Summary of PhD dissertation: «Une étude logique du contrôle appliquée à la programmation fonctionnelle et logique»*

Alexis Saurin

Defended on the 30th of september 2008

Introduction

Proof theory and Programming languages theory are strongly related areas of study. Recent developments in proof theory have led to major advances in the theory of programming languages. The modelling of computation using proofs impacted deeply the foundational studies of programming languages as well as many of their practical issues by providing formal tools to analyze programs properties. Declarative programming languages have been related mainly in two ways to the mathematical theory of proofs: on the one hand, the "computation as proof normalization" paradigm provided a foundation for functional programming languages through the well-known Curry-Howard correspondence [How80]. On the other hand the "computation as proof search" paradigm stands as a foundation for logic programming: the computation of a program is the search for a proof in some deductive system.

- From the "computation as proof normalization" viewpoint, cut-elimination in natural deduction (and more recently in sequent calculus) provided a particularly strong and useful theoretical foundation to functional programming by drawing a correspondence between (i) logic formulas and data/program types, (ii) intuitionistic deduction and program, (iii) cut-reduction and computation step and finally (iv) cut-free proof and results of the computation, or values.

  This also allowed to transfer proof-theoretical results towards theory of programming and was an opportunity to develop formal tools to analyze precisely those languages by using well-established and powerful logical methods. Crossing the bridge between proofs and computations in the other direction was also very fruitful by renewing many questions proof-theorists were considering with respect to cut-elimination results; for isntance, questions related to the efficiency of those proof transformations

*English title: A Logical Study of Control, Applied to Functional Programming and Logic Programming.

emerged, directly related to considerations connected to program complexity.

- In another kind of thoughts, questions related to the (automated, mechanical) search for proofs have originated a programming paradigm in the seventies, namely logic programming, Prolog being its most well-known example. If the automated search for proofs could originally be motivated by the will to help the mathematician in its activity, proof search as a model of computation is much more interested in the very structure of proofs, the dynamics of proof construction and the way this may serve as a universal model of computation.

  In the field of "computation as proof search", one saw also contributions crossing the logic/computation bridge in both directions, in addition to the fact that the very model of computation was historically inspired by logic. For instance, the introduction of the notion of uniform proofs [MNPS91] or focussed proofs in linear logic [And92] which are fundamental structures in sequent calculus proof theory (very much related to Kleene and Curry works on permutabilities of inference rules in the early fifties [Kle52, Cur52]) were inspired by the will to better understand the procedurality of proof search.

The two programming paradigms mentionned above, functional programming and logic programming, are also interesting because their relationships with logic is of a different kind and that this difference finds its root with the notion of cut first introduced by Gentzen [Gen69]:

- with functional programming, or "computation as proof normalization", a program is a proof and the dynamics of computation is to be found in the elimination of the cuts (both in the frameworks biult on natural deduction or on sequent calculus). On the other hand, the result of the computation is a cut-free proof. This paradigm, which is the one of (typed) λ-calculus, relies on the cut-elimination viewpoint of Gentzen's Hauptsatz;

- with logic programming, or "computation as proof search", a program is a collection of formulas (a sequent) and a computation step is the reduction of a sequent into a (set of) sequent(s) by applying an inference rule (bottom-up proof construction). Here, the dynamics of computation is to be found in the search process of a cut-free proof and the final result of a computation is proof. A proof search has a chance to be effective only if ones never used the cut inference rule which is the only inference rule in sequent calculus that truly requires the imagination of the mathematician which is reflected in the non-analycity of the cut-rule. In this way too, proof search relies in a fundamental way on Gentzen's Hauptsatz, but in the cut-admissibility viewpoint of the Hauptsatz: the result of a computation is actually a cut-free proof.

Another interesting difference between those two paradigms is the historical connection they have with logic: whereas logic programming originated from logical consideration, the connection between logic and functional programming came afterwards.

Even though the two paradigms have much to do with Gentzen's Hauptsatz and despite the fact that the results of a computation are similar cut-free proofs, it is difficult to draw a link between their dynamics.

The connections between logic and computation actually becomes much more complex when certain components of real programming languages are concerned.

Most of programming languages allow the programmer to user constructions to control the execution flow:

- in functional programming one may think of exceptions management (constructions such as `try ... with` and `raise`), or managing the continuation of the computation (one may think of the `call/cc` for instance);
- in logic programming one may think of backtracking mechanisms, cut predicate (`!` in Prolog), or other extra-logical predicates that may alter the backtracking behaviour.

One just show the example of `call/cc`:

$$\begin{array}{lcl} E[(\texttt{call/cc})\lambda k.t] & \longrightarrow & E[t\,\{\star_E/k\}] \\ E[(\star_{E'})t] & \longrightarrow & E'[t] \end{array}$$

where E is the computational environment. The effect of `call/cc`, by the creation of $\star_E$ can be understood as a reification of the computational environment within the programming language.

This very fact is part of the explanation why it has revealed to be so difficult to understand logically the control constructions. However, control constructions are often causes of errors in programming and it is thus particularly important that the formal analysis of programs can treat these elements. The difficulties in providing a logical accound of control in programming laguages can be understood as follows:

- it took over twenty years to extend the Curry-Howard correspondence from intuitionistic logic to classical logic, this extension being the key of a start of logical understanding of control in those languages (see griffin's work [Gri90] or Parigot's $\lambda\mu$-calculus [Par92]).
- there is no satisfying solution to adress backtracking and cut predicate proof-theoretically in proof search.

The research we present in this dissertation are at the border of proof theory and programming languages theory and we present contributions both in pure theory and in theoretical computer science.

The dissertation is organized in three parts:

- the first part is dedicated to the "computation as proof normalization" approach. We study $\Lambda\mu$-calculus, an extension of Parigot's $\lambda\mu$-calculus.
- the second part is dedicater to the "computation as proof search" approach with contributions to the understanding of focalization in linear logic.

- the last and third part suggests a radically new look by introducing a framework for "proof search by cut-elimination" in which we can connect the two approaches of the Hauptsatz. This study is carried out in the setting of Ludics [Gir01].

There are several themes that are influencial all along the dissertation:

- First and foremost, the understanding of control mechanism is our first motivation for this study;
- Separation property is originally a topological property that entered the field of proof theory and computation with the work of Böhm on λ-calculus in the late sixties [Bö68]. This is certainly an influential notion: being our starting point on $\lambda\mu$-calculus, separation is also an essential property in the development of Ludics theory. Moreover, separation is related to canonicity of formal system and thus with focalization property (which is actually one of the key of the Separation theorem in Ludics);
- The understanding of the relationships between the cut-elimination and proof-search apporaches is also one of the deep structuring ideas of our work.

Introductory Part

Chapter 1: Notions de théorie de la démonstration

Title: Introduction to proof theory

Chapter 2: Notions de λ-calcul et de $\lambda\mu$-calcul

Title: Introduction to λ-calculus and $\lambda\mu$-calculus

In these two introductory chapters, we provide the reader with the necessary background on proof theory (especially sequent calculus theory) and Church's λ-calculus [Chu41]. In the first chapter, we emphasize the study of sequent calculus (classical sequent calculus LK, intuitionistic sequent calculus LJ and linear sequent calculus LL) and we emphasize the presentation of several key-concepts: constructive proofs, cut rule and cut-elimination, Kleene/Curry permutabilities of inferences, and proof search through a presentation of Gödel completeness theorem for first-order LK. The second chapter, dealing with λ-calculus, is also introducing Parigot's $\lambda\mu$-calculus [Par92], an extension of λ-calculus that allows to carry the Curry-Howard correspondence between proofs and programs to classical logic. This second chapter also contains a detailed account of separation property in pure λ-calculus (also known as Böhm theorem [Bö68]) which will be a crucial notion in the rest of the dissertation.

Part I: $\Lambda\mu$-calcul

Title: $\Lambda\mu$-calculus

In this part, we investigate $\Lambda\mu$-calculus, an extension of Parigot's $\lambda\mu$-calculu that satisfies a separation property. We develop the meta-theory of the calculus (confluence, simple types) and propose an analysis of computations in $\Lambda\mu$-calculus thanks to extensions of proof nets [Gir87, Lau03]. Finally, we develop a stream interpretation for $\Lambda\mu$-calculus and related this calculus with delimited control operators.

Chapter 3: de $\lambda\mu$ à $\Lambda\mu$

Title: From $\lambda\mu$ to $\Lambda\mu$

David and Py [DP01] proved that separation property which holds in λ-calculus is not satisfied in Parigot's $\lambda\mu$-calculus. Indeed, they could witness two canonical $\lambda\mu$-normal forms which are not $\lambda\mu$-equivalent but which cannot be separated by any context.

In this chapter, we start by studying the non-separation result of $\lambda\mu$-calculus and by discussing which heuristics can be adopted when separation fails in a calculus such a λ-calculus or $\lambda\mu$-calculus. Our analysis then leads us to propose an extension of Parigot's $\lambda\mu$-calculus, which we call $\Lambda\mu$-calculus, for which separability is recovered. Indeed, $\Lambda\mu$-calculus will have more separating contexts than $\lambda\mu$-calculus. This new calculus is introduced in the last part of the chapter while the proof of separation is left for the following chapter.

Chapter 4: Théorème de Böhm pour le $\Lambda\mu$-calcul

Title: Böhm Theorem for $\Lambda\mu$-calculus

This chapter is dédicated to the proof of the analogous of Böhm theorem in $\Lambda\mu$-calculus. Our proof is inspired by Joly's proof [Jol00] for λ-calculus rather than by more standard proofs [Kri90]. The key element in the separation proof is the more liberal syntax of $\Lambda\mu$-calculus which allows to consider terms of the form:

$$\langle t, u\rangle_k = \mu\alpha_1 \dots \mu\alpha_k.\lambda x.((x)(t)\alpha_1 \dots \alpha_k)(u)\alpha_1 \dots \alpha_k$$

which are used in the separation process.

We also discuss other heuristics that could be used in order to obtain separation in $\lambda\mu$-calculus.

Chapter 5: Confluence du $\Lambda\mu$-calcul

Title: Confluence theorem for $\Lambda\mu$-calculus

In this chapter, we establish a confluence theorem for $\Lambda\mu$-calculus. It is indeed important to have confluence for the separation of the previous chapter to be significant.

The proof uses commutations arguments and a labelling technique (a potential of *fst*-expansion) which ensure that the *fst*-reduction terminates.

A by-product of our proof is a simplified proof of confluence for $\lambda\mu\eta$-calcul. It is also an occasion to develop the syntactical meta-theory of $\Lambda\mu$-calculus.

Chapter 6: $\Lambda\mu$-calcul simplement typé

Title: Simply typed $\Lambda\mu$-calculus

In chapter 6, we study type systems for $\Lambda\mu$-calcul. We first consider a type system which uses a classical logic typing derived from the one for $\lambda\mu$-calculus.

However, this approach is questionnable by considering that whole classes of terms, among which are terms which are crucial in the proof of separation, cannot be typed. We thus propose another type system, an extended type system, $\Lambda_{\mathcal{S}}$, which allows to type more $\Lambda\mu$-terms. In particular, $\Lambda_{\mathcal{S}}$ allows to type terms of the shape which is used in the proof of separation of chapter 4 and that could not be typed in the classical type system. This type system is peculiar in several aspect and in particular in the fact that there is an equivalence relation on the grammar of types corresponding to an associativity property on type constructors and that express the fact that some terms can both be applied to a term and a stream variable. We study relationships between $\Lambda_{\mathcal{S}}$ and a type system independantly developped by Herblin and Ghilezan [HG08] which does not use associativity.

Finally, we pursue the investigation of $\Lambda_{\mathcal{S}}$ by proving that the standard theorems hold, namely type preservation and strong normalization of the typed calculus and we conclude by proposing some remarks on a type system with second-order quantification.

Chapter 7: Une analyse du $\Lambda\mu$-calcul via des réseaux

Title: An analysis of $\Lambda\mu$-calculus Computations Through Proof-Nets

We study in this chapter the reductions of $\Lambda\mu$-calculus via an encodingin a particular class of proof nets, Streams Associative Nets (SANE). One first define Streams Associative Nets, which are intermediate nets in between usual $MELL$proof nets [Gir87] and Laurent's polarized nets [Lau02]. The connection between $MELL$ nets and polarized nets is obtained by an associativity relations already mentionned above about the type system $\Lambda_{\mathcal{S}}$. Here, it will consist in a real associvity rule between two sorts of $\parr$ connective. We study properties of reductions in Streams Associative Nets which is actually confluent (in a way which is more local than $\Lambda\mu$-calculus) and we then encode pure $\Lambda\mu$-calculus using a process which is similar to the one used to encode pure λ-calcul using $MELL$nets; this encoding allows us to obtain a simulation result of $\Lambda\mu$-calcul by Streams Associative Nets. We finally prove a separation theorem for Streams Associative Nets.

Chapter 8: $\lambda\mu$-calculs, streams et contrôle

Title: $\lambda\mu$-calculi, Streams and control

In this last chapter dealing with $\Lambda\mu$-calculus, we discuss several results the relationships between $\Lambda\mu$-calculus and other $\lambda\mu$-calculi, results which are built on the results of the previous chapters.

First, we shall compare different version of call-by-name $\lambda\mu$-calculi that we encoutered throughout the dissertation: $\lambda\mu$, $\lambda\mu\eta$, $\lambda\mu\epsilon$ and $\Lambda\mu$. We also shall

develop more formally the stream interpretation of $\Lambda\mu$-calcul and of the parallelism with delimited controlvia the correspondence with $\lambda\mu\widehat{\mathsf{tp}}$. In particular, this is established via the definition of an abstract machine for $\Lambda\mu$, that we call $\Lambda\mu$-KAM, and a variant of λ-calculus which possess a construction of streams, the $\Lambda\mathcal{S}$-calculus.

Part II: Focalisation en Logique Linéaire

Title: Focalization in Linear Logic

This part is dedicated to the investigation of the focalization property in Linear Logic which is a fundamental theorem of linear logic with many applications and most importantly, to computation-as-proof-serach and to Ludics. We first propose a proof of the focalization theorem that is structured in such a way that it can be esaily extended to broader setting; this is achieved by defining an abstraction on linear logic sequent proofs, focalization graphs. Focalization graphs naturally lead us to introducing an generalization of the focalization disciplin, which we call multifocalization and for which we study canonicity properties by comparing this approach to MLL proof nets.

Chapter 9: Une preuve modulaire de la focalisation

Title: A Modular Proof of Focalization

In this chapter, we propose a new proof of Andreoli's focalization theorem in linear logic [And90, And92]. Our method is modular and relies on a precise analysis of the properties of permutabilities of inference rules in linear logic.

The proof is built on the notion of focalization graphs, which are an abstraction of linear logic proof retaining only the relevant information to determine wht can be taken as a focus. Using this abstraction, it becomes simple to carry the proof. moreover, one can develop a detailed study of how to polarize the atomic biases in order to treat them in a focussed way.

We illustrate the modularity of this proof in two ways: we first establish the result for MALL and then extend it full linear logic. On the other hand, we explain how to obtain a focalization result for Elementary Linear Logic.

Chapter 10: Multifocalisation

Title: Multifocalization

In this chapter we consider an extension to the focussed system which is naturally suggested by the use of focalization graphs: multifocalization. The basic idea is to replace the using focussing disciplin by a disciplin in which one may focuss on several formulas:

$$\frac{\vdash F_1, \dots F_k : \Delta \Downarrow F_1^{i_1}, \dots F_k^{i_k}, F'_1, \dots F'_l}{\vdash F_1, \dots F_k : \Delta, F'_1, \dots F'_l \Uparrow \cdot} \; MultiFoc \quad i_j \geq 0$$

We then prove that it is possible to define a notion of maximality among multifocussed proofs which provides proofs with canonicity. We give an illustra-

tion of the previous fact by enlightening the relationships between maximally multifocussed proofs and MLL proof nets.

Part III: Programmation Ludique

Title: Ludics Programming

We present a setting for proof search by cut-elimination: proof search will not anymore be guided by a seuqent calculus and an encoding of a program as a sequent, but as a series of tests that will constrain the construction of a proof object which is required to interact with all the tests (that is the cut-elimination have to normlize). This investigation is developped in Ludics [Gir01] and this approach by proofs and counter-proofs allows us to treate interactively, that is logically, the backtracking phenomenon.

Chapter 11: Introduction à la Ludique

Title: Introduction to Ludics

We introduce in this chapter the basic notion a Ludics, an interactive theory introduced by Girard in 2001 [Gir01, Gir06]. Ludics is built on an abstraction of MALL focussed proofs, the designs, which are equipped with an orthogonality relation. The equivalents of formulas, in Ludics, are behaviours, that is sets of designs which are closed by bi-orthogonality.

Chapter 10: Vers une Programmation Ludique: Recherche de Preuve Interactive

Title: Towards Ludics Programming: Interactive Proof Search

We propose in this chapter a framework for a «interactive proof search» that is a proof search which is not guided by a sequent and by search instructions anymore, but guided by an interaction: this is proof search by cut-elimination.

We first motivated our approach by the will to obtain a uniform approach to proof search and then give informal examples in a calculus which is obtained by slightly modifying MALL sequent calculus before moving to Ludics and defining an abstract machine for an interactive construction of designs, the SLAM.

In particular, using the SLAM allors to treat the backtracking in an interactive way, ie. by using only proof objects.

References

[And90] Jean-Marc Andreoli. *Proposition pour une synthèse des paradigmes de la programmation logique et de la programmation par objets.* Thèse de doctorat, Université Paris VI, June 1990.

[And92] Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):297–347, 1992.

[Bö68] Corrado Böhm. Alcune proprietà delle forme $\beta\eta$-normali nel λK-calcolo. *Publicazioni dell'Istituto per le Applicazioni del Calcolo*, 696, 1968.

[Chu41] Alonzo Church. *The Calculi of Lambda-Conversion*. Princeton University Press, 1941.

[Cur52] Haskell B. Curry. The permutability of rules in the classical inferential calculus. *Journal of Symbolic Logic*, 17(4):245–248, 1952.

[DP01] René David and Walter Py. $\lambda\mu$-calculus and Böhm's theorem. *Journal of Symbolic Logic*, 2001.

[Gen69] Gerhard Gentzen. Investigations into logical deductions. In M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131. North-Holland, Amsterdam, 1969.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[Gir01] Jean-Yves Girard. Locus solum. *Mathematical Structures in Computer Science*, 11:301–506, 2001.

[Gir06] Jean-Yves Girard. *Le Point Aveugle: Cours de logique. Tome 1, Vers la perfection; Tome 2, Vers l'imperfection*. Hermann, 2006.

[Gri90] Timothy Griffin. A formulae-as-types notion of control. In *Principles of Programming Languages*. IEEE Computer Society Press, 1990.

[HG08] Hugo Herbelin and Silvia Ghilezan. An approach to call-by-name delimited continuations. In *Principles of Programming Languages*. ACM Sigplan, January 2008.

[How80] William A. Howard. The formulae-as-type notion of construction, 1969. In J. P. Seldin and R. Hindley, editors, *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus, and Formalism*, pages 479–490. Academic Press, New York, 1980.

[Jol00] Thierry Joly. *Codages, séparabilité et représentation de fonctions en λ-calcul simplement typé et dans d'autres systèmes de types*. Thèse de doctorat, Université Paris VII, January 2000.

[Kle52] Stephen Cole Kleene. Permutabilities of inferences in Gentzen's calculi LK and LJ. *Memoirs of the American Mathematical Society*, 10:1–26, 1952.

[Kri90] Jean-Louis Krivine. *Lambda-calcul: Types et Modèles*. Masson, 1990.

[Lau02] Olivier Laurent. *Étude de la polarisation en logique*. Thèse de doctorat, Université Aix-Marseille II, March 2002.

[Lau03] Olivier Laurent. Polarized proof-nets and $\lambda\mu$-calculus. *Theoretical Computer Science*, 290(1):161–188, 2003.

[MNPS91] Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

[Par92] Michel Parigot. $\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction. In *Proceedings of the 1992 International Conference on Logic Programming and Automated Reasonning*, volume 624 of *Lecture Notes in Computer Science*, London, UK, 1992. Springer-Verlag.