IN PARTNERSHIP WITH:
CNRS

Université de Lorraine

Activity Report 2014

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
Architecture, Languages and Compilation

Table of contents

Project-Team PAREO

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

Creation of the Team: 2008 January 01, *updated into Project-Team:* 2011 January 01, end of the Project-Team: 2014 December 31.

1. Members

Faculty Members
Pierre-Etienne Moreau [Team leader, Univ. Lorraine, Professor, HdR]
Christophe Calvès [Univ. Lorraine, until Aug 2014]
Horatiu Cirstea [Univ. Lorraine, Professor, HdR]
Sergueï Lenglet [Univ. Lorraine, Associate Professor]
Sorin Stratulat [Univ. Lorraine, Associate Professor]

PhD Students
Jean-Christophe Bach [Univ. Lorraine, until Sep 2014]
Duc Nguyen Duy [Univ. Tech. Belfort-Montbéliard]
Amira Henaien [Univ. Lorraine]

Administrative Assistants
Laurence Benini [Inria]
Delphine Hubert [Univ. Lorraine]
Martine Kuhlmann [CNRS]

Others
Nauval Atmaja [Univ. Lorraine, from Mar 2014 until Jul 2014]
Pierre Lermusiaux [Inria, from Jun 2014 until Sep 2014]

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, algorithmics, 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.

3. Research Program

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 and the transition rules describing the behaviors of tree automata. Rewriting is used in semantics in order to describe the meaning of programming languages [22] as well as in program transformations like, for example, re-engineering of Cobol programs [31]. 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 [18], a theorem prover [20] or a constraint solver [21]. 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 [23], [14], [12]. We have introduced and studied the notion of *strategy* [13], 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 ρ-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 λ-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 λ-calculus. To palliate this problem, pattern-calculi [28], [25], [19] 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 [16]. 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* project-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. New Software and Platforms

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.

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].

Since 2002, we have developed a new system called *Tom* [27], presented in [11], [12]. 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 [24]. 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. Static Analysis

Participant: Sergueï Lenglet.

6.1.1. Static Analysis for Control Operators

Control operators, such as *call/cc* in Scheme or SML, allow programs to have access and manipulate their execution context. We study the behavioral theory of the $\lambda\mu$-calculus, an extension of the λ-calculus with a control feature similar to *call/cc*. In [6], we define an applicative bisimilarity for the $\lambda\mu$-calculus, demonstrating the differences in the definitions between call-by-name and call-by-value. We give equivalence examples to illustrate how our relations can be used; in particular, we prove David and Py's counter-example, which cannot be proved with the preexisting bisimilarities for the $\lambda\mu$-calculus. The proofs are in the accompanying research report [8], where we also define environmental bisimulations for the calculus.

6.1.2. Polymorphism and Higher-order Functions for XML

In [7], we define a calculus with higher-order polymorphic functions, recursive types with arrow and product type constructors and set-theoretic type connectives (union, intersection, and negation). We study the explicitly-typed version of the calculus in which type instantiation is driven by explicit instantiation annotations. In particular, we define an explicitly-typed λ-calculus with intersection types and an efficient evaluation model for it. The work presented in this article provides the theoretical foundations needed to design and implement higher-order polymorphic functional languages for semi-structured data.

6.2. Termination under Strategies

Participants: Horatiu Cirstea, Sergueï Lenglet, Pierre-Etienne Moreau.

Several approaches for proving the confluence and the termination of term rewriting systems have been proposed [10] and the corresponding techniques have been implemented in tools like Aprove [17] and TTT2 [26]. 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 [15] 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 tools 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. The corresponding tool is available at http://gforge.inria.fr/projects/tom.

6.3. Property-based Testing

Participants: Nauval Atmaja, Horatiu Cirstea, Pierre-Etienne Moreau.

Quality is crucial for software systems and several aspects should be taken into account. Formal verification techniques like model checking and automated theorem proving can be used to guarantee the correctness of finite or infinite systems. While these approaches provide a high level of confidence they are sometimes difficult and expensive to apply. Software testing is another approach and although it cannot guarantee correctness it can be very efficient in finding errors.

We have proposed a property based testing framework for the *Tom* language inspired from the ones proposed in the context of functional programming. The previously developed tool has been improved by integrating it in the *Junit* framework. The tests are thus highly automatized and the library can be smoothly integrated in classical IDEs. The relatively simple shrinking method which searches a smaller counter-example starting from an initial relatively complex one has been also improved. The library is available at http://gforge.inria.fr/projects/tom.

6.4. Inductive Reasoning

Participant: Sorin Stratulat.

6.4.1. Decision Procedures to Prove Inductive Theorems Without Induction

Automated inductive reasoning for term rewriting has been extensively studied in the literature. Classes of equations and term rewriting systems (TRSs) with decidable inductive validity have been identified and used to automatize the inductive reasoning. In [9], we give procedures for deciding the inductive validity of equations in some standard TRSs on natural numbers and lists. Contrary to previous decidability results, our procedures can automatically decide without involving induction reasoning the inductive validity of arbitrary equations for these TRSs, that is, without imposing any syntactical restrictions on the form of equations. We also report on the complexity of our decision procedures. These decision procedures are implemented in our automated provers for inductive theorems of TRSs and experiments are reported.

6.4.2. Implementing Reasoning Modules in Implicit Induction Theorem Provers

In [30], we detail the integration in SPIKE, an implicit induction theorem prover, of two reasoning modules operating over naturals combined with interpreted symbols. The first integration schema is à la Boyer-Moore, based on the combination of a congruence closure procedure with a decision procedure for linear arithmetic over rationals/reals. The second follows a ‘black-box’ approach and is based on external SMT solvers. It is shown that the two extensions significantly increase the power of SPIKE; their performances are compared when proving a non-trivial application.

6.4.3. Building Explicit Induction Schemas for Cyclic Induction Reasoning

In the setting of classical first-order logic with inductive predicates, two kinds of sequent-based induction reasoning are distinguished: cyclic and structural. Proving their equivalence is of great theoretical and practical interest for the automated reasoning community. Previously, it has been shown how to transform any structural proof developed with the LKID system into a cyclic proof using the CLKID$^\omega$ system. However, the inverse transformation was only conjectured. In [29], we provide a simple procedure that performs the inverse transformation for an extension of LKID with explicit induction rules issued from the structural analysis of CLKID$^\omega$ proofs, then establish the equivalence of the two systems. This result is further refined for an extension of LKID with Noetherian induction rules. We show that Noetherian induction subsumes the two kinds of reasoning. This opens the perspective for building new effective induction proof methods and validation techniques supported by (higher-order) certification systems integrating the Noetherian induction principle, like Coq.

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).

We are also involved in the GDR-GPL (CNRS Research Network on Software Engineering), as a member of the FORWAL group and member of the Scientific Board of the GDR.

7.2. European Initiatives

7.2.1. Collaborations in European Programs, except FP7 & H2020

Program: PHC Polonium

Project title: Expressing concurrency through control operators

Duration: January 2015 - December 2016

Coordinator: Sergueï Lenglet

Other partner: Institute of Computer Science, University of Wrocław, Poland

Abstract: The goal of this project is to explore the interplay between concurrency, continuations, and control operators at a fundamental level. We do not restrict ourselves to a specific programming language, but we use more general and well established formal models, namely process calculi (such as the π-calculus) for concurrency, and the λ-calculus (a model of sequential functional programming) for continuations and control operators. We want to find new connections between concurrency and control operators, and especially new ways of implementing concurrent and distributed programs with the help of control operators.

7.3. International Research Visitors

7.3.1. Visits of International Scientists

7.3.1.1. Internships

Nauval Atmaja

Subject: Property Based Testing

Date: from Feb 2014 until Jun 2014

Institution: Erasmus Mundus MSc in Dependable Software Systems

8. Dissemination

8.1. Promoting Scientific Activities

8.1.1. Scientific events selection

8.1.1.1. Conference program committee membership

Horatiu Cirstea:

- PC member of RuleML 2014 (International RuleML Symposium on Rule Interchange and Applications).
- Steering committee of RULE.

Sergueï Lenglet:

- PC member of WPTE'14 (Workshop on Rewriting Techniques for Program Transformations and Evaluation)

Pierre-Etienne Moreau:

- PC member of TERMGRAPH'2014 (8th International Workshop on Computing with Terms and Graphs)
- PC member of WRLA'2014 (10th International Workshop on Rewriting Logic and its Applications)

Sorin Stratulat:

- PC member of CISIS 2014 (7th International Conference on Computational Intelligence in Security for Information Systems)
- PC member of IAS 2014 (10th International Conference on Information Assurance and Security)
- PC member of SYNASC 2014 (16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing)

8.1.1.2. Reviewing activities

Horatiu Cirstea:

- Reviewer for ESOP'2014, ICTAC'2014, LOPSTR'2014

Sergueï Lenglet:

- Reviewer for CSL-LICS'2014, RTA-TLCA'2014, FOSSACS'2015

Pierre-Etienne Moreau:

- Reviewer for TERMGRAPH'2014, WASDETT'2014

8.1.2. Journal

8.1.2.1. Editorial board membership

Pierre-Etienne Moreau:

- Advisory Board of Science of Computer Programming, software track

8.1.2.2. Reviewing activities

Horatiu Cirstea:

- Reviewer for Journal of Symbolic Computation

Sergueï Lenglet:

- Reviewer for Fundamenta Informaticae and HOSC (Higher-Order and Symbolic Computation)

Pierre-Etienne Moreau:

- Reviewer for Electronic Proceedings in Theoretical Computer Science (EPTCS)

Sorin Stratulat:

- Reviewer for Journal of Symbolic Computation

8.2. Teaching - Supervision - Juries

8.2.1. Teaching

Licence: Sergueï Lenglet, teaching at IUT Charlemagne, first year and second year students.

Licence: Horatiu Cirstea, Responsible for the first year of the Licence in Computer Science, Université de Lorraine.

Licence: Pierre-Etienne Moreau, Responsible of the course "Introduction to Algorithms and Programming" (http://www.depinfonancy.net/s5/tcs13), first year at Mines-Nancy (150 students), Université de Lorraine.

Master: Horatiu Cirstea, Responsible for the Master speciality "Logiciels: Théorie, méthodes et ingénierie", Université de Lorraine.

Master: Pierre-Etienne Moreau, Head of the Computer Science Department at Mines Nancy, Université de Lorraine.

Doctorate: Pierre-Etienne Moreau, Implementing Term Rewriting, International School on Rewriting, Valparaíso, Chile.

8.2.2. Supervision

PhD: Jean-Christophe Bach, A formal Island for qualifiable model transformations, Université de Lorraine, September 12th 2014, Pierre-Etienne Moreau

PhD in progress: Duy Duc Nguyen, Aided design of multi-physics and multi-scale systems based on asymptotic methods, Horatiu Cirstea (co-supervised with Michel Lenczner and Federic Zamkotsian)

PhD in progress: Amira Henaien, Certification du raisonnement formel porté sur des systèmes d'information critiques, November 2010, Sorin Stratulat (co-supervised with Maurice Margenstern)

8.2.3. Juries

Horatiu Cirstea:

PhD committee of Julien Ferté, reviewer, "Régularité et contraintes de descendance, équations algébriques+", Marseille 2014

PhD committee of Hernan Vanzetto, examiner, "Automatisation des preuves et synthèse des types pour la théorie des ensembles dans le contexte de TLA+", Nancy 2014

PhD committee of Bin Yang, reviewer, "Contribution to a kernel of a symbolic asymptotic modeling software", Besançon 2014

Pierre-Etienne Moreau:

PhD committee of Jean-Christophe Bach, superviser, "Un îlot formel pour les transformations de modèles qualifiables", Nancy 2014

PhD committee of Faiez Zalila, reviewer, "Methods and Tools for the Integration of Formal Verification in Domain-Specific Languages", Toulouse 2014

PhD committee of Cyrille Wiedling, examiner, "Formal Verification of Advanced Families of Security Protocols: E-Voting and APIs", Nancy 2014

Sorin Stratulat:

PhD committee of Abdelkader Kersani, examiner, "Preuves par induction dans le calcul de superposition", Grenoble 2014

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.

Pierre-Etienne Moreau gave two lectures about “Robotics and Programming” in the ISN course (Informatique et Science du Numérique), in order to help professors of “classes de terminale” to teach this discipline. He organized a three day course about “Algorithms, Programming and Databases” in order to help professors of “classes préparatoires aux grandes écoles” to teach this discipline.

Pierre-Etienne Moreau is member of the national committee for Inria “Médiation Scientifique”. He also participated to “Fête de la Science 2014” at Mines Nancy.

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, pp. 51–65, http://www.loria.fr/~moreau/Papers/BallandKM-AMAST2006.pdf

[2] 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

[3] 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, pp. 523-550

[4] 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, pp. 110–124, http://www.loria.fr/~moreau/Papers/KirchnerKM-2007.pdf

Publications of the year

Doctoral Dissertations and Habilitation Theses

[5] J.-C. Bach. *A formal Island for qualifiable model transformations*, Université de Lorraine, September 2014, https://tel.archives-ouvertes.fr/tel-01081055

International Conferences with Proceedings

[6] D. Biernacki, S. Lenglet. *Applicative Bisimilarities for Call-by-Name and Call-by-Value $\lambda\mu$-Calculus*, in "Mathematical Foundations of Programming Semantics Thirtieth Conference", Ithaca, United States, Elsevier, June 2014, vol. 308, pp. 49 - 64 [*DOI :* 10.1016/j.entcs.2014.10.004], https://hal.inria.fr/hal-01080960

[7] G. Castagna, K. Nguyen, Z. Xu, H. Im, S. Lenglet, L. Padovani. *Polymorphic Functions with Set-Theoretic Types. Part 1: Syntax, Semantics, and Evaluation*, in "POPL’14, 41th ACM Symposium on Principles of Programming Languages", San Diego, United States, January 2014, pp. 5-17 [*DOI :* 10.1145/2535838.2535840], https://hal.archives-ouvertes.fr/hal-00907166

Research Reports

[8] D. Biernacki, S. Lenglet. *Sound and Complete Bisimilarities for Call-by-Name and Call-by-Value Lambda-mu Calculus*, January 2014, n^{o} RR-8447, https://hal.inria.fr/hal-00926100

References in notes

[9] T. AOTO, S. STRATULAT. *Decision Procedures for Proving Inductive Theorems without Induction*, in "16th International Symposium on Principles and Practice of Declarative Programming (PPDP) 2014", Canterbury, United Kingdom, September 2014 [*DOI :* 10.1145/2643135.2643156], https://hal.archives-ouvertes.fr/hal-01098929

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

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

[12] 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, pp. 36–47

[13] 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

[14] 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, pp. 155–185

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

[16] H. CIRSTEA. *Le calcul de réécriture*, Université Nancy II, October 2010, Habilitation à Diriger des Recherches

[17] 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, pp. 133–160

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

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

[20] 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, pp. 1155–1194

[21] 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, 1991, chap. 8, pp. 257–321

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

[23] 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

[24] 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, pp. 207–251, http://www.loria.fr/~moreau/Papers/jfp.ps.gz

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

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

[27] 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, pp. 61–76

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

[29] S. STRATULAT. *Building explicit induction schemas for cyclic induction reasoning*, January 2014, https://hal.archives-ouvertes.fr/hal-00956769

[30] S. STRATULAT. *Implementing Reasoning Modules in Implicit Induction Theorem Provers*, in "International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2014)", Timisoara, Romania, September 2014, https://hal.archives-ouvertes.fr/hal-01098933

[31] 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, pp. 9–18