Activity Report 2012

Team FORMES

Formal Methods for Embedded Systems

RESEARCH CENTER
Paris - Rocquencourt

THEME
Programs, Verification and Proofs

Table of contents

Team FORMES

Keywords: Simulation, Formal Proof, Rewriting Theory, Termination, Verification

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

Creation of the Team: January 01, 2009 .

1. Members

Research Scientists

Frédéric Blanqui [Researcher, HdR]
Vania Joloboff [Senior researcher, team leader since 22/05/12]
Jean-Pierre Jouannaud [Senior researcher, team leader until 21/05/12, HdR]
Jean-François Monin [Senior researcher, HdR]
Guillaume Merle [post doc]

Faculty Members

Ming Gu [Tsinghua professor]
Fei He [Tsinghua assistant professor]
Jianqi Li [Tsinghua assistant professor]
Hehua Zhang [Tsinghua assistant professor]
Hai Wan [Tsinghua assistant professor]

PhD Students

Xiaomu Shi [UJF Grenoble]
Kim-Quyen Ly [UJF Grenoble]
Qian Wang [Tsinghua and École Polytechnique]
Jiaxiang Liu [Tsinghua and École Polytechnique]
Hui Kong [Tsinghua]
Liangze Yin [Tsinghua]
Lianyi Zhang [Tsinghua]
Min Zhou [Tsinghua]
Rui Wang [Tsinghua until 30/06/12]

Administrative Assistants

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

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.

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

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 Year

- The automated termination prover HOT developed by Frédéric Blanqui won the 2012 termination competition in the category “higher-order rewriting union beta”.

3. Scientific Foundations

3.1. Rewriting and Type theory

Coq [42] is one of the most popular proof assistant, in the academia and in the industry. Based on the Extended Calculus of Inductive Constructions, Coq has four 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. We are interested in two challenges that one has to face with the development of formal proofs in Coq: the theoretical status of equality on the one hand, and the confidence one may have in Coq's proofs on the other hand. Our answer to the first challenge is CoqMTU, which isolates equality in a theory T, which must be first order, such as Presburger Arithmetic. Our answer to the second challenge is the (manual) certification of CoqMTU in Coq.

Rewriting is at the heart of proof systems such as the Extended Calculus of Constructions on which Coq is based, since mathematical proofs are made of reasonning steps, expressed by the typing rules of a given proof system, and computational steps, expressed by its rewrite rules. The certification of a proof system involves, in particular, proving three main properties of its rewrite rules: subject reduction (rewriting should preserve types), confluence (computations should be deterministic), and termination -computations must always terminate. The fact that falsity is not provable in a given proof system such as CoqMTU follows from the previous properties, while decidability of type-checking may require further work. These meta-theoretical proofs are indeed very complex, although at the same time very repetitive, depending on both the typing rules and the rewrite rules. A challenging research question here is to develop certification tools aiming at automating these proofs. Building such tools requires new results allowing to check subject-reduction, confluence and termination of higher-order calculi that are found in proof systems. Since subject-reduction is usually easy to check while consistency and decidability of type-checking follow, *in general*, from the others, confluence and termination are two very active research topics in this area. A last challenge to achieve these goals is the formalization itself of proof systems.

3.2. Verification

Model checking is an automatic formal verification technique [38]. In order to apply the technique, users have to formally specify desired properties on an abstract model of the system under verification. Model checkers will check whether the abstract model satisfies the given properties. If model checkers are able to prove or disprove the properties on the abstract model, they report the result and terminate. In practice, however, abstract models can be extremely complicated, model checkers may not conclude with reasonable computational resources.

Compositional reasoning is a way to ameliorate the complexity in abstract models [75]. 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 [41]. 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 [40].

Recently, we propose assume-guarantee reasoning with implicit learning [37]. 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 [53], [54]. 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 [46]. 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.

3.3. 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 [55] is an algorithm which returns a correct yes/no answer to a given input decision problem. Many real-world problems can be reduced to the decision problems, making this technique very practical. For example, Intel and AMD are developing solvers for their circuit verification tools, while Microsoft is developing decision procedures for their code analysis tools.

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

A very popular, effective recent trend is Satisfiability Modulo Theories (SMT) [74], 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 [67], and *eager* method [68]. The eager method encodes an SMT problem into an equi-satisfiable SAT problem, while the lazy method employs different theory solvers for each theory and coordinates them appropriately. The eager method does allow the user to express her problem in a natural way, but does not exploit its logical structure to speed up the computation. The lazy approach is more appealing, and has prompted much interest in algorithms for the various background theories important in practice.

Our SMT solver aCiNO is based on the lazy approach. So far, it provides with two (popular) theories only: linear real arithmetic (LRA) and uninterpreted functions (UF). For efficiency consideration, the solver is implemented in an incremental way. It also invokes an online SAT solver, which is now a modified DPLL procedure, so that recovery from conflicts is possible. Our challenge here is twofold: first, to add other theories of interest for the project, we are currently working on fragments of the theory of arrays [61], [34]. 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 [4] We therefore strongly believe that decision procedures, and SMT provers, should come along with a formal assessment of their correctness. As usual, there are two ways: ensure the correctness of an arbitrary output by proving the code, or deliver for each input a certificate ensuring the correctness of the corresponding output when the checker says so. Developing concise certificates together with efficient certificate checkers for the various decision procedures of interest and their combination with SMT is yet another challenge which is at the heart of the project FORMES.

3.4. 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.

[4] 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.

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 [60] 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 [47], [50], [66]. 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) [36], [23] 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 [35]. Another technique to implement a fast ISS is *dynamic translation* [39], [65], [44] which has been favored by many [63], [44], [64], [65] 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 [28] and GXEMUL [49] that emulate to a large extent the behavior of a particular hardware platform. The technique used in QEMU is a form of dynamic translation. The target code is translated directly into machine code using some pre-determined code patterns that have been pre-compiled with the C compiler. Both QEMU and GXEMUL include many device models of open-source C code, but this code is hard to reuse. The functions that emulate device accesses do not have the same profile. The scheduling process of the parallel hardware entities is not specified well enough to guarantee the compatibility between several emulators or re-usability of third-party models using the standards from the electronics industry (e.g. IEEE 1666).

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

- provide different levels of translation with different degrees of accuracy so that users can choose between accurate and slow (for debugging) or less accurate but fast simulation.

- to take advantage of multi-processor simulation hosts to parallelize the simulation;
- to define intermediate representations of programs that optimize the simulation speed and possibly provide a more convenient format for studying properties of the simulated programs.

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

3.5. Trustworthy Software

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

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

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

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

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

4. Application Domains

4.1. Simulation

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

4.2. Certified Compilation for Embedded systems

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

4.3. Distributed Systems

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

4.4. Security

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

5. Software

5.1. aCiNO

Participants: Fei He [correspondant], Min Zhou.

aCiNO is an SMT (Satisfiability Modulo Theory) solver based on a Nelson-Oppen [62] architecture, and written in C++. Currently, two popular theories are considered: linear real arithmetic (LRA) and uninterpreted functions (UF). A lazy approach is used for solving SMT problem. For efficiency consideration, the solver is implemented in an incremental way. It also invokes an online SAT solver, which is now a modified MiniSAT, so that recovery from conflict is possible.

5.2. CoLoR

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

CoLoR is a Coq [42] library on rewriting theory and termination of more than 72,000 lines of code [4]. It provides definitions and theorems for:

- Mathematical structures: relations, (ordered) semi-rings.
- Data structures: lists, vectors, polynomials with multiple variables, finite multisets, matrices.
- 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.

CoLoR is 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

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

The proof-assistant Coq is based on a complex type theory, which resulted from various extensions of the Calculus of Constructions studied independently from each other. With the collaboration of Bruno Barras, we decided to address the challenge of proving the real type theory underlying Coq, and even, indeed, of its recent extension CoqMT developed in FORMES by Pierre-Yves Strub. To this end, we have studied formally the theory CoqMTU, which extends the pure Calculus of Constructions by inductive types, a predicative hierarchy of universes, and a decidable theory T for some first-order inductive types [1]. Recently, we were able to announce the complete certification of CoqMTU in Coq augmented with appropriate intuitionistic set-theoretic axioms in order to fight Gödel's incompleteness theorem, a work which has not been published yet. As a consequence, Coq and CoqMTU are the first proof assistants which consistency (relative to intuitionistic set theory IZF augmented with the afore-mentioned axioms) is formally entirely proved (in Coq). While previous formal proofs for Coq and other proof assistants all assumed strong normalization, the present one *proves* strong normalization thanks to the new notion of *strongly-normalizing* model introduced by Bruno Barras. While consistency is done already, decidability of type-checking remains to be done. This is a straightforward consequence for Coq, but a non-trivial task for CoqMTU because of the interaction between inductive types and the first-order theory T. It should however be announced around the turn of the year. We consider this work as a major scientific achievement of the team.

5.4. EDOLA

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

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

EDOLA [72] is an integrated tool for domain-specific modeling and verification of PLC applications [70]. It is based on a domain-specific modeling language to describe system models. It supports both model checking and automatic theorem proving techniques for verification. The goal of this tool is to possess both the usability in domain modeling, the reusability in its architecture and the capability of automatic verification.

For the moment, we have developed a prototype of the EDOLA language, which can easily describe the features of PLC applications like the scan cycle mechanism, the pattern of environment model, time constraints and five property patterns. TLA+ [56] was chosen as the intermediate language to implement the automatic verification of EDOLA models. A prototype of EDOLA has also been developed, which comes along with an editor to help writing EDOLA models. To automatically verify properties on EDOLA models, it provides the interface for both a model checker TLC [56] and a first-order theorem prover SPASS [71].

5.5. HOT

Participant: Frédéric Blanqui [correspondant].

HOT is an automated termination prover for higher-order rewrite systems based on the notion of computability closure and size annotation [13]. It won the 2012 competition in the category "higher-order rewriting union beta". The sources are not public.

5.6. Moca

Participant: Frédéric Blanqui [correspondant].

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

Moca is a construction functions generator for OCaml [57] data types with invariants.

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

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

Moca supports two kinds of relations:

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

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

Moca is distributed under QPL on http://moca.inria.fr/.

5.7. Rainbow

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

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

Rainbow is distributed under the CeCILL license on http://color.inria.fr/rainbow.html. See the website for more information.

5.8. SimSoC

Participant: Vania Joloboff [correspondant].

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

SimSoC is based on SystemC standard and uses Transaction Level Modeling for interactions between the simulation models. The current version of SimSoC is based on the open source libraries from the OSCI Consortium: SystemC version 2.2 and TLM 2.0.1 [52], [25]. Hardware components are modeled as TLM models, and since TLM is itself based on SystemC, the simulation is driven by the SystemC kernel. We use standard, unmodified, SystemC (version 2.2), hence the simulator has a single simulation loop.

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

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

5.9. SimSoC-Cert

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

SimSoC-Cert is a set of tools that can automatically generate in various target languages (Coq and C) the decoding functions and the state transition functions of each instruction and addressing mode of the ARMv6 architecture manual [22] (implemented by the ARM11 processor family) but the Thumb and coprocessor instructions. The input of SimSoC-Cert is the ARMv6 architecture manual itself.

Based on this, we first developed *simlight* (8000 generated lines of C, plus 1500 hand-written lines of C), a simulator for ARMv6 programs using no peripheral and no coprocessor. Next, we developed *simlight2*, a fast ARMv6 simulator integrated inside a SystemC/TLM module, now part of SimSoC v0.7.

We can also generate similar programs for SH4 [24] but this is still experimental (work done by Frédéric Tuong in 2011).

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

6. New Results

6.1. Higher-Order Abstract Syntax

This recently started project funded by the National Science Foundation of China aims at setting up a generic infrastructure for representating logical systems and automate their meta-theoretical study. We view a logical system as a type theory made of three components: a language of terms, types being particular terms; a set of typing rules; and a set of computational rules described by typed higher-order rewrite rules.

There are several challenges in this project. The first is to define logical frameworks which are expressive enough -at least as expressive as Girard's System F or Edingburgh's LF- to define the syntax and semantics of rich type theories, such as CoqMTU as an extreme example. A second challenge is to develop new techniques for checking the three main properties of higher-order rewrite rules: type preservation -which is usually easy-, confluence and termination. Our work here has progressed steadily, in paticular with new advanced techniques for checking termination and confluence described next. A third challenge is to formalize these results in Coq, in order to provide proof certificates for particular cases. The fourth challenge is to build a a general infrastructure in Coq in which all these techniques become available in order to study particular logical systems.

As initial steps, we undertook the following formalizations :

- Hua Mei implemented an intensional framework for simply typed lambda-calculus in Coq, where α- and β-conversions have been axiomized.
- Frédéric Blanqui has formalized in Coq the pure lambda-calculus following the definition of Curry and Feys in [43] (named variables and explicit alpha-equivalence), and the proof of termination of β-reduction for simply-typed λ-terms based on computability predicates [51]. To the best of his knowledge, this is the first formalization of the termination of β-reduction using named variables and explicit alpha-equivalence, all the other formalizations using De Bruijn indices [73] or nominal logic [48].
- Qian Wang formalized completely the theory of CoqMTU in Coq augmented with strong set-theoretic axioms in order to get around Gödel's incompleteness theorem. This is described in more details next.

6.2. CoqMTU

The proof-assistant Coq is based on a complex type theory, which resulted from various extensions of the Calculus of Constructions studied independently fromf each other. With Bruno Barras, we decided to address the challenge of proving the real type theory underlying Coq, and even, indeed, its recent extension CoqMT. To this end, we have studied formally the theory CoqMTU, which extends the calculus of Constructions with inductive types, a predicative hierarchy of universes and a decidable theory T for some first-order inductive types for which large elimination is no more available. This work has been published at LICS [1]. It leaves open the question whether large elimination can be accomadated for those inductive types which carry along a decidable theory T. This problem has been solved recently by Wang, who constructed a set-theoretic model of CoqMTU with strong elimination.

6.3. Normal Rewriting

There are many forms of rewriting used in the litterature: plain rewriting (rules are fired via plain pattern matching), rewriting modulo T (rules are fired via pattern matching modulo T), higher-order rewriting (rules are fired via higher-order pattern matching, but apply to simply typed lambda-terms terms provided the redex is of base type and in beta-normal eta-long form). For each of these rewriting mechanisms, there are results describing how to check confluence and termination.

Regarding confluence, these results describe which *critical pairs* must be computed in order to check the confluence property of the rewriting relation, assuming some termination property. In [17], we describe a general abstract result which can then be instantiated to all of the previous cases, and removes the assumptions above for higher-order rewriting. This is done via two novel notions: abstract positional rewriting allows us to capture the notion of critical peak without having to talk about a specific term structure; abstract normal rewriting with a triple (R, S, E) allows us to capture all different forms of rewriting: $S = E = \varnothing$ for plain rewriting; $S = \varnothing$ for rewriting modulo; E is alpha-conversion for higher-order rewriting, while the set of simplifiers S is made of beta-reduction and eta-expansion, R being the set of user-defined rules. Of course, there are other applications of normal rewriting described in the paper: for first-order computations, but also for higher-order computations at higher types, or using eta-reduction instead of eta-expansion, therefore solving a long-standing open problem.

Regarding termination, these results are very preliminary. In a recent paper submitted to ACM Transactions on Computational Logics, we extend the termination proof methods for higher-order computations based on plain pattern matching to higher-order rewriting systems based on higher-order pattern matching. We accomodate, for the one hand, with a weakly polymorphic, algebraic extension of Church's simply typed λ-calculus, and on the other hand, with any use of eta, as a reduction, as an expansion or as an equation. User's rules may be of any type in this type system, either a base, functional, or polymorphic type. Our techniques fit well with higher-order reduction orderings, such as the computability path ordering, but can also be used by other techniques, such as higher-order dependency pairs. All examples of normal higher-order rewrite rules that can be found in the litterature can be treated by our techniques, even those for which termination is by no means obvious to the expert.

6.4. Decreasing Diagrams

Based on the so-called Newman's lemma, the method for checking confluence introduced in the former paragraph applies to terminating computations. A completely different technique based on the so-called Hindley-Rosen's lemma applies when computation do not terminate, and is at the basis of Tait's confluence proof for the pure lambda-calculus. In recent papers, van Oostrom succeeded to capture both within a single framework thanks to the notion of decreasing diagram of a labelled abstract relation [76], see also [11] for an improved proof. Decreasing diagrams are specific convertibility proofs for local peaks, which labels are smaller in some sense than those of the local peak they aim at replacing. Any convertibility proof can then be converted into a confluence proof by recursively replacing its local peaks by their associated decreasing diagrams. Using a subtle characterization of confluence for arbitrary (possibly non-terminating) relations by cofinal derivations due to Klop [11], van Oostrom showed that any confluent relation which convertibility classes are countable, can be labelled in a way that makes it a labelled relation satisfying the decreasing diagram condition.

In [15], we first give a new, simple proof of van Oostrom's initial result based on a subtle well-founded order on conversions, and generalize it to rewriting modulo by using *strongly coherent cliffs* as an analog of decreasing diagrams for peaks. We then extend Klop's cofinal derivations to *cofinal streams*, and prove again a completeness result under the strong coherence assumption. Finally, we derive from these results a new, compact proof of Toyama's theorem that confluence is a modular property of rewriting systems built on disjoint vocabularies, and extend it to rewriting modulo when strong coherence is satisfied.

We are now trying to get rid of the strong coherence assumption by introducing a weaker analog of decreasing diagrams, *decreasing cliffs*. A preliminary result was presented early november at the Japanese Term Rewriting Workshop in Sendai.

This line of work is very promising. We expect it will eventually lead to the solution of an old open problem, the characterization of a class of non-left linear, non-terminating rewrite systems for which confluence is decidable by means of (parallel) critical pairs. We believe that the implementation of such a result would be impact the way confluence proofs are carried out, including in type theory.

6.5. Higher-order Reduction Orderings

Since HORPO , several higher-order reduction orderings have been described, based on either Dershowitz's RPO , Blanqui-Jouannaud-Okada's Computational Closure , and Arts and Giesel' dependency pairs . Our work continues in three different directions:

- CPO is an order for simply typed lambda-terms that allows to show strong normalization of beta-reduction even in presence of higher-order rewrite rules provided these rules decrease in the ordering [32]. It is currently the only automated mechanism that achieves non-trivial computations by turning Girard's computability predicates method into a usable tool. It has been shown that CPO can handle weakly polymorphic type disciplines, as well as inductive types. Recently, we have shown that CPO scales up to dependently typed calculi as LF. We are currently writing a paper describing CPO and its extensions to calculi with inductive and dependent types which should be submitted to a journal by the end of the year.
- Frédéric Blanqui defended his "Habilitation à diriger des recherches" at the University Denis Diderot (Paris 7) on July 13. In [13], he gives a synthetic view on how the notion of computability closure can be used to prove the termination of various kinds of rewrite relations (class rewriting or rewriting with matching modulo), and how it relates with other notions (dependency pairs, semantic labeling, and HORPO, the predecessor of CPO.
- Frédéric Blanqui has developed an automated termination prover called HOT based on the above work on the computability closure and his former work on size annotations [31]. For its first participation, HOT won the international competition on termination in the category "higher-order rewriting union beta".

6.6. Certification of Termination Proofs

Frédéric Blanqui and Kim Quyen Ly continued to work on the development of a new version of Rainbow based on Coq extraction mechanism [59]. We developed a tool generating from an XSD file, Coq and OCaml data structures representing the XML types defined the XSD file, and OCaml parsing functions for generating such data structures from an XML file. The main difficulty was to topologically reorder the XSD type definitions in order to get simple and well defined Coq data structures. We also defined and proved in Coq a function for checking the correctness of termination certificates based on the DP transformation [26]. The main difficulty was to manage the evolution of the arity function along the transformation. Indeed, to simplify the translation of CPF elements into the data structures used in CoLoR [30], we decided to use a fixed but infinite set of symbols [69]. However the arity function need to be updated along the transformations applied to the system. These results are presented in [20].

6.7. Certification of Moca

Frédéric Blanqui has formalized in Coq and proved the correctness and completeness of the construction functions generated by Moca for the theory of groups [29]. The first difficulty is to represent the Moca functions themselves in a faithful way because, in Coq, there is no "when" clauses and "match" constructions are expanded into elementary "case" constructions with no tuple patterns and patterns of depth one only. In addition, Coq termination checker only accepts functions with exactly one structurally decreasing argument, which is generally not the case of Moca functions. The second difficulty is the completeness proof: it requires the use of intermediate data structures for reasoning on normal forms. During his internship, Rémi Nollet (L3, ENS Lyon) improved the representation of OCaml functions by using inductive predicates, and extended the correctness proof to commutative groups.

6.8. First steps towards the certification of an ARM simulator

The simulation of Systems-on-Chip (SoC) is nowadays a hot topic because, beyond providing many debugging facilities, it allows the development of dedicated software before the hardware is available. Low-consumption CPUs such as ARM play a central role in SoC. However, the effectiveness of simulation depends on the faithfulness of the simulator. To this effect, we started to prove significant parts of such a simulator, SimSoC. Basically, on one hand, we develop a Coq formal model of the ARM architecture while on the other hand, we consider a version of the simulator including components written in Compcert-C [58]. Then we prove that the simulation of ARM operations, according to Compcert-C formal semantics, conforms to the expected formal model of ARM. Size issues are partly dealt with using automatic generation of significant parts of the Coq model and of SimSoC from the official textual definition of ARM [3]. A second step was achieved in [12], with the proof a significant instruction (ADC, Add with Carry). A crucial technical issue was then raised: facilitating reasoning by inversion on the rules defined in Compcert-C. Hundreds such steps are required for a single instruction, and each of them generates a dozen of new names. Relying on Coq tactic inversion results in unmanageable scripts, very fragile and difficult to maintain. In 2012 we dealt with this issue by designing our own inversion mechanism, allowing us to improve automation of the proof, while keeping enough command so that interactive steps refer to controlled names. It was then possible to get a much shorter proof on ADC and to prove at least one instruction in each category of the ARM instruction set.

6.9. Certified implementation of BIP

BIP (*Behavior, Interaction, Priority*) is a component-based language designed at VERIMAG for modeling and programming complex embedded systems [27]. A BIP model is essentially a set of atomic components described with explicit states and transitions, composed together in a hierarchical way. The main original feature of BIP lies in a very rich notion of *connector* for defining interactions between components [33]. An efficient implementation of BIP in C++ is already available at VERIMAG.

Building on our previous experience on SimSoC, we started to work on a certified implementation of BIP. Our long term objective is to propose a certified compilation chain from BIP models to embedded code, through a first translation from BIP to Compcert-C.

In 2012 we focused on a simple subset of BIP Currently, we have a first definition of a formal semantics of this subset in Coq, in two versions: an relational version, inspired by a rule-based operational semantics, and a functional version, which specifies a possible implementation of the relational version (in particular, it includes a scheduler). We also produce a Compcert-C code which is expected to behave exactly like the functional semantics, and we started to state and prove corresponding statements on very simple BIP models.

6.10. Formal model and proofs for Netlog protocols

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.

Jean-François Monin, Stéphane Grumbach (formerly LIAMA/Netquest) and Yuxin Deng (Jiaotong University, Shanghai) 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. This framework could be applied to an algorithm constructing a Breadth-First Search Spanning Tree (BFS) in a distributed system [45].

In 2011, Jean-François Monin and Meixian Chen (Jiaotong Shanghai) generalized the model in order to take the removal of datalog facts into account, and used the improved framework to Prim's algorithm. In 2012, this work was slightly improved and published in [16].

6.11. Formalisation of security APIs for mobile phones

This work is in cooperation with Nokia Beijing, who was interested by the application of verification technologies to mobile phones. We decided to focus on security APIs, considering that mobile devices are commonly used by end-users to store their personnal data (e.g., passwords), while running all sort of downloaded applications at the same time.

For 2012, we (including Nokia) agreed to consider devices under Android, though Nokia switched to windows, in order to circumvent copyright issues.

Three models and corresponding sets of APIs for password storage applications on Android were developed. Each model fixes some bugs of the previous one and introduces a new feature. We consider the third model is enough for the basic function and well built to be safe. Then, a full Coq proof of the third model was developed as well as its corresponding API's security property. A suitable abstraction of the application on the phone within its environment is described as a state transition system. Then we proved by induction that the expected secrets actually remain secret at any reachable state.

6.12. Trace Analysis

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

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

We have started to build a trace analysis tool. It includes a language which allows expression of time-related formulas as a subset of LTL, but is simple to formulate expressions. When this language is compiled, the compiler generates two outputs:

- a filter script that will help reduce the size of the trace file.
- a program that analyzes such trace files to find whether the formula is satisfied.

When compiling one trace language input file, it generates a filter script. The filter script is a set of data descriptors. It describes which events from the simulator must be traced and which should be ignored. Then during the simulation, the filter is loaded and only the required output is generated.

We have started to design a trace language and a compiler, and extended the SimSoC simulator to support generation of trace files with a filter. A first version of the trace language compiler has been implemented in OCAML, which generates OCAML programs for trace analysis. In the current version under development, the filters are not yet parallelized with simulation.

7. Bilateral Contracts and Grants with Industry

7.1. Bilateral Contracts with Industry

We obtained a contract of 100 000 Chinese RMB (12 500 Euros) with Nokia Research Center in Beijing to study formal proofs of security API's in Android mobile phones.

8. Partnerships and Cooperations

8.1. National Initiatives

8.1.1. Tsinghua Grant

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

title: An Intensional Logical Framework and Its Implementation

PIs: Jean-Pierre Jouannaud, Jianqi Li

duration: 2011 - 2012

Amount: 100,000 RMB

8.1.2. NSFC Grant

contract: National Science Foundation of China grant 61272002

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

PIs: Jean-Pierre Jouannaud, Jianqi Li

duration : 2013-2016

Amount: 600,000 RMB

8.2. International Initiatives

8.2.1. Inria International Partners

FORMES is an international project from LIAMA in China, located on two sites, Tsinghua University in Beijing, and CAS Shenzhen Institute of Advanced Technologies in Shenzhen. In addition this project has had collaborations with CAS Institute of Software and Harbin Engineering University in 2012.

8.3. International Research Visitors

8.3.1. Visits of International Scientists

FORMES received visiting Pr Nachum Dershowitz from Israel at Tsinghua for a short stay.

8.3.1.1. Internships

Rémi Nollet (L3, ENS Lyon) did an internship at Inria Rocquencourt co-supervised by Frédéric Blanqui and Pierre Weis on the certification of construction functions generated by Moca.

8.3.2. Visits to International Teams

Jean-Pierre Jouannaud, invited in Barcelone, UTC, LSI-Lab, September 2012.

Frédéric Blanqui visited the Institute of Applied Mechanics and Informatics (IAMI) of the Vietnamese Acadamy of Sciences at Ho Chi Minh City.

9. Dissemination

9.1. Scientific Animation

Frédéric Blanqui is member of the steering committe of the international conference on rewriting techniques and its applications (RTA).

Frédéric Blanqui was member of the program committee of the 6th International Workshop on Higher-Order Rewriting (HOR'12).

Jean-Pierre Jouannaud was member of the program committee of WOLLIC'2012.

Jean-Pierre Jouannaud is a member of the steering committee of LICS.

Jean-Pierre Jouannaud is a member of the Advisory Committee of Academia Sinica, Taipei.

Jean-Pierre Jouannaud is a member of the committee for the Ackermann prize (2011–2013).

Jean-Pierre Jouannaud was a member of the committee for the LICS test of time award (2012).

Vania Joloboff was invited speaker at China Open Source Week in Nanjing.

Vania Joloboff was invited speaker at a professional embedded systems workshop in Tokyo.

9.2. Teaching - Supervision - Juries

9.2.1. Teaching

Ming Gu is the director of the School of Software, Tsinghua. She teaches at all levels.

Last year undergraduate: Jianqi Li, An Introduction to Theories of Software, 16 hours, L1, Tsinghua University, China

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

Licence : Jean-François Monin, Introduction to Functional Programming, 25 hours, L3, Beijing Jiatong University, China

Master: He Fei, Formal verification for software systems, 32 hours, Tsinghua University, China

Master: Jianqi Li, The Formal Semantics of Programming Languages, 32 hours 2012, M1, Tsinghua University, China

Master : Jean-François Monin, Complements on Coq, 25 hours, M1-M2, Beijing University (PKU), China

Doctorate : Frédéric Blanqui, Introduction to domain theory and topology, 3 hours, ISCAS, Beijing, China

Doctorate : Jean-François Monin, Coq Summer School, 20 hours, ECNU Shanghai, China

9.2.2. Supervision

PhD & HdR :

PhD in progress : Jiaxiang LIU, Decreasing diagrams for confluence, Sept. 2011 Jean-Pierre Jouannaud

PhD in progress : Qian WANG, A Complete Formalization of Coq Modulo Theory, Sept. 2010, Jean-Pierre Jouannaud

PhD in progress : Xiaomu SHI, Formalisation and Proof of an Instruction Set Simulator, nov. 2009, Jean-François Monin and Vania Joloboff

PhD in progress : Kim Quyen LY, Automated Verification of Termination Certificates, nov. 2010, Frédéric Blanqui

9.2.3. Juries

Jean-Pierre Jouannaud: ENS-Cachan, habilitation, Florent Jacquemard (rapporteur)

Jean-François Monin: Paris-7, PhD, Stéphane Glondu (rapporteur)

Jean-François Monin participated to the recruitment of Chinese students for the polytechnic engineering schools of Grenoble, Marseille, Montpellier and Nice.

9.3. Popularization

Jean-Pierre Jouannaud gave presentations about formal proofs and related topics at Tsinghua, one in the department of applied mathematics in november 2011, and one in the department of computer science in june 2011.

Vania Joloboff has given presentations about simulation at

- Shanghai Fudan University
- Guangzhou Normal University
- a workshop in Japan about Trustworthy Embedded Systems

Jean-François Monin gave presentations about formal methods and our research at FORMES to Jiaotong (Shanghai) in June 2011, UPC (Qingdao) in October 2012, HIT (Harbin) in November 2012 and ECNU (Shanghai) in December 2012.

Jean-François Monin initiated formal agreements between several Chinese universities (Wuhan university, Beijing Jiaotong, UPC, HIT) and the Polytech Group or UJF, and developed the existing formal cooperation between Beihang and UJF.

10. Bibliography

Major publications by the team in recent years

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

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

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

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

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

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

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

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

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

[10] 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.

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

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

Publications of the year

Doctoral Dissertations and Habilitation Theses

[13] F. BLANQUI. *Terminaison des systèmes de réécriture d'ordre supérieur basée sur la notion de clôture de calculabilité*, Université Paris-Diderot - Paris VII, July 2012, Habilitation à Diriger des Recherches, http://hal.inria.fr/tel-00724233.

[14] R. WANG. *Component based modelling method for PLC Control Software*, Tsinghua University, 2012, In Chinese.

Articles in International Peer-Reviewed Journals

[15] J.-P. JOUANNAUD, J. LIU. *From Diagrammatic Confluence to Modularity*, in "Theoretical Computer Science", November 2012, vol. 9032 [*DOI :* 10.1016/J.TCS.2012.08.030], http://hal.inria.fr/hal-00730272.

International Conferences with Proceedings

[16] M. CHEN, J.-F. MONIN. *Formal Verification of Netlog Protocols*, in "TASE", Beijing, China, T. MARGARIA, Z. QIU, H. YANG (editors), IEEE, July 2012, http://hal.inria.fr/hal-00733634.

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

[18] L. LIU, F. FELGNER, G. FREY. *Introducing Explicit Causality in Object-Oriented Hybrid System Modeling*, in "9th International Conference on Modeling, Optimization & SIMulation", Bordeaux, France, June 2012, http://hal.inria.fr/hal-00728581.

[19] W. MENG, F. HE, B.-Y. WANG, Q. LIU. *Thread-Modular Model Checking with Iterative Refinement*, in "NFM 2012 - 4th International Conference on NASA Formal Methods", Norfolk, Virginia, United States, April 2012, http://hal.inria.fr/hal-00730342.

Conferences without Proceedings

[20] F. BLANQUI, K. Q. LY. *Automated verification of termination certificates*, in "15th National Symposium of Selected ICT Problems", Hanoi, Viet Nam, November 2012, http://hal.inria.fr/hal-00763495.

[21] Z. ZUYU, V. JOLOBOFF, X. ZHOU, C. HELMSTETTER. *Fast Dynamic Translation Using LLVM On Multi-Core Hosts*, in "5th Workshop on Architectural and Microarchitectural Support for Binary Translation (AMAS-BT)", Portland, Oregon, United States, ACM (editor), Intel Corporation, June 2012, http://hal.inria.fr/hal-00777156.

References in notes

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

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

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

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

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

[27] A. BASU, S. BENSALEM, M. BOZGA, J. COMBAZ, M. JABER, T.-H. NGUYEN, J. SIFAKIS. *Rigorous Component-Based System Design Using the BIP Framework*, in "IEEE Software", 2011, vol. 28, n^{o} 3, p. 41-48.

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

[29] F. BLANQUI, T. HARDIN, P. WEIS. *On the implementation of construction functions for non-free concrete data types*, in "Proceedings of the 16th European Symposium on Programming, Lecture Notes in Computer Science 4421", 2007.

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

[31] 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, Colloque avec actes et comité de lecture. internationale, http://hal.inria.fr/inria-00100254/en/.

[32] F. BLANQUI, J.-P. JOUANNAUD, A. RUBIO. *The computability path ordering: the end of a quest*, in "7th EACSL Annual Conference on Computer Science Logic - CSL'08", Bertinoro Italie, LNCS, 2008, vol. 5213, http://hal.inria.fr/inria-00288209/en/.

[33] S. BLIUDZE, J. SIFAKIS. *The algebra of connectors: structuring interaction in BIP*, in "EMSOFT '07: Proceedings of the 7th ACM & IEEE international conference on Embedded software", New York, NY, USA, ACM, 2007, p. 11–20, http://doi.acm.org/10.1145/1289927.1289935.

[34] 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.

[35] 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.

[36] 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.

[37] 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/.

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

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

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

[41] 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.

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

[43] H. B. CURRY, R. FEYS. *Combinatory Logic*, North-Holland, 1958.

[44] 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.

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

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

[47] 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.

[48] M. J. GABBAY, A. M. PITTS. *A New Approach to Abstract Syntax Involving Binders*, in "Proceedings of the 14th IEEE Symposium on Logic in Computer Science", 1999.

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

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

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

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

[53] 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/.

[54] 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/.

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

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

[57] 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/.

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

[59] P. LETOUZEY. *Programmation fonctionnelle certifiée: l'extraction de programmes dans l'assistant Coq*, Université Paris-Sud, France, 2004.

[60] 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.

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

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

[63] 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.

[64] 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.

[65] 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.

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

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

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

[69] C. STERNAGEL, R. THIEMANN. *Signature extensions preserve termination - An alternative proof via dependency pairs*, in "Proceedings of the 24th International Conference on Computer Science Logic, Lecture Notes in Computer Science 6247", 2010.

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

[71] 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.

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

[73] N. DE BRUIJN. *Lambda-Calculus Notation with Nameless Dummies: a Tool for Automatic Formula Manipulation with Application to the Church-Rosser Theorem*, in "Indagationes Mathematicae", 1972, vol. 34, n^{o} 5, p. 381-392.

[74] 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.

[75] 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.

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