IN PARTNERSHIP WITH:
CNRS

Université de Lorraine

Activity Report 2013

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.

1. Members

Faculty Members

Pierre-Etienne Moreau [Team leader, Univ. Lorraine, Professor, HdR]
Christophe Calvès [Univ. Lorraine, ATER]
Horatiu Cirstea [Univ. Lorraine, Professor, HdR]
Sergueï Lenglet [Univ. Lorraine, Associate Professor]

External Collaborators

Sorin Stratulat [Univ. Lorraine, Associate Professor]
Claude Kirchner [Inria, Senior Researcher, HdR]
Hélène Kirchner [Inria, Senior Researcher, HdR]

PhD Student

Jean-Christophe Bach [Univ. Lorraine, ATER]

Administrative Assistants

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

Others

Cosay Topaktas [Univ. Lorraine, Master Intern, from Feb 2013 until Jun 2013]
Fellype Vedovato Martins [Inria, Master Intern, from Jun 2013 until Sep 2013]

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 [27] as well as in program transformations like, for example, re-engineering of Cobol programs [33]. 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 [23], a theorem prover [25] or a constraint solver [26]. 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 [28], [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 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 [32], [30], [24] 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 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* [31], 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 [29]. 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 allow programs to have access and manipulate their execution context. Abortive control operators, such as *call/cc* in Scheme or SML, capture the entire execution context (also called continuation), while delimited-control operators, such as *shift* and *reset* captures only a part of the continuation (delimited by reset). We want to prove properties (like equivalences between terms or termination) for languages with these operators, using static analysis.

In [9], [16], we study the behavioral theory of a language with delimited control. More precisely, we define environmental bisimilarities for the delimited-control operators shift and reset. We consider two different notions of contextual equivalence: one that does not require the presence of a top-level control delimiter when executing tested terms, and another one, fully compatible with the original CPS semantics of shift and reset, that does. For each of them, we develop sound and complete environmental bisimilarities, and we discuss up-to techniques.

In [8], we present new proofs of termination of evaluation in reduction semantics (i.e., a small-step operational semantics with explicit representation of evaluation contexts) for System F with control operators. We introduce a modified version of Girard's proof method based on reducibility candidates, where the reducibility predicates are defined on values and on evaluation contexts as prescribed by the reduction semantics format. We address both abortive control operators (*callcc*) and delimited-control operators (*shift* and *reset*) for which we introduce novel polymorphic type systems, and we consider both the call-by-value and call-by-name evaluation strategies.

6.1.2. Polymorphism and higher-order functions for XML

In [11], 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. In a companion paper [21], we define a local type inference system that allows the programmer to omit explicit instantiation annotations, and a type reconstruction system that allows the programmer to omit explicit type annotations. The work presented in the two articles provides the theoretical foundations and technical machinery needed to design and implement higher-order polymorphic functional languages for semi-structured data.

6.2. Model Transformations

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

Model Driven Engineering is a technique that has been applied quite successfully for the design of complex systems. Such systems cannot be released and embedded without complying with the certification required by the application domain: EN 50128 for railways, DO-178C for aeronautics, or ISO 26262 for automotive for instance.

Recently we have developed an extension of *Tom* to support the development of Model Transformations and the generation of traces which are needed to give confidence in the quality of the implemented transformation.

In [12], we present a method, a language and dedicated tooling to ease and to speed up software development based on models transformations. Our approach aims to bridge the gap between general purpose languages and domain specific ones in order to take benefit from both of the two worlds, and to increase software quality. Our approach uses the Tom language which is a shallow extension of general purpose languages. Our proposal allows to write modular transformations whose code is reusable, and which are traceable.

6.3. Property based testing

Participants: Horatiu Cirstea, Pierre-Etienne Moreau, Cosay Topaktas.

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 prosed in the context of functional programming. In the current version relatively simple properties can be already expressed and tested on *Tom* programs. It consists of an exhaustive approach testing all possible input values and guaranteeing that the discovered counter-examples are the smallest ones (the size of the inputs is clearly limited by the execution time) and a random approach where inputs of bigger size could be tested but the minimal counter-example is not guaranteed. A relatively simple shrinking method which searches a smaller counter-example starting from an initial relatively complex one has been also proposed. There is ongoing work on the expressiveness of the property language and the efficiency of the shrinking method. The library is available at http://gforge.inria.fr/projects/tom.

6.4. Nominal Theory

Participant: Christophe Calvès.

Nominal unification is proven to be quadratic in time and space. It was so by two different approaches, both inspired by the Paterson-Wegman linear unification algorithm, but dramatically different in the way nominal and first-order constraints are dealt with.

To handle nominal constraints, Levy and Villaret introduced the notion of replacing while Calvès and Fernández use permutations and sets of atoms. To deal with structural constraints, the former use multi-equation in a way similar to the Martelli-Montanari algorithm while the later mimic Paterson-Wegman.

In [10] we abstract over these two approaches and genralize them into the notion of modality, highlighting the general ideas behind nominal unification. We show that replacings and environments are in fact isomorphic. This isomorphism is of prime importance to prove intricate properties on both sides and a step further to the real complexity of nominal unification.

7. Partnerships and Cooperations

7.1. Regional Initiatives

We participate at the LORIA project entitled "Combining deduction engines into SMT".

7.2. 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).

7.3. International Research Visitors

7.3.1. Internships

Anisia Maria Magdalena Tudorescu

Subject: Integrating SMT solvers into Spike

Date: from Mar 2013 until May 2013

Institution: West Timisoara University (Romania)

Cosay Gurkay Topaktas

Subject: Property Based Testing

Date: from Feb 2013 until Jun 2013

Institution: Erasmus Mundus MSc in Dependable Software Systems

Fellype Vedovato Martins

Subject: Generation of Terms

Date: from Jun 2013 until Sept 2013

Institution: Mines-Nancy, 2nd year student

8. Dissemination

8.1. Scientific Animation

Jean-Christophe Bach:

- Member of the LORIA laboratory council
- Member of the organizing committee of the "Journées GDR–GPL" colocated with the AFADL and CIEL conferences

Christophe Calvès

- Member of the organizing committee of the "Journées GDR–GPL" colocated with the AFADL and CIEL conferences

Horatiu Cirstea:

- PC member of RuleML 2013 (International RuleML Symposium on Rule Interchange and Applications).
- PC member of SCSS 2013 (International Symposium on Symbolic Computation in Software Science).
- Steering committee of RULE.
- Responsible for the Master speciality "Logiciels: Théorie, méthodes et ingénierie".
- Member of the organizing committee of the "Journées GDR–GPL" colocated with the AFADL and CIEL conferences

Sergueï Lenglet:

- Member of the organizing committee of the "Journées GDR–GPL" colocated with the AFADL and CIEL conferences
- Invited speaker at the "Journées LAC"
- Reviewer for the TCS (Theoretical Computer Science) journal

Pierre-Etienne Moreau:

- Member of the GDR–GPL (CNRS Research Group on Software Engineering) board.
- Member of the national committee for Inria "Médiation Scientifique".
- Head of the local committee for Inria "détachements" and "délégations".
- Head of the Computer Science department at Ecole des Mines de Nancy.
- President of the organizing committee of the "Journées GDR–GPL 2013" colocated with the AFADL and CIEL conferences
- PC member of SLE 2013 (6th International Conference on Software Language Engineering), SCSS 2013 (5th International Symposium on Symbolic Computation in Software Science),
- Member of the organizing committee of WASDeTT 2013 (4th International Workshop on Academic Software Development Tools and Techniques)

Sorin Stratulat:

- Member of the LITA Laboratory Council.
- Member of the program committee of the 9th International Conference on Information Assurance and Security (IAS '13)
- Member of the program committee of the 6th International Conference on Computational Intelligence in Security for Information Systems (CISIS'13)
- Member of the program committee of the 5th International Symposium of Symbolic Computation in Software Science (SCSS '13)
- Tutorial speaker at the Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2013)
- Speaker at
  - Workshop on Inductive Theorem Proving 23-24 November 2013, Imperial College London, UK
  - LIX Colloquium on the Theory and Application of Formal Proofs, 5-7 November 2013, Ecole Polytechnique, Palaiseau, France
  - Workshop on Proof Search in Axiomatic Theories and Type Theories (PSATTT), 8 November 2013, Ecole Polytechnique, Palaiseau, France

8.2. Teaching - Supervision - Juries

8.2.1. Teaching

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, France

8.2.2. Supervision

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

PhD in progress : Amira HENAIEN, "Certification du raisonnement formel porté sur des systèmes d’information critiques.", November 1st 2010, Sorin Stratulat

8.2.3. Juries

Horatiu Cirstea:

PhD committee of Henri Debrat, “Certification formelle de la correction d’algorithmes de Consensus”, Nancy 2013

Pierre-Etienne Moreau:

PhD committee of Mathieu Giorgino, reviewer, Toulouse, 2013: “Inductive Representation, Proofs and Refinement of Pointer Structures”

PhD committee of Clément Guy, reviewer, Rennes, 2013: “Facilités de typage pour l’ingénierie des langages”

PhD committee of Pengfei Liu, reviewer, Bordeaux, 2013: “Intégration de politiques de sécurité dans des systèmes ubiquitaires”

PhD committee of Laurent Wouters, Paris, 2013: “Multi-Domain Expert-User Modeling Infrastructure”

8.3. Popularization

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

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 also took part to the “Fête de la science” in October.

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 [15]. Other popularization articles are still under work.

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.

Pierre-Etienne Moreau 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.

[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, pp. 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, pp. 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, pp. 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, pp. 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, pp. 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, pp. 61–76, http://www.loria.fr/~moreau/Papers/MoreauRV-CC2003.ps.gz

Publications of the year

International Conferences with Proceedings

[8] M. BIERNACKA, D. BIERNACKI, S. LENGLET, M. MATERZOK. *Proving termination of evaluation for System F with control operators*, in "COS2013 - First Workshop on Control Operators and their Semantics", Eindhoven, Netherlands, U. DE'LIQUORO, A. SAURIN (editors), Electronic Proceedings in Theoretical Computer Science, Open Publishing Association, September 2013, vol. 127, pp. 15-29, In Proceedings COS 2013, arXiv:1309.0924 [*DOI :* 10.4204/EPTCS.127.2], http://hal.inria.fr/hal-00860954

[9] D. BIERNACKI, S. LENGLET. *Environmental Bisimulations for Delimited-Control Operators*, in "APLAS - 11th Asian Symposium on Programming Languages and Systems - 2013", Melbourne, Australia, C. CHIEH SHAN (editor), LNCS, Springer, 2013, vol. 8301, pp. 333-348, http://hal.inria.fr/hal-00903839

[10] C. CALVÈS. *Unifying Nominal Unification*, in "Rewriting Techniques and Applications", Eindhoven, Netherlands, F. VAN RAAMSDONK (editor), LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, June 2013, vol. 21, pp. 143-157 [*DOI :* 10.4230/LIPICS.RTA.2013.143], http://hal.inria.fr/hal-00926833

[11] 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, 2014, to appear [*DOI :* 10.1145/2535838.2535840], http://hal.inria.fr/hal-00907166

Research Reports

[12] J.-C. BACH. , *Une approche hybride GPL-DSL pour transformer des modèles*, January 2013, 26 p. , Première version d'un article soumis à TSI, http://hal.inria.fr/hal-00786254

[13] D. BIERNACKI, S. LENGLET. , *Sound and Complete Bisimilarities for Call-by-Name and Call-by-Value Lambda-mu Calculus*, Inria, January 2014, n° RR-8447, http://hal.inria.fr/hal-00926100

[14] A. ROUSSEAU, A. DARNAUD, B. GOGLIN, C. ACHARIAN, C. LEININGER, C. GODIN, C. HOLIK, C. KIRCHNER, D. RIVES, E. DARQUIE, E. KERRIEN, F. NEYRET, F. MASSEGLIA, F. DUFOUR, G. BERRY, G. DOWEK, H. ROBAK, H. XYPAS, I. ILLINA, I. GNAEDIG, J. JONGWANE, J. EHREL, L. VIENNOT, L. GUION, L. CALDERAN, L. KOVACIC, M. COLLIN, M.-A. ENARD, M.-H. COMTE, M. QUINSON, M. OLIVI, M. GIRAUD, M. DORÉMUS, M. OGOUCHI, M. DROIN, N. LACAUX, N. ROUGIER, N. ROUSSEL, P. GUITTON, P. PETERLONGO, R.-M. CORNUS, S. VANDERMEERSCH, S. MAHEO, S. LEFEBVRE, S. BOLDO, T. VIÉVILLE, V. POIREL, A. CHABREUIL, A. FISCHER, C. FARGE, C. VADEL, I. ASTIC, J.-P. DUMONT, L. FÉJOZ, P. RAMBERT, P. PARADINAS, S. DE QUATREBARBES, S. LAURENT. , *Médiation Scientifique : une facette de nos métiers de la recherche*, March 2013, 34 p. , http://hal.inria.fr/hal-00804915

Scientific Popularization

[15] M. QUINSON, J.-C. BACH. *L'informatique nomade, c'est la liberté !*, in "Interstices", February 2013, http://hal.inria.fr/hal-00794187

Other Publications

[16] D. BIERNACKI, S. LENGLET. , *Environmental Bisimulations for Delimited-Control Operators*, September 2013, Long version of the corresponding APLAS13 paper, http://hal.inria.fr/hal-00862189

References in notes

[17] 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/

[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, pp. 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° 285, pp. 155–185

[21] G. CASTAGNA, K. NGUYEN, Z. XU, P. ABATE. , *Polymorphic Functions with Set-Theoretic Types. Part 2: Local Type Inference and Type Reconstruction*, November 2013, http://hal.archives-ouvertes.fr/hal-00880744

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

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

[25] 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

[26] 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

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

[28] 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

[29] 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

[30] 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

[31] 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

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

[33] 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