INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team pareo

Formal islands: foundations and applications

Nancy - Grand Est

Theme : Programs, Verification and Proofs

Activity Report

2010

Table of contents

1. Team

Research Scientist
Yves Guiraud [CR INRIA, mis à disposition de l'Institut Camille Jordan à Lyon]

Faculty Members
Horatiu Cirstea [MC Nancy2, HdR]
Pierre-Etienne Moreau [Team Leader, PR INPL, HdR]

External Collaborators
Claude Kirchner [DR INRIA, HdR]
Hélène Kirchner [DR CNRS, HdR]
Sorin Stratulat [MC Metz]

Technical Staff
Jean-Christophe Bach [Ingénieur Jeune Diplômé INRIA until October 30]

PhD Students
Jean-Christophe Bach [Contrat doctoral, since November 1]
Tony Bourdier [CORDI]
Clément Houtmann [MESR until August 31. Postdoctoral fellow at INRIA, since September 1]
Francois Prugniel [Contrat doctoral, since October 1]
Cody Roux [CORDI]
Claudia Tavares [Brazil]

Administrative Assistants
Chantal Llorens [until March 31]
Valérie Genre [since April 1]

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.

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 [47] as well as in program transformations like, for example, re-engineering of Cobol programs [57]. 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 [39], a theorem prover [45] or a constraint solver [46]. 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 (*e.g.*, OBJ), functional programming (*e.g.*, ML) and transition systems (*e.g.*, Murphi).

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 notion 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 [49], [33], [30]. We have introduced and studied the notion of *strategy* [32], 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 [56], [51], [44] 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 [10]. 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*.

3.4. Algebraic rewriting

Rewriting theory, in computer science, and combinatorial algebra, in mathematics, are two research fields that share striking similarities: both study the properties of *intentional* descriptions of complex objects, respectively rewriting systems and presentations by generators and relations. This research direction is devoted to develop a theoretical setting that unifies rewriting and combinatorial algebra, in order to transpose methods of one field to the other one.

Rewriting systems and presentations of algebraic structures have a common generalisation as *polygraphs*, which are cellular specifications of higher-dimensional categories [35]. In this setting, we can describe, in a uniform way, different kinds of objects, such as the following ones:

- A rule-based program that computes the list-splitting function used in the merge-sort algorithm:

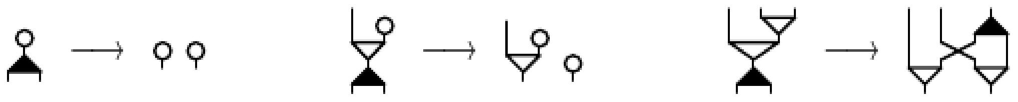

Figure 1.

- The usual presentation by generators and relations of the structure of Hopf algebras, containing, in particular, the following relations:

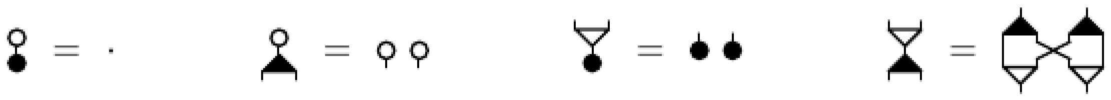

Figure 2.

- The Reidemeister moves for braids, giving a combinatorial description of those topological objects:

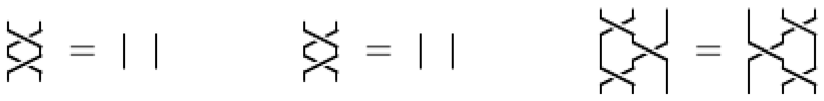

Figure 3.

More precisely, polygraphs are a common algebraic setting for: abstract, word, term and term-graph rewriting systems [35], [40], [6] and, in particular, first-order functional programs [43], [31], [3]; set-theoretic operads, pro(p)s, algebraic theories [40], [6]; Turing machines and Petri nets [40], [42], [31], [3]; formal proofs of propositional calculus and linear logic [41].

In the theoretical setting of polygraphs, the Fox-Squier theory shows that the computational properties of rewriting systems and the mathematical properties of presentations are intimately related to the same topological properties of polygraphs [7]. From this starting point, we progressively establish a correspondence between programs and algebras and we use it to develop applications in different directions:

- Algebraic semantics of programs, such as the new characterisation of evaluation strategies in terms of algebraic resolutions [27]. Here, we want to use well-founded mathematical theories to give a better understanding of programming and, that way, extend the expressiveness of rule-based programming languages.
- Methods from algebraic topology to produce new tools for the static analysis of programs, such as the use of *derivations* to prove termination and bound computational complexity [6], [31], [3]. In this direction, we plan to develop tools from cohomology to classify derivations and, this way, to propose a radically new point of view on computational complexity theory.
- New applications for rewriting, in the field of the formalisation and certification of mathematics, such as the use of rewriting methods to prove coherence theorems or to build resolutions [26], [27].

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, Horatiu Cirstea, Pierre-Etienne Moreau [correspondant], Claudia Tavares.

Since 2002, we have developed a new system called *Tom* [54], presented in [29], [30]. 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 experiences on the efficient compilation of rule-based systems [50]. 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 a *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 algorithm 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.

5.3. Lemuridae

Participant: Clément Houtmann.

Lemuridae is a proof assistant for the sequent calculus instance of superdeduction modulo. It is written in *Tom* and features automatic super-rules derivation with support for axiom directed focussing, automated derivation of induction principles using deduction modulo encoding of higher order logic, as well as some basic automatic tactics. The soundness is ensured by a tiny kernel checking the generated proof trees.

It has been used as a prototyping environment for the developpement of the encoding of Pure Type Systems as well as the simulation of inductive types in superdeduction modulo.

In 2009, the kernel has been entirely rewritten to take advantage of the proofterms developped in [4]. To this end, we have developped *FreshGom*, an extension of the *Tom* language providing mechanisms that ease the manipulation terms containing binders. This allowed us to export *Lemuridae*'s proofs to other formats, including Parigot's $\lambda\mu$-calculus [55] and *Coq* proofterms [38].

Lemuridae is available in the *Tom* subversion repository, under `applications/lemuridae`.

5.4. Cat

Participant: Yves Guiraud [correspondant].

Cat is a library for polygraphic calculus, written in Caml. It has been used, in a joint work with F. Blanqui, to produce an automatic termination prover for first-order functional programs. It translates such a rewriting system into a polygraph and tries to find a derivation proving its termination, using the results of [6], [43]. If possible, it seeks a derivation that proves that the program is polynomial [31], [3]. Cat is also at the basis of Catex.

5.5. Catex

Participant: Yves Guiraud [correspondant].

Catex is a tool for (pdf)Latex, used in the same way as Bibtex, that automatically produces string diagrams from their algebraic expression. It follows the same design as Tom, a Catex file being a Latex file enriched with formal islands corresponding to those algebraic expressions, such as:

```
\deftwocell[red]{delta : 1 -> 2}
\deftwocell[orange]{mu : 2 -> 1}
\deftwocell[crossing]{tau : 2 -> 2}
\twocell{(delta *0 delta) *1 (1 *0 tau *0 1) *1 (mu *0 mu)}
```

Catex dissolves such an island into Latex code, using the PGF/Tikz package. Executed on the result, (pdf)Latex produces the following diagram:

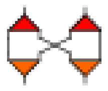

Figure 4.

Catex is distributed through the page: http://www.loria.fr/~guiraudy/catex. We want to extend Catex in two directions. First, to produce diagrams not only for Latex but also for web publications. Then, Catex will be adapted to a tool for the automatic production, in scientific papers, of certified algebraic computations, which are a three-dimensional equivalent of string diagrams.

6. New Results

6.1. Improvement of theoretical foundations

6.1.1. Strategic and constrained rewriting

Participants: Tony Bourdier, Horatiu Cirstea, Hélène Kirchner.

In [48], we introduce the notion of abstract strategies for abstract reduction systems. Adequate properties of termination, confluence and normalization under strategy can then be defined. Thanks to this abstract concept, we draw a parallel between strategies for computation and strategies for deduction. We define deduction rules as rewrite rules, a deduction step as a rewriting step and a proof construction step as a narrowing step in an adequate abstract reduction system. Computation, deduction and proof search are thus captured in the uniform foundational concept of abstract reduction system in which abstract strategies have a clear formalisation.

In the continuation of this work, we proposed in [11] an alternative definition of abstract reduction systems and introduced a general definition of abstract strategies which is extensional in the sense that a strategy is defined explicitly as a set of derivations of an abstract reduction system. We introduce also a more intensional definition supporting the abstract view but more operational in the sense that it describes a means for determining such a set. We characterize the class of extensional strategies that can be defined intensionally. We also give some hints towards a logical characterization of intensional strategies.

We investigated in [24] another generalization of rewriting : constrained rewriting. Constraints over terms are very suitable to define sets of terms thanks to logic formulae. Most of the works on constrained rewriting focus on order, equality, disequality and membership constraints. We extend in this work the usual notion of constrained rewrite systems by constraining rules not only with classical constraints but with any first order formula and perform rewriting according to an interpretation of functional and predicate symbols. We then show that the specification of such interpretations with tree automata offers us a framework sufficiently powerful to build an algorithm for solving a particular case of constrained matching.

6.1.2. Algebraic rewriting

Participant: Yves Guiraud.

In collaboration with P. Malbos (Institut Camille Jordan, Univ. Lyon 1), we have explored two extensions of our polygraphic version of Squier's theory [7].

In [26], we have used rewriting to give a theoretical setting and concrete formal methods to formalise and give constructive proofs of coherence theorems. The first one is Mac Lane's classical result on monoidal categories: the new proof we give is a direct application of [7]. Then, cases like symmetric monoidal categories are a first step towards a "rewriting modulo" version of the same work. Finally, we give a new understanding and a constructive proof of the result for cases like braided monoidal categories.

In [27], we have generalised the work of [7] to any dimension. We have introduced a notion of polygraphic resolution that generalises both usual algebraic resolutions, in combinatorial algebra, but also, more surprisingly, normalisation strategies, as used in rewriting theory and, in particular, in rule-based programming languages. Thus, a functional program can be mathematically defined as a complete cellular model of the functions it computes. This gives a strong mathematical background to the notion of program, together with a constructive way to build resolutions from convergent polygraphs. This work has been presented during an invited conference at the International Congress on Operads and Universal Algebra, held in Tianjin, China, in July 2010. Those results will be further explored to give a mathematical description of the strategies used in Tom, in order to develop methods from algebraic topology to study their computational properties, like termination and complexity.

6.1.3. Mechanized deduction

Participants: Clément Houtmann, Claude Kirchner, Hélène Kirchner, Cody Roux.

Higher order rewrite systems are useful abstractions to model both operational semantics of programming languages and equational reasoning in certain theories. The termination of these systems is very useful for proving correctness of programs, finding decision procedures, and proving consistency of certain theorem proving systems based on type theory.

A well studied method of proving termination is *size based termination* which allows comparison of sizes of arguments in recursive calls by typing.

We investigate a type system related to size based termination, *refinement types*, in which the type of terms in a given data-type contain additional information on the *shape* of the normal forms of the term, in the spirit of abstract interpretation. We show that given a typed higher-order rewrite system, it is possible to build a type-based *dependency graph*, and give a termination criterion by syntactic considerations of this graph, which is sufficiently powerful to express structural decrease. We demonstrate the advantages of this method over other approaches to higher-order dependency pair frameworks. This work is described in a technical report [22].

Superdeduction is a method specially designed to ease the use of first-order theories in predicate logic. The theory is used to enrich the deduction system with new deduction rules in a systematic, correct and complete way. A proof-term language and a cut-elimination reduction already exist for superdeduction, both based on Christian Urban's work on classical sequent calculus. However Christian Urban's calculus is not directly related to the Curry-Howard correspondence contrarily to the $\overline{\lambda}\mu\widetilde{\mu}$-calculus which relates straightaway to the lambda-calculus. In [28], we initiate a further exploration of the computational content of superdeduction proofs, for we extend the $\overline{\lambda}\mu\widetilde{\mu}$-calculus in order to obtain a proofterm language together with a cut-elimination reduction for superdeduction. We also prove strong normalisation for this extension of the $\overline{\lambda}\mu\widetilde{\mu}$-calculus.

6.1.4. Certification of induction proofs

Participant: Sorin Stratulat.

We have defined a methodology for validating implicit induction proofs. In collaboration with Vincent Demange, we gave evidence of the possibility to perform implicit induction-based proofs inside certified reasoning environments, as that provided by the Coq proof assistant. This is the first step of a long term project focused on 1) mechanically certifying implicit induction proofs generated by automated provers like Spike, and 2) narrowing the gap between automated and interactive proof techniques by devising powerful proof strategies inside proof assistants that aim to perform automatically multiple induction steps and to deal with mutual induction more conveniently. Contrary to the current approaches of reconstructing implicit induction proofs into scripts based on explicit induction tactics that integrate the usual proof assistants, our checking methodology is simpler and fits better for automation. The underlying implicit induction principles are separated and validated independently from the proof scripts that consist in a bunch of one-to-one translations of implicit induction proof steps. The translated steps can be checked independently, too, so the validation process fits well for parallelisation and for the management of large proof scripts. Moreover, our approach is more general; any kind of implicit induction proof can be considered because the limitations imposed by the proof reconstruction techniques no longer exist. This result has been firstly presented at the Poster session of 2010 Grande Region Security and Reliability Day, Saarbrucken.

Based on the previous result, an implementation that integrates automatic translators for generating fully checkable Coq scripts from Spike proofs is reported in [18]. The induction ordering underlying the Spike induction principle was defined using *COCCINELLE* [37], a Coq library well suited for modelling mathematical notions needed for rewriting, such as term algebras and RPO. *COCCINELLE* formalises RPO in a generic way using a precedence and a status (multiset/lexicographic) for each function symbol. Spike automatically generates a term algebra starting from Coq function symbols which preserve the precedence of the original Spike symbols. Many useful properties about the RPO orderings have been already provided by *COCCINELLE*. On the other hand, the induction ordering was modelled as a multiset extension of RPO and only few properties about it were provided by *COCCINELLE*. We have proved useful lemmas about it and added them to *COCCINELLE*, for example, the multiset extensions of RPO is stable under substitutions. Finally, every single inference step derived with a restricted version of Spike can be automatically translated into equivalent Coq script. The restricted inference system was powerful enough to prove properties about specifications involving mutually defined functions and to validate a sorting algorithm. The scripts resulted from the translation of these proofs were successfully validated by Coq.

Another improvement of the *COCCINELLE* library was the redefinition of the RPO ordering in order to consider precedencies that take into account equivalent function symbols. The new release of CoLoR (http://color.inria.fr) that compiles with the new version 8.3 of Coq includes the updated version of *COCCINELLE*. This improvement allowed Rainbow, a program developped within the CoLoR project that uses *COCCINELLE*'s formalisation for RPO, to certify more than 30 additional proofs from the set of CPF files generated during the last annual termination competition (http://termination-portal.org/wiki/Termination_Competition).

6.2. Integration of formal methods in programming languages

6.2.1. Formal islands and Tom

Participants: Jean-Christophe Bach, Horatiu Cirstea, Claude Kirchner, Pierre-Etienne Moreau, Claudia Tavares.

In [1] we have proposed a framework which makes possible the integration of formally defined constructs into an existing language. The *Tom* system is an instance of this principle: terms, first-order signatures, rewrite rules, strategies and matching constructs are integrated into Java and C for instance. The high level programming features provided by this approach are presented in [53]. The *Tom* system is documented in [29]. A general overview of the research problem raised by this approach are presented in [52].

One interest of *Tom* is to make the compilation process independent of the considered data-structure. Given any existing data-structure, using a *formal anchor* definition, it becomes possible to match and to apply transformation rules on the considered data-structures.

We have defined a new type system for *Tom* which allows to declare first-order signatures with subtyping. This is particularly useful to encode inheritance relations into algebraic terms. Considering Java as the host language, in [16] we present this type system with subtyping for *Tom*, that is compatible with Java's type system, and that performs both type checking and type inference. We propose an algorithm that checks if all patterns of a *Tom* program are well-typed. In addition, we propose an algorithm based on equality and subtyping constraints that infers types of variables occurring in a pattern. Both algorithms are exemplified and the proposed type system is showed to be sound and complete.

A first application consists in defining algebraic mappings for implementations of meta-models generated by the Eclipse Modeling Framework.

6.2.2. *Extensions of pattern-matching*

Participants: Horatiu Cirstea, Claude Kirchner, Pierre-Etienne Moreau.

Negation is intrinsic to human thinking and most of the time when searching for something, we base our patterns on both positive and negative conditions. In [13] we present the notion of anti-terms, i.e. terms that may contain complement symbols. We present algorithms for solving anti-pattern matching problems in the syntactic case as well as modulo an arbitrary equational theory E, and we study the specific and practically very useful case of associativity, possibly with a unity (AU). To this end, based on the syntacticness of associativity, we present a rule-based associative matching algorithm, and we extend it to AU. This algorithm is then used to solve AU antipattern matching problems. AU anti-patterns are implemented in the *Tom* language and we show some examples of their usage.

6.3. Practical applications

6.3.1. *Security policy analysis*

Participants: Tony Bourdier, 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 in [36], [34] a methodology for specifying, implementing and analysing access control policies using the rewrite based framework *Tom*. This verification approach specify the analysed systems without making a clear distinction between the policies and the corresponding contexts. In [20] we propose 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. We also consider in [21], [25] the operational aspects related to the abstract point of view mentioned above.

We also propose an approach [23] where the specification of a security policy is split into two distinct elements: a security model and a configuration. The security model (expressed as an equational problem) describes how authorization requests must be evaluated depending on security information. The configuration (expressed as a rewriting system) assigns values to security information. We show that this separation eases the formal analysis of security policies and makes it possible to automatically convert a given policy with respect to an alternative security model.

In computer networks, security policies are generally implemented by firewalls. We propose in [19] an original framework based on tree automata which can be used to specify firewalls and which takes into account the network address translation functionality. We show that this framework allows us to perform structural analysis as well as query analysis and comparisons over firewall policies.

6.3.2. Model transformation using rewriting

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

New development chains of critical systems rely on Domain Specific Modeling Languages (DSML) and on qualifiable transformations (insurance that a transformation preserves interesting properties). To specify and to make such transformations we have started to extend *Tom*.

A first part of this extension is an *EMF* [1] mapping generator which allows to use *Tom* with *EMF*. The idea of this tool is to generate *Tom* mappings (*i.e.* an algebraic view) by introspecting *EMF* generated Java code. These mappings can then be used to describe transformations of models that have been created in Eclipse. *Tom-EMF* is documented and available in the *Tom* source distribution.

The second part of this extension is still in development: it consists in the addition of new *Tom* language constructs to express transformations of models. Studying several use cases[2], we have already handwritten the code that should be generated. We are currently considering abstraction of the code in order to make the generation of this one automatic in a near future.

7. Other Grants and Activities

7.1. Regional Initiatives

We obtained a financial support from the Lorraine region for funding the research activities of Tony Bourdier.

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).

We participate and co-animate the "Transformation" group of the GDR–GPL (CNRS Research Group on Software Engineering).

7.2.1. ANR Complice (2009-2012)

Participant: Yves Guiraud.

The ANR project "Complexité implicite, concurrence et extraction" (Complice), headed by Patrick Baillot (CNRS, LIP Lyon), federates researchers from Lyon (LIP), Nancy (LORIA) and Villetaneuse (LCR). The coordinator for the LORIA site is Guillaume Bonfante (Carte).

7.2.2. ANR Ravaj (2007-2010)

Participant: Pierre-Etienne Moreau.

Ravaj (Réécriture et Approximation pour la Vérification d'Applications Java) is an ANR project coordinated by Thomas Genet (Irisa). The goal is to model Java bytecode programs using term rewriting and to use completion techniques to compute the set of reachable terms. Then, it is possible to check some properties related to reachability (in particular safety and security properties) on the modeled system using tree automata intersection algorithms.

7.2.3. ANR SSURF (2007-2010)

Participants: Tony Bourdier, Horatiu Cirstea.

[1] Eclipse Modeling Framework, http://www.eclipse.org/modeling/emf/
[2] available on the Quarteft subversion repository: https://gforge.enseeiht.fr/projects/quarteft/

“SSURF: Safety and Security under FOCAL” is an ANR project coordinated by Mathieu Jaume (LIP6). The SSURF project consists in characterizing and studying the required features that an Integrated Development Environment (IDE) must provide in order not only to obtain software systems in conformance with high Evaluation Assurance Levels (EAL-5, 6 and 7), but also to ease the evaluation process according to various standards (*e.g.* IEC61508, CC, ...). Moreover we aim at developing a formal generic framework describing various security properties, *e.g.* access control policies, together with their implementations using such an IDE. A more detailed presentation is available at http://www-spi.lip6.fr/~jaume/ssurf.html.

7.2.4. ARC CORiAS (2009-2010)

Participants: Horatiu Cirstea, Clément Houtmann, Claude Kirchner, Cody Roux.

The INRIA ARC "Conception et réalisation d’assistants à la preuve fondés sur la surdéduction modulo" (CORiAS), headed by Germain Faure (INRIA, Saclay), federates researchers from Nancy (LORIA) and Saclay (LIX). The coordinator for the LORIA site is Horatiu Cirstea. A detailed presentation is available at http://www.lix.polytechnique.fr/corias/.

7.2.5. ARC ACCESS (2010-2011)

Participant: Horatiu Cirstea.

This project is concerned with the security and access control for Web data exchange, in the context of Web applications and Web services. We aim at defining automatic verification methods for checking properties of access control policies (ACP) for XML, like consistency or secrecy. A more detailed presentation is available at http://acxml.gforge.inria.fr/.

7.2.6. ARC REDO (2009-2010)

Participant: Yves Guiraud.

The INRIA ARC "Redesigning logical syntax" (REDO), headed by Lutz Straßburger (INRIA, LIX Saclay), federates researchers from Bath (Department of Computer Science), Nancy (LORIA) and Saclay (LIX). The coordinator for the LORIA site is François Lamarche (Calligramme). The ARC held a meeting in Nancy on November 16-18. A more detailed presentation is available at, http://www.lix.polytechnique.fr/~lutz/orgs/redo.html

7.2.7. FRAE QUARTEFT (2009-2012)

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

“QUARTEFT: QUAlifiable Real TimE Fiacre Transformations” is a research project founder 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. A main difficulty 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/.

7.3. International Initiatives

Chili. We have an associated team “VanaWeb” that started in 2008 and continues the collaboration initiated during the joint project INRIA-CONICYT (Chili), VANANAA (formerly, COCARS). It is a project on rules and strategies for the hybrid resolution of constraint problems with applications to composition problems for the Web. We have many exchanges with Carlos Castro and his group (UTFSM, Valparaiso, Chile).
Brazil. Project INRIA-CNPq (Brazil), DA CAPO - Automated deduction for the verification of specifications and programs. It is a project on the development of proof systems for the verification of specifications and software components. The coordinators of this project are David Déharbe (UFRN Natal, Brazil) and Christophe Ringeissen (CASSIS). On the french side, DA CAPO also involves the CASSIS project.

8. Dissemination

8.1. Animation of the scientific community

Tony Bourdier:

- Member of the scientific board of the University of Nancy 1.
- Member of the board of the Doctoral School in Computer Science and Mathematics.

Horatiu Cirstea:

- Member of the Loria Laboratory Council.
- PC member of RuleML 2010 (International RuleML Symposium on Rule Interchange and Applications).
- Steering committee of RULE.

Yves Guiraud:

- Animation of the group "Invariants algébriques en informatique" [Algebraic invariants in Computer Science], http://math.univ-lyon1.fr/~guiraud/iai, Institut Camille Jordan, Lyon.

Pierre-Etienne Moreau:

- Co-chair of LDTA 2010 (International Workshop on Language Descriptions, Tools and Applications),
- PC member of SLE 2010 (International Conference on Software Language Engineering), RULE 2010 (International Workshop on Rule-Based Programming), WRLA 2010 (International Workshop on Rewriting Logic and its Applications), IWS 2010 (International Workshop on Strategies in Rewriting, Proving, and Programming).
- Member of the board of the Doctoral School in Computer Science and Mathematics.
- Member of the Direction board of LORIA.
- Head of the local committee for INRIA "détachements" and "délégations".

8.2. Teaching

We do not mention the teaching activities of the various teaching assistants, associate professors and professors of the project who work in various universities of the region.
Tony Bourdier:

- Licence (L3) course in Nancy (ESSTIN) on statistics.
- Supervision of Francois Prugniel (internships ESIAL).

Horatiu Cirstea:

- Master course in Nancy on programming and proving with rule based languages, with Pierre-Etienne Moreau.
- Supervision of Cyrille Cornu (internship École des Mines de Nancy).

Pierre-Etienne Moreau:

- Head of the Computer Science department at École des Mines de Nancy
- Master course in Nancy on programming and proving with rule based languages, with Horatiu Cirstea.

Sorin Stratulat:

– In the frame of the ERASMUS LLP EMaCS project (http://www.ecs-emacs.net) and in collaboration with Gabriel Michel, we analysed in [17] the student mobility from different points of view (geographical, social and cultural) and proposed solutions to improve student mobility in the field of Computer Science. Some preliminary results about concrete case studies concerning French-German double degree programs at the ISFATES-DFHI institute were presented.

8.3. Communications in conferences

Tony Bourdier:

– “Constrained rewriting in recognizable theories”, Journées Nationales GEOCAL-LAC, March 2010, Nice.

– “Formal analysis of firewalls using tree automata techniques”, Grande Region Security and Reliability Day, March 2010, Saarbrucken, Germany.

Horatiu Cirstea:

– “On Formal Specification and Analysis of Security Policies”, Grande Region Security and Reliability Day, March 2010, Saarbrucken, Germany.

– “Rule-based Specification and Analysis of Security Policies”, 5th International Workshop on Security and Rewriting Techniques - SecRet’10, June 2010, Valencia, Spain.

Yves Guiraud:

– “Higher-dimensional normalisation strategies for acyclicity”, invited conference, Operads and Universal Algebra, Tianjin, China, July 2010.

Pierre-Etienne Moreau:

– Invited talk at "Journées Nationales 2010 du GDR GPL" : "Combiner Java et réécriture, c’est possible et utile", march 2010, Pau.

Cody Roux:

– “Dependent types as higher order dependency pairs”, Journées Nationales GEOCAL-LAC, March 2010, Nice.

8.4. Theses

– Horatiu Cirstea, “Le calcul de réécriture”, HDR

– Clément Houtmann, “Représentation et interaction des preuves en superdéduction modulo”, PhD

– Paul Brauner, “Fondements et mise en œuvre de la Super Déduction Modulo”, PhD

8.5. Thesis and admission committees

Pierre-Etienne Moreau:

– Paul Brauner: “Fondements et mise en œuvre de la Super Déduction Modulo”, PhD, March 2010, INPL,

– Julien Blond: “Modélisation et implantation d’une politique de sécurité d’un OS multi-niveaux via une traduction de FoCaLize vers C”, PhD, December 2010, Paris 6.

– Laurent Hubert: “Foundations and Implementation of a Tool Bench for Static Analysis of Java Bytecode Programs”, PhD, December 2010, Rennes 1.

– Member of the recruitment committee for a Professor position at ESIAL (Nancy 1).

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] G. BONFANTE, Y. GUIRAUD. *Polygraphic programs and polynomial-time functions*, in "Logical Methods in Computer Science", 2009, vol. 5, n^{o} 2:14, p. 1-37.

[4] 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.

[5] 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.

[6] Y. GUIRAUD. *Termination orders for 3-dimensional rewriting*, in "Journal of Pure and Applied Algebra", 2006, vol. 207, n^{o} 2, p. 341-371.

[7] Y. GUIRAUD, P. MALBOS. *Higher-dimensional categories with finite derivation type*, in "Theory and Applications of Categories", 2009, vol. 22, n^{o} 18, p. 420-478.

[8] 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.

[9] 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

[10] H. CIRSTEA. *Le calcul de réécriture*, Université Nancy II, October 2010, http://hal.inria.fr/tel-00546917/en.

Articles in International Peer-Reviewed Journal

[11] 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 [*DOI :* 10.4204/EPTCS.15.1], http://arxiv.org/abs/1001.4427, http://hal.inria.fr/inria-00494636/en.

[12] G. BUREL, C. KIRCHNER. *Regaining Cut Admissibility in Deduction Modulo using Abstract Completion*, in "Information and Computation", February 2010, vol. 208, n° 2, p. 140-164 [*DOI :* 10.1016/J.IC.2009.10.005], http://hal.inria.fr/inria-00132964/en.

[13] 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° 5, p. 523-550 [*DOI :* 10.1016/J.JSC.2010.01.007], http://hal.inria.fr/inria-00429226/en.

International Peer-Reviewed Conference/Proceedings

[14] F. DURÁN, M. ROLDAN, J.-C. BACH, E. BALLAND, M. VAN DEN BRAND, J. R. CORDY, S. EKER, L. ENGELEN, M. DE JONGE, K. TRYGVE KALLEBERG, L. C. KATS, P.-E. MOREAU, E. VISSER. *The Third Rewrite Engines Competition*, in "WRLA 2010", Cyprus Paphos, Springer-Verlag, 2010, vol. 6381/2010, p. 243-261 [*DOI :* 10.1007/978-3-642-16310-4_16], http://www.springerlink.com/content/436358m836203763/, http://hal.inria.fr/inria-00552037/en.

[15] H. KIRCHNER, F. KIRCHNER, C. KIRCHNER. *Constraint Based Strategies*, in "18th International Workshop on Functional and Constraint Logic Programming - WFLP 2009", Brazil Brasilia, S. ESCOBAR (editor), Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2010, vol. 5979, p. 13-26, http://hal.inria.fr/inria-00494531/en.

[16] C. KIRCHNER, P.-E. MOREAU, C. TAVARES. *A Type System for Tom*, in "The Tenth International Workshop on Rule-Based Programming", Brazil Brasilia, I. MACKIE, A. M. MOREIRA (editors), March 2010, vol. 21 [*DOI :* 10.4204/EPTCS.21.5], http://hal.inria.fr/inria-00426439/en.

[17] G. MICHEL, S. STRATULAT. *Good reasons to implement transnational European diploma programs in Computer Science*, in "ICEUTE'2010 (First International Conference on EUropean Transnational Education)", 2010, p. 135–143.

[18] S. STRATULAT. *Integrating Implicit Induction Proofs into Certified Proof Environments*, in "Integrated Formal Methods - IFM 2010", France Nancy, D. MERY, S. MERZ (editors), Lecture Notes in Computer Science, Springer Berlin / Heidelberg, October 2010, vol. 6396, p. 320-335, http://www.springerlink.com/index/32221517Q3T88366.pdf, http://hal.inria.fr/inria-00525187/en.

Workshops without Proceedings

[19] T. BOURDIER. *Formal analysis of firewalls using tree automata techniques*, in "2010 Grande Region Security and Reliability Day", Germany Saarbrücken, March 2010, http://hal.inria.fr/inria-00460462/en.

[20] T. BOURDIER, H. CIRSTEA, M. JAUME, H. KIRCHNER. *On Formal Specification and Analysis of Security Policies*, in "2010 Grande Region Security and Reliability Day", Germany Saarbrücken, March 2010, http://hal.inria.fr/inria-00429240/en.

[21] T. BOURDIER, H. CIRSTEA, M. JAUME, H. KIRCHNER. *Rule-based Specification and Analysis of Security Policies*, in "5th International Workshop on Security and Rewriting Techniques", Spain Valencia, June 2010, http://hal.inria.fr/inria-00552221/en.

Research Reports

[22] C. ROUX. *Refinement Types as Higher Order Dependency Pairs*, LORIA, 01 2011, http://hal.inria.fr/inria-00552046/en/.

Other Publications

[23] T. BOURDIER. *Specification, analysis and transformation of security policies via rewriting techniques*, 2010, 22 pages, http://hal.inria.fr/inria-00525761/en.

[24] T. BOURDIER, H. CIRSTEA. *Constrained rewriting in recognizable theories*, 2010, 19 pages, http://hal.inria.fr/inria-00456848/en.

[25] T. BOURDIER, H. CIRSTEA, M. JAUME, H. KIRCHNER. *Formal Specification and Validation of Security Policies*, 2010, 16 pages, http://hal.inria.fr/inria-00507300/en.

[26] Y. GUIRAUD, P. MALBOS. *Coherence in monoidal track categories*, 2010, Mathematical Structures in Computer Science (to appear), 32 pages, http://hal.inria.fr/hal-00470795/en.

[27] Y. GUIRAUD, P. MALBOS. *Higher-dimensional normalisation strategies for acyclicity*, 2010, 46 pages, http://hal.inria.fr/hal-00531242/en.

[28] C. HOUTMANN. *Superdeduction in Lambda-bar-mu-mu-tilde*, 2010, 11 pages, http://hal.inria.fr/inria-00498744/en.

References in notes

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

[30] 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.

[31] G. BONFANTE, Y. GUIRAUD. *Intensional properties of polygraphs*, in "Electronic Notes in Theoretical Computer Science", 2008, vol. 203, n^{o} 1, p. 65-77.

[32] 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.

[33] 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.

[34] T. BOURDIER, H. CIRSTEA, P.-E. MOREAU, A. SANTANA DE OLIVEIRA. *Analysis of Lattice-Based Access Control Policies using Rewiting Systems and Tom.*, in "1st Luxembourg Day on Security and Reliability", Luxembourg, 2009.

[35] A. BURRONI. *Higher-dimensional word problems with applications to equational logic*, in "Theoretical Computer Science", 1993, vol. 115, n^{o} 1, p. 43-62.

[36] H. CIRSTEA, P.-E. MOREAU, A. SANTANA DE OLIVEIRA. *Rewrite Based Specification of Access Control Policies*, in "3rd International Workshop on Security and Rewriting Techniques - SecReT 2008", Pittsburgh, USA, 2009, vol. 234, p. 37-54.

[37] E. CONTEJEAN, P. COURTIEU, J. FOREST, O. PONS, X. URBAIN. *Certification of Automated Termination Proofs*, in "Frontiers of Combining Systems", 2007, p. 148–162.

[38] COQ DEVELOPMENT TEAM. *The Coq Proof Assistant, version 8.2*, 2009, http://coq.inria.fr.

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

[40] Y. GUIRAUD. *Présentations d'opérades et systèmes de réécriture*, Université Montpellier 2, 2004.

[41] Y. GUIRAUD. *The three dimensions of proofs*, in "Annals of Pure and Applied Logic", 2006, vol. 141, n^{o} 1-2, p. 266-295.

[42] Y. GUIRAUD. *Two polygraphic presentations of Petri nets*, in "Theoretical Computer Science", 2006, vol. 360, n^{o} 1-3, p. 124-146.

[43] Y. GUIRAUD. *Polygraphs for termination of left-linear term rewriting systems*, 2007, Preprint.

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

[45] 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.

[46] 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.

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

[48] C. KIRCHNER, F. KIRCHNER, H. KIRCHNER. *Strategic Computations and Deductions*, in "Festchrift in honor of Peter Andrews", Studies in Logic and the Foundations of Mathematics, Elsevier, 2008.

[49] 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.

[50] 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.

[51] 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.

[52] P.-E. MOREAU. *Programmation et confiance*, Institut National Polytechnique de Lorraine - INPL, 06 2008, http://tel.archives-ouvertes.fr/tel-00337408/en/.

[53] P.-E. MOREAU, A. REILLES. *Rules and Strategies in Java*, in "7th International Workshop on Reduction Strategies in Rewriting and Programming - WRS 2007 Proceedings of the 7th International Workshop on Reduction Strategies in Rewriting and Programming (WRS 2007) Electronic Notes in Theoretical Computer Science", France Paris, J. GIESL (editor), Elsevier, 2008, vol. 204, p. 71-82, http://hal.inria.fr/inria-00185698/en/.

[54] 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.

[55] M. PARIGOT. *Lambda-mu-calculus: An algorithmic interpretation of classical natural deduction*, in "Logic Programming and Automated Reasoning", St Petersburg, Russia, A. VORONKOV (editor), Springer, 1992, p. 190-201.

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

[57] 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.