INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Project-Team Parsifal

Preuves Automatiques et Raisonnement sur des SpécIFicAtions Logiques

Futurs

THEME SYM

Activity Report

2007

Table of contents

1. Team

Parsifal is an INRIA-Futurs project-team located at the CNRS laboratory LIX (Laboratoire d'Informatique de l'Ecole Polytechnique). Two additional people join the team at the end 2007: Stephane Lengrand, CNRS CR2 (1 January 2008) and Matteo Capelletti INRIA post doc (1 December 2007).

Head of project-team
Dale Miller [DR INRIA-Futurs]

Research scientist
Joëlle Despeyroux [CR INRIA-Sophia]
Lutz Straßburger [CR INRIA-Futurs]

Administrative assistant
Isabelle Biercewicz

PhD student
David Baelde [Allocataire École Normale Supérieure, Lyon, since 1 September 2005]
Olivier Delande [Bourse Monge École Polytechnique, since 1 October 2006]
Vivek Nigam [Allocataire Mobius, since 1 October 2006]
Alexis Saurin [Allocataire École Normale Supérieure, Paris, since 1 October 2004]

Post-doctoral fellow
Kaustuv Chaudhuri [Post Doctorant INRIA, 1 Nov 2006 - 31 October 2007]
Laurent Méhats [Post Doctorant INRIA, since 1 September 2007]

Invited researchers
Chuck Liang [Associate Professor, Hofstra University, NY, USA, 1 - 30 June 2007]

Student intern
Nicolas Guenot [M2 MPRI, 5 March 2007 - 31 August 2007]
Zachery Snow [MS student from the University of Minnesota, 15 May - 31 July 2007.]

2. Overall Objectives

2.1. Main Themes

The aim of the Parsifal team is to develop and exploit the theories of proofs and types to support the specification and verification of computer systems. To achieve these goals, the team works on several level.

- The team has expertise in *proof theory* and *type theory* and conducts basic research in these fields: in particular, the team is developing results that help with the automation of deduction and with the formal manipulation and communication of proofs.
- Based on experience with computational systems and theoretical results, the team *designs* new logical principles, new proof systems, and new theorem proving environments.
- Some of these new designs are appropriate for *implementation* and the team works at developing prototype systems to help validate basic research results.
- By using the implemented tools, the team can develop examples of specifications and verification to test the success of the design and to help suggest new logical and proof theoretic principles that need to be developed in order to improve ones ability to specify and verify.

The foundational work of the team focuses on the proof theory of classical, intuitionistic, and linear logics making use, primarily, of sequent calculus and deep inference formalisms. A major challenge for the team is the reasoning about computational specifications that are written in a relational style: this challenge is being addressed with the introduction of some new approaches to dealing with induction, co-induction, and generic judgments. Another important challenge for the team is the development of normal forms of deduction: such normal forms can be used to greatly enhance the automation of search (one only needs to search for normal forms) and for communicating proofs (and proof certificates) for validation.

The principle application areas of concern for the team currently are in functional programming (e.g., λ-calculus), concurrent computation (e.g., π-calculus), interactive computations (e.g., games), and biological systems.

2.2. Highlights of the year

The team has released the Bedwyr model checker that can be used to work with linguistic expressions involving bound variables in a completely declarative fashion. The system comes with several examples, including a declarative and direct implementation of bisimulation checking for the finite π-calculus. The team has also produced a number of papers on extending and applying the important notion of *focusing proofs*.

3. Scientific Foundations

3.1. General Overview

Keywords: *proof normalization*, *proof search*.

In the specification of computational systems, logics are generally used in one of two approaches. In the *computation-as-model* approach, computations are encoded as mathematical structures, containing such items as nodes, transitions, and state. Logic is used in an external sense to make statements *about* those structures. That is, computations are used as models for logical expressions. Intensional operators, such as the modals of temporal and dynamic logics or the triples of Hoare logic, are often employed to express propositions about the change in state. This use of logic to represent and reason about computation is probably the oldest and most broadly successful use of logic in computation.

The *computation-as-deduction* approach, uses directly pieces of logic's syntax (such as formulas, terms, types, and proofs) as elements of the specified computation. In this much more rarefied setting, there are two rather different approaches to how computation is modeled.

The *proof normalization* approach views the state of a computation as a proof term and the process of computing as normalization (know variously as β-reduction or cut-elimination). Functional programming can be explained using proof-normalization as its theoretical basis [38] and has been used to justify the design of new functional programming languages [20].

The *proof search* approach views the state of a computation as a sequent (a structured collection of formulas) and the process of computing as the process of searching for a proof of a sequent: the changes that take place in sequents capture the dynamics of computation. Logic programming can be explained using proof search as its theoretical basis [45] and has been used to justify the design of new logic programming languages [44].

The divisions proposed above are informal and suggestive: such a classification is helpful in pointing out different sets of concerns represented by these two broad approaches (reductions, confluence, etc, versus unification, backtracking search, etc). Of course, a real advance in computation logic might allow us merge or reorganize this classification.

Although type theory has been essentially designed to fill the gap between these two kinds of approaches, it appears that each system implementing type theory up to now only follows one of the approaches. For example, the Coq system implementing the Calculus of Inductive Constructions (CIC) uses proof normalization while the Twelf system [54], implementing the Edinburgh Logical Framework (LF, a sub-system of CIC), follows the proof search approach (normalization appears in LF, but it is much weaker than in, say, CIC).

The Parsifal team works on both the proof normalization and proof search approaches to the specification of computation.

3.2. Reasoning about logic specifications

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

Once a computational system (e.g., a programming language, a specification language, a type system) is given a logic (relational) specifications, how do we reason about the formal properties of such specifications? New results in proof theory are being developed to help answer this question.

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 [32] use classical higher-order logic while systems such as Isabelle/ZF [53] 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 is obfuscated.

Despeyroux, Pfenning, Leleu, and Schürmann proposed two different type theories [2], [1] based on modal logic in which expressions (possibly with binding) live in the functional space $A \rightarrow B$ while general functions (for case and iteration reasoning) live in the full functional space $\Box A \rightarrow B$. These works give a possible answer to the problem of extending the Edinburgh Logical Framework, well suited for describing expressions with binding, with recursion and induction principles internalized in the logic (as done in the Calculus of Inductive Constructions). However, extending these systems to dependent types seems to be difficult (see [28] where an initial attempt was given).

The LINC logic of [59] appears to be a good meta-logical setting for proving theorems about such logical specifications 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 [29]. 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* (also called *fixed points*), 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 [35] and, independently, by Girard [31]. Later McDowell, Miller, and Tiu have made substantial extensions to our understanding of this concept [40], [5], [47]. Tiu and Momigliano [48], [59] 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 [39], [41], [5], it became clear that when the object-logic supports λ-tree syntax, the *generic judgment* [47], [8] 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 [8].

3.3. Focused proof systems

Keywords: *focused proof systems*.

There is a great deal of non-determinism that is present in the search for proofs (in the sense of automated deduction). The non-determinism involved with generating lemmas is one extreme: when attempting to prove one formula it is possible to generate a potential lemma and then attempt to prove it and to use it to prove the original formula. In general, there are no clues as to what is a useful lemma to construct. The famous "cut-elimination" theorem say that it is possible to prove theorems without using lemmas (that is, by restricting to *cut-free* proofs). Of course, cut-free proofs are not appropriate for all domains of computation logic since they can be vastly larger than proofs containing cuts. Even when restricting to cut-free proofs make sense (as in logic programming, model checking, and some areas of automated reasoning), the construction of cut-free proofs still contains a great deal of non-determinism.

Structuring the non-deterministic choices within the search for cut-free proofs has received increasing attention in recent years with the development of *focusing proofs systems*. In such proof systems, there is a clear separation between non-deterministic choices for which no backtracking is required ("don't care non-determinism") and choices were backtracking may be required ("don't know non-determinism"). Furthermore, when a backtrackable choice is required, that choice actually extends over a series of inference rules, representing a "focus" during the construction of a proof. One focusing-style proof systems was developed within the early work of providing a proof-theoretic foundations for logic programming via *uniform proofs* [45]. The first comprehensive analysis of focusing proofs was done in linear logic by Andreoli [22]. There it was shown that proofs are constructed in two alternating phases: a *negative phase* in which don't-care-non-determinism is done and a *positive phase* in which a focused sequence of don't-know-non-determinism choices is applied.

Since a great deal of automated deduction (in the sense of logic programming, type inference, and model checking) is done in intuitionistic and classical logic, there is a strong need to have comprehensive focusing results for these logics as well. In linear logic, the assignment of inference rules to the positive and negative phases is canonical (only the treatment of atomic formulas is left as a non-canonical choice). Within intuitionistic and classical logic, a number of inference rules do not have canonical treatments. Instead, several focusing-style proof systems have been developed one-by-one for these logics. A general scheme for putting all of these choices together has recently been developed within the team and will be described below.

3.4. Proof structure and proof certificates

Keywords: *proof certificates*.

There has been a good deal of concern in the proof theory literature on the nature of proofs as objects. An example of such a concern is the question as to whether or not two proofs should be considered equal. Such considerations were largely of interest to philosophers and logicians. Computer scientists started to get involved with the structure of proofs more generally in essentially two ways. The first is the extraction of algorithms from constructive proofs. The second is the use of proof-like objects to help make theorem proving systems more sophisticated (proofs could be stored, edited, and replayed, for example).

It was not until the development of the topic of *proof carrying code (PCC)* that computer scientists from outside the theorem proving discipline took a particular interesting in having proofs as actual data structure within computations. In the PCC setting, proofs of safety properties need to be communicated from one host to another: the existence of the proof means that one does not need to trust the correctness of a piece of software (for example, that it is not a virus). Given this need to produce, communicate, and check proofs, the actual structure and nature of proofs-as-objects becomes increasingly important.

Often the term *proof certificate* (or just *certificate*) is used to refer to some data structure that can be used to communicate a proof so that it can be checked. A number of proposals have been made for possible structure of such certificates: for examples, proof scripts in theorem provers such as Coq are frequently used. Other notions include oracle [52] and fixed points [21].

The earliest papers on PCC made use of logic programming systems Twelf [51] and λProlog [23]. It seems that the setting of logic programming is a natural one for exploring the structure of proofs and the trade-offs between proof size and the need for run-time proof search. For example, there is a general trade-off between the size of a proof object and the amount of search one must do to verify that an object does, in fact, describe a proof. Exploring such trade-off should be easy and natural in the proof search setting where such search is automated. In particular, focused proof systems should be a large component of such an analysis.

3.5. Deep Inference and Categorical Axiomatizations

Keywords: *category theory*, *deep inference*.

Deep inference [33], [34] is a novel methodology for presenting deductive systems. Unlike traditional formalisms like the sequent calculus, it allows rewriting of formulas deep inside arbitrary contexts. The new freedom for designing inference rules creates a richer proof theory. For example, for systems using deep inference, we have a greater variety of normal forms for proofs than in sequent calculus or natural deduction systems. Another advantage of deep inference systems is the close relationship to categorical proof theory. Due to the deep inference design one can directly read off the morphism from the derivations. There is no need for a counterintuitive translation.

One reason for using categories in proof theory is to give a precise algebraic meaning to the identity of proofs: two proofs are the same if and only if they give rise to the same morphism in the category. Finding the right axioms for the identity of proofs for classical propositional logic has for long been thought to be impossible, due to “Joyal’s Paradox”. For the same reasons, it was believed for a long time that it it not possible to have proof nets for classical logic. Nonetheless, Lutz Straßburger and François Lamarche provided proof nets for classical logic in [3], and analyzed the category theory behind them in [36]. In [9] and [57], one can find a deeper analysis of the category theoretical axioms for proof identification in classical logic. Particular focus is on the so-called *medial rule* which plays a central role in the deep inference deductive system for classical logic.

The following research problems are investigated by members of the Parsifal team:

- Find deep inference system for richer logics. This is necessary for making the proof theoretic results of deep inference accessible to applications as they are described in the previous sections of this report.
- Investigate the possibility of focusing proofs in deep inference. As described before, focusing is a way to reduce the non-determinism in proof search. However, it is well investigated only for the sequent calculus. In order to apply deep inference in proof search, we need to develop a theory of focusing for deep inference.
- Use the results on deep inference to find new axiomatic description of categories of proofs for various logics. So far, this is well understood only for linear and intuitionistic logics. Already for classical logic there is no common accepted notion of proof category. How logics like LINC can be given a categorical axiomatisation is completely open.

3.6. Proof Nets and Combinatorial Characterization of Proofs

Keywords: *cographs*, *correctness criteria*, *proof nets*, *relation webs*.

Proof nets are abstract (graph-like) presentations of proofs such that all "trivial rule permutations" are quotiented away. More generally, we investigate combinatoric objects and correctness criteria for studying proofs independently from syntax. Ideally the notion of proof net should be independent from any syntactic formalism. But due to the almost absolute monopoly of the sequent calculus, most notions of proof nets proposed in the past were formulated in terms of their relation to the sequent calculus. Consequently we could observe features like “boxes” and explicit “contraction links”. The latter appeared not only in Girard’s proof nets [30] for linear logic but also in Robinson’s proof nets [55] for classical logic. In this kind of proof nets every link in the net corresponds to a rule application in the sequent calculus.

The concept of deep inference allows to design entirely new kinds of proof nets. Recent work by Lamarche and Straßburger [56] and [4] have extended the theory of proof nets for multiplicative linear logic to multiplicative linear logic with units. This seemingly small step—just adding the units—had for long been an open problem, and the solution was found only by consequently exploiting the new insights coming from deep inference. A proof net no longer just mimicks the seqent calculus proof tree, but rather an additional graph structure that is put on top of the formula tree (or sequent forest) of the conclusion. The work on proof nets within the team is focused on the following two directions

- Extend the work of Lamarche and Straßburger to larger fragments of linear logic, containing the additives, the exponentials, and the quantifiers.
- Finding (for classical logic) a notion of proof nets that is deductive, i.e., can effectively be used for doing proof seach. An important property of deductive proof nets must be that the correctness can be checked in linear time. For the classical logic proof nets by Lamarche and Straßburger [3] this takes exponential time (in the size of the net). We hope that eventually deductive proof nets will provide a "bureaucracy-free" formalism for proof search.

3.7. A logic for systems biology

Keywords: *biology*, *hybrid logic*, *linear logic*, *stochastic pi-calculus*, *systems biology*, *temporal logic*.

Systems in molecular biology, such as those for regulatory gene networks or protein-protein interactions, can be seen as state transition systems that have an additional notion of *rate* of change. Methods for specifying such systems is an active research area. However, to our knowledge, no logic (more powerful than the boolean logic) have been proposed so far to both specify and reason about these systems.

One current and prominent method uses process calculi, such as the stochastic π-calculus, that has a built in notion of rate [26]. Process calculi, however, have the deficiency that reasoning about the specifications is external to the specifications themselves, usually depending on simulations and trace analysis.

Kaustuv Chaudhuri and Joëlle Despeyroux are considering the problem of giving a *logical* instead of a *process-based* treatment both to specify and to reason about biological systems in a uniform linguistic framework. The logic they have proposed, called HyLL, is an extension of (intuitionistic) linear logic with a modal situated truth that may be reified by means of the $\downarrow$ operator from *hybrid logic*. A variety of semantic interpretation can be given to this logic, including the rates and the delay of formation.

The expressiveness of the logic has been demonstrated on small examples and first meta-theoretical properties of the logic have been proven. Considerable work needs to be done before this proposal succeeds as a natural logical framework for systems biology. Remaining work mainly includes the description of larger examples (requiring more specifications of usual biological notions), and automating reasoning about the specifications. It also includes further studies of the meta-theoretical properties of the logic, and of course eventual extensions of the logic (for example to get branching semantics).

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 as a logic program, which means that it is usually easy to animated such semantic specifications in direct and natural ways. Given the natural duality between finite success and finite failure (given a proof theoretic foundations in papers such as [6] and [46]) it is also possible to describe model checking systems from a proof theoretic setting.

One application area for this work is, thus, the development of model checking software that can work on linguistic expressions that may contain bound variables. Specific applications could be towards checking bisimulation of λ-calculus and π-calculus expressions.

More about a prototype model checker in this area is described below.

4.2. Mechanized metatheory

Keywords: *mechanized metatheory*.

There has been increasing interest in the international community with the use of formal methods to provide proofs of properties of programs and entire programming languages. The example of proof carrying code is one such example. Two more examples for which the team's efforts should have important applications are the following two challenges.

Tony Hoare's Grand Challenge titled "Verified Software: Theories, Tools, Experiments" has as a goal the construction of "verifying compilers" to support a vision of a world where programs would only be produced with machine-verified guarantees of adherence to specified behavior. Guarantees could be given in a number of ways: proof certificates being one possibility.

The PoplMark challenge [24] envisions "a world in which mechanically verified software is commonplace: a world in which theorem proving technology is used routinely by both software developers and programming language researchers alike." The proposers of this challenge go on to say that a "crucial step towards achieving these goals is mechanized reasoning about language metatheory." There is clearly a strong overlap in the goals of this challenge and those of part of the Parsifal team.

4.3. Biological systems

Keywords: *biology*, *systems biology*.

When one looks at systems of biochemical reactions in molecular biology, such as gene-protein and protein-protein interaction systems, one observes two basic phenomena: state change (where, for example, two or more molecules interact to form other molecules) and delay. Each of the state changes has an associated delay before the state change is observed, or, more precisely, a probability distribution over possible delays: the rate of the change. A system of biochemical reactions can therefore be seen as a stochastic computation.

The HyLL logic proposed this year by Kaustuv Chaudhuri and Joëlle Despeyroux is a first attempt at providing a logical framework for both specifying and reasoning about such computations.

5. Software

5.1. Bedwyr

Participants: Dale Miller, David Baelde [correspondant].

In order to provide some practical validation of the formal results mentioned above regarding the logic LINC and the quantifier ∇, we picked a small but expressive subset of that logic for implementation. While that subset did not involve the proof rules for induction and co-induction (which are difficult to automate) the subset did allow for model-checking style computation. During 2006 and 2007, the Parsifal team, with contributions from our close colleagues at the University of Minnesota and the Australian National University, designed and implemented the Bedwyr system for doing proof search in that fragment of LINC. This system is organized as an open source project and is hosted on INRIA's GForge server. It has been described in the conference papers [58] and [11]. This systems, which is implemented in OCaml, has been download about 200 times since it was first released.

Bedwyr is a generalization of logic programming that allows model checking directly on syntactic expressions 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.

Bedwyr has served well to validate the underlying theoretical considerations while at the same time providing a useful tool for exploring some applications. 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.

5.2. Taci

Participants: David Baelde, Dale Miller, Zachery Snow.

During the summer of 2007, Baelde (LIX PhD student) and visiting intern Zachery Snow (PhD student from the University of Minnesota) built a prototype theorem prover, called *Taci*, that we are currently using "in-house" to experiment in a number of large examples and a few different logics. We hope to make the tool available eventually once we have settled into the exact logic that it should support.

6. New Results

6.1. Reasoning about operational semantics

Participants: David Baelde, Dale Miller.

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 [43], 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 validate at least some of the design ideas presented in [43]. These results are encouraging and the team plans to develop those themes to a greater extent.

6.2. Effective implementation of model checking with linguistic expressions

Participants: David Baelde, Dale Miller, Vivek Nigam.

The effort to implement Bedwyr provided the team with a number of challenges, including dealing with how best to implement λ-terms to effectively 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 [25].

6.3. Focusing proof systems for linear, intuitionistic, and classical logics

Participants: David Baelde, Dale Miller, Alexis Saurin.

When working with proof search and logic programming within linear logic, the completeness of *focusing proofs* by Andreoli [22] provides a critical normal form for proofs in all of linear logic. Chaudhuri and his colleagues have applied the focusing proof system of linear logic to forward and backward reasoning systems. Miller and Saurin have provided a new and modular proof of focusing for linear logic in [15]. This proof allows for direct extensions: for example, the focusing result in Baelde and Miller's paper [12] were based on this new modular proof.

In contrast, a flexible and general definition of focusing proofs in intuitionistic and classical proofs have not been provided. Althought there had been a number of focusing-style proof systems that have been defined for these two logics, a general framework to relate all of them was needed. In [13], Liang and Miller provided just such a framework. In particular, the proof systems LJF and LKF were introduced that provides for a great deal of flexibility in the description of how focusing could be done in these two logics. In particular, polarities could be mixed (a result that was a challenge to get for intuitionistic logic). Many other focusing systems for intuitionistic logic could be mapped compositionally into the LJF via the simple insertion of "delay" operators (simple logical expressions that stop the focusing process).

6.4. Tables, lemmas, and focused proofs

Participants: Dale Miller, Vivek Nigam.

To help validate our energies at exploring focusing proof systems, the team has looked at various applications of such proof theoretic results. Miller and Nigam in [14] have shown how focusing proof systems can be exploited to provide a declarative approach to the use of tables in proof search (addressing the issue of lemma *reuse* instead of *reproof* in the case of atomic lemmas). They also showed how a table of lemmas can be used to give a new form of proof certificate.

6.5. Game semantics for proof search with multiplicative connectives

Participants: Olivier Delande, Dale Miller.

There appears to be a very close connection between focusing proof systems and certain kinds of game semantics for proof search. The team has been working on understanding a "neutral approach to proof and refutation". In [46], Miller and Saurin attempted to use game semantics to provide a "neutral" approach to proof and refutation. Their approach worked well for additive games (which are essentially Hintikka games). If the nature of multiplicatives were greatly restricted (to simple "guards"), then this game theoretic approach worked well again. It was unclear, however, how to extend that effort to handle general multiplicative connectives. This past year, Olivier Delande has been developing a solution to this problem that involves several innovations in the earlier notion of games. In particular, since games are no longer determinate (games might end in a draw), game playing must continue even after one player has failed (in order to find out if the other play wins or draws). A paper on this work is planned for the end of 2007.

6.6. A Proof theory of induction and co-induction

Participants: David Baelde, Dale Miller.

Systems like the model-checker Bedwyr provides properties of computational system by exploring all of its finite behaviors. As a result, most such systems cannot handle infinite state spaces and, hence, cannot handle the vast majority of computer systems. Baelde and Miller [12] [19] have been exploring a proof theory for induction and coinduction within linear logic: given the cut-elimination and focusing results that they have obtained, it should be possible to developing some effective tools to help automate proofs that require induction and coinduction: at least when the required invariants are easy to guess.

6.7. Proof nets for second order multiplicative linear logic

Participant: Lutz Straßburger.

Exploring the ideas of [4] which allowed including the units into the theory of proof nets, Lutz Straßburger developed a new theory of proof nets that also includes the quantifiers, without relying on boxes, as in Girard's original work. The results are not yet published, but available from Straßburger's webpage[1].

6.8. Combinatorial characterization of the medial rule

Participant: Lutz Straßburger.

Geometric or combinatoric correctness criteria are important for studying proofs independently from syntax. In [17], Lutz Straßburger gives such a criterion for the medial rule which is central in the deep inference presentation for classical logic [27]. This means that there are now two independent approaches towards a notion of proof identification: First, via algebraic considerations, i.e., categories [9], and second, via combinatorial or graph-theoretical considerations, i.e., proof nets and correctness criteria.

6.9. Deep inference for hybrid logics

Participant: Lutz Straßburger.

Hybrid languages are modal languages which use formulas to refer to specific points in a model. There is a fast growing community because of many applications. In [18] Lutz Straßburger presents a deep inference deductive system for hybrid logic. Thus, the rich proof theory related to deep inference is made available for hybrid logics, which so far have mainly been studied via model theory. The HyLL logic proposed by Kaustuv Chaudhuri and Joëlle Despeyroux for systems biology (see below) also provides a proof theoretic presentation of hybrid logics, in a traditional natural and sequent style.

6.10. A logic for systems biology

Participants: Kaustuv Chaudhuri, Joëlle Despeyroux.

Kaustuv Chaudhuri and Joëlle Despeyroux have proposed a logic, called HyLL, for reasoning in and about reactions in systems and molecular biology. Such reaction systems can be imagined as state transition systems where the transitions are equipped with a stochastic rate function. In HyLL, these transitions are linear implications (from linear logic). The rate is determined using a modal judgement that makes the rate of formation explicit. The modal judgement is then reified by means of the $\downarrow$ connective from hybrid logic. They have demonstrated the use of HyLL on the repressilator, an example of a regulatory gene network. The expressivity of HyLL have been explicated by giving an embedding of the stochastic π-calculus. A paper has been submitted and a technical report is in preparation.

6.11. Separation in the $\lambda\mu$-calculus

Participant: Alexis Saurin.

The $\lambda\mu$-calculus has been proposed in recent years by M. Parigot as a term calculus inspired by classical logic (similar to the way that the λ-calculus is inspired by intuitionistic logic. Later on, David & Py proved that the separation rules failed for the $\lambda\mu$-calculus, leading to the need to fix the calculus. In [16], Saurin showed a way to regaining the separation theorem by a natural extension of the $\lambda\mu$-calculus to the $\Lambda\mu$-calculus.

6.12. Static analysis via proof theory

Participant: Dale Miller.

[1] http://www.lix.polytechnique.fr/~lutz/

Static analysis of logic programs can provide useful information for programmers and compilers. Typing systems, such as in λProlog [49] [50], 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 [42], 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 [37].

7. Other Grants and Activities

7.1. Actions nationales

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

Participants: Dale Miller, Lutz Straßburger.

The ANR-project 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. *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 (USA). 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, Italy) and Brigitte Pientka (McGill University, Canada).

7.2.2. *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.

7.2.3. 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.

7.2.4. Hurbert Curien: PAI Amadeus 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 (Austria) and Michel Parigot at CNRS-PPS.

7.2.5. Hurbert Curien: PAI Germaine De Stael “Deep Inference and the Essence of Proofs”

Participants: Dale Miller, Lutz Straßburger.

This collaboration between Paris and Bern (Germany) aims at exploring some questions in the structure and identity of proofs. People involved in the Paris area are Lutz Straßburger, Dale Miller, Alexis Saurin, David Baelde, Michel Parigot, Stéphane Lengrand, and Séverine Maingaud. People involved in Bern are Kai Brünnler, Richard McKinley, and Phiniki Stouppa.

8. Dissemination

8.1. Services to the Scientific Community

8.1.1. Organization of Conferences and Workshops

Joëlle Despeyroux co-organized the ACM SIGPLAN-SIGACT international conference on Principles of Programming Languages (POPL), with Martin Hofmann (LMU university, Munich, Germany). The conference was held in Nice, from 14 to 20 January.

Lutz Straßburger organized a workshop on "The Realm of Cut Elimination" on May 14, 2007 at LIX. This workshop was part of the PAI Amadeus with TU Wien.

Lutz Straßburger organized a workshop on "Theory and Application of Deep Inference" on June 21, 2007 at LIX. This workshop is part of (and partially financed by) the ANR project INFER on "Theory and Application of Deep Inference", and the PAI Germaine De Stael project on "Deep Inference and the Essence of Proofs". It served as first meeting of the participants of these projects.

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.

- LFMTP'07: Workshop on Logical Frameworks and Meta-Languages: Theory and Practice, August, Bremem, Germany.
- WoLLIC'07: Fourteenth Workshop on Logic, Language, Information and Computation, Rio de Janeiro, 2-5 July.
- CADE-21: 21st Conference on Automated Deduction, 17 - 20 July Bremen, Germany.

8.1.4. Invited talks at Conferences or Workshops

David Baelde and Dale Miller were invited speakers at the ICMS Workshop on Mathematical Theories of Abstraction, Substitution and Naming in Computer Science, Edinburgh, UK, 25-29 May 2007.

Dale Miller and Lutz Straßburger were invited participants to the meeting *Collegium Logicum 2007: Proofs and Structures*, 24-25 October 2007, Vienna, Austria.

8.2. Teaching

From December 17 to December 22, 2007, Lutz Straßburger teaches a course on "Introduction to Deep Inference and Proof Nets" (together with Paola Bruscoli, University of Bath) at the Technische Universität Dresden for the International MSc Program in Computational Logic.

Dale Miller 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, 2007).

8.3. Evaluation

Dale Miller was an external reporter for the Habilitation of Agata Ciabattoni (Technische Universität Wien), March 2007, and was an external reporter (rapporteur) for the PhD thesis of Sébastien Briais, École Polytechnique Fédérale de Lausanne, 17 Dec 2006.

8.4. Internship supervision

From March to August 2007, Nicolas Guenot (MPRI) was writing his Master's thesis on "Proof search, multi-focussing and deep inference for linear logic" under the supervision of Lutz Straßburger.

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, http://www-sop.inria.fr/parsifal/Joelle.Despeyroux/papers/mscs00.ps.

[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, http://www-sop.inria.fr/parsifal/Joelle.Despeyroux/papers/tcs00.ps.

[3] F. LAMARCHE, L. STRASSBURGER. *Naming Proofs in Classical Propositional Logic*, in "Typed Lambda Calculi and Applications, TLCA 2005", P. URZYCZYN (editor), LNCS, vol. 3461, Springer, 2005, p. 246–261, http://www.lix.polytechnique.fr/~lutz/papers/namingproofsCL.pdf.

[4] 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/pdf/cs.LO/0605054.

[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, 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.

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

[8] 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, http://www.lix.polytechnique.fr/Labo/Dale.Miller/papers/tocl-nabla.pdf.

[9] L. STRASSBURGER. *On the Axiomatisation of Boolean Categories with and without Medial*, Accepted for publication in TAC, 2007, http://www.lix.polytechnique.fr/~lutz/papers/medial.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.

Year Publications

Publications in Conferences and Workshops

[11] D. BAELDE, A. GACEK, D. MILLER, G. NADATHUR, A. TIU. *The Bedwyr system for model checking over syntactic expressions*, in "21th Conference on Automated Deduction", F. PFENNING (editor), LNAI, n^o 4603, Springer, 2007, p. 391–397, http://www.lix.polytechnique.fr/Labo/Dale.Miller/papers/cade2007.pdf.

[12] D. BAELDE, D. MILLER. *Least and greatest fixed points in linear logic*, in "LPAR 2007: Logic for Programming, Artificial Intelligence, and Reasoning", Lecture Notes in Computer Science, vol. 4790, 2007, p. 92-106, http://www.lix.polytechnique.fr/Labo/Dale.Miller/papers/lpar07final.pdf.

[13] C. LIANG, D. MILLER. *Focusing and Polarization in Intuitionistic Logic*, in "CSL 2007: Computer Science Logic", J. DUPARC, T. A. HENZINGER (editors), LNCS, vol. 4646, Springer-Verlag, 2007, p. 451–465, http://www.lix.polytechnique.fr/Labo/Dale.Miller/papers/csl07liang.pdf.

[14] D. MILLER, V. NIGAM. *Incorporating tables into proofs*, in "CSL 2007: Computer Science Logic", J. DUPARC, T. A. HENZINGER (editors), LNCS, vol. 4646, Springer-Verlag, 2007, p. 466–480, http://www.lix.polytechnique.fr/Labo/Dale.Miller/papers/csl07nigam.pdf.

[15] D. MILLER, A. SAURIN. *From proofs to focused proofs: a modular proof of focalization in Linear Logic*, in "CSL 2007: Computer Science Logic", J. DUPARC, T. A. HENZINGER (editors), LNCS, vol. 4646, Springer-Verlag, 2007, p. 405–419, http://www.lix.polytechnique.fr/Labo/Dale.Miller/papers/csl07saurin.pdf.

[16] A. SAURIN. *Typing streams in the $\Lambda\mu$-calculus: extended abstract*, in "Proceedings of the Short Papers Session at LPAR 2007", October 2007, http://www.eleves.ens.fr/home/saurin/Recherche/lpar07saurin.pdf.

[17] L. STRASSBURGER. *A Characterisation of Medial as Rewriting Rule*, in "Term Rewriting and Applications, RTA'07", F. BAADER (editor), LNCS, vol. 4533, Springer-Verlag, 2007, p. 344–358, http://www.lix.polytechnique.fr/~lutz/papers/CharMedial.pdf.

[18] L. STRASSBURGER. *Deep Inference for Hybrid Logic*, in "International Workshop on Hybrid Logic 2007 (Part of ESSLLI'07)", 2007, http://www.lix.polytechnique.fr/~lutz/papers/hybrid.pdf.

Miscellaneous

[19] D. BAELDE, D. MILLER. *Least and greatest fixed points in linear logic: extended version*, Technical report, available from the first author's web page, April 2007, http://www.lix.polytechnique.fr/~dbaelde/productions/pool/mumall_draft_long.pdf.

References in notes

[20] S. ABRAMSKY. *Computational Interpretations of Linear Logic*, in "Theoretical Computer Science", vol. 111, 1993, p. 3–57.

[21] E. ALBERT, G. PUEBLA, M. V. HERMENEGILDO. *Abstraction-Carrying Code*, in "LPAR 2004: Logic for Programming, Artificial Intelligence, and Reasoning", Lecture Notes in Computer Science, vol. 3452, Springer, 2004, p. 380–397.

[22] J.-M. ANDREOLI. *Logic Programming with Focusing Proofs in Linear Logic*, in "Journal of Logic and Computation", vol. 2, n^{o} 3, 1992, p. 297–347.

[23] A. W. APPEL, A. P. FELTY. *Lightweight Lemmas in Lambda Prolog*, in "16th International Conference on Logic Programming", MIT Press, November 1999, p. 411–425.

[24] B. E. AYDEMIR, A. BOHANNON, M. FAIRBAIRN, J. N. FOSTER, B. C. PIERCE, P. SEWELL, D. VYTINIOTIS, G. WASHBURN, S. WEIRICH, S. ZDANCEWIC. *Mechanized Metatheory for the Masses: The PoplMark Challenge*, in "Theorem Proving in Higher Order Logics: 18th International Conference", LNCS, Springer-Verlag, 2005, p. 50–65.

[25] D. BAELDE, A. GACEK, D. MILLER, G. NADATHUR, A. TIU. *A User Guide to Bedwyr*, November 2006, http://gforge.inria.fr/docman/view.php/367/705/userguide.pdf.

[26] R. BLOSSEY, L. CARDELLI, A. PHILLIPS. *A Compositional Approach to the Stochastic Dynamics of Gene Networks*, in "Transactions on Computational Systems Biology", vol. IV, n^{o} 3939, 2006, p. 99–122.

[27] K. BRÜNNLER, A. F. TIU. *A Local System for Classical Logic*, in "LPAR 2001", R. NIEUWENHUIS, A. VORONKOV (editors), LNAI, vol. 2250, Springer-Verlag, 2001, p. 347–361.

[28] 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, http://www-sop.inria.fr/parsifal/Joelle.Despeyroux/papers/imla99.ps.

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

[30] J.-Y. GIRARD. *Linear Logic*, in "Theoretical Computer Science", vol. 50, 1987, p. 1–102.

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

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

[33] A. GUGLIELMI. *A System of Interaction and Structure*, in "ACM Trans. on Computational Logic", vol. 8, n^{o} 1, 2007.

[34] A. GUGLIELMI, L. STRASSBURGER. *Non-commutativity and MELL in the Calculus of Structures*, in "Computer Science Logic, CSL 2001", L. FRIBOURG (editor), LNCS, vol. 2142, Springer-Verlag, 2001, p. 54–68.

[35] 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.

[36] F. LAMARCHE, L. STRASSBURGER. *Constructing free Boolean categories*, in "Proceedings of the Twentieth Annual IEEE Symposium on Logic in Computer Science (LICS'05)", 2005, p. 209–218.

[37] 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.

[38] P. MARTIN-LÖF. *Constructive Mathematics and Computer Programming*, in "Sixth International Congress for Logic, Methodology, and Philosophy of Science, Amsterdam", North-Holland, 1982, p. 153–175.

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

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

[41] 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.

[42] 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.

[43] 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.

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

[45] D. MILLER, G. NADATHUR, F. PFENNING, A. SCEDROV. *Uniform Proofs as a Foundation for Logic Programming*, in "Annals of Pure and Applied Logic", vol. 51, 1991, p. 125–157.

[46] D. MILLER, A. SAURIN. *A game semantics for proof search: Preliminary results*, in "Proceedings of the Mathematical Foundations of Programming Semantics (MFPS05)", Electr. Notes Theor. Comput. Sci, n^{o} 155, 2006, p. 543–563, http://www.lix.polytechnique.fr/Labo/Dale.Miller/papers/mfps05.pdf.

[47] 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.

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

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

[50] 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.

[51] G. C. NECULA. *Proof-carrying code*, in "Conference Record of the 24th Symposium on Principles of Programming Languages 97, Paris, France", ACM Press, 1997, p. 106–119.

[52] G. C. NECULA, S. P. RAHUL. *Oracle-based checking of untrusted software.*, in "POPL", 2001, p. 142–154.

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

[54] F. PFENNING, C. SCHÜRMANN. *System Description: Twelf — A Meta-Logical Framework for Deductive Systems*, in "16th Conference on Automated Deduction, Trento", H. GANZINGER (editor), LNAI, n^{o} 1632, Springer, 1999, p. 202–206.

[55] E. P. ROBINSON. *Proof Nets for Classical Logic*, in "Journal of Logic and Computation", vol. 13, 2003, p. 777–797.

[56] L. STRASSBURGER, F. LAMARCHE. *On Proof Nets for Multiplicative Linear Logic with Units*, in "Computer Science Logic, CSL 2004", J. MARCINKOWSKI, A. TARLECKI (editors), LNCS, vol. 3210, Springer-Verlag, 2004, p. 145–159.

[57] 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.

[58] 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, http://www.lix.polytechnique.fr/Labo/Dale.Miller/papers/eshol05.pdf.

[59] 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.