
IN PARTNERSHIP WITH:
**CNRS**

**Université de Lorraine**

Activity Report 2012

# Project-Team PAREO

## Formal islands: foundations and applications

IN COLLABORATION WITH: Laboratoire lorrain de recherche en informatique et ses applications (LORIA)

RESEARCH CENTER
**Nancy - Grand Est**

THEME
**Programs, Verification and Proofs**

# Table of contents

# Project-Team PAREO

**Keywords:** Programming Languages, Compiling, Formal Methods, Type Systems, Security, Proofs Of Programs

*Creation of the Project-Team:* January 01, 2008 , *Updated into Project-Team:* January 01, 2011 .

# 1. Members

**Faculty Members**

Horatiu Cirstea [Professor, Université de Lorraine, HdR]
Pierre-Etienne Moreau [Team Leader, Professor, Université de Lorraine, HdR]
Sergueï Lenglet [Associate Professor, Université de Lorraine, since September 1]

**External Collaborators**

Claude Kirchner [Senior Researcher, Inria, HdR]
Hélène Kirchner [Senior Researcher, Inria, HdR]
Sorin Stratulat [Associate Professor, Université de Lorraine]

**PhD Students**

Jean-Christophe Bach [CORDI QUARTEFT]
Claudia Tavares [Brazil, until March 2]

**Post-Doctoral Fellow**

Christophe Calvès [ATER, Université de Lorraine, since September 1]

**Administrative Assistant**

Laurence Benini

# 2. Overall Objectives

## 2.1. Overall Objectives

The PAREO team aims at designing and implementing tools for the specification, analysis and verification of software and systems. At the heart of our project is therefore the will to study fundamental aspects of programming languages (logic, semantics, algorithmic, etc.) and to make major contributions to the design of new programming languages. An important part of our research effort will be dedicated to the design of new fundamental concepts and tools to analyze existing programs and systems. To achieve this goal we focus on:

- the improvement of theoretical foundations of rewriting and deduction;
- the integration of the corresponding formal methods in programming and verification environments;
- the practical applications of the proposed formalisms.

## 2.2. Highlights of the Year

BEST PAPER AWARD :
[14] **A Unified View of Induction Reasoning for First-Order Logic in Turing-100, The Alan Turing Centenary Conference**. S. STRATULAT.

# 3. Scientific Foundations

## 3.1. Introduction

It is a common claim that rewriting is ubiquitous in computer science and mathematical logic. And indeed the rewriting concept appears from very theoretical settings to very practical implementations. Some extreme examples are the mail system under Unix that uses rules in order to rewrite mail addresses in canonical forms (see the `/etc/sendmail.cf` file in the configuration of the mail system) and the transition rules describing the behaviors of tree automata. Rewriting is used in semantics in order to describe the meaning of programming languages [28] as well as in program transformations like, for example, re-engineering of Cobol programs [36]. It is used in order to compute, implicitly or explicitly as in Mathematica or MuPAD, but also to perform deduction when describing by inference rules a logic [24], a theorem prover [26] or a constraint solver [27]. It is of course central in systems making the notion of rule an explicit and first class object, like expert systems, programming languages based on equational logic, algebraic specifications, functional programming and transition systems.

In this context, the study of the theoretical foundations of rewriting have to be continued and effective rewrite based tools should be developed. The extensions of first-order rewriting with higher-order and higher-dimension features are hot topics and these research directions naturally encompass the study of the rewriting calculus, of polygraphs and of their interaction. The usefulness of these concepts becomes more clear when they are implemented and a considerable effort is thus put nowadays in the development of expressive and efficient rewrite based programming languages.

## 3.2. Rule-based programming languages

Programming languages are formalisms used to describe programs, applications, or software which aim to be executed on a given hardware. In principle, any Turing complete language is sufficient to describe the computations we want to perform. However, in practice the choice of the programming language is important because it helps to be effective and to improve the quality of the software. For instance, a web application is rarely developed using a Turing machine or assembly language. By choosing an adequate formalism, it becomes easier to reason about the program, to analyze, certify, transform, optimize, or compile it. The choice of the programming language also has an impact on the quality of the software. By providing high-level constructs as well as static verifications, like typing, we can have an impact on the software design, allowing more expressiveness, more modularity, and a better reuse of code. This also improves the productivity of the programmer, and contributes to reducing the presence of errors.

The quality of a programming language depends on two main factors. First, the *intrinsic design*, which describes the programming model, the data model, the features provided by the language, as well as the semantics of the constructs. The second factor is the programmer and the application which is targeted. A language is not necessarily good for a given application if the concepts of the application domain cannot be easily manipulated. Similarly, it may not be good for a given person if the constructs provided by the language are not correctly understood by the programmer.

In the *Pareo* group we target a population of programmers interested in improving the long-term maintainability and the quality of their software, as well as their efficiency in implementing complex algorithms. Our privileged domain of application is large since it concerns the development of *transformations*. This ranges from the transformation of textual or structured documents such as XML, to the analysis and the transformation of programs and models. This also includes the development of tools such as theorem provers, proof assistants, or model checkers, where the transformations of proofs and the transitions between states play a crucial role. In that context, the *expressiveness* of the programming language is important. Indeed, complex encodings into low level data structures should be avoided, in contrast to high level notions such as abstract types and transformation rules that should be provided.

It is now well established that the notions of *term* and *rewrite rule* are two universal abstractions well suited to model tree based data types and the transformations that can be done upon them. Over the last ten years we have developed a strong experience in designing and programming with rule based languages [29], [20], [18]. We have introduced and studied the notion of *strategy* [19], which is a way to control how the rules should be applied. This provides the separation which is essential to isolate the logic and to make the rules reusable in different contexts.

To improve the quality of programs, it is also essential to have a clear description of their intended behaviors. For that, the *semantics* of the programming language should be formally specified.

There is still a lot of progress to be done in these directions. In particular, rule based programming can be made even more expressive by extending the existing matching algorithms to context-matching or to new data structures such as graphs or polygraphs. New algorithms and implementation techniques have to be found to improve the efficiency and make the rule based programming approach effective on large problems. Separating the rules from the control is very important. This is done by introducing a language for describing strategies. We still have to invent new formalisms and new strategy primitives which are both expressive enough and theoretically well grounded. A challenge is to find a good strategy language we can reason about, to prove termination properties for instance.

On the static analysis side, new formalized typing algorithms are needed to properly integrate rule based programming into already existing host languages such as Java. The notion of traversal strategy merits to be better studied in order to become more flexible and still provide a guarantee that the result of a transformation is correctly typed.

## 3.3. Rewriting calculus

The huge diversity of the rewriting concept is obvious and when one wants to focus on the underlying notions, it becomes quickly clear that several technical points should be settled. For example, what kind of objects are rewritten? Terms, graphs, strings, sets, multisets, others? Once we have established this, what is a rewrite rule? What is a left-hand side, a right-hand side, a condition, a context? And then, what is the effect of a rule application? This leads immediately to defining more technical concepts like variables in bound or free situations, substitutions and substitution application, matching, replacement; all notions being specific to the kind of objects that have to be rewritten. Once this is solved one has to understand the meaning of the application of a set of rules on (classes of) objects. And last but not least, depending on the intended use of rewriting, one would like to define an induced relation, or a logic, or a calculus.

In this very general picture, we have introduced a calculus whose main design concept is to make all the basic ingredients of rewriting explicit objects, in particular the notions of rule *application* and *result*. We concentrate on *term* rewriting, we introduce a very general notion of rewrite rule and we make the rule application and result explicit concepts. These are the basic ingredients of the *rewriting*- or $\rho$-calculus whose originality comes from the fact that terms, rules, rule application and application strategies are all treated at the object level (a rule can be applied on a rule for instance).

The $\lambda$-calculus is usually put forward as the abstract computational model underlying functional programming. However, modern functional programming languages have pattern-matching features which cannot be directly expressed in the $\lambda$-calculus. To palliate this problem, pattern-calculi [34], [31], [25] have been introduced. The rewriting calculus is also a pattern calculus that combines the expressiveness of pure functional calculi and algebraic term rewriting. This calculus is designed and used for logical and semantical purposes. It could be equipped with powerful type systems and used for expressing the semantics of rule based as well as object oriented languages. It allows one to naturally express exception handling mechanisms and elaborated rewriting strategies. It can be also extended with imperative features and cyclic data structures.

The study of the rewriting calculus turns out to be extremely successful in terms of fundamental results and of applications [22]. Different instances of this calculus together with their corresponding type systems have been proposed and studied. The expressive power of this calculus was illustrated by comparing it with similar

formalisms and in particular by giving a typed encoding of standard strategies used in first-order rewriting and classical rewrite based languages like *ELAN* and *Tom*.

# 4. Application Domains

## 4.1. Application Domains

Beside the theoretical transfer that can be performed via the cooperations or the scientific publications, an important part of the research done in the *Pareo* group team is published within software. *Tom* is our flagship implementation. It is available via the Inria Gforge (http://gforge.inria.fr) and is one of the most visited and downloaded projects. The integration of high-level constructs in a widely used programming language such as Java may have an impact in the following areas:

- Teaching: when (for good or bad reasons) functional programming is not taught nor used, *Tom* is an interesting alternative to exemplify the notions of abstract data type and pattern-matching in a Java object oriented course.
- Software quality: it is now well established that functional languages such as Caml are very successful to produce high-assurance software as well as tools used for software certification. In the same vein, *Tom* is very well suited to develop, in Java, tools such as provers, model checkers, or static analyzers.
- Symbolic transformation: the use of formal anchors makes possible the transformation of low-level data structures such as C structures or arrays, using a high-level formalism, namely pattern matching, including associative matching. *Tom* is therefore a natural choice each time a symbolic transformation has to be implemented in C or Java for instance. *Tom* has been successfully used to implement the Rodin simplifier, for the B formal method.
- Prototyping: by providing abstract data types, private types, pattern matching, rules and strategies, *Tom* allows the development of quite complex prototypes in a short time. When using Java as the host-language, the full runtime library can be used. Combined with the constructs provided by *Tom*, such as strategies, this procures a tremendous advantage.

One of the most successful transfer is certainly the use of *Tom* made by Business Objects/SAP. Indeed, after benchmarking several other rule based languages, they decided to choose *Tom* to implement a part of their software. *Tom* is used in Paris, Toulouse and Vancouver. The standard representation provided by *Tom* is used as an exchange format by the teams of these sites.

# 5. Software

## 5.1. ATerm

**Participant:** Pierre-Etienne Moreau [correspondant].

ATerm (short for Annotated Term) is an abstract data type designed for the exchange of tree-like data structures between distributed applications.

The ATerm library forms a comprehensive procedural interface which enables creation and manipulation of ATerms in C and Java. The ATerm implementation is based on maximal subterm sharing and automatic garbage collection.

A binary exchange format for the concise representation of ATerms (sharing preserved) allows the fast exchange of ATerms between applications. In a typical application—parse trees which contain considerable redundant information—less than 2 bytes are needed to represent a node in memory, and less than 2 bits are needed to represent it in binary format. The implementation of ATerms scales up to the manipulation of ATerms in the giga-byte range.

The ATerm library provides a comprehensive interface in C and Java to handle the annotated term data-type in an efficient manner.

We are involved (with the CWI) in the implementation of the Java version, as well as in the garbage collector of the C version. The Java version of the ATerm library is used in particular by *Tom*.

The ATerm library is documented, maintained, and available at the following address: http://www.meta-environment.org/Meta-Environment/ATerms.

## 5.2. Tom

**Participants:** Jean-Christophe Bach, Christophe Calvès, Horatiu Cirstea, Pierre-Etienne Moreau [correspondant], Claudia Tavares.

Since 2002, we have developed a new system called *Tom* [33], presented in [17], [18]. This system consists of a pattern matching compiler which is particularly well-suited for programming various transformations on trees/terms and XML documents. Its design follows our experiments on the efficient compilation of rule-based systems [30]. The main originality of this system is to be language and data-structure independent. This means that the *Tom* technology can be used in a C, C++ or Java environment. The tool can be seen as a Yacc-like compiler translating patterns into executable pattern matching automata. Similarly to Yacc, when a match is found, the corresponding semantic action (a sequence of instructions written in the chosen underlying language) is triggered and executed. *Tom* supports sophisticated matching theories such as associative matching with neutral element (also known as list-matching). This kind of matching theory is particularly well-suited to perform list or XML based transformations for example.

In addition to the notion of *rule*, *Tom* offers a sophisticated way of controlling their application: a strategy language. Based on a clear semantics, this language allows to define classical traversal strategies such as *innermost*, *outermost*, *etc.*. Moreover, *Tom* provides an extension of pattern matching, called *anti-pattern matching*. This corresponds to a natural way to specify *complements* (*i.e.*what should not be there to fire a rule). *Tom* also supports the definition of cyclic graph data-structures, as well as matching algorithms and rewriting rules for term-graphs.

*Tom* is documented, maintained, and available at http://tom.loria.fr as well as at http://gforge.inria.fr/projects/tom.

# 6. New Results

## 6.1. Model transformation

**Participants:** Jean-Christophe Bach, Pierre-Etienne Moreau.

In [10], we have proposed a general method to transform high level models by using *Tom* strategies. High-level models we consider are *EMF-ECore* models that we represent by terms whose mappings have been generated by the *Tom-EMF* tool. The proposed method consists in decomposing a complex transformation into many elementary transformations (*definitions*) encoded by *Tom* strategies. These *definitions* are applied on a source model without any consideration of execution order. Therefore, we proposed a mechanism to address the problem of dependency between elementary transformations without introducing any scheduling between rewriting rules. This mechanism relies on the use of temporary elements which play the roles of the target elements until the last part of the transformation : the *Resolve* phase. The goal of this phase is to find and replace all temporary elements by real target ones, and therefore to reconnect all partial target models obtained during elementary transformations to build the resulting model.

In [11], [15], we presented a first proposal of a high-level transformation language included in *Tom* which implements the aforementioned general method. We used this language to implement an avionic case study — AADL2Fiacre — which was proposed by Airbus for the *quarteFt* project.

## 6.2. Improvements of theoretical foundations

### 6.2.1. Termination under strategies

**Participants:** Horatiu Cirstea, Pierre-Etienne Moreau.

Several approaches for proving the confluence and the termination of term rewriting systems have been proposed [16] and the corresponding techniques have been implemented in tools like Aprove [23] and TTT2 [32]. On the other hand, there are relatively few works on the study of these properties in the context of strategic rewriting and the corresponding results were generally obtained for some specific strategies and not within a generic framework. It would thus be interesting to reformulate these notions in the general formalism we have previously proposed [21] and to establish confluence and termination conditions similar to the ones used in standard rewriting.

We have first focused on the termination property and we targeted the rewriting strategies of the *Tom* language. We propose a direct approach which consists in translating *Tom* strategies into a rewriting system which is not guided by a given evaluation strategy and we show that our systematic transformation preserves the termination. This allowed us to take advantage of the termination proof techniques available for standard rewriting and in particular to use existing termination tools (such as Aprove and TTT2) to prove the termination of strategic rewriting systems. The efficiency and scalability of these latter tool has a direct impact on the performances of our approach especially for complex strategies for which an important number of rewrite rules could be generated. We have nevertheless proposed a meta-level implementation of the automatic transformation which improves significantly the performances of the approach.

### 6.2.2. Automatizing the certification of induction proofs

**Participant:** Sorin Stratulat.

Largely adopted by proof assistants, the conventional induction methods based on explicit induction schemas are non-reductive and local, at schema level. On the other hand, the implicit induction methods used by automated theorem provers allow for lazy and mutual induction reasoning. In collaboration with Amira Henaien [13], we devised a new tactic for the Coq proof assistant able to perform automatically implicit induction reasoning. By using an automatic black-box approach, conjectures intended to be manually proved by the certifying proof environment that integrates Coq are proved instead by the `Spike` implicit induction theorem prover. The resulting proofs are translated afterwards into certified Coq scripts.

As a case study, conjectures involved in the validation of a non-trivial application [35] have been successfully and directly certified by Coq using the `Spike` tactic. The proofs of more than 60% of them have been performed completely automatically, i.e., the Coq user does not need to provide any argument to the tactic. On the other hand, its application is limited to Coq specifications transformable into conditional specifications whose axioms can be oriented into rewrite rules.

### 6.2.3. Cyclic proofs by induction methods

**Participant:** Sorin Stratulat.

In a first-order setting, two different 'proof by induction' methods are distinguished: the conventional induction, based on explicit induction schemas, and the implicit induction, based on reductive procedures. In [14], we proposed a new cycle-based induction method that keeps their best features, i.e., performs local and non-reductive reasoning, and naturally fits for mutual and lazy induction. The heart of the method is a proof strategy that identifies in the proof script the subset of formulas contributing to validate the application of induction hypotheses. The conventional and implicit induction are particular cases of our method.

## 6.3. Integration of formal methods in programming languages

### 6.3.1. Multi-focus strategies

**Participants:** Jean-Christophe Bach, Christophe Calvès, Horatiu Cirstea, Pierre-Etienne Moreau.

Like most rewriting engines, *Tom* patterns combined with traversal strategies, gives the possibility to match and rewrite at any position in a given term. We have extended this classical approach with multi-focus strategies which enable us to match and rewrite several positions simultaneously. More precisely, the action performed at a given position can depend on the other positions involved in the corresponding strategy. This extension is particularly well-suited for programming-language semantics specification, semantics which usually require gathering several subterms (code, memory, input/output channels, ...) to perform one action.

The multi-focus library is a conservative extension of *Tom* standard strategies and provides combinators to handle multi-position traversal, matching and rewriting. Compared to the original *Tom* strategy library, the multi-focus version provides global backtracking. The library is available at http://gforge.inria.fr/projects/tom.

### *6.3.2. Formal islands grammars parsing*

**Participants:** Jean-Christophe Bach, Pierre-Etienne Moreau.

Extending a language by embedding within it another language presents significant parsing challenges, especially if the embedding is recursive. The composite grammar is likely to be nondeterministic as a result of tokens that are valid in both the host and the embedded language. In [9], we examined the challenges of embedding the *Tom* language into a variety of general-purpose high level languages. The current parser of *Tom* is complex and difficult to maintain. In this paper, we described how *Tom* can be parsed using island grammars implemented with the Generalised LL (*GLL*) parsing algorithm. The grammar is, as might be expected, ambiguous. Extracting the correct derivation relies on a disambiguation strategy which is based on pattern matching within the parse forest. We described different classes of ambiguity and proposed patterns to solve them.

## 6.4. Security policies specification and analysis

**Participants:** Horatiu Cirstea, Hélène Kirchner, Pierre-Etienne Moreau.

Access control policies, a particular case of security policies should guarantee that information can be accessed only by authorized users and thus prevent all information leakage. We proposed [12] a framework where the security policies and the systems they are applied on are specified separately but using a common formalism. This separation allows not only some analysis of the policy independently of the target system but also the application of a given policy on different systems. In this framework, we propose a method to check properties like confidentiality, integrity or confinement over secure systems based on different policy specifications.

# 7. Partnerships and Cooperations

## 7.1. National Initiatives

We participate in the “Logic and Complexity” part of the GDR–IM (CNRS Research Group on Mathematical Computer Science), in the projects “Logic, Algebra and Computation” (mixing algebraic and logical systems) and “Geometry of Computation” (using geometrical and topological methods in computer science).

### *7.1.1. FRAE QUARTEFT (2009-2012)*

**Participants:** Jean-Christophe Bach, Horatiu Cirstea, Pierre-Etienne Moreau.

“QUARTEFT: QUAlifiable Real TimE Fiacre Transformations” is a research project funded by the FRAE (Fondation de Recherche pour l’Aéronautique et l’Espace). A first goal is to develop an extension of the Fiacre intermediate language to support real-time constructs. A second goal is to develop new model transformation techniques to translate this extended language, Fiacre-RT, into core Fiacre. One of the main difficulties consists in proposing transformation techniques that could be verified in a formal way. A more detailed presentation is available at http://quarteft.loria.fr/dokuwiki/.

## 7.2. International Research Visitors

### *7.2.1. Visits of International Scientists*

Cooperation with Prof. Mark van den Brand from Technical University of Eindhoven.

# 8. Dissemination

## 8.1. Scientific Animation

- Jean-Christophe Bach:
  - Reviewer for SLE 2012 (5th International Conference on Software Language Engineering)
- Horatiu Cirstea:
  - PC member of RuleML 2012 (International RuleML Symposium on Rule Interchange and Applications).
  - Steering committee of RULE.
  - Responsible for the Master speciality “Logiciels: Théorie, méthodes et ingénierie”.
- Claude Kirchner:
  - Keynote speaker of RTA 2012 (23rd International Conference on Rewriting Techniques and Applications): “Rho-Calculi for Computation and Logic”.
- Sergueï Lenglet:
  - Presentation at “Journées communes LTP - LAC - LaMHA”
- Pierre-Etienne Moreau:
  - Member of the board of the Doctoral School in Computer Science and Mathematics.
  - Member of the GDR–GPL (CNRS Research Group on Software Engineering) board.
  - Head of the local committee for Inria “détachements” and “délégations”.
  - Head of the Computer Science department at Ecole des Mines de Nancy.
  - PC member of RTA 2012 (23rd International Conference on Rewriting Techniques and Applications), SLE 2012 (5th International Conference on Software Language Engineering), and WRLA 2012 (9th International Workshop on Rewriting Logic and its Applications)
- Sorin Stratulat:
  - PC Member of SYNASC’12 (14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing), IAS’12 (8th International Conference on Information Assurance and Security), and CISIS’12 (5th International Conference on Computational Intelligence in Security for Information Systems).
  - Invited Speaker at PAS’12 (International Seminar on Program Verification, Automated Debugging and Symbolic Computation).
  - Member of the LITA Laboratory Council.

## 8.2. Teaching - Supervision - Juries

### *8.2.1. Teaching*

Licence : Horatiu Cirstea, Structures de données, 20h, L3, Université de Lorraine, Nancy

Licence : Sergueï Lenglet, Algorithmique et strutures de données, 110h, L1, L2, IUT Charlemagne, Nancy

Licence : Sergueï Lenglet, Bases de données, 60h, L1, IUT Charlemagne, Nancy

Licence : Sergueï Lenglet, Analyse et conception de systèmes d'information, 20h, L2, IUT Charlemagne, Nancy

Licence : Pierre-Etienne Moreau, Tronc Commun d'Informatique, 35h, L3, École des Mines de Nancy

Licence : Christophe Calvès, Certificat Informatique et Internet, 40h, L1, Université de Lorraine, Nancy

Master : Horatiu Cirstea, Analyse et conception de logiciels, 100h, M1, Université de Lorraine, Nancy

Master : Horatiu Cirstea, Génie logiciel avancé, 30h, M2, Université de Lorraine, Nancy

Master : Pierre-Etienne Moreau, Software Engineering, 15h, M1, École des Mines de Nancy

Master : Pierre-Etienne Moreau, Bootcamp OO Programming, 45, M1, École des Mines de Nancy

Master : Pierre-Etienne Moreau, Research project, 30h, M2, École des Mines de Nancy

### *8.2.2. Supervision*

PhD : Cláudia TAVARES, "Un système de types pour la programmation par réécriture embarquée", Université de Lorraine, March 2nd 2012, Claude Kirchner et Pierre-Etienne Moreau

PhD : Vincent DEMANGE, "Vers un calcul des constructions pédagogiques", Université de Lorraine, December 7th 2012, Sorin Stratulat et Loïc Colson

PhD in progress : Jean-Christophe BACH, "Transformation de modèles et certification", November 1st 2010, Pierre-Etienne Moreau

### *8.2.3. Juries*

Pierre-Etienne Moreau:

PhD committee of Aurélien Monot, "Vérification de contraintes temporelles de bout-en-bout dans le contexte AutoSar", Nancy 2012

PhD committee of Luc Engelen, "From Napkin Sketches to Reliable Software", Eindhoven 2012

PhD reviewer of Pengfei Liu, "Intégration de politiques de sécurité dans des systèmes ubiquitaires", Bordeaux 2013

PhD committee of Laurent Wouters, "Multi-Domain Expert-User Modeling Infrastructure", Paris 2013

## 8.3. Popularization

Jean-Christophe Bach participated to scientific mediation by proposing several activities to demonstrate the *algorithmic thinking* at the core of the Computer Science without requiring any computer or even electric devices. These activities are the first part of the CSIRL (Computer Science In Real Life) project which aims to popularize computer science and to initiate children, school students and non-scientists into this domain. These activities were presented during the high school students welcome at LORIA and Inria - Nancy Grand Est, and also during APMEP [1] days.

Jean-Christophe Bach was also involved in popularization activities with Interstices [2] by writing short debunking articles ("Idées reçues") for non computer scientists about Church's thesis and Turing's work.

[1] http://www.apmep.asso.fr/
[2] http://interstices.info

# 9. Bibliography

## Major publications by the team in recent years

[1] E. BALLAND, C. KIRCHNER, P.-E. MOREAU. *Formal Islands*, in "11th International Conference on Algebraic Methodology and Software Technology", Kuressaare, Estonia, M. JOHNSON, V. VENE (editors), LNCS, Springer-Verlag, jul 2006, vol. 4019, p. 51–65, http://www.loria.fr/~moreau/Papers/BallandKM-AMAST2006.pdf.

[2] G. BARTHE, H. CIRSTEA, C. KIRCHNER, L. LIQUORI. *Pure Patterns Type Systems*, in " Principles of Programming Languages - POPL2003, New Orleans, USA", ACM, Jan 2003, p. 250–261.

[3] P. BRAUNER, C. HOUTMANN, C. KIRCHNER. *Principles of Superdeduction*, in "Twenty-Second Annual IEEE Symposium on Logic in Computer Science - LiCS 2007", Wroclaw Pologne, IEEE Computer Society, 2007, http://dx.doi.org/10.1109/LICS.2007.37.

[4] H. CIRSTEA, C. KIRCHNER. *The rewriting calculus - Part I and II*, in "Logic Journal of the Interest Group in Pure and Applied Logics", May 2001, vol. 9, n$^{o}$ 3, p. 427-498.

[5] H. CIRSTEA, C. KIRCHNER, R. KOPETZ, P.-E. MOREAU. *Anti-patterns for Rule-based Languages*, in "Journal of Symbolic Computation", February 2010, vol. 54, n$^{o}$ 5, p. 523-550.

[6] C. KIRCHNER, R. KOPETZ, P.-E. MOREAU. *Anti-Pattern Matching*, in "16th European Symposium on Programming", Braga, Portugal, Lecture Notes in Computer Science, Springer, 2007, vol. 4421, p. 110–124, http://www.loria.fr/~moreau/Papers/KirchnerKM-2007.pdf.

[7] P.-E. MOREAU, C. RINGEISSEN, M. VITTEK. *A Pattern Matching Compiler for Multiple Target Languages*, in "12th Conference on Compiler Construction, Warsaw (Poland)", G. HEDIN (editor), LNCS, Springer-Verlag, may 2003, vol. 2622, p. 61–76, http://www.loria.fr/~moreau/Papers/MoreauRV-CC2003.ps.gz.

## Publications of the year

### Doctoral Dissertations and Habilitation Theses

[8] C. TAVARES. *Un système de types pour la programmation par réécriture embarquée*, Université de Lorraine, March 2012, http://hal.inria.fr/tel-00702301.

### International Conferences with Proceedings

[9] A. AFROOZEH, J.-C. BACH, M. VAN DEN BRAND, A. JOHNSTONE, M. MANDERS, P.-E. MOREAU, E. SCOTT. *Island Grammar-based Parsing using GLL and Tom*, in "5th International Conference on Software Language Engineering - SLE 2012", Dresden, Germany, June 2012, http://hal.inria.fr/hal-00722878.

[10] J.-C. BACH, X. CRÉGUT, P.-E. MOREAU, M. PANTEL. *Model Transformations with Tom*, in "LDTA - 12th Workshop on Language Descriptions, Tools and Applications - 2012", Tallinn, Estonia, ACM, 2012, http://hal.inria.fr/hal-00646350.

[11] J.-C. BACH, P.-E. MOREAU, M. PANTEL. *Tom-based tools to transform EMF models in avionics context*, in "ITSLE - Industrial Track of Software Language Engineering 2012", Dresden, Germany, September 2012, http://hal.inria.fr/hal-00730738.

[12] T. BOURDIER, H. CIRSTEA, M. JAUME, H. KIRCHNER. *Formal Specification and Validation of Security Policies*, in "Foundations & Practice of Security", Paris, France, 2012, http://hal.inria.fr/inria-00507300.

[13] A. HENAIEN, S. STRATULAT. *Performing Implicit Induction Reasoning with Certifying Proof Environments*, in "SCSS'2012 - 4th International Symposium on Symbolic Computation in Software Science", Gammarth, Tunisie, December 2012, http://hal.inria.fr/hal-00764909.

[14] *Best Paper*
S. STRATULAT. *A Unified View of Induction Reasoning for First-Order Logic*, in "Turing-100, The Alan Turing Centenary Conference", Manchester, Royaume-Uni, June 2012, http://hal.inria.fr/hal-00763236.

### Other Publications

[15] J.-C. BACH, P.-E. MOREAU, M. PANTEL. *EMF Models Transformations with Tom*, in "5th International Conference on Software Language Engineering - SLE 2012", Dresden, Germany, 2012, This poster gives an overview of work on models transformations by using Tom language. It was presented at SLE 2012, http://hal.inria.fr/hal-00765091.

## References in notes

[16] F. BAADER, T. NIPKOW. *Term Rewriting and All That.*, Cambridge University Press, 1998.

[17] J.-C. BACH, E. BALLAND, P. BRAUNER, R. KOPETZ, P.-E. MOREAU, A. REILLES. *Tom Manual*, LORIA, 2009, 155, http://hal.inria.fr/inria-00121885/en/.

[18] E. BALLAND, P. BRAUNER, R. KOPETZ, P.-E. MOREAU, A. REILLES. *Tom: Piggybacking rewriting on java*, in "18th International Conference on Rewriting Techniques and Applications - (RTA)", Paris, France, Lecture Notes in Computer Science, Jun 2007, vol. 4533, p. 36–47.

[19] P. BOROVANSKÝ, C. KIRCHNER, H. KIRCHNER. *Controlling Rewriting by Rewriting*, in "Proceedings of the first international workshop on rewriting logic - (WRLA)", Asilomar (California), J. MESEGUER (editor), Electronic Notes in Theoretical Computer Science, Sep 1996, vol. 4.

[20] P. BOROVANSKÝ, C. KIRCHNER, H. KIRCHNER, P.-E. MOREAU. *ELAN from a rewriting logic point of view*, in "Theoretical Computer Science", Jul 2002, vol. 2, n$^{o}$ 285, p. 155–185.

[21] T. BOURDIER, H. CIRSTEA, D. DOUGHERTY, H. KIRCHNER. *Extensional and Intensional Strategies*, in "Electronic Proceedings in Theoretical Computer Science", 2010, vol. 15, p. 1–19.

[22] H. CIRSTEA. *Le calcul de réécriture*, Université Nancy II, October 2010, Habilitation à Diriger des Recherches, http://hal.inria.fr/tel-00546917/en.

[23] C. FUHS, J. GIESL, M. PARTING, P. SCHNEIDER-KAMP, S. SWIDERSKI. *Proving Termination by Dependency Pairs and Inductive Theorem Proving*, in "J. Autom. Reasoning", 2011, vol. 47, n$^{o}$ 2, p. 133–160.

[24] J.-Y. GIRARD, Y. LAFONT, P. TAYLOR. *Proofs and Types*, Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 1989, vol. 7.

[25] C. B. JAY, D. KESNER. *First-class patterns*, in "Journal of Functional Programming", 2009, vol. 19, n$^{o}$ 2, p. 191–225.

[26] J.-P. JOUANNAUD, H. KIRCHNER. *Completion of a set of rules modulo a set of Equations*, in "SIAM J. of Computing", 1986, vol. 15, n$^{o}$ 4, p. 1155–1194.

[27] J.-P. JOUANNAUD, C. KIRCHNER. *Solving equations in abstract algebras: a rule-based survey of unification*, in "Computational Logic. Essays in honor of Alan Robinson", Cambridge (MA, USA), J.-L. LASSEZ, G. PLOTKIN (editors), The MIT press, Cambridge (MA, USA), 1991, chap. 8, p. 257–321.

[28] G. KAHN. *Natural Semantics*, Inria Sophia-Antipolis, feb 1987, n$^{o}$ 601.

[29] C. KIRCHNER, H. KIRCHNER, M. VITTEK. *Designing Constraint Logic Programming Languages using Computational Systems*, in "Proc. 2nd CCL Workshop, La Escala (Spain)", F. OREJAS (editor), Sep 1993.

[30] H. KIRCHNER, P.-E. MOREAU. *Promoting Rewriting to a Programming Language: A Compiler for Non-Deterministic Rewrite Programs in Associative-Commutative Theories*, in "Journal of Functional Programming", 2001, vol. 11, n$^{o}$ 2, p. 207–251, http://www.loria.fr/~moreau/Papers/jfp.ps.gz.

[31] J. W. KLOP, V. VAN OOSTROM, R. DE VRIJER. *Lambda calculus with patterns*, in "Theor. Comput. Sci.", 2008, vol. 398, n$^{o}$ 1-3, p. 16–31.

[32] M. KORP, C. STERNAGEL, H. ZANKL, A. MIDDELDORP. *Tyrolean Termination Tool 2*, in "RTA", 2009, p. 295–304.

[33] P.-E. MOREAU, C. RINGEISSEN, M. VITTEK. *A Pattern Matching Compiler for Multiple Target Languages*, in "12th Conference on Compiler Construction - (CC)", G. HEDIN (editor), Lecture Notes in Computer Science, Springer-Verlag, MAY 2003, vol. 2622, p. 61–76.

[34] S. PEYTON-JONES. *The implementation of functional programming languages*, Prentice-Hall, 1987.

[35] M. RUSINOWITCH, S. STRATULAT, F. KLAY. *Mechanical Verification of an Ideal Incremental ABR Conformance Algorithm*, in "J. Autom. Reasoning", 2003, vol. 30, n$^{o}$ 2, p. 53–177.

[36] M. VAN DEN BRAND, A. VAN DEURSEN, P. KLINT, S. KLUSENER, E. A. VAN DER MEULEN. *Industrial Applications of ASF+SDF*, in "AMAST '96", M. WIRSING, M. NIVAT (editors), Lecture Notes in Computer Science, Springer-Verlag, 1996, vol. 1101, p. 9–18.