Software, Tsinghua University, early 2010.
- Jean-Pierre Jouannaud gave a lecture on first-order logic and a lecture on the Curry-Howard correspondence at the 2nd Asian-Pacific Summer School on Formal Methods.
- Jean-François Monin gave two lectures on mathematical induction at the 2nd Asian-Pacific Summer School on Formal Methods.
- Jean-François Monin gave an undergraduate course *Formal Reasoning in Practice* at Tsinghua University (12 lectures, 3 hours each), from March 2010 to June 2010.
- Jean-François Monin gave an undergraduate course *Introduction to Interactive Proof of Software* at Tsinghua University 16 lectures, 2 hours each) from September 2010 to January 2011.

- Jean-François Monin gave 2 lectures on Coq for *Foundations of Programming Languages* at the Institute of Software of the Chinese Academy of Sciences in November 2010.
- Vania Joloboff gave a lecture at Tsinghua University on Virtual Prototyping techniques.
- Vania Joloboff gave a 1 day tutorial at the ARTIST 2010 Summer School in China.

9.5. Long-term visitors

- Cody Roux, PhD at INRIA Nancy in the Pareo team, supervised by Claude Kirchner (INRIA Bordeaux), Gilles Dowek (INRIA Saclay, École Polytechnique) and Frédéric Blanqui visited Formes from June 2 to July 1 to work on his thesis with Frédéric Blanqui.
- Ming-Hsien Tsai, PhD student at National Taiwan University, supervised by Yih-Kuen Tsay (National Taiwan University), visited Formes from March 15 to June 2 to work on the formalization of a numerical abstraction for C programs with Bow-Yaw Wang and Jean-Pierre Jouannaud.
- Jean-Jacques Lévy (INRIA), director of the MSR-INRIA Joint Center, visited FORMES from August 17 to September 17, gave two introductory lectures on λ-calculus at the 2nd Asian-Pacific Summer School on Formal Methods, and six advanced lectures on λ-calculus at Tsinghua University.

9.6. Short-term visitors

- Yu Zhang (USTC, China) gave a talk on April 9 on “Our Research on Verified Software: Compilation and Verification”.
- Xinyu Jiang (USTC, China) gave a talk on April 9 on “Modular Verification of Dynamic Code Loading and Linking”.
- David Pichardie (INRIA Rennes) gave a talk on April 16 on “A Certified Denotational Abstract Interpreter”.
- Afonso Ferreira, CNRS Scientific Coordinator for International Affairs, visited LIAMA from May 19 to May 21.
- Reynald Affeldt (AIST, Japan) gave a talk on May 28 on “Formal verification of cryptographic software in Coq”.
- Rongjie Yan (VERIMAG) gave a talk on May 28 on “Incremental Component-based Construction and Verification using Invariants”.
- Cody Roux (INRIA Nancy) gave a talk on June 25 on “Refinement types as Higher Order Dependency Pairs”.
- Valérie Pécresse, French Minister of Higher Education and Research, visited LIAMA on July 5. Vania Joloboff hosted the visit with Chinese Director Tianzi Jiang.
- Laurent Fribourg (CNRS, ENS Cachan) gave a talk on September 20 on “Simulation + Uncertainty = Model Checking”.
- Patrick Devedjian, French Minister of Reflation, visited LIAMA on October 18. Jean-François Monin gave a presentation of FORMES activities.
- José Meseguer (University of Illinois at Urbana-Champaign) gave a talk on November 15 on “Formalization and Correctness of the PALS Architectural Pattern for Distributed Real-Time Systems”.
- Sophie Quinton (VERIMAG) gave a talk on November 23 on “Reasoning about Safety and Progress using Contracts”.
- Kwangkeun Yi (Seoul National University, South Korea) gave a talk on December 3 on “Multi-Staged Program’s Static Type System and Beyond”.

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, p. 37-92, Journal version of LICS’01 [*DOI :* 10.1017/S0960129504004426], http://hal.inria.fr/inria-00105648/en/.

[2] 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/.

[3] 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.

[4] 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.

[5] 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.

[6] C. HELMSTETTER, F. MARANINCHI, L. MAILLET-CONTOZ. *Full simulation coverage for SystemC transaction-level models of systems-on-a-chip*, in "Formal Methods in System Design", 06 2009, vol. 35, n^{o} Number 2 / October, 2009, p. pages 152-189 [*DOI :* 10.1007/s10703-009-0075-z], http://hal.archives-ouvertes.fr/hal-00429058/en/.

[7] C. JARD, J.-F. MONIN, R. GROZ. *Development of Veda, a Prototyping Tool for Distributed Algorithms*, in "IEEE Transactions on Software Engineering", march 1988, vol. 14, n^{o} 3, p. 339–352.

[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.-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

Doctoral Dissertations and Habilitation Theses

[10] L. LI. *Research on model checking method of counterexample guided predicate abstraction-refinement*, Tsinghua university, 6 2010.

Articles in International Peer-Reviewed Journal

[11] F. BLANQUI. *A size-based termination criterion for dependently-typed higher-order rule-based programs*, in "Mathematical Structures in Computer Science", 2010, Submitted.

[12] F. BLANQUI, C. KIRCHNER, C. RIBA. *On the confluence of lambda-calculus with conditional rewriting*, in "Theoretical Computer Science", 2010, vol. 411, n^{o} 37, p. 3301-3327, http://hal.inria.fr/inria-00509054/en/.

[13] 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", 2010, To appear.

[14] H. ZHANG, S. MERZ, M. GU. *Specifying and Verifying PLC systems with TLA+: a case study*, in "Computers & Mathematics with Applications", 2010, vol. 60, n^{o} 3, p. 695-705, http://hal.archives-ouvertes.fr/hal-00516785/en/.

Invited Conferences

[15] Y.-F. CHEN, E. CLARKE, A. FARZAN, F. HE, M.-H. TSAI, Y.-K. TSAY, B.-Y. WANG, L. ZHU. *Comparing Learning Algorithms in Automated Assume-Guarantee Reasoning*, in "International Symposium On Leveraging Applications of Formal Methods, Verification and Validation", Grèce Crete, B. STEFFEN (editor), 2010, http://hal.inria.fr/inria-00515167/en/.

[16] J.-P. JOUANNAUD, P.-Y. STRUB, L. ZHANG. *Certification of SAT of Solvers in Coq*, in "Guangzhou Symposium on Satisfiability in Logic-Based Modeling", Chine Zuhai, Yuping Shen, Institute of Logic and Cognition Sun Yat-sen University, Guangzhou., 2010, http://hal.inria.fr/inria-00516906/en/.

International Peer-Reviewed Conference/Proceedings

[17] F. BLANQUI, C. HELMSTETTER, V. JOLOBOFF, J.-F. MONIN, X. SHI. *Designing a CPU model: from a pseudo-formal document to fast code*, in "Proceedings of the 3rd Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools", 2010.

[18] 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/.

[19] F. HE, L. ZHU, W. HUNG, X. SONG, M. GU. *Compositional Abstraction Refinement for Timed Systems*, in "IEEE International Symposium on Theoretical Aspects of Software Engineering", Taïwan, Province De Chine Taipei, 2010, http://hal.inria.fr/inria-00516575/en/.

[20] J.-P. JOUANNAUD, J.-Q. LI. *A Computability Path Ordering for Polymorphic Terms*, in "11th International Workshop on Termination", Royaume-Uni Edinburgh, 2010, http://hal.inria.fr/inria-00497405/en/.

[21] J.-P. JOUANNAUD, B. MONATE. *Infinite families of finite string rewriting systems and their confluence*, in "Proc. LPAR 2010", Indonésie Yogyakarta, FERMÜLLER, VORONKOV (editors), SPRINGER, 2010, http://hal.inria.fr/inria-00515395/en/.

[22] 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/.

[23] 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/.

[24] J.-Q. LI. *A Computability Path Ordering for Polymorphic Terms - Short Paper*, in "25th Annual IEEE Symposium on Logic in Computer Science (LICS 2010)", Royaume-Uni Edinburgh, 2010, http://hal.inria.fr/inria-00516858/en/.

[25] J.-F. MONIN. *Proof Trick: Small Inversions*, in "Second Coq Workshop", Royaume-Uni Edinburgh, Y. BERTOT (editor), Yves Bertot, 2010, http://hal.inria.fr/inria-00489412/en/.

[26] P.-Y. STRUB. *Coq modulo theory*, in "Proceedings of the 24th International Conference on Computer Science Logic, Lecture Notes in Computer Science 6247", 2010, http://hal.inria.fr/inria-00497404/en/.

[27] P.-Y. STRUB, Q. WANG. *Coq Modulo Theory - Short Paper*, in "Logic In Computer Science (LICS 2010)", Royaume-Uni Edimbourg, 2010, http://hal.inria.fr/inria-00497794/en/.

[28] H. WAN, Z. DREY, Z. YOU, L. LIU. *Formal Modeling and Verification of Services Managements for Pervasive Computing Environment*, in "The 7th International Conference on Service Systems and Service Management", Japon Tokyo, 2010, http://hal.inria.fr/inria-00516020/en/.

[29] H. WAN, X. SONG, G. CHEN, M. GU. *A Refinement-Based Validation Method for Programmable Logic Controllers*, in "The 10th International Conference on Quality Software", Chine Zhangjiajie, Hunan, 2010, http://hal.inria.fr/inria-00516014/en/.

[30] H. WAN, X. SONG, M. GU. *Parameterized Specification and Verification of PLC Systems in Coq*, in "4th IEEE International Symposium on Theoretical Aspects of Software Engineering", Taïwan, Province De Chine Taipei, 2010, http://hal.inria.fr/inria-00516016/en/.

[31] H. XIAO, V. JOLOBOFF, C. HELMSTETTER. *Simulation of Floating-Point Hardware with Multiple Precision.*, in "ICIIE 11: Proceedings of 2011 International Conference on Information and Industrial Electronics", 2010.

[32] H. ZHANG, M. GU, X. SONG. *Specifying time-sensitive systems with TLA+*, in "COMPSAC 2010 : 34th Annual IEEE Computer Software and Applications Conference", Corée, République De Seoul, 2010, p. 425-430, http://hal.inria.fr/inria-00516164/en/.

[33] M. ZHOU, F. HE, B.-Y. WANG, M. GU. *On Array Theory of Bounded Elements*, in "CAV 2010 - 22nd International Conference on Computer Aided Verification", Royaume-Uni Edinburgh, 2010, http://hal.inria.fr/inria-00517943/en/.

Research Reports

[34] S. SUZUKI, K. KUSAKARI, F. BLANQUI. *Argument Filterings and Usable Rules in Higher-Order Rewrite Systems*, IEICE, 2010, n^{o} SS2010-24, Vol 110, No 169.

Other Publications

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

References in notes

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

[37] *Certification Problem Format*, 2010, http://cl-informatik.uibk.ac.at/software/cpf/.

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

[39] *Termination Competition*, 2010, http://termination-portal.org/wiki/Termination_Competition.

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

[41] M. ABADI, L. LAMPORT. *The Existence of Refinement Mappings*, in "Theoretical Computer Science", 1991, vol. 82, n^{o} 2, p. 253–284.

[42] A. ABEL. *Termination checking with types*, in "Theoretical Informatics and Applications", 2004, vol. 38, n^{o} 4, p. 277-319.

[43] G. BARTHE, M. J. FRADE, E. GIMÉNEZ, L. PINTO, T. UUSTALU. *Type-based termination of recursive definitions*, in "Mathematical Structures in Computer Science", 2004, vol. 14, n^{o} 1, p. 97-141.

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

[45] F. BLANQUI. *A type-based termination criterion for dependently-typed higher-order rewrite systems*, in "15th International Conference on Rewriting Techniques and Applications - RTA'04", Aachen Allemagne, 2004, 15 p, Colloque avec actes et comité de lecture. internationale., http://hal.inria.fr/inria-00100254/en/.

[46] F. BLANQUI. *Decidability of Type-checking in the Calculus of Algebraic Constructions with Size Annotations*, in "14th Annual Conference of the EACSL", Oxford Royaume-Uni, L. ONG (editor), Lecture Notes in Computer Science, Springer Verlag, 2005, vol. 3634, p. 135–150 [*DOI :* 10.1007/11538363_11], http://hal.inria.fr/inria-00000200/en/.

[47] F. BLANQUI, J.-P. JOUANNAUD, A. RUBIO. *HORPO with Computability Closure : A Reconstruction*, in "14th International Conference on Logic for Programming Artificial Intelligence and Reasoning", Yerevan Arménie, LNCS, 10 2007, vol. 4790, http://hal.inria.fr/inria-00168304/en/.

[48] F. BLANQUI, J.-P. JOUANNAUD, P.-Y. STRUB. *Building Decision Procedures in the Calculus of Inductive Constructions*, in "16th EACSL Annual Conference on Computer Science and Logic - CSL 2007", Lausanne Suisse, J. DUPARC, T. HENZIGER (editors), Lecture Notes in Computer Science, Springer Verlag, 2007, vol. 4646, F.: Theory of Computation/F.4: MATHEMATICAL LOGIC AND FORMAL LANGUAGES/F.4.1: Mathematical Logic/F.4.1.7: Proof theory [*DOI :* 10.1007/978-3-540-74915-8_26], http://hal.inria.fr/inria-00160586/en/.

[49] 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/.

[50] M. BOZGA, P. HABERMEHL, R. IOSIF, F. KONEČNÝ, TOMÁŠ. VOJNAR. *Automatic Verification of Integer Array Programs*, in "CAV", A. BOUAJJANI, O. MALER (editors), Lecture Notes in Computer Science, Springer Verlag, 2009, vol. 5643, p. 157–172.

[51] 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.

[52] 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.

[53] 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.

[54] A. CIMATTI, E. CLARKE, F. GIUNCHIGLIA, M. ROVERI. NUSMV*: a new Symbolic Model Verifier*, in "CAV", Lecture Notes in Computer Science, Springer Verlag, 1999, n° 1633, p. 495-499, eds. N. Halbwachs and D. Peled.

[55] E. CLARKE, O. GRUMBERG, S. JHA, Y. LU, H. VEITH. *Counterexample-guided abstraction refinement for symbolic model checking*, in "JACM", 2003, vol. 50, n° 5, p. 752–794.

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

[57] 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.

[58] 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.

[59] 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.

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

[61] 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.

[62] D. DOUGHERTY. *Adding Algebraic Rewriting to the Untyped Lambda Calculus*, in "Information and Computation", 1992, vol. 101, n° 2, p. 251-267.

[63] Z. DREY, C. CONSEL. *A Visual, Open-Ended Approach to Prototyping Ubiquitous Computing Applications*, in "Proceedings of the 8th IEEE Conference on Pervasive Computing and Communications (PERCOM'10)", 2010.

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

[65] L. FOUSSE, G. HANROT, V. LEFÈVRE, P. PÉLISSIER, P. ZIMMERMANN. *MPFR: A multiple-precision binary floating-point library with correct rounding*, in "ACM Trans. Math. Softw.", 2007, vol. 33, n° 2, 13, http://doi.acm.org/10.1145/1236463.1236468.

[66] 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.

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

[68] 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.

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

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

[71] P. HABERMEHL, R. IOSIF, TOMÁŠ. VOJNAR. *A Logic of Singly Indexed Arrays*, in "LPAR", Lecture Notes in Computer Science, Springer Verlag, 2008, vol. 5330, p. 558–573.

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

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

[74] 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.

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

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

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

[78] L. LAMPORT. *The Temporal Logic of Actions*, in "ACM Trans. Program. Lang. Syst.", 1994, vol. 16, n^{o} 3, p. 872-923.

[79] C. LATTNER, V. ADVE. *LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation*, in "Proceedings of the 2004 International Symposium on Code Generation and Optimization (CGO'04)", Palo Alto, California, Mar 2004.

[80] 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/.

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

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

[83] K. L. MCMILLAN. *Interpolation and SAT-based Model Checking*, in "CAV", WARREN A. JR. HUNT, F. SOMENZI (editors), Lecture Notes in Computer Science, Springer Verlag, 2003, vol. 2725, p. 1-13.

[84] 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.

[85] F. MÜLLER. *Confluence of the lambda calculus with left-linear algebraic rewriting*, in "Information Processing Letters", 1992, vol. 41, n° 6, p. 293-299.

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

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

[88] F. NEURAUTER, A. MIDDELDORP, H. ZANKL. *Monotonicity Criteria for Polynomial Interpretations over the Naturals*, in "Proceedings of the 6th International Joint Conference on Automated Reasoning, Lecture Notes in Computer Science 6173", 2010.

[89] 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.

[90] 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.

[91] 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.

[92] 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° 1, p. 70–87, http://doi.acm.org/10.1145/1124713.1124719.

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

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

[95] C. STERNAGEL, R. THIEMANN. *Certified Subterm Criterion and Certified Usable Rules*, in "Proceedings of the 21st International Conference on Rewriting Techniques and Applications, Leibniz International Proceedings in Informatics 6", 2010.

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

[97] A. STUMP, C. W. BARRETT, D. L. DILL, J. LEVITT. *A Decision Procedure for an Extensional Theory of Arrays*, in "LICS '01", IEEE Computer Society, 2001, p. 29–37.

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

[99] 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.

[100] H. XI. *Dependent Types for Program Termination Verification*, in "Journal of Higher-Order and Symbolic Computation", 2002, vol. 15, n^{o} 1, p. 91-131.

[101] 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.

[102] 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.