RESEARCH CENTRE

Saclay - Île-de-France

IN PARTNERSHIP WITH:

CNRS, Ecole Polytechnique

2021

ACTIVITY REPORT

Project-Team

PARTOUT

Proof Automation and RepresenTation: a fOundation of compUtation and deducTion

IN COLLABORATION WITH: Laboratoire d'informatique de l'école polytechnique (LIX)

DOMAIN

Algorithmics, Programming, Software and Architecture

THEME

Proofs and Verification

Contents

Project-Team PARTOUT

Creation of the Project-Team: 2019 December 01

Keywords

Computer sciences and digital sciences

A2.1. – Programming Languages

A2.2. – Compilation

A2.4. – Formal method for verification, reliability, certification

A4.5. – Formal methods for security

A7.2. – Logic in Computer Science

A7.2.1. – Decision procedures

A7.2.2. – Automated Theorem Proving

A7.2.3. – Interactive Theorem Proving

A7.2.4. – Mechanized Formalization of Mathematics

A7.3.1. – Computational models and calculability

A8.1. – Discrete mathematics, combinatorics

A8.11. – Game Theory

Other research topics and application domains

B6.1. – Software industry

1 Team members, visitors, external collaborators

Research Scientists

- Lutz Straßburger [Team leader, Inria, Researcher, HDR]
- Beniamino Accattoli [Inria, Researcher]
- Kaustuv Chaudhuri [Inria, Researcher]
- Ian Mackie [CNRS, Researcher]
- Dale Miller [Inria, Senior Researcher]
- Gabriel Scherer [Inria, Researcher]
- Noam Zeilberger [École polytechnique, Researcher]

Post-Doctoral Fellow

- Marianna Girlando [Inria, until Feb 2021]

PhD Students

- Farah Al Wardani [Inria, from Nov 2021]
- Nicolas Blanco [Université de Birmingham, from Jul 2021]
- Maico Carlos Leberle [Inria, until May 2021]
- Matteo Manighetti [Inria, until Oct 2021]
- Olivier Martinot [Inria]
- Marianela Evelyn Morales Elena [École polytechnique]
- Giti Omidvar [Inria]
- Wendlasida Ouedraogo [Siemens Mobility, CIFRE]
- Jui Hsuan Wu [Institut Polytechnique de Paris, from Oct 2021]

Interns and Apprentices

- Nicolas Chataing [École Normale Supérieure de Paris, from Mar 2021 until Jul 2021]
- Maxime Legoupil [École Normale Supérieure de Paris, from Mar 2021 until Jul 2021]
- Maxime Vemclefs [Inria, from May 2021 until Aug 2021]

Administrative Assistant

- Bahar Carabetta [Inria]

2 Overall objectives

There is an emerging consensus that formal methods must be used as a matter of course in software development. Most software is too complex to be fully understood by one programmer or even a team of programmers, and requires the help of computerized techniques such as testing and model checking to analyze and eliminate entire classes of bugs. Moreover, in order for the software to be maintainable and reusable, it not only needs to be bug-free but also needs to have fully specified behavior, ideally accompanied with formal and machine-checkable proofs of correctness with respect to the specification. Indeed, formal specification and machine verification is the only way to achieve the highest level of assurance (EAL7) according to the ISO/IEC Common Criteria.[1]

Historically, achieving such a high degree of certainty in the operation of software has required significant investment of manpower, and hence of money. As a consequence, only software that is of critical importance (and relatively unchanging), such as monitoring software for nuclear reactors or fly-by-wire controllers in airplanes, has been subjected to such intense scrutiny. However, we are entering an age where we need trustworthy software in more mundane situations, with rapid development cycles, and without huge costs. For example: modern cars are essentially mobile computing platforms, smart-devices manage our intensely personal details, elections (and election campaigns) are increasingly fully computerized, and networks of drones monitor air pollution, traffic, military arenas, etc. Bugs in such systems can certainly lead to unpleasant, dangerous, or even life-threatening incidents.

The field of formal methods has stepped up to meet this growing need for trustworthy general purpose software in recent decades. Techniques such as computational type systems and explicit program annotations/contracts, and tools such as model checkers and interactive theorem provers, are starting to become standard in the computing industry. Indeed, many of these tools and techniques are now a part of undergraduate computer science curricula. In order to be usable by ordinary programmers (without PhDs in logic), such tools and techniques have to be high level and rely heavily on automation. Furthermore, multiple tools and techniques often need to marshaled to achieve a verification task, so theorem provers, solvers, model checkers, property testers, etc. need to be able to communicate with—and, ideally, trust—each other.

With all this sophistication in formal tools, there is an obvious question: what should we trust? Sophisticated formal reasoning tools are, generally speaking, complex software artifacts themselves; if we want complex software to undergo rigorous formal analysis we must be prepared to formally analyze the tools and techniques used in formal reasoning itself. Historically, the issue of trust has been addressed by means of relativizing it to *small* and *simple* cores. This is the basis of industrially successful formal reasoning systems such as Coq, Isabelle, HOL4, and ACL2. However, the relativization of trust has led to a balkanization of the formal reasoning community, since the Coq kernel, for example, is incompatible with the Isabelle kernel, and neither can directly cross-validate formal developments built with the other. Thus, there is now a burgeoning cottage industry of translations and adaptations of different formal proof languages for bridging the gap. A number of proposals have also been made for universal or retargetable proof languages (e.g., Dedukti, ProofCert) so that the cross-platform trust issues can be factorized into single trusted checkers.

Beyond mutual incompatibility caused by relativized trust, there is a bigger problem that the proof evidence that is accepted by small kernels is generally far too detailed to be useful. Formal developments usually occurs at a much higher level, relying on algorithmic techniques such as unification, simplification, rewriting, and controlled proof search to fill in details. Indeed, the most reusable products of formal developments tend to be these algorithmic techniques and associated collections of hand-crafted rules. Unfortunately, these techniques are even less portable than the fully detailed proofs themselves, since the techniques are often implemented in terms of the behaviors of the trusted kernels. We can broadly say that the problem with relativized trust is that it is based on the *operational* interpretation of implementations of trusted kernels. There still remains the question of *meta-theoretic correctness*. Most formal reasoning systems implement a variant of a well known mathematical formalism (e.g., Martin-Löf type theory, set theory, higher-order logic), but it is surprising that hardly any mainstream system has a formalized meta-theory.[2] Furthermore, formal reasoning systems are usually associated with complicated checkers for

[1] http://www.commoncriteriaportal.org/cc/

[2] A prominent exception is HOL-Light, whose implementation has been self-certified—in HOL-Light itself—up to a strong assumption necessary to side-step incompleteness.

side-conditions that often have unclear mathematical status. For example, the Coq kernel has a built-in syntactic termination checker for recursive fixed-point expressions that is required to work correctly for the kernel to be sound. This termination checker evolves and improves with each version of Coq, and therefore the most accurate documentation of its behavior is its own source code. Coq is not special in this regard: similar trusted features exist in nearly every mainstream formal reasoning system.

The PARTOUT project is interested in the principles of deductive and computational formalisms. In the broadest sense, we are interested in the question of *trustworthy and verifiable meta-theory*. At one end, this includes the well studied foundational questions of the meta-theory of logical systems and type systems: cut-elimination and focusing in proof theory, type soundness and normalization theorems in type theory, etc. The focus of our research here is on the fundamental relationships behind the the notions of *computation* and *deduction*. We are particularly interested in relationships that go beyond the well known correspondences between proofs and programs.[3] Indeed, interpreting *computation in terms of deduction* (as in logic programming) or *deduction in terms of computation* (as in rewrite systems or in model checking) can often lead to fruitful and enlightening research questions, both theoretical and practical.

From another end, PARTOUT works on the question of the *essential nature* of deductive or computational formalisms. For instance, we are interested in the question of *proof identity* that attempts to answer the following question: when are two proofs of the same theorem the same? Surprisingly, this very basic question is left unanswered in *proof theory*, the branch of mathematics that supposedly treats proofs as algebraic objects of interest. We also pay particular attention to the combinatorial and complexity-theoretic properties of the formalisms. Indeed, it is surprising that until very recently the λ-calculus, which is the de facto basis of every functional programming language, lacked a good complexity-theoretic foundation, i.e., a cost model that would allow us to use the λ-calculus directly to define complexity classes.

To put trustworthy meta-theory to use, the PARTOUT project also works on the design and implementations of formal reasoning tools and techniques. We study the mathematical principles behind the representations of formal concepts (λ-terms, proofs, abstract machines, etc.), with the goal of identifying the relationships and trade-offs. We also study computational formalisms such as higher-order relational programming that is well suited to the specification and analysis of systems defined in the *structural operational semantics* (SOS) style. We also work on foundational questions about induction and co-induction, which are used in intricate combinations in metamathematics.

3 Research program

Software and hardware systems perform *computation* (systems that process, compute and perform) and *deduction* (systems that search, check or prove). The makers of those systems express their intent using various frameworks such as programming languages, specification languages, and logics. The PARTOUT project aims at developing and using mathematical principles to design better frameworks for computation and reasoning. Principles of expression are researched from two directions, in tandem:

- Foundational approaches, from theories to applications: studying fundamental problems of programming and proof theory.

  Examples include studying the complexity of reduction strategies in lambda-calculi with sharing, or studying proof representations that quotient over rule permutations and can be adapted to many different logics.

- Empirical approaches, from applications to theories: studying systems currently in use to build a theoretical understanding of the practical choices made by their designers.

  Examples include studying realistic implementations of programming languages and proof assistants, which differ in interesting ways from their usual high-level formal description (regarding of sharing of code and data, for example), or studying new approaches to efficient automated proof search, relating them to existing approaches of proof theory, for example to design proof certificates or to generalize them to non-classical logics.

[3] The *Curry-Howard* correspondence.

One of the strengths of PARTOUT is the co-existence of a number of different expertise and points of view. Many dichotomies exist in the study of computation and deduction: functional programming *vs* logic programming, operational semantics *vs* denotational semantics, constructive logic *vs* classical logic, proof terms *vs* proof nets, etc. We do not identify with any one of them in particular, rather with them as a whole, believing in the value of interaction and cross-fertilization between different approaches. PARTOUT defines its scope through the following core tenets:

- An interest in both computation and logic.
- The use of mathematical formalism as our core scientific method, paired with practical implementations of the systems we study.
- A shared belief in the importance of good *design* when creating new means of expression, iterating towards simplicity and elegance.

More concretely, the research in PARTOUT will be centered around the following four themes:

1. **Foundations of proof theory as a theory of proofs.** Current proof theory is not a theory of proofs but a theory of proof systems. This has many practical consequences, as a proof produced by modern theorem provers cannot be considered independent from the tool that produced it. A central research topic here is the quest for proof representations that are independent from the proof system, so that proof theory becomes a proper theory of proofs.

2. **Program Equivalence** We intend to use our proof theoretical insights to deepen our understanding of the structure of computer programs by discovering canonical representations for functional programming languages, and to apply these to the problems of program equivalence checking and program synthesis.

3. **Reasoning with relational specifications of formal systems.** Formal systems play a central role for proof checkers and proof assistants that are used for software verification. But there is usually a large gap between the specification of those formal systems in concise informal mathematical language and their implementation in ML or C code. Our research goal is to close that gap.

4. **Foundations of complexity analysis for functional programs.** One of the great merits of the functional programming paradigm is the natural availability of high-level abstractions. However, these abstractions jeopardize the programmer's predictive control on the performance of the code, since many low-level steps are abstracted away by higher-order functions. Our research goal is to regain that control by developing models of space and time costs for functional programs.

4 Application domains

4.1 Automated Theorem Proving

The Partout team studies the structure of mathematical proofs, in ways that often makes them more amenable to automated theorem proving – automatically searching the space of proof candidates for a statement to find an actual proof – or a counter-example.

(Due to fundamental computability limits, fully-automatic proving is only possible for simple statements, but this field has been making a lot of progress in recent years, and is in particular interested with the idea of generating verifiable evidence for the proofs that are found, which fits squarely within the expertise of Partout.)

4.2 Proof-assistants

Our work on the structure of proofs also suggests ways how they could be presented to a user, edited and maintained, in particular in "proof assistants", automated tool to assist the writing of mathematical proofs with automatic checking of their correctness.

4.3 Programming language design

Our work also gives insight on the structure and properties of programming languages. We can improve the design or implementation of programming languages, help programmers or language implementors reason about the correctness of the programs in a given language, or reason about the cost of execution of a program.

5 Highlights of the year

5.1 Awards

Noam Zeilberger was awarded an ANR PRC grant as project coordinator for the LambdaComb project, which starts in 2022. The aim of the project is to develop some surprising connections between lambda calculus and combinatorics that were discovered over recent years. Partners include labs in Paris (LIX, LIPN, LIGM), Marseille (LIS), and Poland (Jagiellonian), with an overall budget of roughly 285k€.

6 New software and platforms

6.1 New software

6.1.1 MOIN

Name: MOdal Intuitionistic Nested sequents

Keywords: Logic programming, Modal logic

Functional Description: MOIN is a SWI Prolog theorem prover for classical and intuitionstic modal logics. The modal and intuitionistic modal logics considered are all the 15 systems occurring in the modal S5-cube, and all the decidable intuitionistic modal logics in the IS5-cube. MOIN also provides a protptype implementation for the intuitionistic logics for which decidability is not known (IK4,ID5 and IS4). MOIN is consists of a set of Prolog clauses, each clause representing a rule in one of the three proof systems. The clauses are recursively applied to a given formula, constructing a proof-search tree. The user selects the nested proof system, the logic, and the formula to be tested. In the case of classic nested sequent and Maehara-style nested sequents, MOIN yields a derivation, in case of success of the proof search, or a countermodel, in case of proof search failure. The countermodel for classical modal logics is a Kripke model, while for intuitionistic modal logic is a bi-relational model. In case of Gentzen-style nested sequents, the prover does not perform a countermodel extraction.

A system description of MOIN is available at https://hal.inria.fr/hal-02457240

URL: `http://www.lix.polytechnique.fr/Labo/Lutz.Strassburger/Software/Moin/MoinProver.html`

Publication: hal-02457240

Contact: Lutz Strassburger

6.1.2 OCaml

Keywords: Functional programming, Static typing, Compilation

Functional Description: The OCaml language is a functional programming language that combines safety with expressiveness through the use of a precise and flexible type system with automatic type inference. The OCaml system is a comprehensive implementation of this language, featuring two compilers (a bytecode compiler, for fast prototyping and interactive use, and a native-code compiler producing efficient machine code for x86, ARM, PowerPC, RISC-V and System Z), a debugger, and a documentation generator. Many other tools and libraries are contributed by the user community and organized around the OPAM package manager.

URL: `https://ocaml.org/`

Publications: hal-03146495, hal-03510931, hal-03145030, hal-01929508, hal-03125031, hal-00772993, hal-00914493, hal-00914560, inria-00074804, hal-01499973, hal-01499946

Contact: Damien Doligez

Participants: Florian Angeletti, Damien Doligez, Xavier Leroy, Luc Maranget, Gabriel Scherer, Alain Frisch, Jacques Garrigue, Marc Shinwell, Jeremy Yallop, Leo White

6.1.3 Abella

Keyword: Proof assistant

Functional Description: Abella is an interactive theorem prover for reasoning about computations given as relational specifications. Abella is particuarly well suited for reasoning about binding constructs.

URL: `http://abella-prover.org/`

Contact: Kaustuv Chaudhuri

Participants: Dale Miller, Gopalan Nadathur, Kaustuv Chaudhuri, Mary Southern, Matteo Cimini, Olivier Savary-Bélanger, Yuting Wang

Partner: Department of Computer Science and Engineering, University of Minnesota

6.1.4 ocaml-boxroot

Keywords: Interoperability, Library, Ocaml, Rust

Scientific Description: Boxroot is an implementation of roots for the OCaml GC based on concurrent allocation techniques. These roots are designed to support a calling convention to interface between Rust and OCaml code that reconciles the latter's foreign function interface with the idioms from the former.

Functional Description: Boxroot implements fast movable roots for OCaml in C. A root is a data type which contains an OCaml value, and interfaces with the OCaml GC to ensure that this value and its transitive children are kept alive while the root exists. This can be used to write programs in other languages that interface with programs written in OCaml.

URL: `https://gitlab.com/ocaml-rust/ocaml-boxroot`

Contact: Guillaume Munch

Participants: Guillaume Munch, Gabriel Scherer

6.2 New platforms

7 New results

7.1 Game Semantics for Constructive Modal Logic

Participants: Lutz Straßburger.

External Collaborators: Matteo Acclavio (University of Luxembourg), Davide Catta (University of Montpellier)

Continuing our work on constructive and intuitionistic modal logics, we provide the first game semantics for the constructive modal logic CK. We define arenas encoding modal formulas, and we define winning innocent strategies for games on these arenas. Finally we characterize the winning strategies corresponding to proofs in the logic CK. To prove the full-completeness of our semantics, we provide a sequentialization procedure of winning strategies. We also prove their compositionality and showing how our results can be extend to the constructive modal logic CD. The results are published in [23] and [24].

7.2 Combinatorial Proofs and Decomposition Theorems for First-order Logic

Participants: Lutz Straßburger, Jui-Hsuan Wu.

External Collaborators: Dominic Hughes (U.C. Berkeley)

We uncover a close relationship between combinatorial and syntactic proofs for first-order logic (without equality). Whereas syntactic proofs are formalized in a deductive proof system based on inference rules, a combinatorial proof is a syntax-free presentation of a proof that is independent from any set of inference rules. We show that the two proof representations are related via a deep inference decomposition theorem that establishes a new kind of normal form for syntactic proofs. This yields (a) a simple proof of soundness and completeness for first-order combinatorial proofs, and (b) a full completeness theorem: every combinatorial proof is the image of a syntactic proof.

This result is published in the LICS 2021 conference [17]

7.3 Coqlex, an approach to generate verified lexers

Participants: Lutz Straßburger, Wendlasida Ouedraogo.

External Collaborators: Danko Ilik (Siemens)

A compiler consists of a sequence of phases going from lexical analysis to code generation. Ideally, the formal verification of a compiler should include the formal verification of every component of the tool-chain. In order to contribute to the end-to-end verification of compilers, we implemented a verified lexer generator with usage similar to OCamllex. This software-Coqlex-reads a lexer specification and generates a lexer equipped with Coq proofs of its correctness. Although the performance of the generated lexers does not measure up to the performance of a standard lexer generator such as OCamllex, the safety guarantees it comes with make it an interesting alternative to use when implementing totally verified compilers or other language processing tools.

This work has been presented at the ML 2021 workshop [22]

7.4 Perspectives on proof theory and logic programming

Participants: Dale Miller.

For more than thirty years, various researchers, including current and previous members of Partout and Parsifal, have been applying proof theory to multiple topics in computational logic. A survey of that work has been published in [9], an invited submission to the *20th Anniversary Issue of the Theory and Practice of Logic Programming*. This survey documents various ways that proof theory has been applied to logic programming. One can actually see from this history a surprising influence of logic programming also on the development of proof theory. This reciprocal influences between logic programming and proof theory is reported in the paper [10].

7.5 Unification of terms containing bindings

Participants: Dale Miller.

External Collaborators: Tomer Libal, University of Luxembourg, Computer Science Department

Higher-order pattern unification is often used in proof assistants and other computational logic systems to discover appropriate instances of quantifiers when implementing the mechanical search for proofs. Recently, this approach to unification has been expanded to the *functions-as-constructors higher-order unification* setting: this expanded setting continues to be decidable and possesses most general unifiers whenever unifiers exist [7].

7.6 Proof search when equality is a logical connective

Participants: Dale Miller.

External Collaborators: Alexandre Viel

The formulation of equality uses in the team's implementations of model checking and theorem proving identifies equality as a *logical connective*. Other treatments of equality treat it as a non-logical predicate axiomatized by an appropriate theory. Our treatment as a logical connective, however, allows us to have very strong focusing theorems in the setting of arithmetic proofs. The Bedwyr and Abella systems incorporate this approach to equality. The paper [11] provides a proof that such unification, in its most general form, is undecidability.

7.7 Improving Gentzen's LK proof system

Participants: Dale Miller.

External Collaborators: Chuck Liang, Hofstra University, New York, USA.

Gentzen's sequent calculi LK is a landmark proof systems. Given the extensive uses that have been made of LK by computer scientists in recent decades, several undesirable features of this calculus have been identified. Among such features is that its inferences rules are low-level and frequently permute over each other. As a result, large-scale structures within sequent calculus proofs are hard to identify. Liang and Miller [26] present a different approach to designing a sequent calculus for classical logic. Starting with LK, they examined the proof search meaning of its inference rules and classify them as involving either don't care nondeterminism or don't know nondeterminism. Based on that classification, they designed the focused proof system LKF in which inference rules belong to one of two phases of proof construction depending on which flavor of nondeterminism they involve. They then prove that the cut rule and the general form of the initial rule are admissible in LKF. These results can be used to provide simple proofs of various meta-theoretic properties of classical logic, including Herbrand's theorem.

7.8 Extending the Inferno type-inference approach to realistic type-system features

Participants: Gabriel Scherer, Olivier Martinot.

Inferno is a software library from François Pottier (EPI Cambium) to implement constraint-based type-inference in a pleasant, declarative style. It contains a proof-of-concept inference engine for a very small programming language, but it is not obvious how to scale its declarative style to richer language features.

Olivier Martinot, as a PhD student, has been working with Gabriel Scherer on extending the Inferno approach to more language features, hoping to eventually cover a large subset of the OCaml type system. This action is continued from last year.

Gabriel presented a part of this joint work at the "ML Family Workshop" 2021, [21].

7.9 OCaml Constructor unboxing

Participants: Nicolas Chataing (M2 intern), Gabriel Scherer

Constructor unboxing is a proposed data-representation optimization for OCaml; it could improve memory usage in certain cases by eliminating overhead in the memory representation of values. We worked on a prototype implmentation of constructor unboxing. This required solving a decision problem for unfolding of ML datatype declarations in presence of mutual recursion, an interesting scientific result of its own.

This work was presented at the ML workshop [20] and will be presented at the JFLA'22 [19].

7.10 Camlboot

Participants: Nathanaëlle Courant (INRIA Paris, EPI Cambium), Julien Lepiller (Yale University), Gabriel Scherer

Camlboot is a software project to "debootstrap" the OCaml compiler, that is, compile the OCaml compiler (which is itself written in OCaml) without requiring the use of a previous version of the OCaml compiler. It relies on a reference interpreter for OCaml written in a small subset of OCaml, that can be compiled to bytecode by a small, special-purpose compiler written in Scheme. A reference interpreter for OCaml is also a result of independent interest.

7.11 The (In)Efficiency of Interaction

Participants: Beniamino Accattoli.

External Collaborators: Ugo Dal Lago (University of Bologna & Inria) and Gabriele Vanoni (University of Bologna & Inria).

This work studies the time performance of abstract machines for the λ-calculus inspired by the geometry of interaction, building over work by the same authors in 2020. The results are that in general these machines are less efficient than those based on environments. The paper also studies the link between the time of the interaction abstract machine and intersection types, showing how to extract the former from the latter. The type system is then used to show that the inefficiency of the interaction abstract machine is due and proportional to the use of higher-order types.

This work belongs to the research theme *Foundations of complexity analysis for functional programs* and it has been published in [5].

7.12 The Space of Interaction

Participants: Beniamino Accattoli.

External Collaborators: Ugo Dal Lago (University of Bologna & Inria) and Gabriele Vanoni (University of Bologna & Inria).

This work complements the one in the previous subsection by studying the *space* of the interaction abstract machine (shortened to IAM) for the λ-calculus via a new type system based on intersection types, the first such type system able to measure space. The type system is then used to give a strong argument against the conjecture that the space of the IAM is a reasonable space cost model for the λ-calculus, roughly disproving the conjecture.

This work belongs to the research theme *Foundations of complexity analysis for functional programs* and it has been published in [15].

7.13 Strong Call-by-Value is Reasonable, Implosively

Participants: Beniamino Accattoli.

External Collaborators: Andrea Condoluci (Tweag I/O) and Claudio Sacerdoti Coen (University of Bologna).

This work proves that the number of steps taken by the strong call-by-value evaluation strategy of the λ-calculus, which is used in the implementation of the Coq proof assistant, is a reasonable time cost model. The proof rests on a new abstract machine which—thanks to a new mix of sharing techniques—is remarkably efficient: it is the first machine for strong evaluation (that is, evaluation that enters function bodies)working within an overhead *linear* in both the number of β-steps and the size of the initial term. We actually show that on some families of terms the overhead is actually *logarithmic* inthe number of β-steps.

This work belongs to the research theme *Foundations of complexity analysis for functional programs* and it has been published in [13].

7.14 Asymptotic Distribution of Parameters in Trivalent Maps and Linear Lambda Terms

Participants: Noam Zeilberger

External Collaborators: Olivier Bodini, LIPN, Université Sorbonne Paris Nord; Alexandros Singh, LIPN, Université Sorbonne Paris Nord

This work builds on recent surprising connections discovered between lambda calculus and the study of *map enumeration*, which is an active subfield of combinatorics initiated by Bill Tutte in the 1960s. Notably, bijections between different families of linear lambda terms and different families of rooted maps were independently discovered by Bodini, Gardy, and Jacquot (2013) and by Zeilberger and Giorgetti (2015), and have since been the subject of a variety of followup works (for an overview, see the introduction to Zeilberger, "A theory of linear typings as flows on 3-valent graphs", LICS'2018).

In this paper, we dive deeper into the study of the combinatorics of linear lambda calculus, focusing on the analysis of different parameters of lambda terms and their map-theoretic counterparts. For instance, under the bijections mentioned above, *closed subterms* of a linear lambda term correspond to *bridges* in the corresponding map, i.e., edges whose deletion increases the number of connected components. We proved that the limit distribution of the number of closed proper subterms of a random closed linear lambda term is a Poisson distribution of parameter 1 (= the asymptotic probability of having k closed proper subterms is $1/(k!e)$), therefore allowing us to conclude exactly the same for the limit distribution of the number of bridges in a random map, as a surprising application of lambda calculus to graph theory. We also studied the distribution of free variables in open terms, which correspond to 1-valent vertices.

This is work that occurs in the context of Singh's doctoral thesis research, co-supervised by Bodini and Zeilbeger. It has been presented by Singh at various venues including the journées ALEA 2021, the

"Structure Meets Power" workshop at LICS 2021, and the "Combinatorics and Arithmetic for Physics" workshop at IHES, among others. The pre-print [25] has been submitted to the open-access journal *Combinatorial Theory* and is currently under review.

7.15 Subformula linking for intuitionistic logic

Participants: Kaustuv Chaudhuri.

In 2013 we proposed a new method of interactive theorem proving based on the use of *subformula linking*, which involves the use of links between arbitrary subformulas of a goal conjecture (i.e., the theorem being proved) [29]. In a work published at CADE 2021 [16] we show how to extend this technique to intuitionistic logic and a certain kind of intuitionistic type theories.

The main purpose of this work is to build new interactive theorem proving interfaces that are decoupled from the formal languages used to instruct proof verifiers. This allows the same proof building interface to be used across a variety of theorem provers, and also to simplify the instruction that must be given to novice users.

We are currently exploring mechanisms to add induction and higher-order logic to this framework.

8 Bilateral contracts and grants with industry

8.1 Bilateral contracts with industry

8.1.1 CIFRE Thesis Inria - Siemens

Participants: Lutz Straßburger, Wendlasida Ouedraogo.

Title: Optimization of source code for safety-critical systems

Duration: 2020 – 2022

Scientific Responsible: Lutz Straßburger

Industrial Partner: Siemens Mobility, Chatillon

Summary: The goal of the thesis is to develop ways to optimize the performance of software, while not sacrificing the guarantees of safety already provided for non-optimized code. The software that Siemens is using for their self-driving trains (e.g. Metro 14 in Paris) is programmed in Ada. Due to the high safety requirements for the software, the used Ada compiler has to be certified. At the current state of the art, only non-optimized code fulfils all necessary requirements. Because of higher performance needs, we are interested in producing optimized code that also fulfils these reqirements.

Stated most generally, the aim of the thesis is to assure, *at the same time*:

- optimization of execution-time of safety-critical software — safety-critical software is more prone to bad execution-time performance, because most of its actions involve performing checks (i.e., CPU branch instructions), and
- maintaining the safety guarantees from the input source code to the produced binary code — in general, as soon as we decide to use a compiler optimization, the qualification of the compiler no longer applies.

8.2 Bilateral grants with industry

8.2.1 OCaml Software Foundation

Participants: Gabriel Scherer.

The OCaml Software Foundation (OCSF),[4] established in 2018 under the umbrella of the Inria Foundation, aims to promote, protect, and advance the OCaml programming language and its ecosystem, and to support and facilitate the growth of a diverse and international community of OCaml users.

Since 2019, Gabriel Scherer serves as the director of the foundation.

8.2.2 General OCaml funding from Nomadic Labs

Participants: Gabriel Scherer, Olivier Martinot.

Nomadic Labs, a Paris-based company, has implemented the Tezos blockchain and cryptocurrency entirely in OCaml. In 2019, Nomadic Labs and Inria have signed a framework agreement ("contrat-cadre") that allows Nomadic Labs to fund multiple research efforts carried out by Inria groups. Within this framework, we participate to the following grants, in collaboration with the project-team Cambium at INRIA Paris:

Évolution d'OCaml

This grant is intended to fund a number of improvements to OCaml, including the addition of new features and a possible re-design of the OCaml type-checker. This grant funds the PhD thesis of Olivier Martinot on this topic.

Maintenance d'OCaml

This grant is intended to fund the day-to-day maintenance of OCaml as well as the considerable work involved in managing the release cycle.

OCaml-Rust

Title: OCaml/Rust bindings

Duration: 2021-2023

Coordinator: Gabriel Scherer (INRIA Saclay, EPI Partout)

Participants: Guillaume Munch-Maccagnoni (INRIA Rennes, EPI Galinette), Jacques-Henri Jourdan (CNRS, LRI)

Partners: Inria, Nomadic Labs

Inria contact: Gabriel Scherer

Summary: We often want to write hybrid programs with components in several different programming languages. Interfacing two languages typically goes through low-level, unsafe interfaces. The OCaml/Rust project studies safer interfaces between OCaml and Rust.

[4] http://ocaml-sf.org/

Expected Impact: We investigated safe low-level representations of OCaml values on the Rust side, representing GC ownership, and developed a calling convention that reconciles the OCaml FFI idioms with Rust idioms. We also developed Boxroot, a new API to register values with the OCaml GC, for used when interfacing with Rust (and other programming languages) and possibly when writing concurrent programs. This resulted in novel techniques which can benefit other pairs of languages in the future. These works are now integrated in the ocaml-rs interface between OCaml and Rust used in the industry.

9 Partnerships and cooperations

9.1 International initiatives

9.1.1 Inria associate team not involved in an IIL or an international program

COMPRONOM

Title: Combinatorial Proof Normalization

Duration: 2020 ->

Coordinator: Lutz Straßburger

Partners:

- University of Bath

Inria contact: Lutz Strassburger

Summary: This project teams up three research groups at Inria Saclay, the University of Bath, and University College London, who are driven by their joint interest in the development of a combinatorial proof theory which is able to treat formal proofs independently from syntactic proof systems. We plan to focus our research in two major directions: First, study the normalization of combinatorial proofs, with possible applications for the implementation of functional programming languages, and second, study combinatorial proofs for the logic of bunched implications, with the possible application for separation logic and its use in the verification of imperative programs.

9.1.2 STIC/MATH/CLIMAT AmSud project

- "Dynamic Logics: Model Theory, Proof Theory and Computational Complexity (DyLo-MPC)" (joint project beween France, Argentina, Brazil, 2020–2022)

9.2 National initiatives

- ANR JCJC project COCA HOLA: Cost Models for Complexity Analyses of Higher-Order Languages, coordinated by B. Accattoli, 2016–2021, ANR-16-CE40-004-01.

10 Dissemination

10.1 Promoting scientific activities

10.1.1 Scientific events: organisation

General chair, scientific chair

- Dale Miller completed in July 2021 a three year appointment as the General Chair of LICS.

Member of the organizing committees

- Giti Omidvar was student volunteer at POPL'21
- Wendlasida Ouedraogo was student volunteer at POPL'21
- Miller is a member of the Steering Committees of both LICS and LFMTP.
- Chaudhuri serves on the Steering Committee of the International Joint Conference on Automated Reasoning (IJCAR, 2020-2022).
- Marianela Morales was student volunteer at POPL'21 and ICFP'21

10.1.2 Scientific events: selection

Member of the conference program committees

- Lutz Straßburger was member of the PC of Tableaux 2021 and of TLLA 2021
- Miller was on the program committees of RAMiCS 2021 (19th International Conference on Relational and Algebraic Methods in Computer Science) and LPAR-23 (23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning).
- Accattoli was member of the PC of IWC 2021 (International workshop on confluence).
- Chaudhuri was on the program committee of Tableaux 2021
- Marianela Morales was member of the Artifact Evaluation Committee at CAV2021: 33rd International Conference on Computer-Aided Verification.

Reviewer

- Lutz Straßburger was reviewer for Tableaux 2021, TLLA 2021, and CSL 2022
- Zeilberger was reviewer for LICS 2021, MSCS 2021, and CALCO 2021.
- Accattoli was reviewer for LICS 2021 (2 papers), ICFP 2021, FSCD 2021 (2 papers).

10.1.3 Journal

Member of the editorial boards

- Miller is a member of the Advirsory Board of the new Diamond Open Access electronic journal TheoretiCS, which will cover all areas of Theoretical Computer Science.
- Miller is a member of the Journal of Automated Reasoning, published by Springer (since May 2011).
- Miller is an area editor for "Type Theory for Theorem Proving Systems" of the Journal of Applied Logic, published by Elsevier (since 2003).

Reviewer - reviewing activities

- Lutz Straßburger was reviewer for the *Journal of Philosopical Logic* (JLP), and *Logical Methods in Computer Science* (LMCS)
- Miller was a reviewer for the *Annals of Mathematics and Artificial Intelligence*.
- Miller was a reviewer for Deutsche Forschungsgemeinschaft (German Research Foundation).
- Zeilberger was a reviewer for LMCS.
- Accattoli was reviewer for LMCS (2 papers), TOCL (*ACM Transactions on Computational Logic*) and JLAMP (*Journal of Logical and Algebraic Methods in Programming*).

10.1.4 Invited talks

- Miller was an invited speaker at the Seventh Meeting of ANR-FWF Project Ticamore (June) and the PhilMath Seminar, Institut d'histoire et de philosophie des sciences et des techniques (IHPST) (December).
- In March, Miller was invited to speak to both the Online Worldwide Seminar on Logic and Semantics (OWLS) and the Proof Theory Virtual Seminar.
- Zeilberger gave an invited talk as part of the special session on Categorical Type Theory at MFPS 2021. Over the year he also gave invited online seminars at the University of Bath, Tallinn University of Technology, and Masaryk University.
- Accattoli was invited speaker at TLLA 2021 (International Workshop on Trends in Linear Logic and Applications).

10.2 Teaching - Supervision - Juries

10.2.1 Teaching

- Lutz Straßburger was teaching a course at ESSLLI 2022. Course notes can be found here [28]
- Giti Omidvar was teaching assistant for the course INF442 – Algorithms Pour l'analyse de donées en C++ (from the beginning of March to the beginning of June) at Ecole Polytechnique
- Wendlasida Ouedraogo was teaching assistant for the course INF411 at Ecole Polytechnique
- Miller was an instructor for MPRI (Master Parisien de Recherche en Informatique) in the Course 2-1: Logique linéaire et paradigmes logiques du calcul. He taught 12 hours during Spring and 15 hours during Fall 2021.
- Noam Zeilberger taught the third year undergraduate course "Functional Programming" in the Bachelors program at Ecole Polytechnique, and was a teaching assistant for the second year polytechnicien course INF412 "Fondements de l'informatique".
- Accattoli was an instructor for MPRI (Master Parisien de Recherche en Informatique) in the Course 2-1: Logique linéaire et paradigmes logiques du calcul. He taught 15 hours.
- Chaudhuri taught the third year undergraduate course "CSE 302: Compiler Design" at the Ecole polytechnique. He was also a teach assistant for INF412 at the engineering program at the polytechnique.
- Marianela Morales was teaching assistant at the course Computer Programming at École Polytechnique (CSE101), second semester of Bachelor of Science 1

10.2.2 Supervision

- Lutz Straßburger is supervising three PhD students: Giti Omidvar, Marianela Morales, and Wendlasida Ouedraogo
- Miller is supervising three Ph.D. students: Farah Al Wardani, Matteo Manighetti, and Jui-Hsuan Wu.
- Zeilberger is co-supervising two PhD students: Nicolas Blanco (with Paul Blain Levy) and Alexandros Singh (with Olivier Bodini).
- Accattoli completed the supervision of one PhD student, Maico Leberle, who defended in May 2021, and one master student, Maxime Vemclefs, for an internship.
- Chaudhuri is co-supervising the PhD thesis of Farah Al Wardani.

10.2.3 Juries

- Lutz Straßburger was in the jury (as external reviewer) for the PhD defense of Tim Lyon (TU Wien, Austria)
- Miller was a reporter for the Ph.D. defense of Ahmed Bhayat (University of Manchester).

11 Scientific production

11.1 Major publications

[1] B. Accattoli, A. Condoluci and C. S. Coen. 'Strong Call-by-Value is Reasonable, Implosively'. In: 2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS). Rome, Italy: IEEE, 29th June 2021, pp. 1–14. DOI: 10.1109/LICS52264.2021.9470630. URL: https://hal.inria.fr/hal-03475461.

[2] B. Accattoli, U. Dal Lago and G. Vanoni. 'The (In)Efficiency of interaction'. In: *Proceedings of the ACM on Programming Languages* 5.POPL (4th Jan. 2021), pp. 1–33. DOI: 10.1145/3434332. URL: https://hal.inria.fr/hal-03346750.

[3] B. Accattoli, U. D. Lago and G. Vanoni. 'The Space of Interaction'. In: LICS 2021 - 36th Annual ACM/IEEE Symposium on Logic in Computer Science. Vol. 5. Rome, France: IEEE, 4th Jan. 2021, pp. 1–13. DOI: 10.1109/LICS52264.2021.9470726. URL: https://hal.inria.fr/hal-03346767.

[4] L. Straßburger, D. J. D. Hughes and J.-H. Wu. 'Combinatorial Proofs and Decomposition Theorems for First-order Logic'. In: 2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS). Rome, Italy: IEEE, 27th Apr. 2021, pp. 1–13. DOI: 10.1109/LICS52264.2021.9470579. URL: https://hal.inria.fr/hal-03369764.

11.2 Publications of the year

International journals

[5] B. Accattoli, U. Dal Lago and G. Vanoni. 'The (In)Efficiency of interaction'. In: *Proceedings of the ACM on Programming Languages* 5.POPL (4th Jan. 2021), pp. 1–33. DOI: 10.1145/3434332. URL: https://hal.inria.fr/hal-03346750.

[6] M. Girlando, S. Negri and N. Olivetti. 'Uniform labelled calculi for preferential conditional logics based on neighbourhood semantics'. In: *Journal of Logic and Computation* 31.3 (2021), pp. 947–997. DOI: 10.1093/logcom/exab019. URL: https://hal.archives-ouvertes.fr/hal-02330319.

[7] T. Libal and D. Miller. 'Functions-as-constructors higher-order unification: extended pattern unification'. In: *Annals of Mathematics and Artificial Intelligence* (30th Sept. 2021). DOI: 10.1007/s10472-021-09774-y. URL: https://hal.archives-ouvertes.fr/hal-03457303.

[8] S. Marin, M. Morales and L. Straßburger. 'A fully labelled proof system for intuitionistic modal logics'. In: *Journal of Logic and Computation* 31.3 (6th May 2021), pp. 998–1022. DOI: 10.1093/logcom/exab020. URL: https://hal.inria.fr/hal-02390454.

[9] D. Miller. 'A Survey of the Proof-Theoretic Foundations of Logic Programming'. In: *Theory and Practice of Logic Programming* (18th Nov. 2021), pp. 1–46. DOI: 10.1017/S1471068421000533. URL: https://hal.inria.fr/hal-03411144.

[10] D. Miller. 'Reciprocal Influences Between Proof Theory and Logic Programming'. In: *Philosophy & Technology* 34 (2021), pp. 75–104. DOI: 10.1007/s13347-019-00370-x. URL: https://hal.inria.fr/hal-02368867.

[11] D. Miller and A. Viel. 'The undecidability of proof search when equality is a logical connective'. In: *Annals of Mathematics and Artificial Intelligence* (3rd July 2021). DOI: 10.1007/s10472-021-09764-0. URL: https://hal.archives-ouvertes.fr/hal-03457312.

[12] A. Reynaud, G. Scherer and J. Yallop. 'A practical mode system for recursive definitions'. In: *Proceedings of the ACM on Programming Languages* 5.POPL (4th Jan. 2021), pp. 1–29. DOI: 10.1145/3434326. URL: https://hal.inria.fr/hal-03125031.

International peer-reviewed conferences

[13] B. Accattoli, A. Condoluci and C. S. Coen. 'Strong Call-by-Value is Reasonable, Implosively'. In: 2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS). Rome, Italy: IEEE, 29th June 2021, pp. 1–14. DOI: 10.1109/LICS52264.2021.9470630. URL: https://hal.inria.fr/hal-03475461.

[14] B. Accattoli, C. Faggian and G. Guerrieri. 'Factorize Factorization'. In: CSL 2021 - 29th EACSL Annual Conference on Computer Science Logic. Vol. 183. CSL 2021: 29th EACSL Annual Conference on Computer Science Logic. Ljubljana, Slovenia, 25th Jan. 2021. DOI: 10.4230/LIPIcs.CSL.2021.22. URL: https://hal.archives-ouvertes.fr/hal-03044338.

[15] B. Accattoli, U. D. Lago and G. Vanoni. 'The Space of Interaction'. In: LICS 2021 - 36th Annual ACM/IEEE Symposium on Logic in Computer Science. Vol. 5. Rome, France: IEEE, 4th Jan. 2021, pp. 1–13. DOI: 10.1109/LICS52264.2021.9470726. URL: https://hal.inria.fr/hal-03346767.

[16] K. Chaudhuri. 'Subformula Linking for Intuitionistic Logic with Application to Type Theory'. In: *Automated Deduction – CADE 28*. CADE 2021 - 28th International Conference on Automated Deduction. Vol. 12699. Lecture Notes in Computer Science. Pittsburgh, PA (Virtual), United States: Springer International Publishing, 5th July 2021, pp. 200–216. DOI: 10.1007/978-3-030-79876-5_12. URL: https://hal.inria.fr/hal-03528659.

[17] L. Straßburger, D. J. D. Hughes and J.-H. Wu. 'Combinatorial Proofs and Decomposition Theorems for First-order Logic'. In: 2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS). Rome, Italy: IEEE, 27th Apr. 2021, pp. 1–13. DOI: 10.1109/LICS52264.2021.9470579. URL: https://hal.inria.fr/hal-03369764.

National peer-reviewed Conferences

[18] F. Bour, B. Clément and G. Scherer. 'Tail Modulo Cons'. In: JFLA 2021 - Journées Francophones des Langages Applicatifs. Saint Médard d'Excideuil, France, 6th Apr. 2021. URL: https://hal.inria.fr/hal-03146495.

[19] N. Chataing, C. Noûs and G. Scherer. 'Déboîter les constructeurs'. In: Journées Francophones des Langages Applicatifs. Saint-Médard-d'Excideuil, France, 2nd Feb. 2022. URL: https://hal.inria.fr/hal-03510931.

Conferences without proceedings

[20] N. Chataing and G. Scherer. 'Unfolding ML datatype declarations without loops'. In: ML Family Workshop. online, South Korea, 27th Aug. 2021. URL: https://hal.inria.fr/hal-03510898.

[21] O. Martinot and G. Scherer. 'Frozen inference constraints for type-directed disambiguation'. In: ML Family Workshop. online, South Korea, 27th Aug. 2021. URL: https://hal.inria.fr/hal-03510890.

[22] W. Ouedraogo, D. Ilik and L. Straßburger. 'Demo Paper: Coqlex, an approach to generate verified lexers'. In: ML 2021-ACM SIGPLAN Workshop on ML. Online event, United States, 26th Aug. 2021. URL: https://hal.inria.fr/hal-03470713.

Scientific book chapters

[23] M. Acclavio, D. Catta and L. Straßburger. 'Game Semantics for Constructive Modal Logic'. In: *Automated Reasoning with Analytic Tableaux and Related Methods*. Vol. 12842. Lecture Notes in Computer Science. Springer International Publishing, 30th Aug. 2021, pp. 428–445. DOI: 10.1007/978-3-030-86059-2_25. URL: https://hal.inria.fr/hal-03369819.

Reports & preprints

[24] M. Acclavio, D. Catta and L. Straßburger. *Towards a Denotational Semantics for Proofs in Constructive Modal Logic*. 18th Apr. 2021. URL: https://hal.archives-ouvertes.fr/hal-03201439.

[25] O. Bodini, A. Singh and N. Zeilberger. *Asymptotic Distribution of Parameters in Trivalent Maps and Linear Lambda Terms*. 20th Dec. 2021. URL: https://hal.archives-ouvertes.fr/hal-03495894.

[26] C. Liang and D. Miller. *Focusing Gentzen's LK proof system*. 30th Nov. 2021. URL: https://hal.archives-ouvertes.fr/hal-03457379.

[27] M. Manighetti and D. Miller. *Computational logic based on linear logic and fixed points*. 18th Feb. 2022. URL: https://hal.inria.fr/hal-03579451.

11.3 Other

Educational activities

[28] W. Heijltjes and L. Straßburger. 'From Proof Nets to Combinatorial Proofs - A New Approach to Hilbert's 24th Problem'. École thématique. Netherlands, 2nd Aug. 2021. URL: https://hal.inria.fr/hal-03316571.

11.4 Cited publications

[29] K. Chaudhuri. 'Subformula Linking as an Interaction Method'. In: *4th Conference on Interactive Theorem Proving*. Vol. 7998. Lecture Notes in Computer Science. Rennes, France: Springer, July 2013, pp. 386–401. DOI: 10.1007/978-3-642-39634-2_28. URL: https://hal.inria.fr/hal-00937009.