INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Team Parsifal

Preuves Automatiques et Raisonnement sur des SpécIFicAtions Logiques

Futurs

THEME SYM

Table of contents

1. Team

Joint team with LIX (Laboratoire d'Informatique de l'Ecole Polytechnique). Parsifal has been approved as an INRIA team ("équipe") since September 2005.

Team Leader
Dale Miller [DR-INRIA]

Administrative assistant
Catherine Moreau

Vice-head of parsifal
Joëlle Despeyroux [CR. INRIA-Sophia]

Personnel Inria
Lutz Straßburger [CR.]

Ph. D. student
David Baelde [Allocataire École Normale Supérieure, Lyon. Since 1/09/2005.]
Alexis Saurin [Allocataire École Normale Supérieure, Paris. Since 1/10/2004.]
Olivier Delande [École Polytechnique. Since 15/10/2006]
Vivek Nigam [Allocataire Mobius. Since 1/10/2006.]

Post-doctoral fellow
Kaustuv Chaudhuri [Post Doctorant INRIA. Since 1/11/2006.]

Invited researchers
Chuck Liang [Associate Professor, Hofstra University, NY, USA, from 21 September to 15 December]
Andrew Gacek [PhD student from the University of Minnesota from 29 May to 28 July.]
Frank Pfenning [Professor at CMU, USA, in June and July.]

Student intern
William Audry [École Normale Supérieure. From 1/03/2006.]

2. Overall Objectives

2.1. Overall Objectives

Software correctness is a key aspect of many computing systems. For example, computers and software are used to help control nuclear power plants, avionic controls, and automobiles and, in such safety-critical systems, incorrect software can cause serious problems. Similarly, errors in networking software, operating systems, browsers, etc, can leave computer systems open for computer viruses and security breeches. In order to avoid errors in such complex and interacting systems, one must be able to prove the correctness of individual application programs as well as a wide range of software systems that analyze and manipulate them: these range from compilers and linkers, to parsers and type checkers, to high-level properties of entire programming languages. In the face of this increasing need for program correctness, an international community of researchers is developing many approaches to the correctness of software. Formal methods are gaining acceptance as one viable approach to addressing program correctness and this project will focus on using such methods to address this problem.

The Parsifal team aims at elaborating methods and tools for specifying and reasoning about computation systems such as compilers, security protocols, and concurrent programs. A central challenge here is proving properties of programs that manipulate other programs. The specification of such computational systems today is commonly given using operational semantics, supplanting the well-established but restrictive approach using denotational semantics. Operational semantics is generally given via inference rules using relations between different items of the computation, and for this reason, it is an example of a *relational specification*. Inference rules over relations are also used for specifying the static semantics for programming languages as well (type inference, for example). The use of denotational style presentations of computational systems naturally leads to the use of functional programming-based executable specifications. Similarly, the use of inference systems for the presentation of operational semantics provides a natural setting for exploiting logic programming-based implementations.

The Parsifal project will exploit recent developments in proof search, logic programming, and type theory to make the specification of operational semantics more expressive and declarative and will develop techniques and tools for animating and reasoning directly on logic-based specifications. More specifically, the Parsifal project will focus on the following goals.

- Foundations: We plan to exploit proof search in expressive logics for the specification of computations and to develop a single logic for reasoning about proof search specifications. This logic will be based on higher-order intuitionistic logic and will include proof principles for induction and co-induction. We shall also consider its analogous design as a type system.
- Prototypes: We plan to build prototype components needed for the implementation of proof search (including unification, search, tabling, binding and substitution in syntax, etc) and use these components to build specific research prototypes for a range of applications. We shall explore architectures for provers that allow differing amounts of interaction and automation.
- Applications: We will test the feasibility of incorporating our deductive tools into various applications that can benefit from mixing computation and deduction. Application areas we have in mind are security, global computing, proof-carrying code, and mobile code.

Parsifal's focus on formal specification and correctness of computer systems means that this project is supporting INRIA's objective to "guarantee the reliability and security of software intensive systems" (Strategic Plan 2003-2007).

3. Scientific Foundations

3.1. Proof search

Keywords: *linear logic*, *logic programming*, *proof search.*

Participants: David Baelde, Dale Miller, Alexis Saurin, Lutz Straßburger.

The term "proof search" refers to a modern updating of the logic programming paradigm. In particular, that update implies several things that have not been generally embraced by the logic programming community.

- Proof search takes its motivation for new design and analysis from proof theory instead of model theory. Proof theory provides a solid foundation for explaining logic programming generally and has been a productive framework for motivating exciting new programming languages.
- Proof theory allows for relatively easy exploring of such intensional aspects of computation as binders, binder mobility, and resource management.
- Proof search, given its foundations in "finitistic" proof theory often allows one to move quickly to consider implementations using logic variables, unification, backtracking, resource-splitting, explicit substitutions, etc.

- Proof search has a focus on logically correct deduction. In particular, many short cuts that have been easy to implement in, say, Prolog systems, such as negation-as-failure, Prolog's cut, unsound unification, etc, are generally de-emphasized.
- Proof search emphasizes specifications and execution of specifications rather than programming and pure performance. Emphasis is more generally on clear and deductive specifications. There are, however, at least a couple of examples of when the more declarative specification also turned out to provide the more effective implementation (given suitable compilers): see [34] and [32].
- Proof search design and theory also focuses on the meaning of logical connectives and quantifiers. This focus is in contrast to the focus of constraint logic programming (on processing of constraints on terms) and concurrent logic programming language (with their focus on novel readings of logic variables or constraints) [26].

The proof theory of proof search has been used to design various logic programming languages. In the late 1980's, sequent calculus for intuitionistic logic was used to motivate the various extensions now incorporated into λProlog. Later, linear logic programming languages have been introduced: in particular, LO of Andreoli and Pareschi [18], Lolli of Hodas and Miller [31], and Forum of Miller [38].

A key application area for proof search and the above logic programs is the specification of operational semantics. Particularly successful specifications along these lines are those of evaluation and typing for the untyped λ-calculus [3] and reachability within progress calculi, such as the π-calculus [6], [47], [50], [51].

3.2. Reasoning about logic specifications

Keywords: *LINC*, *definitions*, *fixed points*, *generic judgments*, *higher-order abstract syntax*, *lambda-tree syntax*.

Participants: David Baelde, Joëlle Despeyroux, Dale Miller, Alexis Saurin.

Now that meaning is given using logic specifications (relational specifications), how do we reason about the formal properties of such specifications? Addressing this question is a central focus of the theoretical aspects of this project.

The traditional architecture for systems designed to help reasoning about the formal correctness of specification and programming languages can generally be characterized at a high-level as follows: **First: Implement mathematics.** This often involves choosing between a classical or constructive (intuitionistic) foundation, as well as a choosing abstraction mechanism (eg, sets or functions). The Coq and NuPRL systems, for example, have chosen intuitionistically typed λ-calculus for their approach to the formalization of mathematics. Systems such as HOL [29] use classical higher-order logic while systems such as Isabelle/ZF [44] use classical logic. **Second: Reduce programming correctness problems to mathematics.** Thus, data structures, states, stacks, heaps, invariants, etc, all are represented as various kinds of mathematical objects. One then reasons directly on these objects using standard mathematical techniques (induction, primitive recursion, fixed points, well-founded orders, etc).

Such an approach to formal methods is, of course, powerful and successful. There is, however, growing evidence that many of the proof search specifications that rely on such intensional aspects of logic as bindings and resource management (as in linear logic) are not served well by encoding them into the traditional data structures found in such systems. In particular, the resulting encoding can often be complicated enough that the *essential logical character* of a problem can be obfuscated.

We propose to use the LINC logic of [48] as the meta-logic for our system for reasoning about computation systems. The three key ingredients of LINC can be described as follows.

First, LINC is an intuitionistic logic for which provability is described similarly to Gentzen's LJ calculus [27]. Quantification at higher-order types (but not predicate types) is allowed and terms are simply typed λ-terms over $\beta\eta$-equivalence. This core logic provides support for λ-*tree syntax*, a particular approach to *higher-order abstract syntax*. Considering a classical logic extension of LINC is also of some interest, as is an extension allowing for quantification at predicate type.

Second, LINC incorporates the proof-theoretical notion of *definition*, a simple and elegant device for extending a logic with the if-and-only-if closure of a logic specification and for supporting inductive and co-inductive reasoning over such specifications. This notion of definition was developed by Hallnäs and Schroeder-Heister [30] and, independently, by Girard [28]. Later McDowell, Miller, and Tiu have made substantial extensions to our understanding of this concept [36], [3], [40]. Tiu and Momigliano [41], [48] have also shown how to modify the notion of definition to support induction and co-induction in the sequent calculus.

Third, LINC contains a new (third) logical quantifier ∇ (nabla). After several attempts to reason about logic specifications without using this new quantifier [35], [37], [3], it became clear that when the object-logic supports λ-tree syntax, the *generic judgment* [40], [6] and its associated quantifier could provide a strong and declarative role in reasoning. This new quantifier is able to help capture internal (intensional) reasons for judgment to hold generically instead of the universal judgment that holds for external (extensional) reasons. Another important observation to make about ∇ is that if given a logic specification that is essentially a collection of Horn clauses (that is, there is no uses of negation in the specification), there is no distinctions to be made between $\forall$ or ∇ in the premise (body) of semantic definitions. In the presence of negations and implications, a difference between these two quantifiers does arise [6].

3.3. A type theory for proof search

Keywords: *co-induction*, *higher-order abstract syntax*, *induction*, *type theory*.

Participants: Joëlle Despeyroux, Dale Miller.

Type theory can be used to specify and reason about computation using much as LINC is used as a to reason about computation. The meta-level would be, for example, CIC. The applications would in principle be the same, except that conventional type theories are often better suited for proofs in mathematics and less suited for encoding of languages with binders, although Coq can certainly code reachability of, say, the π-calculus [19].

The type theory approach does not need to make a distinction between meta-logic and object-logic, which may make specifications more compactness and elegance. This approach also avoids the duplication of code. However, we like the flexibility given by the existence of two different meta-logics in LINC. Note that the Isabelle system also uses two meta-logics, giving rise to several systems that people can choose: Isabelle/ZF, Isabelle/HOL, etc. Having a single meta-logic is more natural in the context of a rich type theory. Outside this context, we think that both choices make sense.

More points of comparison between the two approaches are listed below.

- Type theory is generally explained using *natural deduction*, whereas proof search usually relies on *sequent calculus*. Normalization of proofs in type theory correspond to computation. In contrast, proof search explores only normal proofs (cut-free proofs) and uses normalization (cut-elimination) for reasoning about computation.
- Type theory provides a fixed, rich notion of proof, usually with dependent types, while there is no a-priori notion of proofs as objects in proof search, although most proof search systems provide primitives for building proof objects.
- The development of proofs using relational specifications require the instantiation of quantified variables. Proof search systems provide unification and backtracking search to fully automate such instantiations. In type theory, such instantiations are usually done interactively.
- In type theory, object level bindings are either coded as functions (a priori, a wrong choice) or names or deBruijn numbers (usually messy), while binding in syntax can be naturally supported by the meta logic in proof search (as in λProlog and Twelf).
- The new ∇ quantifier in LINC describes the intensional behavior of abstraction, which has no (current) correspondence in type theory.

In their first experiments [21], [20], Despeyroux, Hirschowitz, and Felty worked in the Coq system. They defined certain expressions in higher-order abstract syntax as "valid" expressions (using an inductively defined predicate "valid") to describe a well defined sub-part of all the functional terms corresponding roughly to syntactic expressions.

In a second step, Despeyroux, Pfenning, Leleu, and Schürmann proposed two different, yet similar type theories [22], [25], [23] based on modal logic in which expressions live in the functional space $A \to B$ while general functions (for case and iteration reasoning in the proposed systems) live in the full functional space $\Box A \to B$. An initial attempt to extend the systems in [22], [23] to dependent types was given in [24]: clearly more work on that kind of extension is needed. These papers give a possible answer to the problem of extending the Edinburgh Logical Framework (LF is a sub-system of CC where the function space is restricted to λ-trees) with recursion and induction principles.

Pfenning and Schürmann then proposed a different (two levels) system to which it was easier to add dependent types [46], [45].

The previous works mentioned proposed principles for recursion. In the traditional functional approach, in a rich type theory like CIC, those principles naturally come with their counterparts at the level of types: the principles for induction. In proof search, the situation is less clear. Among the works proposing principles for induction in our context, let us cite the similar induction principles independently discovered by Despeyroux & co-workers at the proof level [21], [22], [23] and by Hofmann at the model level [33]. There is also the work by Miller and McDowell [36] and more recently the work by Momigliano and Tiu on LINC [41].

4. Application Domains

4.1. Model checking operational semantics

Keywords: *model checking*, *operational semantics*, *process calculus*, *symbolic bisimulations*.

When operational semantics is presented as inference rules, it can often be encoded naturally into a logic programming setting. The logic programming foundations developed in Parsifal allows for such operational semantics to be animated in direct and natural ways. Recent work on searching fixed points of logic program specifications allow for the automating of bisimulation checking for finitary processes. This foundational work lead to a proof theory [4] and game semantic [39] that allows for the natural duality of finite success and finite failure.

Given its basis in proof theory, an intensional treatment of syntax and bindings in syntax was natural to develop [6]. The design of a working system, Level 0/1, that exploits these foundational ideas is presented in [49], [7]. The resulting system is one of the few systems that can directly implement bisimulation checking for the π-calculus [50], [47].

Since operational semantics can be described declaratively, our work should allow for richer approaches to developing rule forms for operational semantics.

4.2. Mechanized metatheory

The Parsifal project has been developing approaches to meta-logic and theorem proving architecture that should allow for reasoning about the operational semantics of computational systems. Early work along these lines was reported in [3]. The meta-logical framework reported in that paper has been greatly extended by Miller and Tiu [6], [48]. These papers report success in the way such meta-logic can be used interactive to establish important properties concerning operational semantics. This work should naturally lead to addressing the following different applications.

- Model checking as deduction. Being able to see model checking as a special kind of deduction should have many benefits for establishing the correctness of computational systems.
- The POPLmark Challenge: Mechanized Meta-theory for the Masses
- Correctness of security protocols
- Mixing deduction and computation, as in, for example, proof carrying code.

5. Software

5.1. Bedwyr

Participants: Dale Miller, David Baelde [correspondant].

Bedwyr is a generalization of logic programming that allows model checking directly on syntactic expression possibly containing bindings. This system, written in OCaml, is a direct implementation of two recent advances in the theory of proof search.

1. It is possible to capture both finite success and finite failure in a sequent calculus. Proof search in such a proof system can capture both may and must behavior in operational semantics.
2. Higher-order abstract syntax is directly supported using term-level λ-binders, the ∇-quantifier, higher-order pattern unification, and explicit substitutions. These features allow reasoning directly on expressions containing bound variables.

The distributed system comes with several example applications, including the finite π-calculus (operational semantics, bisimulation, trace analysis, and modal logics), the spi-calculus (operational semantics), value-passing CCS, the λ-calculus, winning strategies for games, and various other model checking problems.

While the system has been written to validate certain theoretic results and to help suggest new theoretical directions, we believe that the system can be of use to others interested more in reasoning about computer systems than about proof theory foundations.

Bedwyr is an open source project hosted by INRIA's GForge. We welcome contributions from others of example applications and new features. The developers behind the current distribution are:

- David Baelde and Dale Miller (INRIA and LIX/Ecole Polytechnique)
- Andrew Gacek and Gopalan Nadathur (University of Minneapolis)
- Alwen Tiu (Australian National University and NICTA).

The developers thank Brigitte Pientka, for her advice regarding tabling, and Axelle Ziegler, for her help in translating an early implementation from SML to OCaml. Much of the effort for this work is supported within the context of the Slimmer project, which is supported by funds from INRIA and NSF.

The system's document and sources can be found at http://slimmer.gforge.inria.fr/bedwyr/. Version 1.0 of Bedwyr was released on November 1st, 2006. About 65 downloads of the system were made during the following six weeks.

1. Mixing Finite Success and Finite Failure in an Automated Prover by A. Tui, G. Nadathur, and D. Miller. (pdf)
2. Model checking for pi-calculus using proof search by Alwen Tiu. In Proceedings of CONCUR 2005, LNCS Vol. 3653, pages 36 - 50, Springer-Verlag, 2005. (pdf).
3. A proof theory for generic judgments by Dale Miller and Alwen Tiu. ACM Trans. on Computational Logic, 6(4):749-783, October 2005. (.pdf).

6. New Results

6.1. Static analysis via proof theory

Static analysis of logic programs can provide useful information for programmers and compilers. Typing systems, such as in λProlog [42], [43], have proved valuable during the development of code: type errors often represent program errors that are caught at compile time when they are easier to find and fix than at runtime when they are much harder to repair. Static type information also provides valuable documentation of code since it provides a concise approximation to what the code does.

In [11], Miller described a method by which it is possible to infer that certain relationships concerning collections underlying structured data hold. For collections, both *multisets* and *sets* were used to *approximate* more complicated structures as lists and binary trees. Consider, for example, a list sorting program that maintains duplicates of elements. Part of the correctness of a sort program includes the fact that if the atomic formula $sort(t, s)$ is provable, then s is a permutation of t that is in-order. The proof of such a property is likely to involve inductive arguments requiring the invention of invariants: in other words, this is not likely to be a property that can be inferred statically during compile time. On the other hand, if the lists t and s are approximated by multisets (that is, if we forget the order of items in lists), then it might be possible to establish that if the atomic formula $sort(t, s)$ is provable, then the multiset associated to s is equal to the multiset associated to t. If that is so, then it is immediate that the lists t and s are, in fact, permutations of one another (in other words, no elements were dropped, duplicated, or created during sorting). Such properties based on using multisets to approximate lists can often be inferred statically.

While this work focused exclusively on the static analysis of first-order Horn clauses, it does so by making substitution instances of such Horn clauses that carry them into linear logic. Proofs for the resulting linear logic formulas are then attempted as part of static analysis.

Work such as this suggests that declarative languages might permit rich kinds of static analysis. Such preliminary work provided a partial justification of the need for flexible connections between programming languages and static analysis that is briefly described in [10].

6.2. Reasoning about operational semantics

The operational semantics of programming and specification languages is often presented via inference rules and these can generally be mapped into logic programming-like clauses. Such logical encodings of operational semantics can be surprisingly declarative if one uses logics that directly account for term-level bindings and for resources, such as are found in linear logic. Traditional theorem proving techniques, such as unification and backtracking search, can then be applied to animate operational semantic specifications. Of course, one wishes to go a step further than animation: using logic to encode computation should facilitate formal reasoning directly with semantic specifications. In the paper [12], Miller outlined an approach to reasoning about logic specifications that involves viewing logic specifications as theories in an object-logic and then using a meta-logic to reason about properties of those object-logic theories.

The Bedwyr system (described in the Software section) has been used to valid at least some of the design ideas presented in [12]. These results are encouraging and the team plans to develop those themes to a greater extent.

6.3. Effective implementation of module checking with linguistic expressions

The effort to implement the Bedwyr provided the team with a number of challenges. A number of these challenges involved dealing with how best to implement λ-terms efficiently to support unification, backtracking search, and the "flipping" of variable status (internally, Bedwyr has two provers and the status of variables in these two systems are essentially duals). Most of these implementation details are documented directly in the code and, in part, in the Bedwyr user manual [14].

6.4. Structure and semantics of proofs

Lutz Straßburger continues to probe the structure and semantics of proofs. In particular, in his lectures notes [15], he develops the basics of the structure of proof nets for linear logic for a non-specialist audience. These notes provide a comprehensive perspective on how proofs can be constructed and equated.

Straßburger is also developing semantic approaches capturing proof structures within Boolean categories [13] and within free *-autonomous categories [9]. In a field where almost all that is written about proofs is syntactic (structure induction over the height of proofs, etc), these contributions to the semantics of proofs are important alternative avenues of investigation.

Chuck Liang, a professor from Hofstra University, spent his sabbatical at LIX in the fall of 2006. He and Miller have written the report [16] that describes a general framework for capturing focusing-style proof systems for intuitionistic and classical logic. As a result, a rather strong normal form of cut-free proofs are available. We expect to be able to apply this framework to account for almost all known focusing-style calculi for intuitionistic logic. Applications of this normal form that we are now pursuing involve the structure of implementation of λ-terms and λ-reduction as well as new approaches to proving the completeness of various first-order logic theorem proving strategies.

Miller is beginning the supervision of two PhD students (Delande and Nigam) who are working on understanding the structure of cut-free system in richer and more "practical" senses than simply as "trees-of-inference rules" or as typed λ-terms. In the context of the PCC (proof-carrying-code) paradigm, proofs can come from a wide range of formal proving systems that are not traditional theorem provers (for example, model checkers, static analyzers, etc) and proofs need to meet certain "engineering" goals (compact size, fast checking, reuse, modular, etc). No concrete research results are available currently.

6.5. Proof theory approaches to induction

Systems like model-checkers and the Bedwyr system explores a computational system through all its finite behaviors. As a result, most such systems cannot handle infinite state spaces, many kinds of looping behavior, etc. Baelde and Tiu spent some a month recently attempting to develop a proof theory for cyclic proofs. Our current efforts are being directed, instead, on more conventional notions of proof search (focusing proofs) in the presence of induction.

7. Other Grants and Activities

7.1. Actions nationales

7.1.1. Rossignol: Project ACI Securité

Participant: Dale Miller.

The project ROSSIGNOL started in 2003 and includes the following participants: LIF (responsable: D. Lugiez), INRIA Futurs (responsable: C. Palamidessi), LSV (responsable: F. Jacquemard), VERIMAG (responsable: Y. Lakhnech).

ROSSIGNOL focuses on the foundations of Security Protocols. The goal of this project is the development of abstract models, simple enough to be used for the definition of a comprehensible semantics for the language of security properties. In particular, the project focuses on probabilistic models.

7.1.2. Geocal: ACI-Nouvelles interfaces des mathematiques

Participants: David Baelde, Joëlle Despeyroux, Dale Miller, Alexis Saurin, Lutz Straßburger.

GeoCal collects together 10 research terms in logic, theoretical computer science, and algebraic topology. This team involves LDP (Institut de Mathématiques de Luminy), PPS (Preuves, Programmes, Systèmes, Paris VII), LIF (Laboratoire d'Informatique Fondamentale, modélisation et vérification, Marseille), LIPN (Laboratoire d'Informatique de Paris Nord), LSV (Laboratoire Spécification et Vérification, ENS Cachan), LIP (Laboratoire d'Informatique du Parallélisme, PLUME), INRIA-Futurs, Institut de Mathématiques et Modélisation de Montpellier LORIA (CALLIGRAMME), and INRIA-Sophia (MIMOSA).

7.1.3. INFER: ANR on the Theory and Application of Deep Inference

Participants: Dale Miller, Lutz Straßburger.

The ARN proposal (blanc) titled INFER: Theory and Application of Deep Inference that is coordinated by Lutz Straßburger has been accepted in September 2006. Besides Parsifal, the teams associated with this effort are represented by Francois Lamarche (INRIA-Loria) and Michel Parigot (CNRS-PPS).

7.2. Actions internationales

7.2.1. Vallauris: Integrated Action within the EGIDE/PAI PICASSO program

Participants: David Baelde, Joëlle Despeyroux, Dale Miller, Alexis Saurin, Lutz Straßburger.

The EGIDE/PAI program PICASSO aims at promoting the scientific and technological exchanges between France and Spain. The equip Parsifal is participating, within this program, to a project whose participants are INRIA Futurs (responsables: D. Miller) and the Universidad Politécnica de Madrid (responsables: James Lipton and Manuel Hermenegildo).

The main aims of our project, which we call Vallauris, are the integration of the approaches developed by the INRIA and the UPM teams to the analysis and implementation of Higher-Order Languages (both sequential and concurrent), coinductive techniques (with special emphasis on lazy features), and in the areas of code validation, proof carrying code and security. This project started January 2005.

7.2.2. Slimmer: INRIA funded international team

Participants: David Baelde, Dale Miller.

Slimmer stands for *Sophisticated logic implementations for modeling and mechanical reasoning* is an "Equipes Associées" with seed money from INRIA. This project is initially designed to bring together the Parsifal personnel and Gopalan Nadathur's Teyjus team at the University of Minnesota. Separate NSF funding for this effort has also been awards to the University of Minnesota. We also hope to expand the scope of this project to include other French and non-French sites, in particular, Marino Miculan (University of Udine) and Brigitte Pientka (McGill University).

7.2.3. Mobius: an EU Integrated Project on Proof Carrying Code

Participants: Vivek Nigam, Olivier Delande, Joëlle Despeyroux, Dale Miller.

Mobius stands for "Mobility, Ubiquity and Security" and is a Proposal for an Integrated Project in response to the call FP6-2004-IST-FETPI. This proposal involve numerous site in Europe and was awarded in September 2005. This large, European based project is coordinated out of INRIA-Sophia.

7.2.4. TYPES: coordinated action from IST

Participants: Dale Miller, Joelle Despeyroux.

TYPES is an European project (a coordination action from the IST program) aiming at developing the technology of formal reasoning based on type theory. The project brings together 36 universities and research centers from 8 European countries (France, Italy, Germany, Netherlands, United Kingdom, Sweden, Poland and Estonia). It is the continuation of a number of European projects since 1992. The funding from the present project enables the maintaining of collaboration within the community by supporting an annual workshop, a few smaller thematic workshops, one summer school, and visits of researchers to one another's labs.

7.2.5. Amadeus: PAI on the "Realm of Cut Elimination"

Participants: Dale Miller, Lutz Straßburger.

The "PAI" Amadeus for collaboration between France and Austria has approved the grant "The Realm of Cut Elimination" in November 2006. This proposal will allow for collaborations between the Parsifal team and the groups of Agata Ciabattoni at Technische Universität Wien and Michel Parigot at CNRS-PPS.

7.2.6. Hurbert Curien: PAI Germaine De Stael "Deep Inference and the Essence of Proofs"

Participants: Dale Miller, Lutz Straßburger.

Funding to explore some questions in structure and identity of proofs. People involved from the Paris area: Lutz Straßburger, Dale Miller, Alexis Saurin, David Baelde, Michel Parigot, Stéphane Lengrand, Séverine Maingaud. People from Bern: Kai Brünnler, Richard McKinley, Phiniki Stouppa.

8. Dissemination

8.1. Services to the Scientific Community

8.1.1. Organization of Workshops

Lutz Strassburger and Dale Miller organized the Workshop on Logic Programming and Concurrency, at CIRM, Luminy, in Marseilles. This workshop was part of the Winter School on Geometry of Computation (Geocal'06) held in Luminy from January 30 to March 3.

8.1.2. Editorial activity

Dale Miller has the following editorial duties.

- *Theory and Practice of Logic Programming* Member of Advisory Board since 1999. Cambridge University Press.
- *ACM Transactions on Computational Logic (ToCL)* Area editor for *Proof Theory* since 1999. Published by ACM.
- *Journal of Functional and Logic Programming*. Permanent member of the Editorial Board since 1996. MIT Press.
- *Journal of Logic and Computation.* Associate editor since 1989. Oxford University Press.

8.1.3. Participation in program committees

Dale Miller was a program committee member for the following conferences.

- FSTTCS'06: Foundations of Software Technology and Theoretical Computer Science, Kolkata, India. 13-15 December.
- LPAR-13: 13th International Conference on Logic for Programming Artificial Intelligence and Reasoning, Phnom Penh, Cambodia. 13-17 November.
- LFMTP'06: Workshop on Logical Frameworks and Meta-Languages: Theory and Practice, 16 August.
- TFIT'06: Taiwanese-French Conference on Information Technology, Nancy, France. 28-30 March.
- Geocal Workshop on Logic Programming and Concurrency, 27 February - 3 March, CIRM, Luminy, France (Program Committee co-Chair)

8.1.4. Invited talks at Conferences or Workshops

Dale Miller gave an invited talk at IJCAR during the FLoC meeting in Seattle during 16-21 August. He also gave talks at two workshops: *Geometria della Logica* (Facoltà di Lettere e Filosofia, Università di Roma Tre, 28-29 April) and *Small Workshop on Deep Inference* (Paris, December 1 and 2).

8.2. Teaching

Dale Miller teaches regularly at École Polytechnique. In 2006, he taught a "petite classes" for INF 431 (Informatique Fondamentale - Spring 2003, 2004, 2005) and for INF 542 (Théorie des automates, langages formels, calculabilité - Fall 2003). Miller was also in charge of teaching INF 542 (Automates, Langages, Logique, and Calculabilité). Miller and Jouannaud shared the teaching of the course they designed for "Majeure 2" titled INF 585: Logics for Computer Science.

Dale Miller also co-teaches the course "Logique Linéaire et paradigmes logiques du calcul" in the new masters program MPRI (Master Parisien de Recherche en Informatique) (2004, 2005, 2006).

Dale Miller also served as external evaluator for the following PhD exams: Stéphane Lengrand (from Université Paris VII & University of St Andrews); Nicolas Oury (LRI, University of Paris Sud); and James Brotherston (LFCS, University of Edinburgh).

Lutz Straßburger taught a course on "Proof Nets and the Identity of Proofs" at ESSLLI'06, Malaga, Spain in August 2006.

Dale Miller gave five lectures at MPRI in the class 2-1 "Logique linéaire et paradigmes logiques du calcul" in January and February.

Dale Miller taught a short graduate course on "Proof Search and Computation" at the IT University of Copenhagen from 31 May 2006 to 2 June 2006.

8.3. Popular writing

Team members have also written more popular or more accessible presentations of their work. Lutz Straßburger lecture notes for his course on "Proof Nets and the Identity of Proofs" at ESSLLI'06 are available as technical report [15]. Dale Miller and Roberto Di Cosmo were invited to contribute an article on linear logic to the Stanford Encyclopedia of Philosophy [8]. Dale Miller was invited to contibute a personal and historic account of the "early days" of logic programming. His article appeared in the newsletter of the Association for Logic programming [17].

8.4. Internship supervision

From 1/03/2006, William Audry (École Normale Supérieure) was an intern in the Parsifal team. He has worked on the theme: Non-commutative logics and operational semantics. Andrew Gacek, a PhD student from the University of Minnesota took a three month internship during the summer of 2006 to work on the Bedwyr system.

9. Bibliography

Major publications by the team in recent years

[1] J. DESPEYROUX, P. LELEU. *Recursion over Objects of Functional Type*, in "Special issue of MSCS on "Modalities in Type Theory"", vol. 11, n^{o} 4, August 2001.

[2] J. DESPEYROUX, F. PFENNING, C. SCHÜRMANN. *Primitive Recursion for Higher-Order Abstract Syntax*, in "Theoretical Computer Science (TCS)", vol. 266, n^{o} 1-2, September 2001, p. 1–57.

[3] R. MCDOWELL, D. MILLER. *Reasoning with Higher-Order Abstract Syntax in a Logical Framework*, in "ACM Transactions on Computational Logic", vol. 3, n^{o} 1, January 2002, p. 80-136, http://www.lix.polytechnique.fr/Labo/Dale.Miller/papers/mcdowell01.pdf.

[4] R. MCDOWELL, D. MILLER, C. PALAMIDESSI. *Encoding transition systems in sequent calculus*, in "Theoretical Computer Science", vol. 294, n^{o} 3, 2003, p. 411-437, http://www.lix.polytechnique.fr/Labo/Dale.Miller/papers/tcs97.pdf.

[5] D. MILLER, A. TIU. *A Proof Theory for Generic Judgments: An extended abstract*, in "Proc. 18th IEEE Symposium on Logic in Computer Science (LICS 2003)", P. KOLAITIS (editor). , IEEE, June 2003, p. 118-127, http://www.lix.polytechnique.fr/Labo/Dale.Miller/papers/lics03.pdf.

[6] D. MILLER, A. TIU. *A proof theory for generic judgments*, in "ACM Trans. on Computational Logic", vol. 6, n^{o} 4, October 2005, p. 749–783.

[7] A. TIU, G. NADATHUR, D. MILLER. *Mixing Finite Success and Finite Failure in an Automated Prover*, in "Proceedings of ESHOL'05: Empirically Successful Automated Reasoning in Higher-Order Logics", December 2005, p. 79 - 98.

Year Publications

Articles in refereed journals and book chapters

[8] R. DICOSMO, D. MILLER. *Linear Logic*, in "The Stanford Encyclopedia of Philosophy", E. N. ZALTA (editor). , 2006, http://plato.stanford.edu/archives/fall2006/entries/logic-linear/.

[9] F. LAMARCHE, L. STRASSBURGER. *From Proof Nets to the Free *-Autonomous Category*, in "Logical Methods in Computer Science", vol. 2, n^{o} 4:3, 2006, p. 1–44, http://arxiv.org/cs.LO/0605054.

Publications in Conferences and Workshops

[10] G. T. LEAVENS, J.-R. ABRIAL, D. BATORY, M. BUTLER, A. COGLIO, K. FISLER, E. HEHNER, C. JONES, D. MILLER, S. PEYTON-JONES, M. SITARAMAN, D. R. SMITH, A. STUMP. *Roadmap for Enhanced Languages and Methods to Aid Verification*, in "Fifth International Conference on Generative Programming and Component Engineering (GPCE)", ACM, October 2006, p. 221-235.

[11] D. MILLER. *Collection analysis for Horn clause programs*, in "Proceedings of PPDP 2006: 8th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming", July 2006, p. 179 - 188, http://www.lix.polytechnique.fr/Labo/Dale.Miller/papers/ppdp06.pdf.

[12] D. MILLER. *Representing and reasoning with operational semantics*, in "Proceedings of IJCAR: International Joint Conference on Automated Reasoning", U. FURBACH, N. SHANKAR (editors). , LNAI, vol. 4130, August 2006, p. 4-20, http://www.lix.polytechnique.fr/Labo/Dale.Miller/papers/ijcar06.pdf.

[13] L. STRASSBURGER. *What could a Boolean category be?*, in "Classical Logic and Computation 2006 (Satellite Workshop of ICALP'06)", S. VAN BAKEL (editor). , 2006, http://www.lix.polytechnique.fr/~lutz/papers/medial-kurz.pdf.

Internal Reports

[14] D. BAELDE, A. GACEK, D. MILLER, G. NADATHUR, A. TIU. *A User Guide to Bedwyr*, November 2006.

[15] L. STRASSBURGER. *Proof Nets and the Identity of Proofs*, Research Report, n^{o} 6013, INRIA, 10 2006, https://hal.inria.fr/inria-00107260.

Miscellaneous

[16] C. LIANG, D. MILLER. *On focusing and polarities in linear logic and intuitionistic logic*, Unpublished report, December 2006, http://www.lix.polytechnique.fr/Labo/Dale.Miller/papers/focusli.pdf.

[17] D. MILLER. *Logic and Logic Programming: A Personal Account*, Vol. 19, No. 1, February 2006, http://www.cs.kuleuven.ac.be/~dtai/projects/ALP/newsletter/feb06/nav/articles/Dale/dale.pdf, ALP Newsletter.

References in notes

[18] J.-M. ANDREOLI, R. PARESCHI. *Linear Objects: Logical Processes with Built-In Inheritance*, in "Proceeding of the Seventh International Conference on Logic Programming, Jerusalem", May 1990.

[19] J. DESPEYROUX. *A Higher-order specification of the π-calculus*, in "Proc. of the IFIP International Conference on Theoretical Computer Science, IFIP TCS'2000, Sendai, Japan, August 17-19, 2000.", August 2000.

[20] J. DESPEYROUX, A. FELTY, A. HIRSCHOWITZ. *Higher-order abstract syntax in Coq*, in "Second International Conference on Typed Lambda Calculi and Applications", April 1995, p. 124–138.

[21] J. DESPEYROUX, A. HIRSCHOWITZ. *Higher-order abstract syntax with induction in Coq*, in "Fifth International Conference on Logic Programming and Automated Reasoning (LPAR)", June 1994, p. 159–173.

[22] J. DESPEYROUX, P. LELEU. *Metatheoretic Results for a Modal lambda-Calculus*, in "Journal of Functional and Logic Programming (JFLP)", vol. 2000, n^{o} 1, January 2000.

[23] J. DESPEYROUX, P. LELEU. *Recursion over Objects of Functional Type*, in "Special issue of MSCS on "Modalities in Type Theory"", vol. 11, n^{o} 4, August 2001.

[24] J. DESPEYROUX, P. LELEU. *Primitive recursion for higher-order abstract syntax with dependant types*, in "Informal proceedings of the FLoC'99 IMLA Workshop on Intuitionistic Modal Logics and Applications", June 1999.

[25] J. DESPEYROUX, F. PFENNING, C. SCHÜRMANN. *Primitive Recursion for Higher-Order Abstract Syntax*, in "Theoretical Computer Science (TCS)", vol. 266, n^{o} 1-2, September 2001, p. 1–57.

[26] F. FAGES, P. RUET, S. SOLIMAN. *Phase semantics and verification of concurrent constraint programs*, in "Symposium on Logic in Computer Science", V. PRATT (editor). , IEEE, July 1998.

[27] G. GENTZEN. *Investigations into Logical Deductions*, in "The Collected Papers of Gerhard Gentzen, Amsterdam", M. E. SZABO (editor). , North-Holland Publishing Co., 1969, p. 68-131.

[28] J.-Y. GIRARD. *A Fixpoint Theorem in Linear Logic*, An email posting to the mailing list linear@cs.stanford.edu, February 1992.

[29] M. GORDON. *HOL: A Machine Oriented Formulation of Higher-Order Logic*, Technical report, n^{o} 68, University of Cambridge, July 1985.

[30] L. HALLNÄS, P. SCHROEDER-HEISTER. *A Proof-Theoretic Approach to Logic Programming. II. Programs as definitions*, in "Journal of Logic and Computation", vol. 1, n^{o} 5, October 1991, p. 635-660.

[31] J. HODAS, D. MILLER. *Logic Programming in a Fragment of Intuitionistic Linear Logic*, in "Information and Computation", vol. 110, n^{o} 2, 1994, p. 327-365.

[32] J. S. HODAS, N. TAMURA. *lolliCop — A Linear Logic Implementation of a Lean Connection-Method Theorem Prover for First-Order Classical Logic*, in "Proceedings of IJCAR: International Joint Conference on Automated Reasoning", R. GORÉ, A. LEITSCH, T. NIPKOW (editors). , LNCS, n^{o} 2083, 2001, p. 670-684.

[33] M. HOFMANN. *Semantical analysis of higher-order abstract syntax*, in "14th Annual Symposium on Logic in Computer Science", IEEE Computer Society Press, 1999, p. 204-213.

[34] A. LISITSA. *λleanTAP: Lean Deduction in λProlog*, Technical report, n^{o} ULCS-03-017, University of Liverpool, Department of Computer Science, 2003.

[35] R. MCDOWELL. *Reasoning in a Logic with Definitions and Induction*, Ph. D. Thesis, University of Pennsylvania, December 1997.

[36] R. MCDOWELL, D. MILLER. *Cut-elimination for a logic with definitions and induction*, in "Theoretical Computer Science", vol. 232, 2000, p. 91-119.

[37] R. MCDOWELL, D. MILLER. *A Logic for Reasoning with Higher-Order Abstract Syntax*, in "Proceedings, Twelfth Annual IEEE Symposium on Logic in Computer Science, Warsaw, Poland", G. WINSKEL (editor). , IEEE Computer Society Press, July 1997, p. 434-445.

[38] D. MILLER. *Forum: A Multiple-Conclusion Specification Language*, in "Theoretical Computer Science", vol. 165, n^{o} 1, September 1996, p. 201–232.

[39] D. MILLER, A. SAURIN. *A game semantics for proof search: Preliminary results*, in "Proceedings of the Mathematical Foundations of Programming Semantics (MFPS)", 2005, http://www.lix.polytechnique.fr/Labo/Dale.Miller/papers/mfps05.pdf.

[40] D. MILLER, A. TIU. *A Proof Theory for Generic Judgments: An extended abstract*, in "Proc. 18th IEEE Symposium on Logic in Computer Science (LICS 2003)", IEEE, June 2003, p. 118-127, http://www.lix.polytechnique.fr/Labo/Dale.Miller/papers/lics03.pdf.

[41] A. MOMIGLIANO, A. TIU. *Induction and Co-induction in Sequent Calculus*, in "Post-proceedings of TYPES 2003", S. BERARDI, M. COPPO, F. DAMIANI (editors). , LNCS, n^{o} 3085, January 2003, p. 293 - 308, http://www.lix.polytechnique.fr/Labo/Alwen.Tiu/linc.pdf.

[42] G. NADATHUR, D. MILLER. *An Overview of λProlog*, in "Fifth International Logic Programming Conference, Seattle", MIT Press, August 1988, p. 810-827.

[43] G. NADATHUR, F. PFENNING. *The type system of a higher-order logic programming language*, in "Types in Logic Programming", F. PFENNING (editor). , MIT Press, 1992, p. 245-283.

[44] L. C. PAULSON. *Isabelle: The Next 700 Theorem Provers*, in "Logic and Computer Science", P. ODIFREDDI (editor). , Academic Press, 1990, p. 361-386.

[45] C. SCHÜRMANN, F. PFENNING. *A Coverage Checking Algorithm for LF*, in "Theorem Proving in Higher Order Logics: 16th International Conference, TPHOLs 2003", LNCS, vol. 2758, Springer-Verlag, 2003, p. 120-135.

[46] C. SCHÜRMANN. *Automating the Meta Theory of Deductive Systems*, CMU-CS-00-146, Ph. D. Thesis, Carnegie Mellon University, October 2000.

[47] A. TIU, D. MILLER. *A Proof Search Specification of the π-Calculus*, in "3rd Workshop on the Foundations of Global Ubiquitous Computing", ENTCS, vol. 138, September 2004, p. 79-101.

[48] A. TIU. *A Logical Framework for Reasoning about Logical Specifications*, Ph. D. Thesis, Pennsylvania State University, May 2004, http://www.lix.polytechnique.fr/Labo/Alwen.Tiu/etd.pdf.

[49] A. TIU. *Level 0/1 Prover: A tutorial*, Available online., September 2004.

[50] A. TIU. *Model Checking for π-Calculus Using Proof Search*, in "CONCUR", M. ABADI, L. DE ALFARO (editors). , Lecture Notes in Computer Science, vol. 3653, Springer, 2005, p. 36-50.

[51] A. ZIEGLER, D. MILLER, C. PALAMIDESSI. *A congruence format for name-passing calculi*, in "Proceedings of SOS 2005: Structural Operational Semantics, Lisbon, Portugal", ENTCS, Elsevier Science B.V., July 2005, http://www.lix.polytechnique.fr/parsifal/ziegler05report.pdf.