UNIVERSITÉ PARIS 13
SYNTHÈSE DE TRAVAUX PRÉSENTÉS POUR OBTENIR LE DIPLÔME
HABILITATION À DIRIGER LES RECHERCHES
SPÉCIALITÉ : INFORMATIQUE

Patrick BAILLOT

Logique Linéaire, Types et Complexité Implicite

soutenue le 11 mars 2008

Rapporteurs :

Pierre-Louis	CURIEN
Martin	HOFMANN
C.-H. Luke	ONG

Jury :

Pierre-Louis	CURIEN	CNRS/Université Paris 7
Christophe	FOUQUERÉ	Université Paris 13
Jean-Yves	GIRARD	CNRS/Université Aix-Marseille 2
Martin	HOFMANN	LMU Munich
Jean-Yves	MARION	LORIA INPL Nancy
C.-H. Luke	ONG	Oxford University
Simona	RONCHI DELLA ROCCA	Università di Torino
Jacqueline	VAUZEILLES	Université Paris 13

English title : Linear logic, Types and Implicit Computational Complexity

Laboratoire d'Informatique de Paris-Nord (LIPN), UMR 7030 CNRS,
Institut Galilée, Université Paris 13.

Table des matières

Remerciements

Je tiens tout d'abord à remercier Pierre-Louis Curien, Martin Hofmann et Luke Ong d'avoir accepté d'être les rapporteurs de cette habilitation à diriger les recherches, ainsi que Christophe Fouqueré, Jean-Yves Girard, Jean-Yves Marion, Simona Ronchi Della Rocca et Jacqueline Vauzeilles, d'avoir bien voulu être membres du jury.

Je dois un merci tout particulier aux collaborateurs avec lesquels j'ai mené les travaux de la période couverte par ce mémoire : Kazushige Terui et Ugo Dal Lago, avec qui j'ai entretenu de fructueux échanges entre Paris, Tokyo et Bologne ; Virgile Mogbil ; Vincent Atassi ; Paolo Coppola ; Marco Pedicini.

Ce mémoire retrace l'essentiel de mon parcours scientifique depuis la fin de ma thèse, et plusieurs personnes ont joué un rôle important dans cet itinéraire. Je souhaite remercier Thomas Ehrhard et Laurent Régnier, pour les échanges scientifiques que j'ai eus avec eux et pour leur soutien, ainsi que pour le projet GEOCAL qui a constitué un cadre motivant et fédérateur. Roberto Amadio, en m'intégrant en tant qu'ATER dans son équipe en 2000, et par la suite en coordonnant les projets AS *Mobilité* et surtout CRISS, m'a ouvert à de nouvelles questions et a toujours été de bon conseil. Jean-Yves Girard par ses travaux a fourni le point de départ des recherches présentées ici, en offrant des idées toujours riches de perspectives et d'inspiration ; je le remercie aussi pour son soutien constant. Vincent Danos a continué à me transmettre sa curiosité scientifique et m'a donné un exemple de groupe de travail dynamique. Je suis également reconnaissant à Samson Abramsky et au réseau européen LINEAR pour le postdoctorat effectué à l'université d'Édimbourg en 1999-2000.

Ma gratitude va en outre à Jacqueline Vauzeilles et Christophe Fouqueré qui m'ont accueilli dans le cadre convivial de l'équipe LCR du LIPN, et ont encouragé mes recherches et mes initiatives. Je les remercie notamment de m'avoir incité à présenter cette habilitation.

J'ai une pensée aussi pour les participants du groupe NOCoST, avec qui j'ai toujours plaisir à travailler, notamment Vincent Atassi, Pierre Boudes, Daniel de Carvalho, Paulin Jacobé de Naurois, Olivier Laurent, Damiano Mazza, Virgile Mogbil, Jean-Yves Moyen.

Je voudrais adresser un remerciement à toutes les personnes avec qui j'ai eu des discussions et des échanges scientifiques au cours de cette période, notamment : Guillaume Bonfante, Marie-Renée Fleury-Donnadieu, Claudia Faggian, Marco Gaboardi, Daniel Hirshchkoff, Martin Hyland, Jean-Baptiste Joinet, Yves Lafont, Kazem Lellahi, Jean-Vincent Loddo, Harry Mairson, Simone Martini, François Maurel, Micaela Mayero, Paul-André Melliès, Anass Nagih, Michele Pagani, Romain Péchoux, Luca Roversi, Paul Ruet, Lorenzo Tortora de Falco, Sophie Toulouse, Paolo Tranquilli, Glynn Winskel. Mes excuses aux personnes que j'ai oubliées...

Je voudrais exprimer ma reconnaissance à l'équipe administrative et technique du LIPN, pour leur disponibilité et leur aide quotidienne, et notamment à Brigitte Guéveneux, Jacqueline Giraud et Antonia Wilk.

Merci aux membres de l'équipe LCR et plus généralement du LIPN, qui en font un lieu agréable

et amical.

Enfin j'ai une pensée particulière pour ma famille, pour leur attention et leur affection, ainsi que pour Sophie, pour ses relectures, sa patience et son soutien si précieux.

Articles présentés

Les travaux présentés (disponibles en Annexe) sont les suivants, classés ici par thèmes :

1. Sémantique dénotationnelle de logiques light :
   1 P. Baillot. *Stratified Coherence Spaces : a Denotational Semantics for Light Linear Logic. Theoretical Computer Science*, 318 (1-2), pp.29-55, Elsevier, 2004.
2. Correspondance de Curry-Howard pour logiques light :
   2 P. Baillot, V. Mogbil. *Soft lambda-calculus : a language for polynomial time computation.* Proceedings *Foundations of Software Science and Computation Structures* (FoSSaCS'04), volume 2987 of LNCS, pp.27-41, Springer, 2004.
   3 U. Dal Lago, P. Baillot. *On Light Logics, Uniform Encodings and Polynomial Time. Mathematical Structures in Computer Science*, 16(4), pp. 713-733, Cambridge University Press, 2006.
3. Les logiques light comme systèmes de types : premières méthodes de typage :
   4 P. Baillot. *Type Inference for Light Affine Logic via Constraints on Words. Theoretical Computer Science*, 328(3) :289-323, Elsevier, december 2004.
4. Conception d'un système de types light et inférence de type efficace :
   5 P. Baillot, K. Terui. *Light types for polynomial time computation in lambda calculus* (long version). 35 pp. 10/2007. Soumis pour publication. Une version conférence est parue dans Proceedings *International Symposium on Logic in Computer Science* (LICS'04), pp. 266-275, IEEE Computer Society Press, 2004.
   6 P. Baillot, K. Terui. *A feasible algorithm for typing in Elementary Affine Logic.* Proceedings 7th *International Conference on Typed Lambda-calculi and Applications* (TLCA'05), LNCS 3461, pp.55-70, Springer, 2005.
   7 V. Atassi, P. Baillot, K. Terui. *Verification of Ptime Reducibility for system F Terms : Type Inference in Dual Light Affine Logic. Logical Methods in Computer Science*, 3(4 :10), Special issue on CSL'06, 2007. (Une version conférence était parue auparavant dans les actes de CSL'06).
5. Logiques light et réduction optimale du λ-calcul :
   8 P. Baillot, P. Coppola, U. Dal Lago. *Light Logics and Optimal Reduction : Completeness and Complexity.* Proceedings *International Symposium on Logic in Computer Science* (LICS'07), pp 421–430. IEEE Computer Society Press, 2007.

Ces 8 articles correspondent à 4 articles publiés dans des revues internationales et à 5 articles publiés dans les actes de conférences internationales à comité de lecture (un article de conférence est remplacé par la version de revue correspondante).

Introduction (en Français)

Logique. Mon travail porte essentiellement sur l'étude des liens entre logique et programmes, par le biais des preuves formelles. La théorie de la démonstration est la branche de la logique étudiant les preuves en tant qu'objets mathématiques. Dans les années 1970, la *correspondance de Curry-Howard* a ouvert la voie à une étude calculatoire des preuves en établissant une bijection entre preuves et programmes. Ces derniers sont écrits en λ-calcul, un langage introduit par A. Church dans les années 1930 et qui peut être vu comme un langage de programmation simplifié, à la base des langages fonctionnels actuels comme CAML.

À travers Curry-Howard, l'exécution des programmes du λ-calcul correspond à une réécriture des preuves, appelée *réduction* ou *élimination des coupures*. Cette correspondance est la clé de voûte de nombreuses recherches ultérieures, d'une part sur les logiciels de preuve assistée et d'autre part sur l'étude des langages de programmation fonctionnels. Dans le premier domaine, par exemple, elle permet l'extraction de programmes certifiés corrects à partir de preuves, développées au moyen d'outils comme le logiciel Coq. Dans le deuxième domaine, elle permet une analyse logique des mécanismes de programmation, comme le poymorphisme, la gestion d'exceptions ou les opérateurs de contrôle.

La correspondance de Curry-Howard, dans sa version initiale, s'applique à la logique intuitionniste, un système sans le principe du tiers-exclu ("A ou non-A"). Un des outils principaux pour l'étude du λ-calcul et de la logique intuitionniste est la *sémantique dénotationnelle* ; elle associe à une preuve-programme un invariant du calcul : la fonction représentée par le programme, ou plus généralement un morphisme d'une *catégorie*, dont les objets, eux, correspondent aux formules.

Une étape importante dans l'analyse de la correspondance de Curry-Howard est l'introduction de la *logique linéaire* (LL) en 1987 par J.-Y. Girard, mise à jour grâce à des travaux de sémantique dénotationnelle. Son idée est que les opérations de duplication et d'effacement d'hypothèse sont des règles logiques à part entière : en logique linéaire elles sont donc contrôlées à l'aide de nouveaux connecteurs (ou modalités) !, ?, appelés *exponentielles*. Ce système doit être compris plus comme une décomposition de la logique usuelle que comme une nouvelle logique *stricto sensu*. Par exemple, des traductions des logiques intuitionniste et classique dans la logique linéaire permettent d'étudier finement la dynamique (réduction) de ces dernières. De fait, la logique linéaire est aujourd'hui un des principaux outils fondamentaux dans des domaines comme le λ-calcul, la théorie de la démonstration de la logique classique ou la sémantique dénotationnelle

Un produit dérivé important de la logique linéaire est la syntaxe des *réseaux de preuves*, une représentation des preuves par des graphes. Les réseaux de preuves sont obtenus en laissant de côté certaines informations de séquentialité superflues dans les preuves. Cette approche permet des simplifications dans l'étude de la réduction, vue désormais comme une procédure de réécriture de graphes, et apporte un point de vue géométrique sur cette dernière. Les réseaux de preuves ont aussi ouvert la voie à la *Géométrie de l'interaction* [Gir88], un programme de recherche lancé par

Girard et visant à interpréter la normalisation des preuves comme un flot d'information dans les réseaux de preuves. Par exemple, un modèle possible de géométrie de l'interaction consiste à voir le réseau comme un automate agissant sur des jetons, et l'exécution consiste alors à calculer les chemins empruntés par les jetons (voir par exemple [DHR96]).

Complexité. Nous avons beaucoup mentionné le λ-calcul mais un autre modèle de calcul bien connu est celui des *machines de Turing*, qui permet de représenter les mêmes fonctions que le λ-calcul. La *complexité algorithmique* classifie les programmes sur machine de Turing en fonction du temps ou de l'espace nécessaires pour leur évaluation, rapportés à la taille de l'argument ou entrée du programme (par exemple une liste binaire). Les programmes *Ptime* en particulier sont ceux qui terminent en temps polynomial par rapport à la taille de l'entrée. La *théorie de la complexité* classifie de la même manière les *fonctions* (sur les listes binaires notamment) ou les problèmes (fonctions à valeurs dans $\{0,1\}$) : une fonction est dite *calculable en temps* $\phi(n)$ s'il existe au moins un programme qui la représente et qui termine en temps au plus $\phi(n)$ sur les entrées de taille n. On définit ainsi la classe des fonctions (resp. problèmes) calculables en temps polynomial FP (resp. P), qui est généralement considérée comme la classe des fonctions calculables en pratique (du moins en première approximation). Rappelons qu'un des plus importants problèmes ouverts de l'informatique théorique, P ?=NP, concerne la question de savoir si la classe P est égale à la classe NP des problèmes dont les solutions sont vérifiables en temps polynomial.

Dans cette présentation, il sera plutôt question de la complexité des programmes. Si la complexité algorithmique permet de définir théoriquement les classes de complexité, elle ne nous dit pas *comment* obtenir des programmes Ptime ni, un programme étant donné, comment vérifier s'il s'agit d'un programme Ptime, ou encore comment le *certifier*. Or la complexité est un aspect essentiel des programmes en pratique, car souvent la terminaison après un temps rédhibitoire peut être aussi problématique qu'une non-terminaison. Ces questions sont donc importantes pour des raisons d'efficacité et de sûreté. Or le problème général, étant donné un programme, de décider par exemple s'il est de complexité Ptime est un problème indécidable. Comme pour beaucoup de problèmes d'analyse statique de programmes, si on veut automatiser une telle vérification, on ne peut donc espérer au mieux qu'une méthode donnant une condition suffisante : si le programme est validé par la méthode, alors il est de complexité Ptime, sinon une analyse plus fine est nécessaire. Notons toutefois qu'une telle propriété nous apporte alors en général de l'information sur un nombre infini d'exécutions du programme (l'exécution de ce programme appliqué à toutes les entrées possibles, par exemple toutes les listes).

Le problème de trouver comment générer un programme Ptime est lié à la question de caractériser la classe des fonctions FP, qui a été étudiée par la *complexité implicite* (CI) : celle-ci cherche à définir des langages de programmation ou des calculs pour lesquels par construction tous les programmes ont une certaine borne de complexité, ou alternativement, dans le cadre d'un langage de programmation général, des *critères* pour déterminer si un programme admet une borne de complexité. Initialement la CI a fourni des caractérisations de Ptime et d'autres classes de complexité (Pspace, Logspace ...) qui ne dépendent pas d'un modèle de machine particulier et ne mentionnent pas explicitement de borne sur les ressources (temps, espace mémoire). La borne de complexité est obtenue par contre comme une conséquence de certaines restrictions soit sur des règles logiques, soit sur les structures de contrôle dans le programme ou sur les opérations autorisées sur les données. Ce genre de caractérisation pourrait ainsi permettre de mieux comprendre la nature et les fondements du calcul à ressources bornées.

Cette direction de recherche a été entamée par Bellantoni et Cook [BC92] ainsi que Leivant

[Lei94] qui ont donné dans les années 90 des caractérisations de FP dans le cadre de la récursion primitive en utilisant une discipline de *récursion sûre* (ou *ramification*), consistant à limiter la manière dont les récursions peuvent être imbriquées. Cette ligne de travaux sur le contrôle de la récursion a été continuée avec le λ-calcul [LM93], adaptée à d'autres classes de complexité (voir par exemple [LM94]) et étendue au cadre de calculs avec des types d'ordre supérieur [Hof00, BNS00]. Une autre branche de travaux est celle de N. Jones, dans le cadre de la programmation fonctionnelle, qui a permis de caractériser des classes de complexité par des restrictions sur la structure des programmes ou les opérations autorisées ([Jon99]). Tous ces systèmes permettent de caractériser des classes de complexité de *fonctions* mais peuvent être assez restrictifs de point de vue de la programmation : en effet certains algorithmes Ptime ne sont pas validés, même si la *fonction* représentée peut être calculée par un autre algorithme qui, lui, est validé (mais est souvent moins naturel). Ceci constitue le problème de l'*expressivité intensionnelle* ; il a conduit à étudier des systèmes de CI qui sont plus flexibles pour la programmation, comme les systèmes de types pour le calcul en place (*non-size-increasing* computation) [Hof03] et l'approche des *quasi-interprétations* dans le cadre des systèmes de réécriture [MM00, BMM05, BMM07].

Les recherches en logique linéaire ont rejoint la complexité implicite assez tôt, vers 1992, grâce à l'idée que le contrôle de la duplication dans les preuves-programmes est un moyen de limiter la complexité de leur réduction : en considérant des variantes de la logique linéaire avec des versions "faibles" des modalités, on obtient ainsi des systèmes pour lesquels la réduction des preuves est Ptime, et qui permettent de représenter toutes les fonctions de FP : BLL [GSS92], LLL [Gir98], SLL [Laf04]. Un autre système, ELL [Gir98], correspond à la complexité dite en temps *élémentaire* (c'est-à-dire borné par une fonction $2^{2^{\cdot^{\cdot^{2^n}}}}$ pour une certaine hauteur h). On désigne souvent l'ensemble de ces logiques par le terme général de *logiques light* (même si la *logique linéaire light* désigne en fait spécifiquement le système LLL). Les liens entre LLL et la récursion sûre ont été étudiés dans [MO04], avec en particulier une traduction d'un fragment du système de Bellantoni et Cook [BC92] dans LLL. Notons qu'un aspect intéressant des logiques light par rapport aux autres systèmes de CI est le fait qu'elles intègrent dès le départ le polymorphisme et l'utilisation des types d'ordre supérieur.

Problématique et aperçu des contributions. Les logiques light ont ouvert la perspective d'intégrer dans la correspondance de Curry-Howard l'aspect de la complexité des programmes, comme par le passé d'autres traits de programmation tels le polymorphisme, le contrôle... À la fin des années 90, elles étaient cependant encore perçues par la communauté comme des systèmes sophistiqués, nécessitant de bien maîtriser la logique linéaire et les réseaux de preuves.

Nos travaux présentés ici ont eu pour objectifs de mieux comprendre ces logiques, de les simplifier et de les rapprocher d'autres formalismes de complexité implicite pour la programmation fonctionnelle.

En particulier, nous avons mis en avant leur utilisation possible comme systèmes de types pour les programmes du λ-calcul : si un programme est bien typé, alors il est Ptime. Ainsi le typage peut-il être utilisé comme un critère de vérification statique de la complexité d'un programme, et tester ce critère revient à déterminer si un programme admet un type (*inférence de type*). Un aperçu des résultats obtenus pour un de ces systèmes de types, que nous allons expliquer par la suite, est donné par le schéma de la Fig. 1.

Contexte, Projets. Les travaux présentés ici ont contribué à renforcer les liens entre la communauté de la complexité implicite d'un côté, et celle de la logique linéaire de l'autre (comme l'illustrent en particulier les récents workshops et projets à l'intersection de ces deux thématiques),

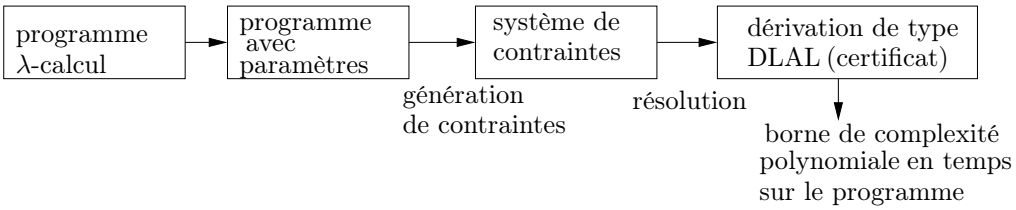

FIG. 1 – Inférence de type pour DLAL.

ainsi qu'à importer dans le domaine de la logique linéaire certaines techniques venant du typage.

Ces recherches ont été menées notamment au travers de collaborations internationales (Japon, Italie) et ont bénéficié de la dynamique de projets nationaux (GEOCAL, CRISS) puis d'un projet ANR *jeunes chercheurs* (NOCoST) que nous avons mis en place. Plusieurs contributions ont été réalisées en collaboration avec de jeunes chercheurs en Postdoctorat au LIPN, et des stages de Master et des thèses de doctorat (une soutenue et une en cours) ont été co-encadrés sur des thèmes en rapport avec ces travaux.

Plan. Nous rappellerons dans le Chap. 1 les définitions et les résultats de la littérature sur les logiques light. Ensuite dans le Chap. 2 nous donnerons un résumé des contributions de nos publications sélectionnées pour cette présentation. Nous ne suivrons pas un ordre strictement chronologique, mais regrouperons certaines directions de recherche de manière soit chronologique, soit thématique, afin de mieux mettre en évidence leur articulation. Enfin le Chap. 3 sera consacré aux perspectives de recherches.

Introduction (in English)

Logic. My work deals essentially with the study of the relations between logic and programs, through the notion of formal proof. Proof theory is the branch of logic studying proofs as mathematical objects. In the 1970s, the *Curry-Howard correspondence* has opened the way to a computational study of proofs by establishing a bijection between proofs and programs. The programs considered are written in λ-calculus, a language introduced by A. Church in the 1930s and which can be seen as a simplified programming language, at the root of functional languages like CAML.

Through the Curry-Howard correspondence, the execution of λ-calculus programs corresponds to a rewriting process on proofs, called *reduction* or *cut elimination*. This correspondence is the cornerstone of several later lines of research, on the one hand on proof-assistants and on the other hand on functional programming languages. In the first area for instance, this correspondence allows to extract from proofs programs that are certified correct, using tools like the Coq software. In the second area, it has allowed for a logical analysis of programming features, like polymorphism, exception handling, control operators.

The Curry-Howard correspondence, in its initial version, is applied to intuitionistic logic, a system without the excluded middle principle ("A or not-A"). One of the main tools for the study of λ-calculus and of intuitionistic logic is *denotational semantics* ; it associates to a proof-program an invariant of computation : the function represented by the program, or more generally a morphism in a *category*, whose objects correspond to formulas.

An important step in the analysis of the Curry-Howard correspondence is the introduction of *linear logic* (LL) [Gir87] in 1987 by J.-Y. Girard, as an outcome of research in denotational semantics. Its idea is that the operations of duplication and erasing of hypothesis are logical rules with first-class status : therefore in linear logic they are controlled by means of new connectives (modalities) !, ?, called *exponentials*. This system must be understood more like a decomposition of usual logic than like a proper new logic. For instance, some embeddings of intuitionistic and classical logics in linear logic allow to analyze accurately the dynamics (reduction) of these systems. Linear logic is now indeed one of the main conceptual tools in such areas as λ-calculus, proof theory of classical logic and denotational semantics.

One important by-product of linear logic is the notion of *proof-net*, a representation of proofs by graphs. Proof-nets are obtained by removing some irrelevant sequential information in proofs. The proof-net approach thereby allows for some simplifications in the study of reduction, that can seen as a graph-rewriting process, and gives a geometrical point of view on reduction. Proof-nets have also opened the way to *Geometry of interaction* (GoI) [Gir88], a program initiated by Girard aiming at interpreting proof normalization as a flow of information in the proof-net. For instance, one possible model of GoI consists in seeing the proof-net as an automaton acting on tokens, and execution is performed by computing the paths followed by the tokens (see e.g. [DHR96]).

Complexity. We have mentioned λ-calculus, but another well-known computational model is

that of *Turing machines*. It allows to represent the same functions as λ-calculus. Computational complexity classifies programs on Turing machines according to the amount of time or memory space needed for their evaluation, expressed relatively to the size of the argument or input of the program (for instance a binary list). *Ptime* programs in particular are those that terminate in polynomial time with respect to the size of the input. Complexity theory classifies in the same way the *functions* (on binary lists for instance) or the *problems* (functions valued in $\{0,1\}$) : a function is *computable in time* $\phi(n)$ if there is at least one program representing it and terminating in time at most $\phi(n)$ on inputs of size n. One defines in this way the class FP (resp. P) of functions (resp. problems) computable in polynomial time, which is commonly considered as the class of feasible functions (at least in first approximation). Let us recall that one of the most important open problems of theoretical computer science, P ?= NP, is the question whether or not the class P is equal to the class NP of problems whose solutions can be checked in polynomial time.

In this report, we will be concerned essentially with the complexity of programs. If computational complexity allows to define complexity classes, it does not tell us *how* to obtain polynomial time programs, nor given a program, how to test if it is Ptime, or how to certify it. Still complexity is a crucial aspect of programs in practice, because concretely termination after a very long computation time can, in some situations, be as problematic as non-termination. These questions are therefore essential for efficiency and reliability of software. However the problem, given a program, to decide for instance if it is Ptime, is undecidable. Hence as for most static analysis problems for programs, if we wish to automatize such a verification, we cannot hope for much more than a method giving sufficient conditions : if the program is validated by the method, then it is of Ptime complexity, otherwise a finer analysis is needed. Let us observe however that such a property gives us in general some information on an infinite number of executions of the program (executions of the program applied to all possible valid inputs, for instance all binary lists).

The problem of finding how to generate a Ptime program is related to the issue of characterizing the class of FP functions, which has been addressed by *implicit computational complexity* (ICC) : this research line aims at defining programming languages or calculi for which, by construction, all programs have a certain complexity bound, or alternatively, in the setting of a given programming language, at identifying some criteria to determine if a program admits a certain complexity bound. Initially ICC provided characterizations of Ptime and other complexity classes (Pspace, Logspace ...) which do not rely on a particular machine-model and do not refer to explicit bounds on resources (time, memory space). Instead, the complexity bound comes as a consequence of some restrictions on either logical rules, control structures in programs or operations allowed on data. This kind of foundational characterization might thus lead to a better understanding of the nature of resource-bounded computation.

This direction was started by Bellantoni and Cook [BC92] and Leivant [Lei94] who gave in the early 90s some characterizations of FP within primitive recursion by using a *safe recursion* (or *ramification*) discipline, limiting the way recursions can be nested. This line of research on taming of recursion was continued with λ-calculus [LM93], adapted to other complexity classes (*e.g.* [LM94]) and extended to the setting of calculi with higher-order types [Hof00, BNS00]. Another research line was conducted by N. Jones within the setting of functional languages to characterize complexity classes by restrictions on the program structure or on the operations allowed [Jon99]. All these systems allow to characterize complexity classes of *functions* but are quite limitative from a programming point of view : some common Ptime algorithms are not validated, even if the *function* represented can be computed by another algorithm, which is validated (but is probably less natural).

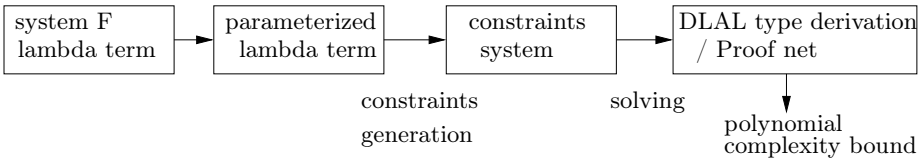

FIG. 2 – DLAL type inference.

This is the problem of *intensional expressivity* ; it has led to the study of ICC systems that are more flexible for programming, like type systems for *non-size-increasing* computation [Hof03] and the approach of *quasi-interpretations* within the framework of term rewriting systems [MM00, BMM05, BMM07].

Research in linear logic has met implicit computational complexity quite early, around 1992, thanks to the idea that the control of duplication in the proofs-programs is a means of limiting the complexity of their reduction : by considering variants of linear logic with "weak" versions of modalities, one obtains systems for which reduction of proofs is Ptime, and which allow to represent all functions of FP : BLL [GSS92], LLL [Gir98], SLL [Laf04]. Another system, ELL [Gir98], corresponds to the *elementary* complexity class (time bounded by a function $2^{2^{\cdot^{\cdot^{2^n}}}}$ for a certain height h). We will refer to all these systems by the generic term of *light logics* (even if *light linear logic* is in fact just one of these systems, LLL). The relations between LLL and safe recursion have been studied in [MO04], with in particular an embedding of a fragment of Bellantoni and Cook's safe recursion system [BC92] into LLL. Let us note that one interesting aspect of the light logics approach with respect to other ICC systems is that they include polymorphism and the usage of higher-order types.

Position of the addressed problems and brief overview of contributions. Light logics have opened a perspective on the insertion of the complexity aspect of programs (as previously other features like polymorphism, control) into the Curry-Howard correspondence. In the late 90s they were still seen however as sophisticated systems, requiring a good expertise on linear logic and proof-nets.

The works that we present here have aimed at improving the understanding of these logics, at simplifying them and at relating them to other approaches of implicit computational complexity for functional programs.

We have put forward in particular their possible usage as type systems for λ-calculus : if a program is well-typed in a certain system, then it is Ptime. Therefore, typing can be used as a static criterion for verifying an upper bound on the complexity of a program, and testing this criterion boils down to testing if the program admits a type (*type inference* problem). A sample of the results obtained for one of these systems, which we shall explain in the sequel, is given on Fig. 2.

Context, Projects. The works presented here have contributed to strengthening the links between the community of implicit computational complexity on the one hand, and that of linear logic on the other (as illustrated in particular by the recent workshops and the projects at the intersection of these two fields), as well as to import in the area of linear logic some techniques coming from typing.

These researches have been carried out in particular through international collaborations (Japan, Italy) and have benefitted from the dynamics first of national projects (GEOCAL, CRISS) and then of a *Young Researchers* project of the ANR (NOCoST) that we have set up. Several contributions in particular have been realized with Postdoc researchers at LIPN, and some supervisions or joint

supervisions of Master thesis and PhD thesis have also been carried out in relation with these works.

Plan. We will first in Chap. 1 recall the background definitions and results on light logics. Then in Chap. 2 we will give an overview of our research contributions out of the selected list of publications. We will not follow a strict chronolgical order but we will instead gather some directions together either on a thematical or a chronological basis, so as to illustrate their articulations. Finally Chap. 3 will be devoted to research perspectives.

Chapitre 1

Background on light logics

Throughout the works that we will discuss later, we will refer to three systems : Elementary, Light and Soft linear logic. Therefore, we shall start by recalling the definition of these systems and their main properties.

1.1 Elementary linear logic

The system of *Elementary linear logic* has been introduced in [Gir98] and then studied with an alternative presentation in [DJ03]. It has the advantage of being a simple system with a complexity-bounded cut elimination procedure, that allows to characterize the elementary complexity class. In fact the intuitionistic version of this system enjoys the same complexity properties as the initial system, and since it is easier to relate to λ-calculus intuitions, we will use this second system (we will proceed similarly for light and soft linear logic). Moreover, adding full weakening to obtain an affine system, following [Asp98], does not change its properties, makes programming more convenient and enables to omit additive connectives.

So by EAL (resp. ELL) we will designate here intuitionistic multiplicative elementary affine (resp. linear) logic. We will proceed in the following way : first define the system EAL, then describe the representation of *data types*, the *proof-nets* to study the complexity properties, then explain the use of iteration, and finally describe the expressivity of the system.

The language of EAL formulas is the same one as that of intuitionistic (multiplicative exponential) linear logic :

$$A, B ::= \alpha \mid A \multimap B \mid A \otimes B \mid !A \mid \forall \alpha.A$$

We will use the notation $!^k A = !\ldots!A$ (k times). The sequent calculus presentation of EAL is given on Fig. 1.1.

Observe that in EAL, there is no rule for *dereliction* nor *digging* : the principles $!A \multimap A$ and $!A \multimap !!A$ are *not* valid in this system.

To facilitate the computational intuitions for the study of EAL, we can annotate proofs with λ-terms as on Fig. 1.2. We will come back later in more detail on the relations between light logics and λ-calculus or extended λ-calculi (see Section 2.1.2). The terms of λ-calculus are given by the following grammar :

$$t, u ::= x \mid \lambda x.t \mid (t\ u)$$

We use the following notations :

$$\frac{}{A \vdash A}\ Id \qquad \frac{\Gamma_1 \vdash A \quad A, \Gamma_2 \vdash C}{\Gamma_1, \Gamma_2 \vdash C}\ Cut$$

$$\frac{\Gamma_1 \vdash A_1 \quad A_2, \Gamma_2 \vdash C}{\Gamma_1, A_1 \multimap A_2, \Gamma_2 \vdash C}\ \multimap l \qquad \frac{A_1, \Gamma \vdash A_2}{\Gamma \vdash A_1 \multimap A_2}\ \multimap r$$

$$\frac{A[B/\alpha], \Gamma \vdash C}{\forall \alpha.A, \Gamma \vdash C}\ \forall l \qquad \frac{\Gamma \vdash A}{\Gamma \vdash \forall \alpha.A}\ \forall r \ \ (\alpha \text{ not free in } \Gamma)$$

$$\frac{\Gamma \vdash C}{A, \Gamma \vdash C}\ Weak \qquad \frac{!A, !A, \Gamma \vdash C}{!A, \Gamma \vdash C}\ Cntr$$

$$\frac{\Gamma \vdash A}{!\Gamma \vdash !A}\ !r$$

FIG. 1.1 – Sequent calculus for EAL

– $\lambda x_1 x_2.t$ stands for $\lambda x_1.\lambda x_2.t$,
– leftward bracketing in applications : $(t\ u\ v)$ stand for $((t\ u)\ v)$.

$$\frac{}{x{:}A \vdash x{:}A}\ Id \qquad \frac{\Gamma_1 \vdash u{:}A \quad x{:}A, \Gamma_2 \vdash t{:}C}{\Gamma_1, \Gamma_2 \vdash t[u/x]{:}C}\ Cut$$

$$\frac{\Gamma_1 \vdash u{:}A_1 \quad x{:}A_2, \Gamma_2 \vdash t{:}C}{\Gamma_1, y{:}A_1 \multimap A_2, \Gamma_2 \vdash t[(y\ u)/x]{:}C}\ \multimap l \qquad \frac{x{:}A_1, \Gamma \vdash t{:}A_2}{\Gamma \vdash \lambda x.t{:}A_1 \multimap A_2}\ \multimap r$$

$$\frac{x{:}A[B/\alpha], \Gamma \vdash t{:}C}{x{:}\forall \alpha.A, \Gamma \vdash t{:}C}\ \forall l \qquad \frac{\Gamma \vdash t{:}A}{\Gamma \vdash t{:}\forall \alpha.A}\ \forall r \ \ (\alpha \text{ not free in } \Gamma)$$

$$\frac{\Gamma \vdash t{:}C}{A, \Gamma \vdash t{:}C}\ Weak \qquad \frac{x{:}!A, y{:}!A, \Gamma \vdash t{:}C}{z{:}!A, \Gamma \vdash t[z/x, z/y]{:}C}\ Cntr$$

$$\frac{\Gamma \vdash t{:}A}{!\Gamma \vdash t{:}!A}\ !$$

FIG. 1.2 – Sequent calculus for EAL typing

Recall that system F is the polymorphic type system corresponding by the Curry-Howard correspondence to second-order intuitionistic logic. The language of system F types is given by :

$$T, U ::= \alpha \mid T \to U \mid \forall \alpha.T$$

There is a forgetful map $(.)^- : EAL \to F$ obtained by removing exponentials. If $A^- = T$ we say that A is a *decoration* of T in EAL.

Data types in EAL. They can be adapted from data types of system F by using the following decorations :

– Unary integers

F :	EAL :
N^F	N^{EAL}
$\forall\alpha.(\alpha \to \alpha) \to (\alpha \to \alpha)$	$\forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha)$

Example : The usual Church integers, like the example of $\underline{2}$, indeed have type N^{EAL} in EAL :

$$\underline{2} = \lambda f.\lambda x.(f\ (f\ x))\ .$$

– Binary lists

F :	EAL :
W^F	W^{EAL}
$\forall\alpha.(\alpha \to \alpha) \to (\alpha \to \alpha) \to (\alpha \to \alpha)$	$\forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha)$

Example : The Church encodings of binary lists, like below the example $\underline{w}$ for the list $w = [1, 0, 0]$, have type W^{EAL} :

$$\underline{w} = \lambda s_0.\lambda s_1.\lambda x.(s_1\ (s_0\ (s_0\ x)))\ .$$

In the rest of this Section we will simply write N (resp. W) for N^{EAL} (resp. W^{EAL}).

Examples of terms. Let us give a few examples of terms corresponding to EAL derivations and representing operations on unary integers.

Addition		
add	$=$	$\lambda nmfx.(n\ f\ (m\ f\ x))$
	$:$	$N \multimap N \multimap N$
Multiplication		
$mult$	$=$	$\lambda nmf.(n\ (m\ f))$
	$:$	$N \multimap N \multimap N$
Squaring		
$square$	$=$	$\lambda nf.(n\ (n\ f))$
	$:$	$!N \multimap !N$

Proof-nets for ELL. We will represent proofs by *proof-nets*. For this purpose it will be convenient to consider proof-nets of *classical* ELL and to represent intuitionistic ELL proofs by their embeddings in this system.

The grammar of classical ELL formulas is given by

$$A, B ::= \alpha \mid \alpha^\perp \mid A \wp B \mid A \otimes B \mid !A \mid ?A \mid \forall\alpha.A \mid \exists\alpha.A$$

Here $A \multimap B$ can be defined as $A^\perp \wp B$. The exponential rules for classical ELL are given on Fig. 1.3.

We translate intuitionistic ELL derivations into proof-nets, via their translation into classical ELL derivations defined in the natural way according to the following translation of sequents :

$$B_1, \ldots, B_n \vdash A \quad \longrightarrow \quad \vdash B_1^\perp, \ldots, B_n^\perp, A$$

The ELL *proof-structures* are defined by the nodes of Fig. 1.4 ; each !, ? node (resp. $\forall$ node) has to be a door of a ${}_b s$-box (resp. quantifier box) as on Fig. 1.5 (resp. Fig. 1.6). Two boxes have

$$\frac{\vdash \Gamma}{\vdash ?A, \Gamma}\ Weak \qquad \frac{\vdash ?A, ?A, \Gamma}{\vdash ?A, \Gamma}\ Cntr$$

$$\frac{\vdash A, B_1, \ldots, B_n}{\vdash\ !A, ?B_1, \ldots, ?B_n}\ !$$

FIG. 1.3 – Exponential rules for classical ELL

to be disjoint, or one is included in the other. Instead of quantifiers with boxes one could also use the proof-nets with quantifiers of [Gir91] (with jumps). Observe that the !-box of ELL is not the same one as the !-box of ordinary linear logic since it adds an exponential (! or ?) to all formulas $A, B_1, \ldots, B_n$ premise of a door of the box.

An ELL *proof-net* is then a proof-structure corresponding to an ELL sequent calculus proof. They can also be characterized by a correctness criterion, in a way similar to that of linear logic proof-nets [Gir87].

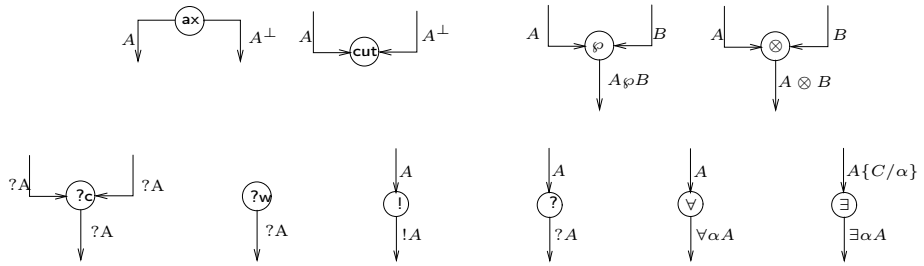

FIG. 1.4 – Nodes for ELL proof structures

The *depth* $d(e)$ of an edge (resp. a node) e in a proof-net R is the number of exponential boxes containing it in R. The *depth* $d(R)$ of the proof-net R is the maximum depth of its nodes and its *size* $|R|$ is its number of nodes.

Proof-net reduction in ELL and its properties. The contraction and weakening reduction steps are given on Fig. 1.7 and are the same as for linear logic proof-nets. The *box-box* reduction step is on Fig.1.8 and corresponds to a merging of boxes. The other reduction steps (multiplicative and axiom steps) are the same as for linear logic proof-nets.

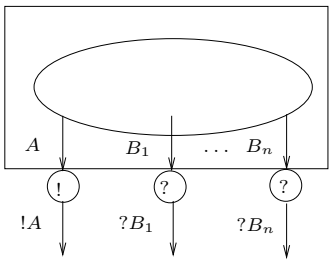

FIG. 1.5 – !-box in ELL

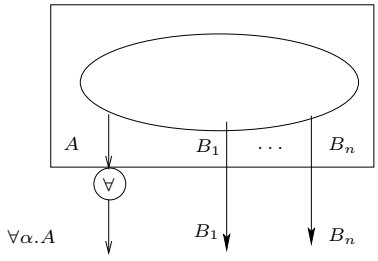

FIG. 1.6 – ∀-box in ELL

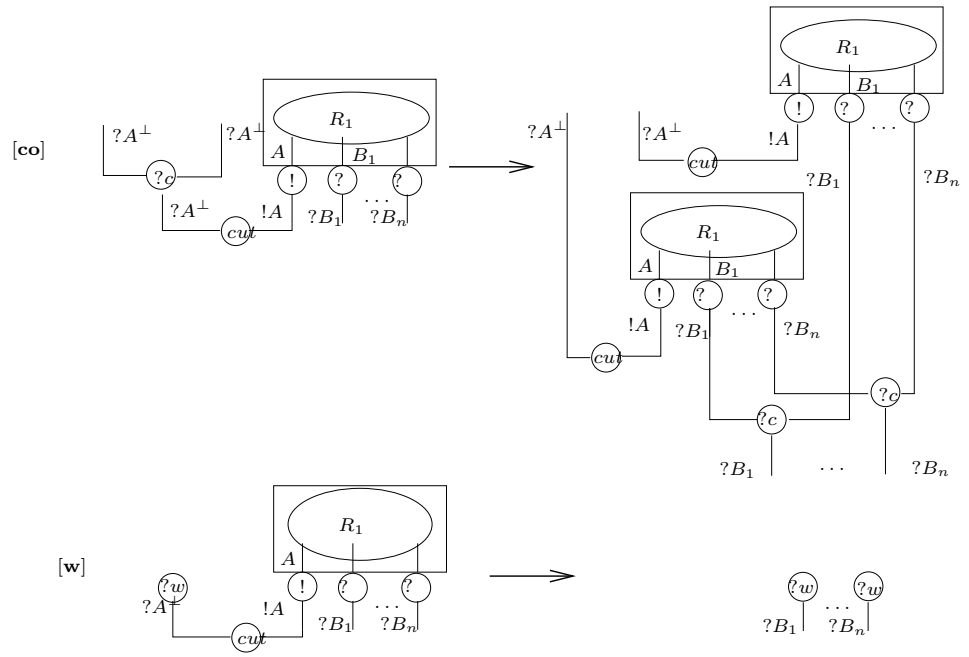

FIG. 1.7 – Contraction and weakening reduction steps

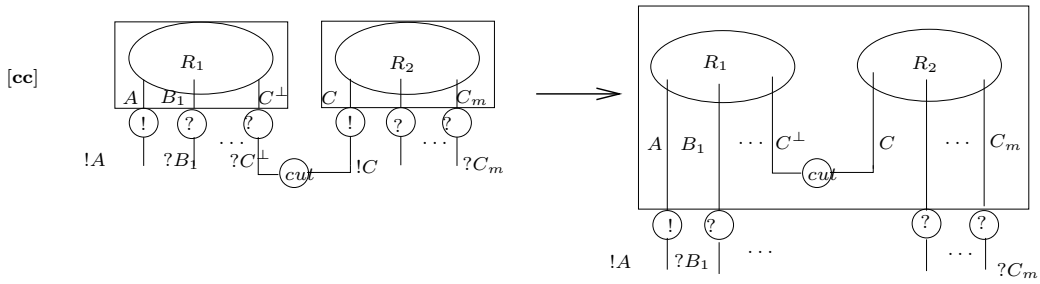

FIG. 1.8 – Box-box reduction step

One can handle full weakening (EAL) in the intuitionistic case by using suitable proof-nets (see [AR02]), but we content ourselves here with ELL for simplicity.

Now, by examining the various reduction steps, we can observe the following property :

Proposition 1 (Stratification) *The depth of an edge of an ELL proof-net does not change after a reduction step.*

Indeed an edge can be erased or duplicated but in any case, the resulting corresponding edges will be at the same depth. Note that this is not true in LL : the depth of an edge can decrease (during a dereliction step) or increase (during a box-box step).

Let us define the function $K(.,.)$ by : $K(0,n) = n$, $K(k+1,n) = 2^{K(k,n)}$. Then we have $K(k,n) = 2^{2^{\cdots^{2^n}}}$ with height k.

Theorem 2 *If R is an ELL proof-net of depth d, then R can be reduced into its normal form in less than $K(d+1,|R|)$ steps.*

Note that :
- the height of the tower in the bound $K(d+1,|R|)$ only depends on the depth, and not on the size ;
- in this bound, there is no reference to the formulas occuring in the proof-net ;
- this bound is in fact obtained by applying a reduction strategy *by levels* (see [Gir98]), that is to say by reducing cuts at depth 0, then at depth 1 etc. until depth d.

Iteration in EAL. As in system F we can define in EAL for any A an iterator $iter_A$:

$$iter_A = \lambda f x n.\ (n\ f\ x)\ : !(A \multimap A) \multimap !A \multimap N \multimap !A$$

Then we have : $(iter_A\ F\ t\ \underline{n}) \to (F\ (F \dots (F\ t) \dots)$ (n times)

Example : By using the previous term for multiplication typed in EAL, one can define a term *double* which represents the doubling function on unary integers, with type : $double : N \multimap N$.

By applying the iterator of type N to this term *double*, one defines then an exponentiation function :

$$exp = \lambda n.(iter_N\ double\ \underline{1}\ n)\ : N \multimap !N.$$

Note that the term *exp* cannot however be iterated in EAL, because its type is not of the form $A \multimap A$ or $!(A \multimap A)$, for any A.

In practice, for programming we will need *coercions* on data types. For instance, we can define a coercion from N to $!N$ by :

$$coerc_1 = (iter_N\ succ\ \underline{0})\ : N \multimap !N$$

More generally one can define similarly : $coerc_i : N \multimap !^i N$, for $i \in \mathbb{N}$.

As a consequence, a term $\vdash t : !^i N \multimap A$ can be replaced by a term $\vdash t' : N \multimap A$ which is extensionally equal, obtained by defining : $t' = \lambda n.(t\ (coerc_i\ n))$.

Representation of functions in EAL. Let Π be an EAL proof of $x : N \vdash t : !^l N$. We say that π *represents function* $f : \mathbb{N} \to \mathbb{N}$ if :

for any integer n the proof obtained by applying a cut rule between $\vdash \underline{n} : N$ and Π reduces to $\vdash \underline{n'} : !^l N$, where $n' = f(n)$.

We recall the definition of elementary functions :

Definition 1 *A function f is elementary recursive if there exists an integer h and a Turing machine $\mathcal{M}$ which computes f in time bounded by $K(h, n)$.*

We have seen with Theorem 2 that a proof-net R of depth d can be reduced in a number of steps $O(K(d+1, |R|))$. It follows from that point that if we have an EAL typable term $\vdash t : N \multimap !^k N$ then t represents an elementary recursive function.

Conversely, all elementary recursive functions can be represented in EAL and we have :

Theorem 3 *The functions representable in EAL are exactly the elementary recursive functions.*

This statement is a natural adaptation of the results of [Gir98, DJ03].

Statements like that of Theorem 3 are often in implicit computational complexity divided in two parts : the *complexity soundness* property with respect to a complexity class $\mathcal{C}$ (here the class of elementary functions) is the fact that all functions representable in the calculus belong to the class $\mathcal{C}$; the *complexity completeness* property w.r.t. $\mathcal{C}$ is the fact that all the functions of the class $\mathcal{C}$ are representable in the calculus.

1.2 Light linear logic

Light linear logic has been introduced by Girard in [Gir98], to characterize Ptime computation. Here we consider its intuitionistic version, that we denote LLL and its intuitionistic affine version LAL following [Asp98].

The language of LAL formulas is given by :

$$A, B ::= \alpha \mid A \multimap B \mid A \otimes B \mid !A \mid \S A \mid \forall \alpha . A$$

We will write $\S^k A = \S \ldots \S A \quad (k \text{ times})$.

The sequent calculus for LAL is given on Fig. 1.9. In the rule (§) we can have $\Gamma = \emptyset$ or $\Delta = \emptyset$.

Observe that the rule § can be thought of as a kind of multiple dereliction (acting on the formulas of Γ), with a *marker* §. Intuitively, the occurrences of § keep track in the formula of the dereliction steps that have been needed (typically, in order to do contractions). Moreover, as the principle $!A \multimap \S A$ is valid, it suggests that from a typing point of view $!A$ can be thought of as a subtype of $\S A$.

If we also consider the connective $\otimes$, note that the principle $!A \otimes !B \multimap !(A \otimes B)$ is *not* provable in LAL, while it is in EAL.

There is a translation $(.)^e$ from LAL to EAL defined in the following way : $(\S A)^e = !A^e$, $(.)^e$ commutes to the other connectives. It is easy to check that $(.)^e$ maps LAL derivations to EAL derivations and is compatible with reduction.

As for EAL, we can associate λ-terms to LAL derivations as shown on Fig. 1.10.

$$\frac{}{A \vdash A}\ Id \qquad \frac{\Gamma_1 \vdash A \quad A, \Gamma_2 \vdash C}{\Gamma_1, \Gamma_2 \vdash C}\ Cut$$

$$\frac{\Gamma_1 \vdash A_1 \quad A_2, \Gamma_2 \vdash C}{\Gamma_1, A_1 \multimap A_2, \Gamma_2 \vdash C}\ \multimap l \qquad \frac{A_1, \Gamma \vdash A_2}{\Gamma \vdash A_1 \multimap A_2}\ \multimap r$$

$$\frac{A[B/\alpha], \Gamma \vdash C}{\forall \alpha.A, \Gamma \vdash C}\ \forall l \qquad \frac{\Gamma \vdash A}{\Gamma \vdash \forall \alpha.A}\ \forall r \ \ (\alpha \text{ not free in } \Gamma)$$

$$\frac{\Gamma \vdash C}{\Delta, \Gamma \vdash C}\ Weak \qquad \frac{!A, !A, \Gamma \vdash C}{!A, \Gamma \vdash C}\ Cntr$$

$$\frac{B \vdash A}{!B \vdash !A}\ ! \qquad \frac{\vdash A}{\vdash !A}\ !$$

$$\frac{\Gamma, \Delta \vdash A}{!\Gamma, \S\Delta \vdash \S A}\ \S$$

FIG. 1.9 – Sequent calculus for LAL

Data types in LAL. As in the case of EAL they can be adapted from the data types of system F :

– Unary integers

$$\begin{array}{cc} \text{EAL :} & \text{LAL :} \\ N^{EAL} & N^{LAL} \\ \forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha), & \forall\alpha.!(\alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha)\ . \end{array}$$

– Binary lists

$$\begin{array}{cc} \text{EAL :} & \text{LAL :} \\ W^{EAL} & W^{LAL} \\ \forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha), & \forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha)\ . \end{array}$$

In the rest of this section we will write simply N (resp. W) for N^{LAL} (resp. W^{LAL}).

Example : The term for addition on unary integers that we have seen with EAL can be also typed in LAL :

$$add \ : \ N \multimap N \multimap N$$

We can also write and type a doubling function :

$$\begin{array}{rcl} double & = & \lambda nfx.(n\ f\ (n\ f\ x)) \\ & : & !N \multimap \S N \end{array}$$

Proof-nets for LLL. We represent proofs by proof-nets for *classical* LLL, as we have done for ELL. The grammar of classical LLL formulas is given by :

$$A, B ::= \alpha \mid \alpha^{\perp} \mid A \wp B \mid A \otimes B \mid !A \mid ?A \mid \S A \mid \bar{\S} A \mid \forall\alpha.A \mid \exists\alpha.A$$

The connectives $\S$ and $\bar{\S}$ are dual :

$$(\S A)^{\perp} = \bar{\S} A^{\perp}, \quad (\bar{\S} A)^{\perp} = \S A^{\perp}.$$

$$\frac{}{x:A \vdash x:A}\ Id \qquad \frac{\Gamma_1 \vdash u:A \quad x:A, \Gamma_2 \vdash t:C}{\Gamma_1, \Gamma_2 \vdash t[u/x]:C}\ Cut$$

$$\frac{\Gamma_1 \vdash u:A_1 \quad x:A_2, \Gamma_2 \vdash t:C}{\Gamma_1, y:A_1 \multimap A_2, \Gamma_2 \vdash t[(y\ u)/x]:C}\ \multimap l \qquad \frac{x:A_1, \Gamma \vdash t:A_2}{\Gamma \vdash \lambda x.t:A_1 \multimap A_2}\ \multimap r$$

$$\frac{x:A[B/\alpha], \Gamma \vdash t:C}{x:\forall\alpha.A, \Gamma \vdash t:C}\ \forall l \qquad \frac{\Gamma \vdash t:A}{\Gamma \vdash t:\forall\alpha.A}\ \forall r \quad (\alpha \text{ not free in } \Gamma)$$

$$\frac{\Gamma \vdash t:C}{\Delta, \Gamma \vdash t:C}\ Weak \qquad \frac{x:!A, y:!A, \Gamma \vdash t:C}{z:!A, \Gamma \vdash t[z/x, z/y]:C}\ Cntr$$

$$\frac{x:B \vdash t:A}{x:!B \vdash t:!A}\ ! \qquad \frac{\vdash t:A}{\vdash t:!A}\ !$$

$$\frac{\Gamma, \Delta \vdash t:A}{!\Gamma, \S\Delta \vdash t:\S A}\ \S$$

FIG. 1.10 – Sequent calculus for LAL typing

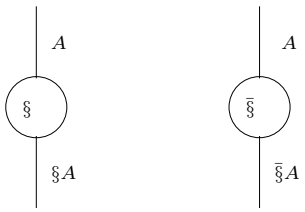

FIG. 1.11 – § and $\overline{\S}$ nodes

We use the same nodes as for ELL proof-structures, extended with two new nodes for § and $\overline{\S}$ (Fig. 1.11) that have to be used with §-boxes (Fig. 1.12). The !-box is displayed on Fig. 1.13 ; it has zero or one ?-node.

As usual, an *LLL proof-net* is a proof-structure corresponding to an LLL sequent calculus derivation.

Proof-net reduction in LLL and its properties. Proof-net reduction in LLL is defined as in ELL, with moreover the case of cut $\S/\overline{\S}$ described on Fig. 1.14.

As ELL proof-nets, LLL proof-nets satisfy the stratification property. Moreover we have the following main property :

Theorem 4 ([Gir98]) *If R is an LLL proof-net, with depth d, then the reduction of R into its normal form can be done in $O((d+1) \cdot |R|^{2^{d+1}})$ steps.*

Note that :
– if the depth is fixed, then the number of steps is polynomial in $|R|$;
– this bound is obtained by a strategy *by levels* ([Gir98]).

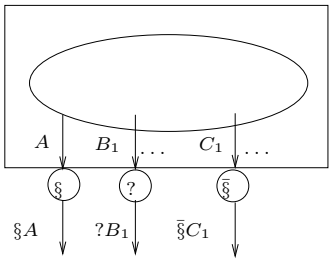

FIG. 1.12 – §-box

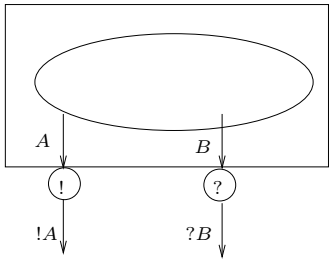

FIG. 1.13 – !-box

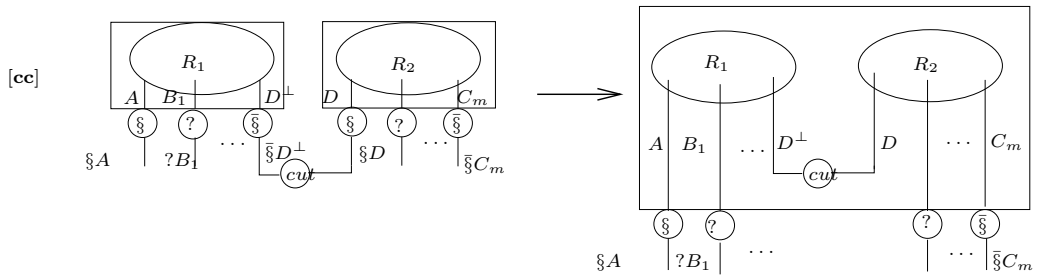

FIG. 1.14 – § reduction step

In fact the bound $O((d+1)\cdot|R|^{2^{d+1}})$ holds not only for the reduction strategy by levels, but for *any* reduction strategy, as it has been shown later in [Ter01, Maz06].

Iteration in LAL. We can define the iterators as in EAL and obtain as type in LAL :

$$iter_A = \lambda f x n.\ (n\ f\ x)\ : !(A \multimap A) \multimap \S A \multimap N \multimap \S A$$

Example :

Using the term $add : N \multimap N \multimap N$ seen before, we can type $(add\ \underline{2}) : N \multimap N$. Then we can define another term for the doubling function, that we denote $double'$:

$$double' = \lambda n.(iter_N\ (add\ \underline{2})\ \underline{0}\ n)\ : N \multimap \S N$$

Observe that neither this term $double'$ nor the previous one $double$ can be iterated in LAL, since they do not have a type of the form $A \multimap A$ or $!(A \multimap A)$.

Similarly we have : $m : N \vdash (add\ m) : N \multimap N$, so $m : !N \vdash (add\ m) : !(N \multimap N)$. From that we can define a term for multiplication, by iteration :

$$mult' = \lambda m n.(iter_N\ (add\ m)\ \underline{0}\ n)\ : !N \multimap N \multimap \S N.$$

On the other hand the previous term $mult$ that we used in EAL is not typable with type $N \multimap N \multimap N$ in LAL.

We can also define a coercion from N to $\S N$, by :

$$coerc_1 = (iter_N\ succ\ \underline{0})\ : N \multimap \S N.$$

More generally there are similar coercions, of the form :

$$coerc_{i,j} : N \multimap \S^{i+1}!^{j}N, \text{ with } i,j \geq 0.$$

We can use for instance coercions to obtain a term for multiplication with type $N \multimap N \multimap \S^2 N$. For that, first use the (§) rule to get $mult' : \S!N \multimap \S N \multimap \S^2 N$. Then define :

$$mult'' = \lambda nm.(mult'\ (coerc_{0,1}\ n)(coerc_{0,0}\ m)) : N \multimap N \multimap \S^2 N.$$

From this term, by applying successively rules (§) and $(Cntr)$, we obtain a term for the doubling function :

$$square'' : !N \multimap \S^3 N.$$

We can proceed in a similar way to define coercions for type W. Therefore, for handling unary functions on binary lists, it is sufficient to use types of the form $W \multimap \S^k W$, for $k \in \mathbb{N}$.

Representation of functions in LAL. Recall that for representing the polynomial time functions, it is important to represent integers as binary lists.

Let Π be an LAL proof of $x : W \vdash t : \S^k W$. We say that Π *represents function* $f : \{0,1\}^\star \to \{0,1\}^\star$ if :

for any w of $\{0,1\}^\star$, the proof obtained by applying a cut rule between $\vdash \underline{w} : W$ and Π reduces to $\vdash \underline{w'} : \S^k W$, where $w' = f(w)$.

We then have :

Proposition 5 (Completeness) *If $f : \{0,1\}^* \to \{0,1\}^*$ belongs to FP, then there exists an integer k and a proof Π of $x : W \vdash t : \S^k W$ representing f.*

See [AspertiRoversi02] for a detailed proof.

So finally we have :

Corollary 6 *The functions representable in LAL are exactly the functions of FP, that is to say computable in polynomial time on a Turing machine.*

Note that even if a term t is typable in LAL, to execute it with the expected complexity bound we need to *compile* it first into a proof-net, by translating its type derivation into a proof-net ; this will be discussed again later (see Section 2.4).

Observe also that a term can be reducible in polynomial time when applied to any binary list, and however not be typable in LAL. The completeness Proposition 5 only states that for any polynomial time *function*, there is at least one term typable in LAL and representing it.

1.3 Soft linear logic

Soft linear logic has been introduced by Lafont in [Laf04]. Here we will consider its intuitionistic affine version SAL, without additive connectives. The language of formulas is the same one as for EAL. We present the system directly as a sequent calculus system for typing λ-terms, on Fig. 1.15.

The rule (mplex) is called *multiplexing*.

If we also use the $\otimes$ connective, we have that the following principles are not valid in SAL :

$$\frac{}{x:A \vdash x:A}\ Id \qquad \frac{\Gamma_1 \vdash u:A \quad x:A, \Gamma_2 \vdash t:C}{\Gamma_1, \Gamma_2 \vdash t[u/x]:C}\ Cut$$

$$\frac{\Gamma_1 \vdash u:A_1 \quad x:A_2, \Gamma_2 \vdash t:C}{\Gamma_1, y:A_1 \multimap A_2, \Gamma_2 \vdash t[(y\ u)/x]:C}\ \multimap l \qquad \frac{x:A_1, \Gamma \vdash t:A_2}{\Gamma \vdash \lambda x.t:A_1 \multimap A_2}\ \multimap r$$

$$\frac{x:A[B/\alpha], \Gamma \vdash t:C}{x:\forall\alpha.A, \Gamma \vdash t:C}\ \forall l \qquad \frac{\Gamma \vdash t:A}{\Gamma \vdash t:\forall\alpha.A}\ \forall r \ \ (\alpha \text{ not free in } \Gamma)$$

$$\frac{\Gamma \vdash t:C}{\Delta, \Gamma \vdash t:C}\ Weak$$

$$\frac{x_1:A_1, \ldots, x_n:A_n, \vdash t:A}{x_1:!A_1, \ldots, x_n:!A_n, \vdash t:!A}\ ! \qquad \frac{x_1:A, \ldots, x_n:A, \Gamma \vdash t:C}{z:!A, \Gamma \vdash t[z/x_1, \ldots, z/x_n]:C}\ mplex\ (n \geq 0)$$

FIG. 1.15 – Sequent calculus for SAL typing

$$!A \multimap !A \otimes !A \ \text{(Contraction)}, \qquad !A \multimap !!A \ \text{(digging)}.$$

However, the following principle is valid for any $k \geq 0$:

$$!A \multimap (A \otimes ... \otimes A)$$

Data types in SAL. One can use the following data types in SAL, derived from system F :

– Unary integers :

$$N = \forall\alpha.!(\alpha \multimap \alpha) \multimap (\alpha \multimap \alpha)$$

– Booleans :

$$B = \forall\alpha.\ \alpha \multimap \alpha \multimap \alpha$$

– Binary lists :

$$W = \forall\alpha.!(B \multimap \alpha \multimap \alpha) \multimap (\alpha \multimap \alpha)$$

Note that actually the data types used in [Laf04] are not the same ones, but this article uses Soft linear logic with additive connectives.

Properties. We do not describe here the proof-nets for SLL, but for more detail one can consult [Laf04].

The following results are directly adapted from [Laf04] (see also [Gab07] for a sequent calculus presentation) :

Theorem 7 *If $\mathcal{D}$ is an SAL proof of depth d, then $\mathcal{D}$ can be normalized in a number of steps bounded by $O(n^d)$, where $n = |\mathcal{D}|$.*

From that we get :

Corollary 8 *If we have $\vdash_{SAL} t : !W \multimap B$, then t represents a Ptime predicate.*

A converse property holds :

Theorem 9 (Completeness) *If $f : \{0,1\}^\star \to \{0,1\}$ is a Ptime predicate, then there exists a term t with an SAL derivation of $\quad \vdash t : !W \multimap B$, representing f.*

The completeness result has been proved for general SLL in [Laf04] and multiplicative SLL in [MT03].

Chapitre 2

Summary of the research works presented

2.1 Denotational semantics for light logics (1999-2000)

2.1.1 Models

Motivations. The goal of the work [Bai04a] was to undertake a semantical analysis of light logics by the approach of denotational semantics. One motivation for this is to provide a more abstract understanding of these systems, which could for instance suggest some simplifications or help to decide between various "design choices" which do not have influence on the complexity properties. Before that, only semantics of provability (formulas) had been studied for light logics, with the framework of phase spaces ([KOS97]).

In the paper [Bai04a] the systems studied are propositional ELL and LLL. Two categories are defined :

- $\mathbb{SCOH}$: objects : stratified coherence spaces ; morphisms : stratified cliques.
- $\mathbb{BSCOH}$: objects : measured stratified coherence spaces ; morphisms : locally bounded stratified cliques.

These categories give sound models respectively of ELL and LLL. Let us describe them briefly. A *stratified coherence space* (s.c.s.) X is given by a sequence $X = (X^i, \phi_i)_{i\in\mathbb{N}}$ where the X^i are coherence spaces. This sequence is stationary and can therefore intuitively be thought of as finite : there exists an integer d (*depth*) such that $\forall i \geq d,\ X^i = X^d$. A stratified coherence space is meant to provide more information than an ordinary coherence space in the following sense : X^d is seen as the main space, while $X^{d-1}, \ldots, X^0$ are approximations that provide lesser information. The partial maps $\phi_i : |X^i| \to |X^{i-1}|$ are given together with the space, and indicate how points of $|X^i|$ are identified in $|X^{i-1}|$.

Now, a subset $S \subseteq |X^i|$ is mapped by ϕ_i to $\phi_i(S) \subseteq |X^{i-1}|$. We say that $S \subseteq |X^d|$ is a *stratified clique* of X if :

- S is a clique of X^d,
- for any $0 \leq i \leq d-1$, $\phi_{i+1} \circ \phi_{i+2} \circ \cdots \circ \phi_d(S)$ is a clique of X^i.

Stratified cliques compose, by using relational composition, and we get a category $\mathbb{SCOH}$.

The interpretation of formulas is then done in the following way :

- constructions $\otimes$, $\oplus$ are performed *level-by-level* ;

– § is a *shift* operation : $\S X$ is defined by $(\S X)^0 = 1$, $(\S X)^{i+1} = X^i$, with $\phi'_1 : |X^0| \to |1|$ the constant identity map (recall that 1 is the coherence space with only one point, $|1| = \{*\}$) ;
– ! is obtained by the level-by-level ordinary multiset $!_m$ construction on coherence spaces, followed by a shift : $(!X)^0 = 1$, $(!X)^{i+1} = !_m X^i$;
– the $(.)^\perp$ construction is defined level-by-level.

Actually, it is convenient to denote for $\S X$ elements of $|(\S X)^{i+1}|$ as singleton elements over $|X^i|$:

$$|(\S X)^{i+1}| = \{[x],\ x \in |X^i|\}.$$

We obtain that $\mathbb{SCOH}$ is a sound model of ELL, but it does not offer in a natural way the structure to be a model of LL. For instance, the natural candidate relations on $!X \multimap X$ and $!X \multimap !!X$ for derelictions and digging are not stratified cliques.

The category $\mathbb{SCOH}$ is a model both of ELL and LLL, and so it does not distinguish these two systems. In order to exhibit a category that is a sound model of LLL but not of ELL, we want to account in a more subtle way for the properties of the '!' modality of LLL, and in particular rule out the monoidalness principle $!A \otimes B \multimap !(A \otimes B)$.

For that, the idea we exploit is to take into account the *input/output* dependencies in the number of '*threads*' of computation. We intend here '*threads*' in the sense of game semantics, and they correspond in the coherence semantics setting to the elements of multisets in the spaces representing $!A$ / $?A$ formulas. We define *locally bounded cliques* with the intuition that they keep bounded the difference between the number of input and output threads in all computations.

To formalize this intuition, we add some structure on the spaces : a *stratified measured coherence space* (s.m.c.s.) X is a sequence $(X^i, s_i, \phi_i)_{i\in\mathbb{N}}$ where the s_i are *measuring functions* :

$$\forall i \in \mathbb{N}, \quad s_i : |X^{i+1}| \to \mathbb{Z}.$$

The constructions on s.m.c.s. follow those on stratified coherence spaces, but one has to specify how the constructions act on measuring functions. The most important cases are those of the '!' construction and the duality :

$$s_0^{!X}([x_1, \ldots, x_n]) = n \text{ and } s_{i+1}^{!X}([x_1, \ldots, x_n]) = \sum_{k=1}^{n} s_i^X(x_k),$$

$$s_i^{X^\perp} = -s_i^X.$$

The other cases are defined by :

$$\begin{aligned} s_i^{X\Box Y}(x,y) &= s_i^X(x) + s_i^Y(y) \text{ for } \Box = \otimes, \wp, \\ s_i^{X\Box Y} &= [s_i^X, s_i^Y] \text{ for } \Box = \&, \oplus, \\ s_i^1(*) &= s_i^\perp(*) = 0 \end{aligned}$$

For the § construction we have :
– for $i \geq 0$, $(\S X)^{i+1} = X^i$,
– $(\S X)^0 = 1$,
– for $i \geq 0$, $\quad s_{i+1}^{\S X}([x]) = s_i^X(x)$, and $s_0^{\S X}([x]) = 1$.

Now we formally define locally bounded cliques by :

Definition 2 *Let f be a stratified clique on a m.s.c.s. X. We say that f is* locally bounded *if for any i, for any x in f^i the following integers set is bounded :*

$$s_i(\phi_i^{-1}(\{x\}) \cap f^{i+1}).$$

For more explanation on the intuitions behind this definition see [Bai04a]. We obtain that :

Proposition 10 *Locally bounded stratified cliques are preserved by composition and $\mathbb{BSCOH}$ is a category.*

Finally, the article [Bai04a] shows that $\mathbb{BSCOH}$ is a sound model of LLL, but does not validate the monoidalness principle and hence is not a model of ELL.

Related works. There have been a few works on denotational semantics for light logics. Murawski and Ong have in [MO00] given a game model (in the so-called AJM style) for LLL with a full completeness result : any strategy of the category described is the interpretation of an LLL proof. In [LTdF06] Laurent and Tortora de Falco introduced models for ELL and SLL which are subcategories of the relational model and for which they proved *relative completeness results* characterizing ELL and SLL within LL : for instance any LL proof-net whose interpretation in the relational model belongs to the first subcategory (*obsessional cliques*) is a proof of ELL. Redmond [Red07] has given a categorical axiomatisation for models of SLL. De Carvalho in his PhD Thesis [dC07] has given a model for untyped light λ-calculus (see Section 2.2) based on intersection types.

A related line of works is that of realizability models for light logics or ICC calculi. These models have been used to prove the complexity soundness of several systems, initially by Hofmann in [Hof00] for a higher-order type system built for safe recursion, then by Hofmann and Scott for BLL [HS04], and in [DLH05] by Dal Lago and Hofmann for SAL, EAL, LAL and the linear type system LFPL for non-size-increasing computation from [Hof03]. The realizability approach was also used by Schöpp in [Sch07] : in this work the author defined a logic inspired by BLL, stratified bounded affine logic, and which characterizes the complexity class Logspace. The realizability technique, based on game-semantics ideas, is used here to show the complexity soundness part.

2.1.2 From denotational semantics to type systems

One drawback of the semantics we have considered is that they do not identify as many proofs-programs as we would hope for. Indeed several proofs of LLL for instance might have essentially the same computational contents and still be interpreted in $\mathbb{BSCOH}$ by distinct morphims. This happens in particular if the two proofs Π, Π' do not have the same conclusion (that is to say the same type, as programs), even if these conclusions differ only by § modalities. Think for instance of a proof Π of $N \vdash \S^2 N$ representing a function on integers, and of $\Pi' : \S N \vdash \S^3 N$ obtained by applying to Π a § rule : Π and Π' can then be applied to the same arguments (up to § rules) and have the same behaviour ; nevertheless the denotational semantics does not identify them ...

This leads to the issue of the *computational contents* of light logics proofs. Indeed these proofs carry both an algorithmic part and some information about the quantitative behaviour of the algorithm. But moreover, they might stipulate some conditions on the inputs expected by the proof-program : we will refer to that as *domain specification*. Take for instance a proof $\Pi : (N \multimap \S N) \vdash \S^2 N$. The proof Π expects as input a proof of $(N \multimap \S N)$, and thus for instance we cannot apply it to a proof of $(N \multimap \S^3 N)$.

Intuitively the algorithm implemented by the proof π is described by the *intuitionistic skeleton* of the proof (of type $(N \to N) \vdash N$ in the example), that is to say the intuitionistic proof corresponding to π, while the domain specification and some quantitative properties are described by the *exponential decoration* of the proof.

This suggests two possible ways of studying light logics :

1. **by the Curry-Howard isomorphism** : define some term calculi for various light logics and study their dynamics ; these calculi can take the form of λ-calculi extended with constructs allowing to handle the exponentials.
2. **as type systems for λ-calculus** : this approach dissociates the algorithmic part from the domain specification and quantitative behaviour description. The idea is to use as calculus the λ-calculus (untyped, or with Church simple types or system F types) and as type system *à la Curry* the light logic.

Approach 1. has been explored in [Rov98, Ter01] for instance for light affine logic and in [BM04] for soft linear logic (see Section 2.2). It is convenient in particular to study the dynamics of reduction, as for instance in [Ter01].

Approach 2. was initiated in [CM01, Bai02] respectively for elementary and light affine logic and then has been the object of several works [Bai04b, BT04, CM06, CDLRDR05, ABT07, GMRDR08].

Note that an advantage of the typing approach 2. is precisely that several proofs differing only by exponentials will correspond to the same λ-term. However, a delicate point is the fact that the dynamics of proof reduction might not coincide with that of β-reduction, in particular subject-reduction or complexity bound on β-reduction can be problematic issues (see Section 2.4).

2.2 Curry-Howard correspondence for light logics (2003-2005)

Soft λ-calculus. By adopting the approach based on the Curry-Howard correspondence mentioned above, K. Terui (Tokyo) had defined in [Ter01] an extended λ-calculus, the *light λ-calculus*, whose typed version corresponds to light affine logic LAL. This calculus fits in the tradition of extended λ-calculi proposed previously for linear logic, like for instance the one of Benton-Bierman-de Paiva-Hyland [BBdPH93]. It contains specific constructions corresponding to exponentials (modalities) of LAL.

Lafont had introduced in [Laf04] Soft linear logic SLL, which he had studied with the proof-net syntax. Following the approach adopted for LAL by Terui, we defined with V. Mogbil (LIPN) an extended λ-calculus for SAL (the affine version of SLL), called *soft λ-calculus* [BM04], whose terms are defined from a class of *soft pseudo-terms* generated by the following grammar :

$$t, t' ::= x \mid \lambda x.t \mid (t\ t') \mid !t \mid \text{let } t \text{ be } !x \text{ in } t'$$

If $\overrightarrow{t}$ and $\overrightarrow{x}$ are finite sequences of same length $(t_1, \ldots, t_n)$ and $(x_1, \ldots, x_n)$, then let $\overrightarrow{t}$ be $!\overrightarrow{x}$ in t' stands for let t_1 be $!x_1$ in let t_2 be $!x_2$ in $\ldots t'$. In the case where $n = 0$, let $\overrightarrow{t}$ be $!\overrightarrow{x}$ in t' is t'.

The reduction rules for pseudo-terms are given on Fig 2.2. The *depth* of a pseudo-term t is the maximal nesting of !(.) constructs inside it. The class of *soft λ-terms* is then defined as a subclass of soft pseudo-terms satisfying certain syntactic conditions (see [BM04]).

Soft pseudo-terms can be typed in SAL. The typing rules are given on Fig. 2.1 ; they are obtained from the ones of Section 1.3 by replacing λ-calculus terms by soft pseudo-terms. A typed pseudo-term is then a soft λ-term and corresponds to a proof (or proof-net) of SAL.

$$\frac{}{x : A \vdash x : A}\ (\text{Id}) \qquad \frac{\Gamma \vdash t : A \quad \Delta, x : A \vdash u : B}{\Gamma, \Delta \vdash u[t/x] : B}\ (\text{Cut})$$

$$\frac{\Gamma, x : B \vdash t : C \quad \Delta \vdash u : A}{\Gamma, \Delta, y : A \multimap B \vdash t[(yu)/x] : C}\ (\multimap \text{l}) \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \multimap B}\ (\multimap \text{r})$$

$$\frac{x : A[C/\alpha], \Gamma \vdash t : B}{x : \forall\alpha.A, \Gamma \vdash t : B}\ (\forall \text{ l}) \qquad \frac{\Gamma \vdash t : B}{\Gamma \vdash t : \forall\alpha.B}\ (\forall \text{ r})\ (\alpha \text{ not free in } \Gamma)$$

$$\frac{\Gamma \vdash t : B}{\Gamma, x : A \vdash t : B}\ (\text{Weak}) \qquad \frac{x_1 : A, \ldots, x_n : A, \Gamma \vdash t : B}{y : !A, \Gamma \vdash \text{let } y \text{ be } !x \text{ in } t[x/x_1, \ldots, x_n] : B}\ (\text{mplex})$$

$$\frac{x_1 : A_1, \ldots, x_n : A_n \vdash t : B}{y_1 : !A_1, \ldots, y_n : !A_n \vdash \text{let } \overrightarrow{y} \text{ be } !\overrightarrow{x} \text{ in } !t : !B}\ (!)$$

FIG. 2.1 – Typing rules in SAL for soft λ-terms

(β) :	$((\lambda x.t)\ u) \ \to^1\ t[u/x]$
(!) :	$\text{let } !u \text{ be } !x \text{ in } t \ \to^1\ t[u/x]$
(com1) :	$\text{let } (\text{let } t_1 \text{ be } !y \text{ in } t_2) \text{ be } !x \text{ in } t_3 \ \to^1\ \text{let } t_1 \text{ be } !y \text{ in } (\text{let } t_2 \text{ be } !x \text{ in } t_3)$
(com2) :	$(\text{let } t_1 \text{ be } !x \text{ in } t_2)t_3 \ \to^1\ \text{let } t_1 \text{ be } !x \text{ in } (t_2\ t_3)$

FIG. 2.2 – Reduction rules for soft λ-terms

This allowed us to illustrate the computational intuitions at work in SLL. The untyped soft λ-calculus admits a *strong polytime reduction property* : at fixed depth d, there is a polynomial such that for any term of depth d, any reduction of this term is performed in polynomial time.

Our second goal with the introduction of soft λ-calculus was to examine if the extension of SAL with recursive types (or type fixpoints) could allow for a more flexible programming style. Recursive types are indeed common practice in functional programming, and it follows from [BM04] that soft λ-calculus typed in SAL_μ (SAL with recursive types) enjoys the same properties as the one typed in SAL. It might at first sound a little surprising that even with fixpoints the system SAL_μ still offers termination and the polynomial bound, but observe that this simply comes from the fact that these properties are already enjoyed by the underlying untyped calculus (soft λ-calculus or proof-nets). This is actually a common feature of ELL, LLL, SLL pointed out for proof-nets of these systems in [Gir98, Laf04]. Therefore in this setting the advantage of the types (polymorphic or with fixpoints) is not to ensure termination with bounded complexity but to guarantee the absence of deadlock during reduction. This guarantees in particular that a suitably typed term will evaluate to a value.

In SAL_μ one can consider two data types for lists (for example) : one following the Church encoding of integers $\mathcal{L}(A)$ and the other one $L(A)$ using recursive types (and defined using connectives $\otimes$ and $\oplus$).

$$\mathcal{L}(A) = \forall\alpha.!(A \multimap \alpha \multimap \alpha) \multimap \alpha \multimap \alpha, \qquad L(A) = \mu X.(1 \oplus (A \otimes X)).$$

However, both of them have a drawback :

– $\mathcal{L}(A)$ allows to define an iteration on lists (`fold`), but does not allow to define a `cons` function with type $\mathcal{L}(A) \multimap A \multimap \mathcal{L}(A)$;

– $L(A)$ on the other hand allows to define the basic functions on lists, like `cons` with type $L(A) \multimap A \multimap L(A)$, but does not admit an associated iteration scheme.

Therefore we introduced a new family of data types to overcome these two limitations. In the case of lists we defined :

$$N[L(A)] = \forall\alpha.!(\alpha \multimap \alpha) \multimap \alpha \multimap (L(A) \otimes \alpha)$$

An element of this type contains a list and a unary integer. The way it is used is that the integer is meant to give a bound on the size of the associated list. In this way, this type allows to define an iteration and at the same time to support basic operations on lists. The compromise is that one needs to be able to maintain the invariant on the property of the integers and lists in the values.

We have shown that these data types allowed to program in a slightly more liberal way than the previous ones, and in particular to nest some iterations (in certain cases), contrarily to the general discipline of ramified systems such as [BC92]. To illustrate that we have described in the paper how to program the insertion sort, with a first iteration to define the insertion and a second one for the sorting term.

This work has been carried out in the frameworks of the (national) projects CRISS and GEOCAL.

Besides, Daniel de Carvalho in his PhD thesis [dC07], which I have co-supervised with Thomas Ehrhard, has continued the study of the light λ-calculus of Terui. Indeed, even if the untyped version of this calculus terminates in polynomial time, LAL types are still useful for these terms because they guarantee that a well-typed term will evaluate into a result (as stressed above for soft λ-calculus). In contrast, evaluation of untyped terms can lead to some deadlocks (that can be seen as pattern-matching errors). In order to characterize which are the terms of light λ-calculus that can be evaluated without deadlock, de Carvalho has introduced a system of intersection types, more flexible than the initial type system LAL. He has shown in [dC05, dC07] that the light λ-terms typable in this system are exactly those that can be evaluated without reaching a deadlock. Moreover this has enabled him to build, starting from the types, a denotational model of light λ-calculus [dC07]. Finally let us stress that these contributions only correspond to one chapter of de Carvalho's PhD thesis, which also contains other results on models of linear logic and untyped λ-calculus.

Another work which uses light λ-calculus is our article [BP06] with Pedicini where we defined an extension of this calculus for computation over reals in the Blum-Shub-Smale model (BSS), following [BCdNM03]. We showed that this calculus allows to characterize the BSS class of polynomial time functions.

Fixpoints and expressivity. The results of complexity soundness and completeness on light logics recalled before in Chap. 1 hold with the particular conventions fixed for the data types and the representation of functions, inspired from system F. However, as illustrated in particular by the work [BM04] discussed above it can be interesting in some cases to consider extensions of light logics (like those with fixpoints) and/or other data types.

One might then wonder on the one hand how far we can enrich light logics while at the same time keeping the complexity properties, and on the other hand how much we can restrict them and still keep completeness for the class of functions FP. To answer these questions in a way which is not dependent on a specific choice of data types nor on that of the representation of functions, we must make precise what we consider as a suitable representation scheme for functions.

In [DLB06] with Dal Lago we analyzed these questions for various fragments of intuitionistic light affine logic (LAL) and extensions of this logic with fixpoints of formulas. The motivation for this investigation was to contribute to the identification of well-behaved fragments/extensions of

LAL that could then in future works be used for various purposes like typing and type inference, proof of program termination or proof-search.

In order to consider a general enough but nevertheless reasonable notion of encoding, we used the definition of *uniform encoding* introduced by Dal Lago in [DL03].

A *uniform encoding* $\mathcal{E}(f)$ of a function $f : (\{0,1\}^*)^n \to \{0,1\}^*$ into a logic consists of :

- A proof π with conclusion $A_1, \ldots, A_n \vdash B$ (where $A_1, \ldots, A_n, B$ can be different) ;
- For every $i \in \{1, \ldots, n\}$, a correspondence Φ_i between elements of $\{0,1\}^*$ and cut-free proofs of conclusion $\vdash A_i$, which is logspace computable.
- A correspondence Ψ between cut-free proofs of conclusion $\vdash B$ and elements of $\{0,1\}^*$, which is logspace computable.

Moreover the following diagram must commute :

$$\begin{array}{ccc} \{0,1\}^* \times \ldots \times \{0,1\}^* & \xrightarrow{f} & \{0,1\}^* \\ \downarrow{\scriptstyle \Phi_1} \quad \downarrow{\scriptstyle \Phi_n} & & \uparrow{\scriptstyle \Psi} \\ A_1 \ , \ \ldots \ , \ A_n & \xrightarrow{\pi} & B \end{array}$$

So the encodings considered in Chap. 1 for logics EAL, LAL, SAL and based on data types inspired from system F are uniform encodings, but they are not the only ones. Note in particular that the types used for the various arguments and for the result need not in general be the same ones.

We then say that a logic $\mathcal{L}$ is *polytime sound* (resp. *polytime complete*) if the class of functions $f : (\{0,1\}^*)^n \to \{0,1\}^*$ uniformly encodable in $\mathcal{L}$ is included in FP (resp. if this class contains FP).

Let us denote as $\mathbf{LAL}_{-\circ\otimes\forall}$ (resp. $\mathbf{LAL}_{-\circ}$) the fragment of LAL restricted to formulas built only with connectives $-\circ$, $\otimes$, $\forall$ (resp. only with the connective $-\circ$). We showed in [DLB06] that $\mathbf{LAL}_{-\circ\otimes\forall}$ is not polytime sound (in the sense defined above) by exhibiting a function of exponential-size output which can be represented with a uniform encoding. On the other hand, we showed that the fragment $\mathbf{LAL}_{-\circ}$ is not polytime complete, by using a method similar to that used in simply typed λ-calculus (applying a Theorem of Statman) to establish that the equality predicate is not representable.

In order to obtain polytime sound and complete fragments, we considered versions of fixpoint and quantifiers rules where these connectives can only be used on *linear formulas*, that is to say formulas containing no occurrence of ! and § connectives. We denote the class of linear formulas by $\mathcal{L}$. The rules for linear fixpoints and quantifiers are the following ones :

$$\frac{\Gamma, C[A/\alpha] \vdash B \quad A \in \mathcal{L}}{\Gamma, \overline{\forall}\alpha.C \vdash B}\ L_{\overline{\forall}} \qquad \frac{\Gamma \vdash C \quad \alpha \notin FV(\Gamma)}{\Gamma \vdash \overline{\forall}\alpha.C}\ R_{\overline{\forall}}$$

$$\frac{\Gamma, A[\overline{\mu}\alpha.A/\alpha] \vdash B \quad A \in \mathcal{L}}{\Gamma, \overline{\mu}\alpha.A \vdash B}\ L_{\overline{\mu}} \qquad \frac{\Gamma \vdash A[\overline{\mu}\alpha.A/\alpha] \quad A \in \mathcal{L}}{\Gamma \vdash \overline{\mu}\alpha.A}\ R_{\overline{\mu}}$$

We then showed that the systems $\mathbf{LAL}_{-\circ\otimes\overline{\forall}\overline{\mu}}$ and $\mathbf{LAL}_{-\circ\otimes\overline{\mu}}$ defined from these connectives are polytime sound and complete. The idea used to prove the completeness part is that : $\mathbf{LAL}_{-\circ\otimes}$ itself is already sufficient to represent any polynomial on unary integers ; moreover $\mathbf{LAL}_{-\circ\otimes\overline{\forall}\overline{\mu}}$ allows to encode transition functions of Turing machines. Combined together, these two results imply that $\mathbf{LAL}_{-\circ\otimes\overline{\forall}\overline{\mu}}$ is polytime complete. The same argument can also be employed for $\mathbf{LAL}_{-\circ\otimes\overline{\mu}}$, by using a more subtle encoding of the transition function of a Turing machine into this system. The main results obtained are summarized on Fig. 2.3.

Properties w.r.t. uniform encodings	$\mathbf{LAL}_{\multimap}$	$\mathbf{LAL}_{\multimap\otimes\forall}$	$\mathbf{LAL}_{\multimap\otimes\overline{\forall\mu}}$	$\mathbf{LAL}_{\multimap\otimes\overline{\mu}}$
polytime soundness	Yes	No	Yes	Yes
polytime completeness	No	Yes, [Gir98, AR02]	Yes	Yes

FIG. 2.3 – Summary of polytime soundness and completeness of fragments and extensions of LAL ([DLB06])

2.3 Light logics as type systems : first typing methods (2000-2003)

We now turn to the study of light logics as type systems for ordinary λ-calculus, in the line of approach 2 described in Section 2.1.2.

An algorithm based on a boxing criterion. The work [Bai02] was the first one to address the problem of type-search in propositional LAL. This algorithm and the one given by Coppola and Martini in [CM01] for EAL were developed independently [1] . Both algorithms use integer programming, but they are based on different principles.

Even though the article [Bai02] is now somehow superseded by [Bai04b] and [BT05], we briefly describe it here because it was an intermediate step in this research direction. The algorithm of [Bai02] showed that the following problem is decidable : let t be a simply typed λ-term in normal form ; can it be *decorated* in a LAL typed λ-term ? The result also allowed for some form of domain specification (for instance requiring the input domain to be of integer type).

The limitations of this work were that :

1. the algorithm was given only for terms in normal form (no β-redex) ;
2. it applied to simple types and propositional LAL, not to polymorphic types ;
3. no explicit complexity bound was given on the algorithm.

Another drawback concerns LAL as a type system for λ-calculus, rather than the algorithm itself : this type system does not satisfy good properties with respect to β-reduction, in particular subject-reduction. We will come back on this issue in Section 2.4.

Observe that point 1. above does not mean that the algorithm is useless : we are typically interested in deciding whether a term expecting a data and returning a data (say binary lists) is typable in LAL and thus Ptime. Asking for this term to be in normal form simply means that the program has been preprocessed and that all β-redexes have been reduced.

Limitation 1. has later been overcome in [Bai04b] and limitations 2., 3. have been solved in [ABT07] by changing the type system. This will be detailed in the next Sections.

Idea of the algorithm.

The algorithm of [Bai02] as well as that of [CM01] is based on the idea of *exponential proof decoration* :

1. LAL types can be seen as decorations of simple types with exponentials !, § ;
2. LAL type derivations can be seen as intuitionistic proofs decorated with exponential rules.

[1] Both works were presented at the TMR Linear workshop of Bertinoro in 2001.

Observe that this kind of decoration is different from generic translations like that of intuitionistic or classical logic in linear logic (*e.g.* [DJS97]) : here the decoration and the number of exponentials needed is not determined by the source formulas, but it depends on the proof. This kind of exponential proof decoration was explored in [DJS94] for the case of intuitionistic proofs translated in linear logic.

The approach of [Bai02] to find LAL decorations of an intuitionistic type derivation follows the following steps :

1. propose a simple type derivation for the λ-term (there are several possible choices because of the placement of contraction rules),
2. decorate the simple types with parameters (meant to range over $\mathbb{N}$) for exponentials !, §,
3. decide how to instantiate the parameters and where to use the !, § rules so as to obtain a valid derivation.

The approach developed uses the syntax of intuitionistic proof-nets. The possible derivations considered in 1. are thus *skeletons* of proof-nets (there are no boxes yet) ; they differ one from another by the placement of contractions (sharing). In step 3 the problem thus becomes how to place !-boxes and §-boxes so as to obtain a valid LAL proof-net.

A difficulty in step 2. for decorating formulas with integer parameters is that we might need in LAL in general an arbitrary number of alternations of ! and § exponentials. Thus parameterized types of the form $\S^{n_1}!^{m_1}A$ or $\S^{n_2}!^{m_2}\S^{n_1}!^{m_1}!A$ for instance, might not be sufficient in general.

The approach adopted in [Bai02] was to restrict to a bounded number of alternations of § and ! in formulas, which anyway was sufficient for LAL typability when considering normal λ-terms. We will talk again about this issue later when discussing [Bai04b] and [BT04].

The key idea exploited by the algorithm is to :

– use an intermediate syntax of proof-nets *with doors* but without explicit boxes,
– establish a criterion allowing to decide whether for such a proof-net valid boxes can be reconstructed.

The idea of this criterion is that a box can be seen as an opening door together with some (possibly zero) matching closing doors. The fact that the boxes can only be nested (and not overlap) imposes that if we follow a path from the root of the net to a variable, whenever we cross a closing door, we must have crossed before the opening door of the corresponding box, and closing doors match the opening door of their respective box following a FIFO stack discipline. This condition can thus be expressed as a bracketing-style condition on paths and is necessary and sufficient for boxes to be definable and nested. Actually, to be valid, the boxes also have to be compatible with the other constructions of the proof-net, in particular with λ bindings in the λ-calculus presentation.

This would be sufficient for EAL, but with LAL an extra delicate point is that !-boxes must have at most one closing door. This condition is of a different kind than the previous ones because it cannot be expressed on each path independently, but needs instead to be expressed on all subpaths starting from a given opening door. Finally, [Bai02] provides a criterion deciding if a proof-net with doors corresponds to an LAL proof-net with valid boxes ; let us call that a *boxing criterion.*

In order to decide, using this boxing criterion, if a skeleton of proof-net can be decorated in a valid LAL proof-net, we then consider a parameterized proof-net, defined as follows :

1. before each node, a parameter n is introduced for the number of boxes,
2. for each edge : its simple type A is replaced with a parameterized LAL type which is a decoration of A.

In 1. we actually know in advance, depending on the kind of node and the proof-net, if the potential doors should be opening or closing doors. The parameter n is meant to range over $\mathbb{N}$.

Each map $\phi : Par \to \mathbb{N}$ on the set of parameters allows then naturally to instantiate the parameterized proof-net R into a tentative proof-net $\phi(R)$. Using the boxing criterion and type unification conditions, one can define a set of constraints $\mathcal{C}^R$ such that :

$$\phi(R) \text{ is a valid LAL proof-net iff } \phi \text{ satisfies the constraints } \mathcal{C}^R.$$

The constraints considered here are a subset of Presburger arithmetic constraints and are thus decidable : see [Bai02] for more details. Universal quantification and disjunction are needed to express the (non-local) condition that !-boxes have at most one auxiliary door.

Type inference for LAL with word constraints. A limitation of the algorithm of [Bai02] is that it tries to decorate simply typed λ-terms only by considering particular parameterizations where in a type, the number of alternations between ! / § is bounded *a priori*. This is related to the fact that the input λ-term is supposed to be in normal form.

To overcome these limitations and establish the decidability of type inference in propositional LAL in full generality, the later article [Bai04b] then adopted another approach (developed in 2002-2003). Actually, the typing system we consider is not exactly LAL but a slight generalization of it, $LAL_{\leq}$, using subtyping. The subtyping is based on the idea that $!A \leq \S A$. However, this system is very close to LAL because any $LAL_{\leq}$ typable term is η-equivalent to an LAL typable term.

The method of [Bai04b] is also based on the idea of exponential decoration of proofs, but does not use the boxing criterion, contrarily to [Bai02]. It starts from the observation (in the same spirit as [CRDR05]) that in a skeleton of proof-net (that is to say an untyped proof-net without boxes), there is only a finite number of *positions* of possible boxes, where a box position is determined by the entrance and exit of the box. By contrast, the number of possible boxes is infinite because each position can correspond to an arbitrary number of boxes.

The technique of [Bai04b] then proceeds in the following way :

1. choose a compatible set of box positions on the skeleton of the proof-net, among the finite set of possibilities ;
2. for this set of positions, determine for each position what is the number and the kind (! or §) of the boxes used : this will be called a *distribution of boxes*.

Given a position, the boxes used in this position can be represented by a (finite) word on the alphabet $\{!, \S\}$. A distribution of boxes for a compatible set of positions is then given by a set of words $w_1, \ldots, w_n$ on $\{!, \S\}$.

To be valid, a proof-net with types must satisfy a certain number of conditions of unification on types, in particular for contraction and application nodes. For a distribution of boxes represented by words $w_1, \ldots, w_n$ these conditions can be translated into inequation constraints on words. The order $\leq$ considered here is the one induced letter-by-letter by $! \leq \S$.

So we finally express the typability in LAL of a simply typed λ-term by the satisfiability of a constraints system on binary words. A theorem then shows that the class of constraints used is decidable.

The algorithm of [Bai04b] therefore shows that the problem of type inference in propositional LAL is decidable. However it leaves certain issues unsolved :

1. The type system LAL for λ-calculus is not completely satisfactory, because the complexity properties are only guaranteed for proof-nets.
2. The algorithm of [Bai04b] handles propositional LAL but not second-order (polymorphism).
3. We have shown that type inference in LAL is decidable, but we have not given any bound on the complexity of this inference problem.

We will show in the sequel how our later works have answered these three points.

2.4 Design of a light type system and efficient type inference (2003-2007)

The type system DLAL. We have stressed the fact that LAL is not really satisfactory as a type system for the λ-calculus : indeed it does not guarantee the property of subject-reduction for β-reduction and the β-reduction of a well-typed term might need exponential time, as shown in [BT04]. To execute a term typed in LAL in polynomial time, one actually needs before to compile it into an LAL proof-net and then use the reduction of proof-nets.

In order to obtain a system offering better properties for β-reduction, we then proposed with Terui (Tokyo) in [BT04] to revisit the type system. For that, we defined a new system called DLAL (*Dual Light Affine Logic*), without exponential '!' but with two arrows, linear $\multimap$ and non-linear $\Rightarrow$. The grammar of DLAL types is given by :

$$A, B ::= \alpha \mid A \multimap B \mid A \Rightarrow B \mid \S A \mid \forall \alpha.A$$

The judgements used for DLAL typing are of the form $\Gamma; \Delta \vdash t : C$. The intended meaning is that variables in Δ are (affine) linear, that is to say that they have at most one occurrence in the term, while variables in Γ are non-linear. We give the typing rules as a natural deduction system, displayed on Fig. 2.4. We have :

– for ($\forall$ i) : (*) α does not appear free in Γ, Δ.

– in the ($\Rightarrow$ e) rule the r.h.s. premise can also be of the form $;\vdash u : A$ (u has no free variable).

DLAL derivations can be translated into LAL proofs by using the following translation on formulas :

– $(A \Rightarrow B)^* = !A^* \multimap B^*$,

– $(.)^*$ commutes to the other connectives.

The *depth* of a DLAL derivation $\mathcal{D}$ is the maximal number of premises of ($\S$ i) and r.h.s. premises of ($\Rightarrow$ e) in a branch of $\mathcal{D}$.

The data types in DLAL can be directly adapted from those of LAL :

$$\begin{aligned} \mathbf{N} &= \forall \alpha.(\alpha \multimap \alpha) \Rightarrow \S(\alpha \multimap \alpha), \\ \mathbf{W} &= \forall \alpha.(\alpha \multimap \alpha) \Rightarrow (\alpha \multimap \alpha) \Rightarrow \S(\alpha \multimap \alpha). \end{aligned}$$

We have shown [BT04] the fact that DLAL admits the property of subject-reduction for β-reduction and the following main property :

Theorem 11 (Polynomial time strong normalization) *Let t be a λ-term which has a typing derivation $\mathcal{D}$ of depth d in DLAL. Then t reduces to the normal form in at most $O(|t|^{2^d})$ reduction steps and in time $O(|t|^{2^{d+1}})$ on a Turing machine. This result holds independently of which reduction strategy we take.*

$$\frac{}{;x : A \vdash x : A}\ (\text{Id})$$

$$\frac{\Gamma;\Delta, x : A \vdash t : B}{\Gamma;\Delta \vdash \lambda x.t : A \multimap B}\ (\multimap \text{ i}) \qquad \frac{\Gamma_1;\Delta_1 \vdash t : A \multimap B \quad \Gamma_2;\Delta_2 \vdash u : A}{\Gamma_1,\Gamma_2;\Delta_1,\Delta_2 \vdash tu : B}\ (\multimap \text{ e})$$

$$\frac{\Gamma, x : A;\Delta \vdash t : B}{\Gamma;\Delta \vdash \lambda x.t : A \Rightarrow B}\ (\Rightarrow \text{ i}) \qquad \frac{\Gamma;\Delta \vdash t : A \Rightarrow B \quad ;z : C \vdash u : A}{\Gamma, z : C;\Delta \vdash tu : B}\ (\Rightarrow \text{ e})$$

$$\frac{\Gamma_1;\Delta_1 \vdash t : A}{\Gamma_1,\Gamma_2;\Delta_1,\Delta_2 \vdash t : A}\ (\text{Weak}) \qquad \frac{x_1 : A, x_2 : A, \Gamma;\Delta \vdash t : B}{x : A, \Gamma;\Delta \vdash t[x/x_1, x/x_2] : B}\ (\text{Cntr})$$

$$\frac{;\Gamma,\Delta \vdash t : A}{\Gamma;\S\Delta \vdash t : \S A}\ (\S \text{ i}) \qquad \frac{\Gamma_1;\Delta_1 \vdash u : \S A \quad \Gamma_2; x : \S A, \Delta_2 \vdash t : B}{\Gamma_1,\Gamma_2;\Delta_1,\Delta_2 \vdash t[u/x] : B}\ (\S \text{ e})$$

$$\frac{\Gamma;\Delta \vdash t : A}{\Gamma;\Delta \vdash t : \forall\alpha.A}\ (\forall \text{ i})\ (*) \qquad \frac{\Gamma;\Delta \vdash t : \forall\alpha.A}{\Gamma;\Delta \vdash t : A[B/\alpha]}\ (\forall \text{ e})$$

FIG. 2.4 – Natural deduction system for DLAL

Moreover DLAL enjoys like LAL a completeness result :

Theorem 12 (FP completeness) *If a function $f : \{0,1\}^\star \to \{0,1\}^\star$ is computable in time $O(n^{2^d})$ by a Turing machine for some d, then there exists a λ-term t such that $\vdash_{DLAL} t : \mathbf{W} \multimap \S^{2d+1}\mathbf{W}$ and t represents f.*

The system DLAL therefore transfers to λ-calculus the properties of LAL proofs.

In fact [BT07] makes more precise the relationship between DLAL and LAL : on the one hand DLAL exactly corresponds to a subsystem of LAL, and on the other hand DLAL is in a certain sense as expressive intensionally as LAL because there exists a generic translation $(\,\cdot\,)^\bullet$ from LAL to DLAL. This translation is based on an encoding of the connective ! with $\multimap$ and $\Rightarrow$ using polymorphism :

- $(!A)^\bullet = \forall\alpha.((A^\bullet \Rightarrow \alpha) \multimap \alpha)$, where α is fresh.
- $(\,\cdot\,)^\bullet$ commutes to the other connectives.

Let us stress that the intensional expressivity of DLAL and LAL, as that of the other light logics, is modest. However we showed in [BT07] that insertion sort can be represented by a λ-calculus program typable in DLAL, in a rather satisfactory way. This relies on the typing of the insertion program with type $\S A \multimap L(A) \multimap L(A)$, where $L(A)$ is the natural type for lists given by $L(A) = \forall(A \multimap \alpha \multimap \alpha) \Rightarrow \S(\alpha \multimap \alpha)$.

The design of the system DLAL was also motivated by the perspective of a system for which type inference could be easier than that given in [Bai04b] for LAL. For that, we have started by re-examining the simpler problem of type inference in elementary affine logic EAL.

Type inference in EAL. Decidability of type inference in EAL has been proven in [CM01, CRDR05] but with algorithms of complexity at least exponential. As EAL only has one exponential '!' instead of two for LAL, the principle of typing by decoration is simpler than for LAL. Exploiting the idea of the boxing criterion of [Bai02], we have defined in [BT05] a simple procedure for typing in EAL, with the following new characteristics :

- the decoration with parameters n is done directly on the λ-terms (instead of proof-nets in [Bai02]) ;

– parameters are valued in $\mathbb{Z}$: a positive (resp. negative) value corresponds to a series of opening doors (resp. closing doors).

Let us describe the procedure in more detail. In fact, to be precise, the type system considered is not exactly EAL, but $EAL^\star$, that is to say a restriction of EAL which essentially does not allow for sharing (which anyway would be problematic to handle through the type system). We address the following problem :

Problem 1 (EAL decoration) *Given a closed simply typed term $\vdash M : T$, does there exist an EAL decoration A of T such that $\vdash M : A$ is a valid $EAL^\star$ typing for M ?*

(Here the closedness assumption is only for readability)

The first step of our approach is to give a characterization of terms typed in $EAL^\star$. For that, it is convenient to consider a richer term language, that we call language of *pseudo-terms*[2] :

$$t, u ::= x \mid \lambda x.t \mid (t\ u) \mid !t \mid \bar{!}t$$

The idea is that ! constructs correspond to main doors of boxes in proof-nets while $\bar{!}$ constructs correspond to auxiliary doors of boxes. But note that there is no information in the pseudo-terms to link occurrences of ! and $\bar{!}$ corresponding to the same box.

In practice, we need to handle pairs (t, Γ) of a pseudo-term t together with an *assignment* Γ, which is a mapping from variables of t (free or bound) to $EAL^\star$ types. We then define three classes of conditions on a pseudo-term t :

1. *bracketing conditions* on t,
2. *scope conditions* on t,
3. *typing conditions* on (t, Γ).

Conditions 1 and 3 constitute the *boxing criterion*. We show that : a pair (t, Γ) corresponds to a valid $EAL^\star$ type derivation if and only if conditions 1, 2, and 3 are satisfied.

This characterization is then convenient to search for suitable decorations by using the parameterization approach. To perform this search, we use *parameters* : $\mathbf{n}, \mathbf{m}, \mathbf{k}, \ldots$ The *parameterized pseudo-terms* are then defined by :

$$a ::= x \mid \lambda x.t \mid (t\ t), \qquad t ::= !^{\mathbf{n}} a,$$

where $\mathbf{n}$ is a parameter (and not an integer). To parameterize types, we use *linear combinations of parameters* $\mathbf{c}, \mathbf{d}, \ldots$ defined by :

$$\mathbf{c} ::= 0 \mid \mathbf{n} \mid \mathbf{n} + \mathbf{c}.$$

The *parameterized types* are defined by :

$$A ::= !^{\mathbf{c}}\alpha \mid !^{\mathbf{c}}(A \multimap A).$$

An *instantiation* ϕ is a map from parameters to $\mathbb{Z}$.

Now given a parameterized pseudo-term t and a parameterized assignment Σ, we can associate to them two classes of linear inequations/equations $\mathcal{C}^b(t)$ and $\mathcal{C}^{typ}(t, \Sigma)$ expressing respectively the facts that an instantiation ϕ is such that $\phi(t)$ satisfies the boxing criterion (conditions 1 and 2) and $(\phi(t), \phi(\Sigma))$ satisfies the typing conditions (condition 3).

[2]They are different from the soft pseudo-terms considered in Section 2.2.

Denote $\mathcal{C}(t,\Sigma) = \mathcal{C}^b(t) \cup \mathcal{C}^{typ}(t,\Sigma)$. By analyzing the form of the constraints, we show that this system has a solution in $\mathbb{Z}$ if and only if it has one in $\mathbb{Q}$. Therefore, using Kachian's Theorem [Kac79, Kar84] we obtain that this constraints system can be solved in time polynomial in $(|t|+|\Sigma|)$, by using linear programming.

Finally, given a simply typed term $M : B$ as in Problem 1, by freely decorating the term and its types with parameters, we get a parameterized term $\overline{M}$ and a parameterized assignment $\overline{\Gamma}$. Using the previous results and the solving argument for $\mathcal{C}(\overline{M},\overline{\Sigma})$, we obtain that :

Theorem 13 *[BT05] The Problem 1 whether the simply typed term $M : B$ can be decorated in $EAL^\star$ can be solved in time polynomial in the simple type derivation of $M : B$.*

This allows to give an $EAL^\star$ type inference algorithm by first inferring the principal simple type and then using the decoration algorithm.

During his Master thesis [Ata05], Vincent Atassi has implemented this $EAL^\star$ type inference algorithm in OCAML, using a standard solver. In a Chapter of his on-going PhD thesis, which I am co-advising with Jacqueline Vauzeilles, he has shown in particular that type inference in EAL can be generalized to a type system EAL_{coer} extended with implicit coercions. Indeed programming in EAL requires in practice many coercions which make tedious the writing even of simple algorithms, and which do not have any real computational content. Type inference in EAL_{coer} thus allows to delegate to the type system this task of data management and hence to write algorithms in a more natural way.

Type inference in DLAL. We have stressed above certain limits of the type inference procedure for LAL of [Bai02] : it is not defined for second-order types and does not have a reasonable complexity. Starting from the technique of [BT05] with Atassi and Terui we then addressed in [ABT06, ABT07] the issue of type inference in DLAL (with second-order) for terms typed in system F. Our idea was that since type inference with polymorphism has difficulties of its own (recall that type inference in system F is undecidable [Wel99]), it was better to decompose the difficulty by assuming the system F type already given together with the term, for instance provided by the user or by a specific partial algorithm for system F typing. Let us now state more precisely the problem addressed.

The language $\mathcal{L}_F$ of system F types is :

$$T, U ::= \alpha \mid T \to U \mid \forall \alpha . T \ .$$

The terms of system F (*à la Church*) are built as follows (here we write M^T to indicate that the term M has type T) :

$$x^T \quad (\lambda x^T . M^U)^{T \to U} \quad ((M^{T \to U}) N^T)^U \quad (\Lambda \alpha . M^U)^{\forall \alpha . U} \quad ((M^{\forall \alpha . U}) T)^{U[T/\alpha]},$$

with the condition that when building a term $\Lambda\alpha.M$, α does not occur free in the types of free term variables of M (the *eigenvariable condition*).

The problem considered in [ABT07] is then the following one :

Problem 2 (DLAL typing) *Given a closed term M^T of system F, does there exist a decoration A in DLAL of T such that we have $\vdash_{DLAL} M : A$?*

(Here also the closedness assumption is only for readability.)

The principle of the algorithm that we designed is illustrated by Fig. 2 from the introduction. An advantage of DLAL with respect to LAL is that it does not use alternations of exponentials !/§, and this allows to restrict the search space for types. Our method uses a parameterization with on the one hand boolean parameters $\mathbf{b_i}$ (for the distinction between $\multimap$ and $\Rightarrow$) and on the other hand integer parameters $\mathbf{n_j}$, for the exponentials § and the doors of boxes. As previously, using the boxing criterion and unification typability is reduced to the satisfiability of a constraints system $\mathcal{C}$. These constraints are however more involved than the previous ones for $EAL^\star$. Actually, to be more precise, the different kinds of constraints generated correspond to different subtasks of the algorithm :

- analysing where non-linear application (and hence !-box) is needed : this is expressed by *boolean constraints* ;
- searching for a suitable distribution of boxes (! or § boxes) : this corresponds to finding values for integer parameters (corresponding to doors) in such a way that constraints corresponding to validity of boxes are satisfied.

In this way a set of constraints on boolean ($\mathbf{b_i}$) and integer ($\mathbf{n_i}$) parameters is associated to the term, expressing its typability. It contains :

boolean constraints,	*e.g.*	$\mathbf{b_1} = \mathbf{b_2}, \quad \mathbf{b_1} = \mathbf{0}$
linear constraints,	*e.g.*	$\sum_i \mathbf{n_i} \geq 0$
mixed boolean/linear constraints,	*e.g.*	$\mathbf{b_1} = 1 \Rightarrow \sum_i \mathbf{n_i} \geq 0.$

Because of the mixed boolean/linear constraints, it is not at first obvious whether these constraints have a lower complexity than NP. An efficient resolution method for solving the constraints system is however given by the following two-step procedure :

1. boolean phase : search for the *minimal* solution to the boolean constraints. This corresponds to doing a linearity analysis (determine which applications are linear and which ones are non-linear).
2. linear programming phase : once the constraints system is instantiated with the boolean solution, we get a linear constraints system, that can be solved with linear programming methods. This corresponds to finding a concrete distribution of boxes satisfying all the conditions.

This resolution procedure is correct and complete and it can be performed in polynomial time w.r.t. the size of the original system F term, using the same kind of argument as for $EAL^\star$ in [BT05] for the solving of the linear inequations. Any solution of the constraints system gives a valid DLAL type derivation. This allowed us to conclude with the following main result :

Theorem 14 *Given a system F term M^T, it is decidable in time polynomial in the size of M whether there is a decoration A of T such that $\vdash_{DLAL} M : A$.*

Atassi has written an OCAML program implementing this algorithm, which we have tested on significant examples (see [ABT07]) and which is available from :

`http://www-lipn.univ-paris13.fr/~atassi/`.

The table of Fig. 2.5 finally summarizes the results that we have obtained for type inference in the various systems.

Context and related works. Our activity in this research line on type inference in light logics has been carried out in large part in the frameworks of the projects GEOCAL, CRISS and more recently NOCoST. The setting of CRISS in particular has motivated our work by analogy with other

Type systems	LAL	LAL	EAL	DLAL
Complexity guaranteed by typing	Ptime	Ptime	Elementary	Ptime
Reference	[Bai02]	[Bai04b]	[BT05]	[ABT07]
Constraints for inference	Presburger arithmetic	word constraints	linear inequations	mixed boolean/linear constraints
Complexity of the inference algorithm	Not known (*)	Not known (*)	Ptime	Ptime

FIG. 2.5 – Summary : type inference algorithms for light logics.
(*) at least exponential.

implicit computational complexity systems for which the issue of inference of complexity certificates is raised in a similar way. For instance in the case of *quasi-interpretations* for rewriting systems, inference of quasi-interpretation (also called *synthesis*) can with some conditions be performed by constraints solving [Ama05, BMMP05].

The research line on type systems derived from light logics and corresponding type inference algorithms has seen several contributions since 2001. The papers [CM06, CRDR05] have dealt with type inference for EAL. Here a motivation was in particular the application of EAL to optimal reduction (see Section 2.5). Then [CDLRDR05] has defined a new type assignment system derived from EAL in order to ensure subject-reduction, considering a call-by-value λ-calculus. More recently in [GR07] Gaboardi and Ronchi Della Rocca have addressed an issue analogous to that of [BT04] but for SLL : the authors defined a type system STA from SLL which ensures subject-reduction and a Ptime complexity bound on β-reduction. A first investigation on type inference for STA was led by Gaboardi in [Gab07] with the goal of inferring at the same time the polymorphic structure and the exponential decoration of the type.

Finally the system STA was also used in [GMRDR08] where Gaboardi, Marion and Ronchi Della Rocca gave an extension STA_B of this system for a λ-calculus with a conditional operator and proved that this language characterizes the class Pspace. Interestingly enough this work, together with the system of [Sch07] for Logspace, illustrated the fact that the light logic approach is also able, besides time complexity classes, to characterize space complexity classes.

2.5 Light logics and optimal reduction of λ-calculus (2006-2007)

Let us recall that *optimal reduction* of λ-calculus, following J.-J. Lévy, refers to an evaluation method which does a minimal work in the sense that no redex is duplicated. J. Lamping [Lam90] has given in 1990 the first algorithm realizing optimal reduction, based on a graph-rewriting procedure defined by local rewriting steps. Gonthier, Abadi and Lévy [GAL92] have in 1992 explained this algorithm and its correctness by using proof-nets of linear logic and a particular model of geometry of interaction, *context semantics*. Asperti has observed that for terms typed in EAL Lamping's algorithm admitted a simplified version : Lamping's *abstract algorithm*. By the way this has been one of the main motivations for the study of EAL in later works. Therefore, terms typed in EAL or LAL offer (beside β-reduction) at least two execution methods :

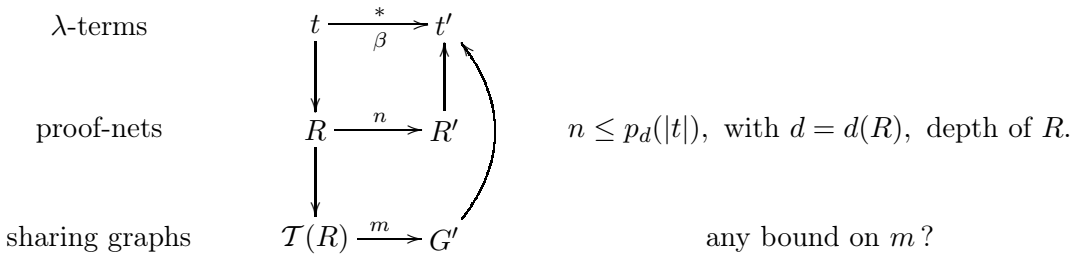

FIG. 2.6 – Optimal reduction for LAL/EAL typable terms.

– (i) execution by means of proof-net reduction (EAL or LAL proof-nets),
– (ii) execution by means of Lamping's abstract algorithm for optimal reduction.

However, even if (ii) is optimal in the sense of Lévy, no bound was known on its number of execution steps, while (ii) is not optimal in the sense of Lévy, but however offered an explicit elementary (for EAL) or Ptime (for LAL) bound.

With Coppola and Dal Lago we have then in [BCDL07] reconciliated these two viewpoints (respectively that of Lévy-optimality and that of quantitative bounds) by proving that for terms typable in LAL (resp. EAL), Lamping's abstract algorithm also offered a polynomial (resp. elementary) time bound. This has been established by using as technical tool a notion of weight inspired from [DL06] and geometry of interaction/context semantics, which allowed to relate together the cost of execution by means of proof-nets and that of execution by means of Lamping's algorithm. Note that while context semantics was used in [GAL92] to prove a *qualitative property* (correctness), here it is also employed to prove a *quantitative property*. This result moreover illustrated the fact that for a term which we know is typable in LAL, the information provided by boxes is not needed to evaluate this term within the polynomial bound.

Let us explain in a little more detail the result and the outline of the proof. We consider type assignment in systems EAL_μ and LAL_μ, the extensions of EAL and LAL with fixpoints of formulas (recursive types). If a term t has a type derivation, one can associate to it a proof-net R : the normal form t' of t can be computed by reducing R into its normal form R' and then extracting t' from R'. This is illustrated by the top square of the diagram of Fig. 2.6. Let us examine a corresponding procedure for sharing graphs.

From proof-nets to sharing graphs. The sharing graph is obtained from the proof-net essentially by removing the boxes and replacing contraction nodes by *fans*. The delicate point is how to set the indices for the fans. In the paper we define a class of admissible translations from proof-nets to sharing graphs, subsuming the previous ones from the literature : for that, the only condition required is that *two fans of same index in the sharing graph have to come from contractions of same depth in the proof-net*. We denote by $\mathcal{T}^{\mathsf{LAL}}_{\mathsf{ASR}}(.)$ (resp. $\mathcal{T}^{\mathsf{EAL}}_{\mathsf{ASR}}(.)$) such a translation for LAL_μ (resp. EAL_μ) proof-nets. In particular, the most common translation is to use the depth of the contraction node in the proof-net as index for the fan in the sharing graph.

The abstract algorithm. The rewriting rules of Lamping's abstract algorithm are recalled on Fig. 2.7. The specificity of this version of the algorithm w.r.t. Lamping's general algorithm is that the indices on fans remain unchanged ; no computing for updating of indices is thus required. This is the rewriting relation used on sharing graphs on the bottom line of Fig. 2.6. Once we have

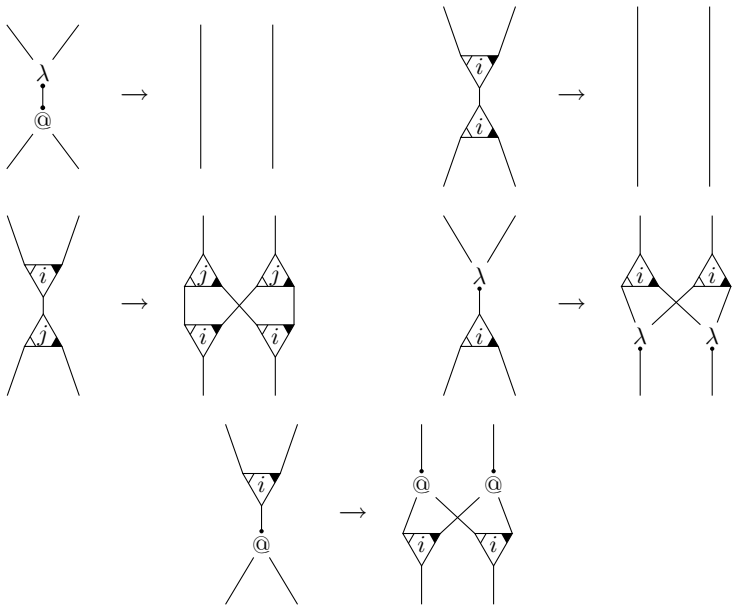

FIG. 2.7 – Rewriting rules for sharing graphs

reached a normal sharing graph, we can extract from it a λ-term by using a *read-back procedure*. Note that since read-back procedures of the literature had been designed for sharing graphs coming from generic translations of λ-terms, which is not the case here (because our translation of λ-terms into sharing graphs is based on type derivations), we had to define a specific read-back procedure.

The main contributions of the article [BCDL07] are to show :

1. that Lamping's abstract algorithm is correct for terms typable in LAL_μ or EAL_μ ;
2. that the number of steps of the sharing graph rewriting on Fig. 2.6 (denoted m on the Figure) is, at fixed depth, in the case of LAL_μ (resp. EAL_μ) polynomial in $|t|$ (resp. elementary in $|t|$).

More formally, the Theorem corresponding to point 2 is the following one :

Theorem 15 *For every natural number n, there is a polynomial (resp. elementary function) $e_n : \mathbb{N} \to \mathbb{N}$ such that for every term t typable in LAL (resp. EAL), if R is a proof-net corresponding to a type derivation of t, then any reduction of the sharing graph $\mathcal{T}^{\mathsf{LAL}}_{\mathsf{ASR}}(R)$ (resp. $\mathcal{T}^{\mathsf{EAL}}_{\mathsf{ASR}}(R)$) has a length bounded by $e_{d(R)}(|R|)$, where $d(R)$ is the depth of R.*

Point 1 above (correctness) is proven in a somewhat 'classical' way, in the style of [GAL92] using geometry of interaction (GoI). We actually use for that a simple GoI/context semantics tailored for EAL and following [BP01]. The paths considered for this argument and characterized by GoI are, as in [GAL92], paths from conclusion to conclusion in the graph.

The quantitative result 2 requires a more involved proof. The idea is to use an indirect argument based on a notion of *weight* following that introduced by Dal Lago in [DL06]. The weight W_G of a sharing graph is an integer corresponding intuitively to the sum for each node in the graph of all possible copies of it created during reduction. Formally, the weight is defined statically by counting

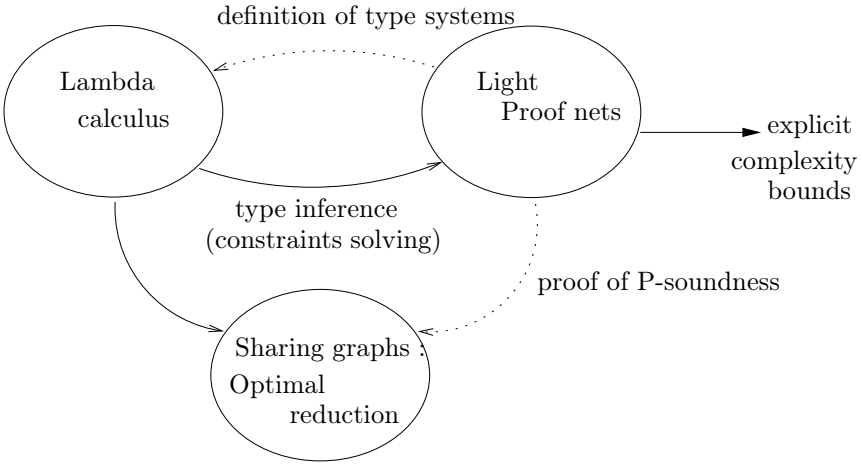

FIG. 2.8 – λ- terms and Light proof-nets.

on the graph certain classes of paths, characterized by using again geometry of interaction. However the paths considered here are not only conclusion-to-conclusion paths as in the correctness argument, but they can start or end in internal nodes. Then the argument proceeds in two steps :

- bound the number of steps of Lamping's algorithm by using the weight (Prop. 7 in [BCDL07]),
- bound the weight itself : for a sharing graph coming from an LAL_μ or EAL_μ proof-net R, this is obtained by using the bound on proof-net reduction and studying the decrease of weight during proof-net reduction (Prop. 8 and 9 in [BCDL07]).

Putting these two points together and using the fact that LAL_μ (resp. EAL_μ) proof-nets have (at fixed depth) a polynomial (resp. elementary) bound on normalisation, one gets Theorem 15.

This work has been carried out during the Postdoctoral Fellowship of Dal Lago and a visit of Coppola (Udine) at University Paris 13 in 2006.

To sum up, the relationships between λ-calculus, LAL proof-nets and sharing graphs are illustrated on Fig. 2.8.

Chapitre 3

Research perspectives

The future research directions that I would like to explore can be gathered in two main axes : on the one hand (3.1) directions related to the applications of linear logic to ICC, and on the other hand (3.2) some issues related more generally to ICC characterizations, the study of the limitations of these characterizations and the investigation of their possible extension to the setting of concurrent systems.

3.1 Linear logic and implicit computational complexity

Linear logic by levels and program extraction from proofs. With Damiano Mazza we have recently (during his Postdoctoral Fellowship at LIPN) introduced a new system of proof-nets called *linear logic by levels* (L^3) [BM07]. Our goal was to generalize the proof-nets of ELL and LLL by seeing them as subsets of a larger class defined by simpler conditions, L^3. More precisely we show that while in LLL the complexity property is guaranteed by a sophisticated system of boxes (!, §), we can replace a large part of these boxes (§ ones) by some informations of *levels* in proof-nets. We prove that the resulting system, L^3, admits an elementary complexity and that a subsystem, L^4, generalizing LLL, admits a polynomial time complexity. The goal we aim at is twofold :

1. on the one hand, a simplified system can help in improving our understanding of the principles at work in LLL,
2. on the other hand, these proof-nets should allow to define a type system which would be clearer than DLAL and would also guarantee a Ptime complexity.

We plan to explore in the sequel point 2., that is to say a type system for λ-calculus derived from L^4, built as DLAL [BT04] had been built from LLL. This system would guarantee a Ptime bound on typable λ-terms, but the constraints to solve for type inference should be simpler. Indeed they will essentially correspond to unification conditions, because the part of constraints needed for ensuring validity of boxes will be largely reduced.

On the other hand it would be interesting to study light logics in the setting of *program extraction from proofs*. Our idea is to use the principle of *programming with proofs* : one starts with a specification, of which we prove the totality by using a particular proof system, which here would be based on a light logic, and finally we extract from the proof a λ-calculus program. In this case the light logic would guarantee that the extracted program is of Ptime complexity. It would thus be an extraction of *feasible program*.

Recall that this idea had been already proposed by Girard in [Gir98] with *light naive set theory*, which has also been studied by Terui [Ter04]. It is however interesting to reexamine this approach in the framework of a second-order logic, like for instance a light adaptation of the system AF2 (Krivine). One of the difficulties for experimenting this approach lies in the fact that to write a proof of totality in a light logic system, a delicate task is to use the exponentials (modalities) in a suitable way. But the results on type inference could probably be used to automatize at least partially this task, in the setting of an interactive proof system. The recent work on linear logic by levels [BM07] mentioned above should certainly be useful in this direction, because we think that it will allow to define a simplified proof system, with fewer sequentiality conditions on the exponentials (corresponding to boxes).

If this attempt leads to relevant results in the setting of second-order logic, it will be worthwhile afterwards to examine whether these ideas could also be applied to the setting of the *calculus of inductive constructions*, which is at the basis of the Coq proof-assistant. Indeed one could then try to delineate certain classes of proofs in Coq (with some extra informations) for which the program extraction would produce certified feasible programs.

Optimal reduction and complexity. Our work [BCDL07] has shown that for λ-calculus terms typable in LAL, Lamping's algorithm for optimal reduction performs a Ptime evaluation. This has allowed to join the two research lines of optimal reduction on the one hand and implicit computational complexity on the other. It is however only a first step and several questions remain open concerning the applications of optimal reduction techniques to implicit computational complexity. First, the complexity bound obtained in [BCDL07] for optimal reduction in EAL/LAL is of the same order of complexity as that for the proof-nets, but it is not exactly the same bound. Therefore, is it possible to prove that the bound for proof-net reduction is also valid for optimal reduction ? Second, the bound obtained in this article does not take into account the final *readback* work allowing to extract the concrete value of the result from the "compressed" representation as a sharing graph. It is very likely that the cost of the readback can also be explicitly bounded, but this remains to be proven.

These two extensions of our results could perhaps be established by using the same tool as in the previous article, that is to say geometry of interaction. If it is not sufficient another possibility could be to try to use more syntactical techniques, by considering translations between graph rewriting systems, following the kind of approach of [GMM03].

Besides it is also natural to wonder to which other systems these results on optimal reduction could be extended. For instance would there be a Ptime bound also for optimal reduction of terms typable in SLL [Laf04, GR07] or in the new system L^4 [BM07] ? This latter objective will probably require to extend to L^4 techniques for ELL/LLL used in [BCDL07] and for that to define and study a geometry of interaction for L^4.

Denotational semantics and game semantics. There seems to remain quite some work to be done on the semantic side for the study of implicit computational complexity. Indeed on the side of syntax we now have at hand a good variety of calculi and logics corresponding to different complexity classes, but which are difficult to compare one to the other or to combine ; moreover they sometimes rely on criteria which are quite sophisticated. On the other hand there are semantics which have good mathematical properties and which present features related to quantitative aspects of computation :

– game semantics, with the notion of *play* which allows to represent explicitly the interaction between program and user/system ;
– coherence or relational models, in which the number of repetitions of an argument is explicit (in the multiset versions).

We know that these models are related to abstract machines for functional programs and to the execution time in these machines, as this has been shown for games (see e.g. [DHR96, Bai99, CH98]) or for the multiset relational semantics (see in particular [dC07]). It would thus be challenging to exhibit some subcategories of such models in which the morphisms would correspond to algorithms of a given complexity class, for instance Ptime. This could perhaps allow to give a more abstract account of complexity-bounded computation, and we would like to find models of this kind in which we could interpret several distinct ICC systems. Recall that a game model for LLL had been given by Murawski and Ong in [MO00]. They proved for this model a full completeness result : any strategy comes from an LLL proof. Our goal would rather be to define a complexity-bounded model, but which would not necessarily be reducible to an already-known ICC calculus, and instead would rely on more general principles. In collaboration with O. Laurent and P. Boudes we have made some first steps in this research direction, working in the framework of game semantics. This perspective was one of the motivations of the NOCoST project (ANR), but since it seems to be quite a difficult and general problem it is more likely to be continued by a longer term research line.

Note also that recent work by Girard (see e.g. [Gir07]) in geometry of interaction in the setting of Von Neumann algebras seems to suggest possible new foundations for ICC and the characterization of Ptime. As previous models of geometry of interaction had deeply influenced game semantics and provided techniques for the study of λ-calculus, a different approach to geometry of interaction could thus be the basis for new insights and techniques of this kind.

Extension of type inference. Our work on type inference in light logics suggests several further research directions. For DLAL, type inference has been studied for a system with polymorphic types [ABT07] (second-order quantification) and a related work has been carried out in his PhD Thesis [Gab07] by Gaboardi (Torino) for a system derived from SLL (the type system STA [GR07]). In this latter case, the goal is also to infer the polymorphic type structure together with the modalities. Besides we know that these systems can be extended with recursive types (that is to say with type fixpoints) as in LAL_μ, and keep the same properties. It would thus be natural to study inference in these systems with recursive types, or in a similar extension of the system L^4 mentioned previously.

On the other hand, it appears that the techniques we have developed for inference in these systems for light logics, by constraints solving, are not specific to this setting. They could be adapted to typing in general linear logic or in a type system based on modal logics (like S4). We think that the approach based on the boxing criterion [Bai02, BT05] can be applied to these new settings, even if this may require to consider new classes of constraints, more general (not only linear constraints or mixed boolean/linear constraints as in [ABT07]). However it seems possible to define in this way Ptime type inference algorithms. A work in collaboration with M. Hofmann on this topic is in progress.

Until now, our type inference methods by constraints generation have been developed on a case-by-case basis for each system (LAL, EAL, DLAL) by each time considering some different kinds of constraints. It would on the mid-term be more motivating to try to develop an integrated method, that would in a single run generate only one constraints system, which we could then analyze or solve to determine in which systems and how the program is typable : LL, EAL, DLAL, SLL, L^4...Ideally

we would like in this way to statically determine some upper bounds on the time and space needed for the evaluation of the program, as well as possibly some indications on which evaluation method to apply in order to satisfy these bounds.

Comparing different kinds of criteria. We have explained that some of our type inference methods (for EAL, DLAL ...), which are used then for ICC criteria, rely on a *boxing criterion* (coming from [Bai02, BT05]). This criterion indicates if and in which way the program can be divided into *boxes*, which each corresponds to a logical (sequential) step in the typing process. Therefore the boxing criterion bears an analogy with the *correctness criteria* of linear logic proof-nets. There exist many variants of correctness criteria (the first one [Gir87] is due to Girard and the most commonly used one, to Danos-Régnier) ; their goal is to indicate necessary and sufficient conditions for a graph (*proof structure*) to be sequentializable in a linear logic proof-net.

We thus have three different kinds of criteria, informally related : ICC criteria, boxing criteria, correctness criteria. The third class of criteria applies to proof structures and the two first ones to λ-calculus terms. We can think of extending the second class of criteria to proof-structures. This would thus suggest to examine more closely the analogies between the boxing and the correctness criteria. Could a boxing criterion for instance suggest a new correctness criterion, which would proceed by generation and solving of constraints ? Conversely, as the theory of correctness criteria is now well-established, could it help to shed a new light on the more recent theory of boxing and ICC criteria ?

3.2 Analysis and extension of implicit computational complexity criteria

Abstract study of implicit complexity criteria. In the paper [BDLM06] with Dal Lago (Bologna) and Moyen (LIPN), we have started to analyze the behaviour of programs (first-order term rewriting systems) satisfying the implicit complexity criterion for the characterization of Ptime from [BMM07] (that we call here the *P-criterion*), based on *quasi-interpretations*. For that we have used a notion of abstraction on programs, called *blind abstraction*. We would like to continue this line of research by studying various ICC criteria and aim at :

- for a given criterion, finding out necessary conditions on the behaviour of programs satisfying the criterion, for instance by using various notions of abstractions on programs ;
- possibly, for certain criteria giving some necessary and sufficient conditions on the behaviour of the program for it to meet the criterion.

There are several motivations for that. First this approach could help in understanding the theoretical limitations of these implicit complexity criteria, and allow for instance to state *a priori* that certain families of programs will *not* be characterizable by a given criterion. On the other hand this could also be a way to compare together different ICC criteria, by comparing the respective classes of programs that they each allow to delineate. Finally from a pragmatic viewpoint, we can use these characterizations in order to obtain ideas on how to extend or generalize the initial criterion considered. This is for instance what we have done already in the case of the P-criterion [BDLM06], by generalizing the notion of recursive path ordering (RPO) that is used in the P-criterion and proving that the complexity property remains valid with this weaker assumption.

Another interesting direction in the general study of ICC criteria deals with the difficulty of the associated decision problems. Recall that the general problem, given a program, to decide if it is

Ptime (or of another complexity) is, as the halting problem, undecidable (provided we are considering a sufficiently rich programming language). The various ICC criteria only give a sufficient condition for this property to hold. They differ from each other by their intensional expressivity (the number and variety of programs validated) and the difficulty of the associated decision problem : given a program, does it satisfy the criterion ? For instance the system DLAL and the P-criterion with quasi-interpretations [BMM07] both characterize Ptime functions, but they have associated decision problems with upper bounds, respectively Ptime and doubly exponentials (with quasi-interpretations searched in a particular function algebra with bounded polynomial degree, [BMMP05]). The intensional expressivity of the P-criterion is also larger than that of DLAL (even if they cannot be compared formally because they do not apply to the same source language). It seems likely that the expressivity of a criterion and the difficulty of its decision problem are inversely correlated : the more general a criterion is, the more difficult we expect the testing of this criterion to be. However is this really necessary and can this intuition be expressed formally ? For instance could one provide some general laws relating expressivity of the criterion and difficulty of the decision problem ? Could one define some families of criteria, with calibrated expressivity and difficulty, in such a way to be able to adapt the granularity of the analysis on the program by picking up one criterion in the family ? The article [Asp08] for instance might be interesting for these questions since it adapts the notions of computability theory in order to handle decision problems on complexity issues.

Implicit complexity for concurrent systems. Implicit computational complexity has been until now investigated mainly for functional programming and more recently, in some settings of imperative programming. We would like to study how this kind of approach could be adapted to the setting of concurrent processes, like for instance to the framework of *π-calculus* or that of the *ambient calculus*. Some works have been carried out by Amadio and co-authors [AD07] in a synchronous framework, with a notion of global instant. In this context they have used some methods from quasi-interpretations to guarantee bounds on the reactivity (*feasible reactivity*). These works constitute an interesting step, but we would like to examine whether some properties could also be obtained in the setting of an asynchronous calculus. The goal would be to statically guarantee certain quantitative properties (with respect to the size of the initial system), like for instance a bound on the size of processes during execution, or possibly a bound on the number of execution steps before terminating or, say, returning to a given state.

A possible starting direction could be to take inspiration from logical systems like LLL or L^4 and to try to define some analogous notions of boxes or levels in the process calculus considered. For that an interesting intermediary step could be to use the work of Ehrhard and Laurent [EL07] on the encoding of π-calculus into *differential interaction nets*. One could try indeed to adapt the approach of L^4 (initially designed for proof-nets) to differential interaction nets, and through the translation, to import it into π-calculus.

Conclusion

The works we presented here have contributed to the understanding of how linear logic can be used for the guarantee of complexity properties, in particular : how to show with a formal proof that a program is feasible (of Ptime complexity) ?

For that in a first part of our work we have studied the properties of the light logics derived from linear logic. We have given denotational semantics of the systems ELL and LLL based on coherence spaces, and which are specific to these systems in the sense that they are not models of ordinary linear logic. We have then explored the computational properties of light logics through the Curry-Howard approach. We have defined a term calculus corresponding to the system SAL (affine version of SLL) and, using this term calculus, we have studied the expressivity of SAL extended with type fixpoints. We have then considered the extension of LAL (affine variant of LLL) with type fixpoints and explored the classes of functions representable in various fragments of this system, by considering a general notion of encoding called uniform encoding. In particular, we have shown that with this notion of encoding, LAL without second-order quantifiers and with a linear notion of fixpoints characterizes the class FP of polynomial time functions.

In a second part, we have recast the study of light logics as a study of λ-calculus, by using these logics as type systems. A type for a λ-calculus program becomes in this setting a certificate that the program can be evaluated with a certain complexity bound (possibly by evaluating it using proof-nets). Deciding if a program admits such a certificate becomes then a type inference problem, which we have studied for several systems. In order to solve it we have adapted to this setting the classical method of type inference by constraints generation and solving. We have in this way established for several systems an upper bound on the complexity of this decision problem, by using various constraints resolution methods. We have shown in particular that type inference for (propositional) LAL is decidable. We have then introduced DLAL, a type system corresponding to a fragment of LAL but with better properties : DLAL ensures Ptime bounds for β-reduction and satisfies subject-reduction. We have shown that type inference for the systems EAL (for elementary complexity) and DLAL can be performed in polynomial time, respectively by solving linear constraints or mixed boolean/linear constraints. Finally we have studied the evaluation of EAL and LAL typable programs by using Lamping's algorithm for optimal reduction, and shown that this evaluation admits an elementary (for EAL) and polynomial (for LAL) time complexity bound.

Bibliographie

[ABT06] V. Atassi, P. Baillot, and K. Terui. Verification of Ptime reducibility for system F terms via Dual Light Affine Logic. In *Proceedings of CSL'06*, number 4207 in LNCS, pages 150–166. Springer, 2006.

[ABT07] V. Atassi, P. Baillot, and K. Terui. Verification of Ptime reducibility for system F terms : type inference in Dual Light Affine Logic. *Logical Methods in Computer Science*, 3(4 :10) :1–32, 2007.

[AD07] R. M. Amadio and F. Dabrowski. Feasible reactivity in a synchronous pi-calculus. In Michael Leuschel and Andreas Podelski, editors, *Proceedings of the 9th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP'07)*, pages 221–230. ACM, 2007.

[Ama05] R. M. Amadio. Synthesis of max-plus quasi-interpretations. *Fundam. Inform.*, 65(1-2) :29–60, 2005.

[AR02] A. Asperti and L. Roversi. Intuitionistic light affine logic. *ACM Transactions on Computational Logic*, 3(1) :1–39, 2002.

[Asp98] A. Asperti. Light affine logic. In *Proceedings LICS'98*, pages 300–308. IEEE Computer Society, 1998.

[Asp08] A. Asperti. The intensional content of Rice's theorem. In *Proceedings of ACM Symposium on Principles of Programming Languages POPL'08*, 2008.

[Ata05] V. Atassi. Sous-typage et coercitions dans ELL. Master's thesis, Université Paris 13, 2005.

[Bai99] P. Baillot. *Approches dynamiques en sémantique de la logique lineaire : jeux et géometrie de l'interaction.* PhD thesis, Université Aix-Marseille II, January 1999.

[Bai02] P. Baillot. Checking polynomial time complexity with types. In *Proceedings of International IFIP Conference on Theoretical Computer Science 2002*, pages 370–382, Montreal, 2002. Kluwer Academic Press.

[Bai04a] P. Baillot. Stratified coherence spaces : a denotational semantics for Light Linear Logic. *Theoretical Computer Science*, 318(1-2) :29–55, 2004. Une version préliminaire a été présentée au Workshop ICC'00, Santa Barbara (USA), Juin 2000.

[Bai04b] P. Baillot. Type inference for Light Affine Logic via constraints on words. *Theoretical Computer Science*, 328(3) :289–323, 2004.

[Bai07] P. Baillot. From Proof-Nets to Linear Logic Type Systems for Polynomial Time Computing. In *Proceedings of Typed Lambda Calculi and Applications (TLCA'07)*, volume 4583 of *LNCS*, pages 2–7. Springer, 2007. Invited talk.

[BBdPH93] P.N. Benton, G.M. Bierman, V.C.V. de Paiva, and J.M.E. Hyland. A term calculus for intuitionistic linear logic. In *Proceedings TLCA'93*, volume 664 of *LNCS*. Springer Verlag, 1993.

[BC92] S. Bellantoni and S. Cook. New recursion-theoretic characterization of the polytime functions. *Computational Complexity*, 2 :97–110, 1992.

[BCDL07] P. Baillot, P. Coppola, and U. Dal Lago. Light Logics and Optimal Reduction : Completeness and Complexity. In *Proceedings of Symposium on Logic in Computer Science (LICS'07)*, pages 421–430. IEEE Computer Society, 2007.

[BCdNM03] O. Bournez, F. Cucker, P. Jacobé de Naurois, and J.-Y. Marion. Computability over an arbitrary structure. sequential and parallel polynomial time. In Andrew D. Gordon, editor, *FoSSaCS*, volume 2620 of *Lecture Notes in Computer Science*, pages 185–199. Springer, 2003.

[BDLM06] P. Baillot, U. Dal Lago, and J.-Y. Moyen. On quasi-interpretations, blind abstractions and implicit complexity. Technical Report hal-00023668, HAL preprint, 2006. Presented at Workshop on Logic and Computational Complexity (LCC'06). 23 pp., https ://hal.ccsd.cnrs.fr/ccsd-00023668.

[BM04] P. Baillot and V. Mogbil. Soft lambda-calculus : a language for polynomial time computation. In *Proceedings of the 7th International Conference on Foundations of Software Science and Computation Structures (FOSSACS'04)*, volume 2987 of *Lecture Notes in Computer Science*, pages 27–41. Springer Verlag, 2004.

[BM07] P. Baillot and D. Mazza. Linear logic by levels and bounded time complexity. 49 pp., submitted, december 2007.

[BMM05] G. Bonfante, J.-Y. Marion, and J.-Y. Moyen. Quasi-interpretations and small space bounds. In Jürgen Giesl, editor, *RTA*, volume 3467 of *Lecture Notes in Computer Science*, pages 150–164. Springer, 2005.

[BMM07] G. Bonfante, J.-Y. Marion, and J.-Y. Moyen. Quasi-interpretations, a way to control resources. *Theoretical Computer Science*, 2007. To appear.

[BMMP05] G. Bonfante, J.-Y. Marion, J.-Y. Moyen, and R. Pechoux. The synthesis of quasi-interpretations. Presented at the International Workshop on Logic and Computational Complexity LCC'05, Chicago, 2005.

[BNS00] S. Bellantoni, K.-H. Niggl, and H. Schwichtenberg. Higher type recursion, ramification and polynomial time. *Annals of Pure and Applied Logic*, 104(1-3), 2000.

[BP01] P. Baillot and M. Pedicini. Elementary complexity and geometry of interaction. *Fundamenta Informaticae*, 45(1-2) :1–31, 2001.

[BP06] P. Baillot and M. Pedicini. An Embedding of the BSS Model of Computation in Light Affine Lambda-Calculus. 8th International Workshop on Logic and Computational C omplexity (LCC'06) (Satellite de LICS'06), Seattle, Prepublication HAL ccsd-00085547, August 2006.

[BT04] P. Baillot and K. Terui. Light types for polynomial time computation in lambda-calculus. In *Proceedings of Symposium on Logic in Computer Science (LICS'04)*, pages 266–275. IEEE Computer Society, 2004.

[BT05] P. Baillot and K. Terui. A feasible algorithm for typing in Elementary Affine Logic. In *Proceedings of Typed Lambda Calculi and Applications (TLCA'05)*, volume 3461 of *LNCS*, pages 55–70. Springer, 2005.

[BT07] P. Baillot and K. Terui. Light types for polynomial time computation in lambda calculus. (long version). Submitted. 35 pages., October 2007.

[CDLRDR05] P. Coppola, U. Dal Lago, and S. Ronchi Della Rocca. Elementary affine logic and the call-by-value lambda calculus. In *Proc. TLCA'05*, LNCS, pages 131–145, 2005.

[CH98] P.-L. Curien and H. Herbelin. Computing with abstract böhm trees. In *Fuji International Symposium on Functional and Logic Programming*, pages 20–39, 1998.

[CM01] P. Coppola and S. Martini. Typing lambda-terms in elementary logic with linear constraints. In *Proceedings TLCA'01*, volume 2044 of *LNCS*, 2001.

[CM06] P. Coppola and S. Martini. Optimizing optimal reduction : A type inference algorithm for elementary affine logic. *ACM Trans. Comput. Log.*, 7(2) :219–260, 2006.

[CRDR05] P. Coppola and S. Ronchi Della Rocca. Principal typing for lambda calculus in elementary affine logic. *Fundamenta Informaticae*, 65(1-2) :87–112, 2005.

[dC05] D. de Carvalho. Intersection types for light affine lambda calculus. *Electr. Notes Theor. Comput. Sci.*, 136 :133–152, 2005.

[dC07] D. de Carvalho. *Sémantiques de la logique linéaire et temps de calcul*. PhD thesis, Université Aix-Marseille II, September 2007. available from `http://iml.univ-mrs.fr/~carvalho/`.

[DHR96] V. Danos, H. Herbelin, and L. Regnier. Game semantics & abstract machines. In *Proceedings of LICS'96*, pages 394–405, 1996.

[DJ03] V. Danos and J.-B. Joinet. Linear logic and elementary time. *Information and Computation*, 183(1) :123–137, 2003.

[DJS94] V. Danos, J.-B. Joinet, and H. Schellinx. On the linear decoration of intuitionistic derivations. *Archive for Mathematical Logic*, 33(6), 1994.

[DJS97] V. Danos, J.-B. Joinet, and H. Schellinx. A new deconstructive logic : Linear logic. *J. Symb. Log.*, 62(3) :755–807, 1997.

[DL03] U. Dal Lago. On the expressive power of light affine logic. In Carlo Blundo and Cosimo Laneve, editors, *ICTCS*, volume 2841 of *Lecture Notes in Computer Science*, pages 216–227. Springer, 2003.

[DL06] U. Dal Lago. Context semantics, linear logic and computational complexity. In *Proc. of LICS 2006*, pages 169–178. IEEE Computer Society, 2006.

[DLB06] U. Dal Lago and P. Baillot. On Light Logics, Uniform Encodings and Polynomial Time. *Mathematical Structures in Computer Science*, 16(4) :713–733, 2006. (Une version préliminaire avait été présentée au *Workshop on Logics for Resources, Processes, and Programs* LRPP'04).

[DLH05] U. Dal Lago and M. Hofmann. Quantitative models and implicit complexity. In *Proc. of FSTTCS 2005*, volume 3821 of *LNCS*, pages 189–200. Springer, 2005.

[EL07] T. Ehrhard and O. Laurent. Interpreting a finitary pi-calculus in differential interaction nets. In Luís Caires and Vasco Thudichum Vasconcelos, editors, *CONCUR*, volume 4703 of *Lecture Notes in Computer Science*, pages 333–348. Springer, 2007.

[Gab07] M. Gaboardi. *Linearity : an Analytic Tool in the Study of Complexity and Semantics of Programming Languages*. PhD thesis, University of Turin/ INPL Nancy, December 2007.

[GAL92] G. Gonthier, M. Abadi, and J.-J. Lévy. The geometry of optimal lambda reduction. In *Proc. of POPL 1992*, pages 15–26. ACM, 1992.

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

[Gir88] J.-Y. Girard. Geometry of interaction I : an interpretation of system F. In Ferro and al, editors, *Proceedings of A.S.L. Meetings*, Padova, 1988. North-Holland.

[Gir91] J.-Y. Girard. Quantifiers in linear logic II. In G. Corsi and G. Sambin, editors, *Nuovi problemi della logica e della filosofia della scienza, Volume II*. CLUEB, Bologna(Italy), 1991. Proceedings of the conference with the same name, Viareggio, 8-13 gennaio 1990.

[Gir98] J.-Y. Girard. Light linear logic. *Information and Computation*, 143 :175–204, 1998.

[Gir07] J.-Y. Girard. Truth modality intersubjectivity. talk at the Siena Workshop on Linear logic (workshop dedicated to J.-Y. Girard), May 2007.

[GMM03] S. Guerrini, S. Martini, and A. Masini. Coherence for sharing proof-nets. *Theoretical Computer Science*, 294(3) :379–409, 2003.

[GMRDR08] M. Gaboardi, J.-Y. Marion, and S. Ronchi Della Rocca. A logical account of pspace. In *Proceedings of Symposium on Principles of Programming Languages (POPL'08)*. ACM, 2008.

[GR07] M. Gaboardi and S. Ronchi Della Rocca. A soft type assignment system for *ambda* -calculus. In Jacques Duparc and Thomas A. Henzinger, editors, *CSL*, volume 4646 of *Lecture Notes in Computer Science*, pages 253–267. Springer, 2007.

[GSS92] J.-Y. Girard, A. Scedrov, and P. Scott. Bounded linear logic : A modular approach to polynomial time computability. *Theoretical Computer Science*, 97 :1–66, 1992.

[Hof00] M. Hofmann. Safe recursion with higher types and BCK-algebra. *Ann. Pure Appl. Logic*, 104(1-3) :113–166, 2000.

[Hof03] M. Hofmann. Linear types and non-size-increasing polynomial time computation. *Inf. Comput.*, 183(1) :57–85, 2003.

[HS04] M. Hofmann and P. J. Scott. Realizability models for BLL-like languages. *Theor. Comput. Sci.*, 318(1-2) :121–137, 2004.

[Jon99] N. D. Jones. Logspace and ptime characterized by programming languages. *Theoretical Computer Science*, 228 :151–174, 1999.

[Kac79] L. G. Kachian. A polynomial algorithm for linear programming. *Soviet Mathematics Doklady*, 20 :191–194, 1979.

[Kar84] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4) :373–396, 1984.

[KOS97] M. I. Kanovich, M. Okada, and A. Scedrov. Phase semantics for light linear logic (extended abstract). In *Mathematical foundations of programming semantics (Pittsburgh, PA, 1997)*, volume 6 of *Electron. Notes Theor. Comput. Sci.* Elsevier, Amsterdam, 1997.

[Laf04] Y. Lafont. Soft linear logic and polynomial time. *Theoretical Computer Science*, 318(1–2) :163–180, 2004.

[Lam90] J. Lamping. An algorithm for optimal lambda calculus reduction. In *Proc. of POPL 1990*, pages 16–30. ACM, 1990.

[Lei94] D. Leivant. Predicative recurrence and computational complexity I : word recurrence and poly-time. In *Feasible Mathematics II*, pages 320–343. Birkhauser, 1994.

[LM93] D. Leivant and J.-Y. Marion. Lambda-calculus characterisations of polytime. *Fundamenta Informaticae*, 19 :167–184, 1993.

[LM94] D. Leivant and J.-Y. Marion. Ramified recurrence and computational complexity ii : Substitution and poly-space. In Leszek Pacholski and Jerzy Tiuryn, editors, *CSL*, volume 933 of *Lecture Notes in Computer Science*, pages 486–500. Springer, 1994.

[LTdF06] O. Laurent and L. Tortora de Falco. Obsessional cliques : a semantic characterization of bounded time complexity. In *Proceedings of the twenty-first annual IEEE symposium on Logic In Computer Science*, pages 179–188. IEEE Computer Society Press, 2006.

[Mau03] F. Maurel. Nondederministic Light Logics and NP-time. In *Proceedings of TLCA'03*, LNCS. Springer, 2003.

[Maz06] D. Mazza. Linear logic and polynomial time. *Mathematical Structures in Computer Science*, 16(6) :947–988, 2006.

[MM00] J.-Y. Marion and J.-Y. Moyen. Efficient first order functional program interpreter with time bound certifications. In Michel Parigot and Andrei Voronkov, editors, *LPAR*, volume 1955 of *Lecture Notes in Computer Science*, pages 25–42. Springer, 2000.

[MN02] H. Mairson and P. M. Neergard. LAL is square : Representation and expressiveness in light affine logic, 2002. Presented at the 4th International Workshop on Implicit Computational Complexity.

[MO00] A. S. Murawski and C.H. L. Ong. Discreet games, Light Affine Logic and PTIME computation. In *Proceedings CSL'00*, volume 1862 of *LNCS*. Springer Verlag, 2000.

[MO04] A. S. Murawski and C.-H. L. Ong. On an interpretation of safe recursion in light affine logic. *Theor. Comput. Sci.*, 318(1-2) :197–223, 2004.

[MT03] H. Mairson and K. Terui. On the computational complexity of cut-elimination in Linear logic. In *Proceedings of ICTCS 2003*, volume 2841 of *LNCS*, pages 23–36. Springer, 2003.

[Red07] B. F. Redmond. Multiplexor categories and models of soft linear logic. In Sergei N. Artëmov and Anil Nerode, editors, *LFCS*, volume 4514 of *Lecture Notes in Computer Science*, pages 472–485. Springer, 2007.

[Rov98] L. Roversi. A Polymorphic Language which is Typable and Poly-step. In *Proceedings of the Asian Computing Science Conference (ASIAN'98)*, volume 1538 of *LNCS*, pages 43–60. Springer Verlag, December 1998.

[Sch07] U. Schöpp. Stratified bounded affine logic for logarithmic space. In *LICS*, pages 411–420. IEEE Computer Society, 2007.

[Ter01] K. Terui. Light affine lambda calculus and polytime strong normalization. In *Proceedings LICS'01*. IEEE Computer Society, 2001.

[Ter04] K. Terui. Light affine set theory : A naive set theory of polynomial time. *Studia Logica*, 77(1) :9–40, 2004.

[Wel99] J. B. Wells. Typability and type checking in system F are equivalent and undecidable. *Ann. Pure Appl. Logic*, 98(1-3), 1999.

Résumé :

Les travaux présentés dans ce mémoire s'inscrivent dans le cadre de l'approche logique de la programmation, par la correspondance de Curry-Howard, et visent à rendre compte dans cette perspective de la complexité calculatoire, et en particulier du calcul en temps polynomial (Ptime). La *correspondance de Curry-Howard*, ou correspondance preuves-programmes, relie l'étude des preuves formelles à celle du λ-*calcul*, qui constitue le fondement des langages de programmation fonctionnelle comme CAML. Nous abordons le thème de la *complexité implicite*, qui cherche à définir des calculs ou des langages de programmation pour lesquels tous les programmes admettent par construction une borne de complexité donnée, telle que Ptime. Notre outil principal est la *logique linéaire*, qui rend finement compte des opérations sur les données, en particulier au moyen de modalités spécifiques pour la duplication. Des variantes de cette logique, appelées *logiques light*, comme les systèmes LLL (Girard) ou SLL (Lafont), fournissent des systèmes de complexité implicite caractérisant la classe de complexité Ptime.

Nos contributions ont porté sur plusieurs aspects de l'étude des logiques light. En amont d'une part, nous avons étudié la sémantique dénotationnelle et les possibilités calculatoires de ces systèmes (extension de ces systèmes avec des points fixes ; calcul de termes pour SLL).

D'autre part nous avons analysé l'application de ces logiques comme systèmes de types pour le λ-calcul. Dans un premier temps, nous avons considéré le système LAL (variante de LLL), dans lequel si un programme de λ-calcul est bien typé, alors il peut être exécuté en temps polynomial (Ptime). Nous avons montré que le fait de déterminer si un programme admet un type (inférence de type) dans LAL (sans polymorphisme) est un problème décidable.

Dans un deuxième temps, nous avons défini à partir de LAL un nouveau système de types, DLAL, admettant de meilleures propriétés vis-à-vis du λ-calcul, et certifiant aussi une complexité Ptime. Nous avons considéré le problème de l'inférence de type d'abord dans le système EAL (correspondant au temps élémentaire) puis dans DLAL, et fourni dans les deux cas un algorithme d'inférence efficace (Ptime), basé sur la programmation linéaire.

Enfin, nous avons étudié l'évaluation des programmes de EAL et LAL par le biais de la *réduction optimale* : cette méthode s'appuie sur la réécriture de graphes et décompose l'exécution en étapes purement locales. Nous avons démontré qu'elle permettait bien d'évaluer les programmes de EAL et LAL avec les bornes de complexité attendues, respectivement élémentaire et Ptime.

Ce mémoire s'appuie notamment sur des travaux en collaboration avec V. Atassi, P. Coppola, U. Dal Lago, V. Mogbil et K. Terui.