INRIA

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

Staff member INRIA
Joëlle Despeyroux [CR. INRIA-Sophia. Since 1/9/2005.]
Lutz Straßburger [CR. Since 1/12/2005.]

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.]
Axelle Ziegler [Allocataire École Normale Supérieure, Paris. Joint with Comète. Since 1/10/2004.]

Post-doctoral fellow
Tom Chothia [Post Doctorant CNRS. From 1/9/2004 until 20/12/2005. Joint with Comète and TANC.]

Invited researchers
Claudia Faggian [Professor, Padova University. During May, June, and November 2005.]
Andrew Gacek [PhD student, University of Minnesota. From 26/09/05 until 30/09/05.]
James Lipton [Professor at Wesleyan University. From 05/12/2005 until 07/12/2005.]
Brigitte Pientka [Professor, McGill University. From 2/05/2005 until 6/05/2005]
Elaine Pimentel [Assoc. Prof., Univ. Federal de Minas Gerais, Brazil. From 9/12/2005 until 29/12/2005.]

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 deemphasized.
- 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 [33] and [31].

- 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) [24].

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 [17], Lolli of Hodas and Miller [30], and Forum of Miller [39].

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 [36] and reachability within progress calculi, such as the π-calculus [11], [45], [48], [15].

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, Axelle Ziegler.

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 [28] use classical higher-order logic while systems such as Isabelle/ZF [42] 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 [46] 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 [25]. 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 [29] and, independently, by Girard [27]. Later McDowell, Miller, and Tiu have made substantial extensions to our understanding of this concept [35], [36], [40]. Tiu and Momigliano [41], [46] 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 [34], [37], [36], it became clear that when the object-logic supports λ-tree syntax, the *generic judgment* [40], [11] 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 [11] .

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 [1].

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 [19], [18], 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 [20], [23], [21] 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 [20], [21] to dependent types was given in [22]: 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 [44], [43].

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 [19], [20], [21] and by Hofmann at the model level [32]. There is also the work by Miller and McDowell [35] 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 [38] and game semantic [12] that allows for the natural duality of finite success and finite failure.

Given its bases on proof theory, an intensional treatment of syntax and bindings in syntax was natural to develop [11]. The design of a working system, Level 0/1, that exploits these foundational ideas is presented in [47], [14]. The resulting system is one of the few systems that can directly implement bisimulation checking for the π-calculus [48], [45].

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 [36]. The meta-logical framework reported in that paper has been greatly extended by Miller and Tiu [11], [46]. 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. Level 0/1

Alwen Tiu and Gopalan Nadathur were members of the Parsifal team during various parts of 2004. During their visits, the team developed a new architecture for automatically computing with expressions involving bound variables within fixed points of finite processes. The system was implemented in Standard ML and is called *Level 0/1*. This system has been used to do model checking style deduction, search for winning strategies in game playing, and bisimulation checking for the π-calculus. The system is described in [14] and is available from the Web.

Recently David Baelde and Axelle Ziegler have reimplemented this system in Ocaml. Extensions and improvements with this system are planned for the Ocaml version.

6. New Results

6.1. The LINC meta-logic

A new meta-logic that plays a central role in the Parsifal plans to reason about operational semantics is the LINC logic. This logic was presented in the PhD thesis of Alwen Tiu [46]: part of it is also presented in [11] and [41]. The major results reported have been the cut-elimination theorem for LINC (hence, its consistency) and the development of numerous examples that indicate that this logic can serve the central purpose that we have planned for it.

6.2. Format rules for mobility

Given that operational semantic specifications can be given elegantly within LINC, it has been possible to provide generalize existing rule format rules, such as the tyft/tyxt format, that capture binder mobility in process calculus. As a consequence of her internship (joint between Parsifal and Comète), Axelle Ziegler developed just such a format rule [15]. As a result, open bisimulation for a wide range of process calculi can be inference automatically from rules determined to be of this extended format style.

6.3. A game semantics for proof search

An exciting area of foundational work is to establish connections between the proof search paradigm and the recent work on *game semantics*, on *ludics* [26], and the general and rather old notions of *logic and interaction*. Using these ideas might allow us to take a neutral approach to proof and refutation: that is, one does some kind of computation with logical expressions and when finished, one finds out if one has a proof or a refutation. For example, consider asking Prolog the query G: if Prolog returns saying "yes", we can claim that Prolog has found a proof. If Prolog returns saying "no", then we would normally say that it has failed to find a proof of G. In fact, given some insights from LINC, it is, in fact, possible to say in the latter case (that is, in the finite failure case), that Prolog actually constructed a proof of $\neg G$. This fundamental duality between success and failure has always been a challenge to traditional proof search and proof theory. Alexis Saurin and Miller [12] have provided a game theoretic foundations for various interesting and computationally useful subsets of linear logic. For example, proof search with the usual specification for bisimulation is assigned the usual game theoretical semantics.

6.4. Specifications for sequent calculus

Elaine Pimentel (an occasional visitor to Parsifal) and Miller have been developing means of specifying object-level logics in linear logic is such a way that the meta-theory of linear logic can be used to infer properties of the specified object-logic. In particular, they have presented sufficient conditions that guarantee that cuts and non-atomic initial rules can be eliminated [13].

6.5. Processes-as-formulas

Since the advent of linear logic in 1987, numerous computer scientists have been working at trying to map process calculus directly into formulas of linear logic. Such encoding help to reveal surprisingly close relationships between what relations are possible in logic and what is important in operational semantics.

A recent addition to this linear of work was the internship report of David Baelde [16]. He should there that the various forms of reachability in asynchronous processes calculi can be encoded naturally in proof theory. In particular, he provided an encoding of the join calculus into linear logic and showed how to motivate CCS style prefixing using a non-canonical modal operator used for synchronization.

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, Axelle Ziegler.

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).

7.2. 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, Axelle Ziegler.

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, Axelle Ziegler.

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: David Baelde, 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.

8. Dissemination

8.1. Services to the Scientific Community

8.1.1. Organization of Workshops

Miller helped to organize WIPS 2005 (Workshop on Implementations of Proof Search) held at the University of Minnesota on 1 July 2005 and MoveLog 2005 (Workshop on Mobile Code Safety and Program Verification Using Computational Logic Tools) held at Barcelona, 5 October.

8.1.2. Editorial activity

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

Despeyroux was a program committee member for Merlin 2005 (Workshop on MEchanized Reasoning about Languages with varIable biNding).

Miller was a program committee member for the following conferences.

- Merlin 2005: Workshop on MEchanized Reasoning about Languages with varIable biNding.
- LPAR 2005: 11th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning. Montevideo, Uruguay, 14-18 March.
- CSL 2005: 14th Annual Conference of the European Association for Computer Science Logic, 22-25 August, Oxford, UK.
- CADE 2005: Conference on Automated Deduction, Tallinn, Estonia, 22-27 July.

8.1.4. Invited talks at Conferences or Workshops

Miller has been invited to speak at the following meetings.

- Algebraic Process Calculus: The first 25 years and beyond, Bertinoro, Italy, 5 August 2005.
- Structure and Deduction 2005: an ICALP workshop, Lisbon, 16-17 July 2005.
- 2nd Taiwanese-French Conference in Information Technologies, Tainan, Taiwan, 23-25 March 2005.

8.2. Teaching

Miller teaches regularly at École Polytechnique. In 2005, he taught two “petite classes”; one for INF 542 (Théorie des automates, langages formels, calculabilité) and one for INF 431 (Informatique Fondamentale - Spring 2003, 2004, 2005) and for INF 542 (Théorie des automates, langages formels, calculabilité - Fall 2003). Miller and Jouannaud taught in new course they designed for “Majeure 2” titled INF 585: Logics for Computer Science.

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).

8.3. Internship supervision

From 1/04/2005 until 30/9/2005, David Baelde (MPRI, École Normale Supérieure, Lyon) was an intern in the Parsifal team.

9. Bibliography

Major publications by the team in recent years

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] D. MILLER. *Forum: A Multiple-Conclusion Specification Language*, in "Theoretical Computer Science", vol. 165, nº 1, September 1996, p. 201–232.

[9] 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.

[10] 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, http://www.lix.polytechnique.fr/Labo/Dale.Miller/papers/fguc04workshop.pdf.

Articles in refereed journals and book chapters

[11] D. MILLER, A. TIU. *A proof theory for generic judgments*, in "ACM Transactions on Computational Logic", vol. 6, nº 4, October 2005, p. 749–783, http://www.lix.polytechnique.fr/Labo/Dale.Miller/papers/tocl-nabla.pdf.

Publications in Conferences and Workshops

[12] 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.

[13] E. PIMENTEL, D. MILLER. *On the specification of sequent systems*, in "LPAR 2005: 12th International Conference on Logic for Programming, Artificial Intelligence and Reasoning", LNAI, nº 3835, 2005, p. 352-366, http://www.lix.polytechnique.fr/Labo/Dale.Miller/papers/lpar05.pdf.

[14] 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.

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

Internal Reports

[16] D. BAELDE. *Logique linéaire et algèbre de processus*, Technical report, INRIA Futurs, LIX and ENS, 2005, http://www.lix.polytechnique.fr/parsifal/baelde05stage.ps.

Bibliography in notes

[17] 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.

[18] 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.

[19] 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.

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

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

[22] 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.

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

[24] 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.

[25] 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.

[26] J.-Y. GIRARD. *Locus solum*, in "Mathematical Structures in Computer Science", vol. 11, n° 3, June 2001, p. 301-506.

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

[28] M. GORDON. *HOL: A Machine Oriented Formulation of Higher-Order Logic*, Technical report, n° 68, University of Cambridge, July 1985.

[29] 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° 5, October 1991, p. 635-660.

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

[31] 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° 2083, 2001, p. 670-684.

[32] 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.

[33] A. LISITSA. *λleanTAP: Lean Deduction in λProlog*, Technical report, n° ULCS-03-017, University of Liverpool, Department of Computer Science, 2003.

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

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

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

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

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

[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° 3085, January 2003, p. 293 - 308, http://www.lix.polytechnique.fr/Labo/Alwen.Tiu/linc.pdf.

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

[43] 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.

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

[45] 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, http://www.lix.polytechnique.fr/Labo/Dale.Miller/papers/fguc04workshop.pdf.

[46] 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.

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

[48] 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.