Under consideration for publication in Math. Struct. in Comp. Science

On Light Logics, Uniform Encodings and Polynomial Time

Ugo Dal Lago†

Dipartimento di Scienze dell'Informazione, Università di Bologna,

Mura Anteo Zamboni 7, 40127 Bologna, Italy.

`dallago@cs.unibo.it`

Patrick Baillot‡

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

Intsitut Galilée, 99 avenue Jean-Baptiste Clément, 93430 Villetaneuse, France.

`patrick.baillot@lipn.univ-paris13.fr`

Received 27 July 2005; Revised 31 January 2006

Light Affine Logic is a variant of Linear Logic with a polynomial cut-elimination procedure. We study the extensional expressive power of Light Affine Logic with respect to a general notion of encoding of functions, in the setting of the Curry-Howard correspondence. We consider Light Affine Logic with both fixpoints of formulae and second-order quantifiers and analyze the properties of polytime soundness and polytime completeness for various fragments of this system. We show in particular that the implicative propositional fragment is not polytime complete, if we add some reasonable conditions on the encodings. Following previous work, we show that second order leads to polytime unsoundness. We then introduce simple constraints on second order quantification and fixpoints, proving the obtained fragments to be polytime sound and complete.

1. Introduction

Characterizing the class of functions a logic can represent helps in understanding the computational expressive power of the logic. If the system under consideration enjoys a Curry-Howard correspondence, the analysis can be even more important — the class of representable functions becomes the class of functions the underlying programming language can compute. These investigations become a crucial issue in the context of light

† Work partially supported by project FOLLIA (MIUR).
‡ Work partially supported by projects GEOCAL (ACI), NO-CoST (ANR), *Interaction et Complexité* (cooperation project CNR-CNRS, no 16251).

logics, which have been defined precisely for capturing relevant function classes, namely complexity classes.

Light Linear Logic (**LLL**, (Girard 1998)) has been proposed by Girard as a variant of Linear Logic (**LL**, (Girard 1987)) characterizing the class **FP** of deterministic polynomial time functions. It has been later simplified by Asperti into Light Affine Logic (**LAL**, (Asperti 1998)). The limited computational power of **LLL** (or **LAL**) is obtained by considering a weaker modality ! for resource re-use than in plain Linear Logic. **LAL** has been the subject of many investigations from syntactical, semantical and programming language perspectives (Murawski & Ong 2004, Murawski & Ong 2000, Roversi 2000, Terui 2002). Another line of research in that direction is Lafont's Soft Linear Logic (**SLL**, (Lafont 2004)) which is another variant of **LL** for polynomial time.

Still, one can observe that these characterizations of **FP** via the Curry-Howard correspondence (in **LLL**, **LAL** or **SLL**) only hold provided data are encoded by bounded-depth proofs (the notion of depth is linked to the modalities and to the notion of box). Recently, Mairson and Neergaard (Neergard & Mairson 2002) proved that dropping the bounded box-depth assumption makes **LAL** complete for doubly exponential time. In their setting, data are represented by proofs having unbounded box-depth and different conclusions. Alternative notions of encodings have also been considered in (Mairson & Terui 2003) for various subsystems of **LL**.

An important point is that the encodings in (Girard 1998, Asperti & Roversi 2002) make an extensive use of second-order quantification, which allows programming with polymorphism in the style of System **F**. This is an elegant and general approach, but second-order quantification comes with difficulties of its own, which are not related to **LAL** itself. For instance it makes the issues of provability decision problems, type-inference or semantics far more delicate. One can wonder how much of the power of second-order is really needed in **LAL** to get polynomial time expressivity.

This question is all the more sensible as **LAL** and **SLL** are compatible with another feature: fixpoints. Indeed, fixpoints of formulae were from the beginning one of the intuitions underlying the definition of **LLL** (see the introduction of (Girard 1998)). They are also *definable* in Light Set Theory (**LST**, see (Girard 1998, Terui 2004)), in which they can be used to write function definitions; one can then prove the termination of such functions in **LST** . Alternatively, when considering **LAL** and **SLL** as type systems, fixpoints correspond to recursive types. In particular the expressivity of **SLL** with fixpoints has been examined in (Baillot & Mogbil 2004).

So, as several notions of encodings and a large range of connectives and computational features are available in **LAL**, we think it is important to set up on a *reasonable* notion of an encoding and then to determine the expressivity of small fragments of this logic. This will help to identify well-behaved fragments that might then be used for various purposes like type inference, proof of program termination or proof-search.

In a previous work (Dal Lago 2003), we started focusing our attention to constrained representation schemes, called *uniform coding schemes*. We proved, in particular, that Light Affine Logic is not polytime sound if the power of second order quantification is fully exploited.

The encodings presented in (Girard 1998, Asperti & Roversi 2002, Baillot & Mogbil

2004) fit into the definition of uniform encodings, while some of those in (Neergard & Mairson 2002, Mairson & Terui 2003) do not. In the latter for instance different function calls are encoded by (cut-free) proofs having different conclusions (in the style of boolean circuits, with one circuit for each size of input). This comes with no surprise, since the authors were interested in studying the complexity of cut-elimination as a computational problem. However, those encodings are not acceptable if we want to study the expressive power of a logic as a programming language, as we do.

Our notion of uniformity is rather general. In particular, we do not impose any constraint on the shape of formulae for inputs and outputs. This is in contrast to similar results from the literature (Fortune, et al. 1983, Leivant & Marion 1993).

In this paper, we systematically investigate the expressive power of (various fragments of) Light Affine Logic, always considering uniform coding schemes. First of all, we prove that if we require some (fairly reasonable) conditions on the notion of encoding, the propositional implicative fragment of **LAL** is *not* complete for **FP** . Then we introduce simple constraints on second order quantification and fixpoints, proving the obtained fragments to be polytime sound and complete.

A preliminary version of this work was presented at the workshop on *Logics for Resources, Processes, and Programs*, 2004 (Dal Lago & Baillot 2004).

2. Uniform Encodings

In this short section we recall the notion of uniform encoding introduced in (Dal Lago 2003).

A uniform encoding $\mathcal{E}(f)$ of $f : (\{0,1\}^*)^n \to \{0,1\}^*$ 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 suitable correspondence Φ_i between elements of $\{0,1\}^*$ and cut-free proofs having conclusions $\vdash A_i$. These correspondences must be computable in logarithmic space;
- A correspondence Ψ between cut-free proofs having conclusion $\vdash B$ and elements of $\{0,1\}^*$. This correspondence must be logspace computable.

Clearly, the following diagram should commute

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

This definition is strongly inspired by the Curry-Howard correspondence.

For example, consider second-order implicative Intuitionistic Logic, that is to say System **F** by the Curry-Howard correspondence. Let W stand for a type for binary words, for instance:

$$W = \forall \alpha.(\alpha \to \alpha) \to (\alpha \to \alpha) \to (\alpha \to \alpha).$$

Then a proof of the sequent $W, \ldots, W \vdash W$ gives a uniform encoding of a function $f : (\{0,1\}^*)^n \to \{0,1\}^*$. But these are not the only uniform encodings in system F, since we can define other ones by considering other representations of binary lists. In particular the types for the various arguments and for the result need not be the same.

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

3. Syntax

Following existing literature, we will use the intuitionistic variant of Light Affine Logic **LAL**, called **ILAL**, as our reference system. *Formulae* are generated by the grammar

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

where α ranges over a set $\mathcal{L}$ of *atoms*. *Sequents* have the form $A_1, \ldots, A_n \vdash B$, where $A_1, \ldots, A_n, B$ are all formulae.

An **ILAL** *proof* is simply a tree whose nodes are labelled with sequents according to **ILAL** rules. A proof π having conclusion $\Gamma \vdash A$ is sometimes denoted as $\pi : \Gamma \vdash A$. We will call *size* of the proof, denoted as $|\pi|$, the number of rules in the proof.

We will study various fragments of **ILAL**. The core will be $\mathbf{ILAL}_{\multimap}$, which is reported in Figure 1.

Recall that in **ILAL** the contraction rule is restricted, as in Linear Logic, to !-marked formulae (rule C from Figure 1). The main difference with Linear Logic lies in the way !-marked formulae are introduced, which is more constrained: with rules $P_!^1$, $P_!^2$, ! modalities are introduced in the same time on lhs and rhs formulae, and the sequent must have at most one formula on the lhs. Alternatively one can introduce ! modalities on the l.h.s. using the $P_\S$ rule, but the remaining formulae must be marked with the new modality §.

The modality § can be seen as a kind of degenerate !, in the sense that it does not allow for contraction. The rule $P_\S$ is a weak analogue of the dereliction rule of Linear Logic. Recall that the following principles (resp. called *dereliction*, *digging*) are *not* provable in **LAL** , whereas they are in Linear Logic :

$$!A \vdash A, \quad !A \vdash !!A.$$

These restricted rules for the modalities are the key to the complexity bound on the cut-elimination procedure, which we will see with Theorem 1.

To this core $\mathbf{ILAL}_{\multimap}$, we can add other connectives, obtaining more powerful logics. For example, we can add tensor ($\otimes$, see Figure 2) and second order quantification ($\forall$, see Figure 3). Another interesting connective that can be added to the logic is the fixpoint operator (μ, see Figure 4). Note that from L_μ and R_μ, the rules L'_μ and R'_μ of Figure 5 can be derived; a derivation of L'_μ is given in Figure 6.

In this way, we can build several fragments of Intuitionistic Light Affine Logic, such as $\mathbf{ILAL}_{\multimap\otimes\forall}$ or $\mathbf{ILAL}_{\multimap\otimes\forall\mu}$. It is important to stress that **ILAL** admits cut-elimination. This stems from two points:

— all connectives admit cut-elimination steps. In particular the number of rules decreases with one step of cut-elimination on a $\mu\alpha.A$ formula.

Identity and Cut.

$$\frac{}{A \vdash A}\ I \qquad \frac{\Gamma \vdash A \quad \Delta, A \vdash B}{\Gamma, \Delta \vdash B}\ U$$

Structural Rules.

$$\frac{\Gamma \vdash A}{\Gamma, B \vdash A}\ W \qquad \frac{\Gamma, !A, !A \vdash B}{\Gamma, !A \vdash B}\ C$$

Implicative Logical Rules.

$$\frac{\Gamma \vdash A \quad \Delta, B \vdash C}{\Gamma, \Delta, A \multimap B \vdash C}\ L_{\multimap} \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B}\ R_{\multimap}$$

Exponential Logical Rules.

$$\frac{A \vdash B}{!A \vdash !B}\ P_!^1 \qquad \frac{\vdash A}{\vdash !A}\ P_!^2 \qquad \frac{\Gamma, \Delta \vdash A}{!\Gamma, \S\Delta \vdash \S A}\ P_\S$$

Fig. 1. Implicative Intuitionistic Light Affine Logic, $\mathbf{ILAL}_{\multimap}$

$$\frac{\Gamma, A, B \vdash C}{\Gamma, A \otimes B \vdash C}\ L_{\otimes} \qquad \frac{\Gamma \vdash A \quad \Delta \vdash B}{\Gamma, \Delta \vdash A \otimes B}\ R_{\otimes}$$

Fig. 2. Tensor Logical Rules

— a particular strategy allows to eliminate all cuts: see (Girard 1998, Asperti & Roversi 2002).

Note that fixpoints added to Intuitionistic Logic or Linear Logic break the cut-elimination property, whereas this is not the case with **LAL**. This is because in this system, like in Elementary, Light or Soft Linear Logic (Lafont 2004, Baillot & Mogbil 2004) the cut-elimination argument does not depend on the size of the cut formulae but on the size and depth of proofs (the latter will be defined soon). Actually this remark was one motivation originally stressed by Girard for the definition of Light linear logic (see (Girard 1998)).

ILAL can also be thought of as a type assignment system for the following term calculus:

$$M, N ::= x \mid \lambda x.M \mid MM \mid (M, M) \mid \ \mathbf{let}\ \ M\ \ \mathbf{be}\ \ (x, x)\ \ \mathbf{in}\ \ M$$

In this setting, rules for !, §, $\forall$ and μ do not influence the underlying term. When this does not cause ambiguity, we will denote an **ILAL** sequent calculus proof as the term it types. If $A_1, \ldots, A_n \vdash B$ types term M, then the free variables appearing in M will be named $x_1, \ldots, x_n$ resp. of type $A_1, \ldots, A_n$. Most results about **ILAL** are traditionally given on proof-nets, which are handy in studying the dynamics of proofs (see (Asperti &

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

Fig. 3. Second Order Rules

$$\frac{\Gamma, A[\mu\alpha.A/\alpha] \vdash B}{\Gamma, \mu\alpha.A \vdash B} L_\mu \qquad \frac{\Gamma \vdash A[\mu\alpha.A/\alpha]}{\Gamma \vdash \mu\alpha.A} R_\mu$$

Fig. 4. Fixpoint Rules

$$\frac{\Gamma, \mu\alpha.A \vdash B}{\Gamma, A[\mu\alpha.A/\alpha] \vdash B} L'_\mu \qquad \frac{\Gamma \vdash \mu\alpha.A}{\Gamma \vdash A[\mu\alpha.A/\alpha]} R'_\mu$$

Fig. 5. Derivable unfolding rules

Roversi 2002)). Nevertheless, we chose to present **ILAL** as a sequent calculus, in order to keep a concise presentation. Many sequent calculus proofs differing only in the order of application of rules could correspond to the same proof-net. Anyway here we are just using sequent calculus as a convenient notation and one can without any problem convert the proofs into proof-nets if one wants to examine the normalization issues.

Definition 1. Given an **ILAL** proof π, the *box-depth* $\partial(\pi)$ of π is the maximum integer n such that there is a path in π from a leaf to the root which crosses n instances of rules $P^1_!$, $P^2_!$ or $P_\S$.

It is easy to check that this definition of box-depth is equivalent to the one traditionally given on **ILAL** proof-nets (Asperti & Roversi 2002), namely the maximal nesting level of boxes in the proof-net.

An **ILAL** fragment is said to be *reflective* if there is a function f (from sequents to natural numbers) such that $\partial(\pi) \leq f(\Gamma \vdash A)$ whenever $\pi : \Gamma \vdash A$ is a cut free proof. This means that given a formula, one can bound the box-depth of cut-free proofs with this conclusion.

Now, we recall the main result of **ILAL**

Theorem 1 (ILAL normalization complexity, (Asperti & Roversi 2002)). The normalization of an **ILAL** proof π can be done in time $O(|\pi|^k)$, where the exponent k only depends on $\partial(\pi)$.

As a direct consequence we have:

Proposition 1. Any reflective fragment of **ILAL** is polytime sound.

$$\cfrac{\cfrac{\cfrac{}{A[\mu\alpha.A/\alpha] \vdash A[\mu\alpha.A/\alpha]} I}{A[\mu\alpha.A/\alpha] \vdash \mu\alpha.A} R_\mu \qquad \Gamma, \mu\alpha.A \vdash B}{\Gamma, A[\mu\alpha.A/\alpha] \vdash B} U$$

Fig. 6. A derivation for rule L'_μ

4. The Full Case

Let us start with the fragment $\mathbf{ILAL}_{\multimap\otimes\forall}$. We know from (Asperti & Roversi 2002) that this fragment is polytime complete. In spite of that, it is not reflective due to the rule $L_\forall$, which can be used to build proofs with fixed conclusion but arbitrary box-depth. Indeed, $\mathbf{ILAL}_{\multimap\otimes\forall}$ is polytime unsound if the full power of second-order quantification is exploited, as we are going to show.

Binary lists can be represented in $\mathbf{ILAL}_{\multimap\otimes\forall}$ by cut-free proofs with conclusion

$$\mathit{SOBinaryLists} = \forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha)$$

The cut-free proof with conclusion $\vdash \mathit{SOBinaryLists}$ corresponding to string $s \in \{0,1\}^*$ will be denoted by $\ulcorner s \urcorner$.

The encodings of functions considered in (Asperti & Roversi 2002) used sequents of the form $\mathit{SOBinaryLists} \vdash \S^k \mathit{SOBinaryLists}$, with k an integer. These are particular uniform encodings, but in the present paper we will also consider other ones.

Lemma 1. For every $n \in \mathbb{N}$, there is a cut-free $\mathbf{ILAL}_{\multimap\otimes\forall}$ proof

$$\rho_n : \mathit{SOBinaryLists} \vdash \S^{n+1}\mathit{SOBinaryLists}$$

such that ρ_n is a uniform encoding of the function $p^n : \{0,1\}^* \to \{0,1\}^*$, where $p^n(s) = 1^{|s|^n}$ for every $s \in \{0,1\}^*$.

Proof. In this proof, we will denote *SOBinaryLists* simply by BL. Since the case $n = 0$ is trivial, we can assume $n \geq 1$. For every $m \geq 1$, we can inductively define Γ_m as follows. First of all, $\Gamma_1 = BL$; moreover, $\Gamma_m = \Gamma_{m-1}, \S^{m-2}!BL$ for every $m > 1$. Similarly, A_1 denotes BL, while for every $m > 1$ $A_m = A_{m-1} \otimes \S^{m-2}!BL$. We now prove, by induction on m, that there is a proof $\sigma_m : \Gamma_m \vdash \S^m BL$ encoding function $f_m : (\{0,1\}^*)^m \to \{0,1\}^*$ where $f_m(s_1, \ldots, s_m) = 1^{|s_1|\cdots|s_m|}$ for every $s_1, \ldots, s_m \in \{0,1\}^*$. If $m = 1$, then σ_m is

$$\dfrac{\varphi : \vdash !(BL \multimap BL) \quad \varphi : \vdash !(BL \multimap BL) \quad \dfrac{\dfrac{\psi : \vdash BL \quad \overline{BL \vdash BL}}{BL \multimap BL \vdash BL}}{\S(BL \multimap BL) \vdash \S BL}}{BL \vdash \S BL}$$

where

— φ is the proof

$$\dfrac{\dfrac{\xi : BL \vdash BL}{\vdash BL \multimap BL}}{\vdash !(BL \multimap BL)}$$

— ξ is a cut-free proof encoding function $g : \{0,1\}^* \to \{0,1\}^*$ with $g(s) = 1s$.

— ψ encodes the string ε.

If $m > 1$, then σ_m is

$$\dfrac{\theta \quad \theta \quad \dfrac{\dfrac{\psi : \vdash BL \quad \sigma_{m-1} : \Gamma_{m-1} \vdash \S^{m-1} BL}{BL \multimap BL, !BL, \ldots, \S^{m-3}!BL \vdash \S^{m-1} BL}}{\S(BL \multimap BL), \S!BL, \ldots, \S^{m-2}!BL \vdash \S^m BL}}{\Gamma_m \vdash \S^m BL}$$

where:

— θ is the proof

$$\dfrac{\dfrac{\omega : BL, BL \vdash BL}{BL \vdash BL \multimap BL}}{!BL \vdash !(BL \multimap BL)}$$

— ω encodes the function $h : (\{0,1\}^*)^2 \to \{0,1\}^*$ such that $h(s,t) = st$ for every $s, t \in \{0,1\}^*$.

We are now able to build ρ_n:

$$\dfrac{\dfrac{\dfrac{\eta : A_n \vdash A_n}{\vdash A_n \multimap A_n}}{\vdash !(A_n \multimap A_n)} \quad \dfrac{\dfrac{\eta : A_n \vdash A_n}{\vdash A_n \multimap A_n}}{\vdash !(A_n \multimap A_n)} \quad \dfrac{\dfrac{\tau : \vdash A_n \quad \dfrac{\sigma_n : \Gamma_n \vdash \S^n BL}{A_n \vdash \S^n BL}}{A_n \multimap A_n \vdash \S^n BL}}{\S(A_n \multimap A_n) \vdash \S^{n+1} BL}}{BL \vdash \S^{n+1} BL}$$

Here, η and τ are generalizations of φ and ψ, respectively. Notice that ρ_n, as we have defined it, is cut-free and can be built in logarithmic space (on n). □

We have just proved that, for every $n \in \mathbb{N}$, ρ_n uniformly encodes the function p^n. Now, if all the different ρ_n had the same type, it would be easy to build a proof δ such that $\delta(\rho_n)$ reduces to $\rho_n(\ulcorner 11 \urcorner)$, then normalizing to a proof similar to $\lceil m \rceil$ where $m = 1^{2^n}$. Actually, every ρ_n has a conclusion which is different from the conclusion of any other ρ_m. This problem, however, can be circumvented by building another sequence of proofs $\{\chi_n\}_{n \in \mathbb{N}}$. Every such χ_n behaves similarly to ρ_n, but all the proofs in the sequence have the same conclusion. In this way, we can find a uniform encoding inside $\mathbf{ILAL}_{\multimap\otimes\forall}$ of an intrinsically exponential function over the algebra $\{0,1\}^*$:

Proposition 2. There is a function $f : \{0,1\}^* \to \{0,1\}^*$ which can be uniformly represented in $\mathbf{ILAL}_{\multimap\otimes\forall}$ and is not computable in polynomial time.

Proof. $f : \{0,1\}^* \to \{0,1\}^*$ is the function defined by letting

$$f(s) = 1^{2^{|s|}}$$

whenever $s \in \{0,1\}^*$. Clearly, f cannot belong to **FP**, because the length of the output is exponential in the length of the input. For every $n \in \mathbb{N}$, the proof χ_n is defined as follows:

$$\dfrac{\dfrac{\dfrac{\rho_n : BL \vdash \S^{n+1}BL \qquad \dfrac{}{\alpha \vdash \alpha}}{BL, \S^{n+1}BL \multimap \alpha \vdash \alpha}}{BL, \forall\beta.(\beta \multimap \alpha) \vdash \alpha}}{\vdash BL \multimap (\forall\beta.(\beta \multimap \alpha)) \multimap \alpha}$$

where ρ_n is as in Lemma 1. For every $m \in \mathbb{N}$, the proof τ_m is defined as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\lceil 1^m \rceil : \vdash BL}{\vdash \S^{\lceil lgm \rceil+1}BL} \qquad \dfrac{}{\alpha \vdash \alpha}}{\S^{\lceil lgm \rceil+1}BL \multimap \alpha \vdash \alpha}}{\forall\beta.(\beta \multimap \alpha) \vdash \alpha}}{\vdash (\forall\beta.(\beta \multimap \alpha)) \multimap \alpha}$$

Let now π be the proof:

$$\dfrac{\lceil 11 \rceil : \vdash BL \qquad \dfrac{}{(\forall\beta.(\beta \multimap \alpha)) \multimap \alpha \vdash (\forall\beta.(\beta \multimap \alpha)) \multimap \alpha}}{BL \multimap (\forall\beta.(\beta \multimap \alpha)) \multimap \alpha \vdash (\forall\beta.(\beta \multimap \alpha)) \multimap \alpha}$$

Now, consider the following diagram

$$\begin{array}{ccc} \{0,1\}^* & \xrightarrow{\quad f \quad} & \{0,1\}^* \\ \downarrow{\scriptstyle \Phi} & & \uparrow{\scriptstyle \Psi} \\ BL \multimap (\forall\beta.(\beta \multimap \alpha)) \multimap \alpha & \xrightarrow{\pi} & (\forall\beta.(\beta \multimap \alpha)) \multimap \alpha \end{array}$$

Φ is the function defined by letting $\Phi(s) = \chi_{|s|}$ for every $s \in \{0,1\}^*$; Ψ is defined by letting $\Psi(\tau_m) = 1^m$ and $\Psi(\rho) = \varepsilon$ whenever ρ is not in the form τ_m. Both Φ and Ψ are obviously logspace computable. The above diagram can be easily checked to be commuting. □

The question we consider now is: can we restrict $\mathbf{ILAL}_{\multimap\otimes\forall}$ to reach a polytime sound and complete system? This is the main subject of this paper. The solution which has been pursued in (Dal Lago 2003) consisted in restricting the class of permitted *encodings*, forbidding the use of $L_\forall$ in proofs representing inputs and outputs. Here, we use a different approach: we try to restrict the *logic*, without modifying coding schemes.

5. $\mathbf{ILAL}_{\multimap}$ and Polynomial Time

How much can we restrict $\mathbf{ILAL}_{\multimap\otimes\forall}$ while keeping polytime completeness? Let us start by considering the smallest fragment, $\mathbf{ILAL}_{\multimap}$. In this section, we will prove that, under reasonable assumptions on the encodings, $\mathbf{ILAL}_{\multimap}$ is not polytime complete.

$\mathbf{ILAL}_{\multimap}$ can be seen as a type assignment system for pure lambda-calculus. If a pure lambda-term M can be typed by an $\mathbf{ILAL}_{\multimap}$ proof, then it is simply-typable. Moreover, if M can be typed by a cut-free $\mathbf{ILAL}_{\multimap}$ proof, then it is necessarily a β-normal form, but can possibly contain η-redexes.

An encoding of f into $\mathbf{ILAL}_{\multimap}$ is said to be *extensional* if all the correspondences $\Phi_1, \ldots, \Phi_n, \Psi$ map distinct elements of $\{0,1\}^*$ to $\mathbf{ILAL}_{\multimap}$ proofs whose underlying lambda-terms are distinct and η-normal.

Now, we can recall a theorem by Statman:

Theorem 2 (Finite Completeness Theorem, (Statman 1982)). Let M be a closed term having simple type A. There exists a finite model $\mathcal{M}(M)$ such that $\mathcal{M}(M) \models M = N$ if and only if $M =_{\beta\eta} N$.

The function *equality* : $\{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^*$ is defined by

$$equality(s,t) = \begin{cases} 1 & if \;\; s = t \\ 0 & otherwise \end{cases}$$

Now we get:

Proposition 3. *equality* is not extensionally encodable into $\mathbf{ILAL}_{\multimap}$.

Proof. Basically we use the fact that an extensional encoding of a function f in $\mathbf{ILAL}_{\multimap}$ induces a corresponding encoding of f into the simply typed lambda-calculus. Suppose, by way of contradiction, that an extensional encoding $\mathcal{E}$ of *equality* into $\mathbf{ILAL}_{\multimap}$ exists. Then, there are simply typable closed terms

$$\begin{array}{ccl} M & : & A \rightarrow B \rightarrow C \\ T & : & C \\ F & : & C \\ T & \neq_{\beta\eta} & F \end{array}$$

where T and F respectively encode the values 0 and 1; and for every $s \in \{0,1\}^*$, there are simply typable closed terms encoding s respectively in types A and B:

$$\begin{array}{ccl} P(s) & : & A \\ Q(s) & : & B \end{array}$$

such that, for every $s, t \in \{0,1\}^*$ with $s \neq t$,

$$\begin{array}{ccl} MP(s)Q(s) & \rightarrow^*_\beta & T \\ MP(t)Q(s) & \rightarrow^*_\beta & F \end{array}$$

From the extensionality hypothesis, we know that both T and F are η-normal. Now, call $\mathcal{M}$ the model $\mathcal{M}(T)$ obtained by Theorem 2 applied to the term T. It is a finite model, so there must be $s, t \in \{0,1\}^*$ with $s \neq t$ such that $\mathcal{M}$ interprets both $P(s)$ and $P(t)$ by the same semantical value. Obviously, $MP(s)Q(s) =_{\beta\eta} T$ and $MP(t)Q(s) =_{\beta\eta} F$; so, by soundness we have:

$$\begin{array}{ccl} \mathcal{M} & \models & MP(s)Q(s) = T \\ \mathcal{M} & \models & MP(t)Q(s) = F. \end{array}$$

But $\mathcal{M}$ must interpret $MP(s)Q(s)$ and $MP(t)Q(s)$ in the same way, so it follows that:

$$\mathcal{M} \models T = F.$$

By Theorem 2 this implies $T =_{\beta\eta} F$, hence a contradiction. □

6. Polynomials and $\mathbf{ILAL}_{\multimap\otimes}$

Because of the previous negative result (section 5), from now on we will consider fragments containing $\mathbf{ILAL}_{\multimap\otimes}$. A necessary condition for polytime completeness is the ability to represent polynomials. In this section we will show that $\mathbf{ILAL}_{\multimap\otimes}$ is sufficient for representing polynomials using a Church-style encoding for numerals.

In fact, throughout the paper when speaking of polynomials we will mean polynomials with positive integer coefficients.

For every $\mathbf{ILAL}_{\multimap\otimes}$ formula A, $PInt(A)$ will be the type $!(A \multimap A) \multimap \S(A \multimap A)$. The class of *integer formulae* is the smallest class satisfying the following conditions:

- For every formula A, $PInt(A)$ is an integer formula;
- If B is an integer formula, then $!B$ and $\S B$ are integer formulae.

In other words, integer formulae are given by the following grammar:

$$B ::= PInt(A) \mid !B \mid \S B$$

where A ranges over $\mathbf{ILAL}_{\multimap\otimes}$ formulae.

Lemma 2. For every $\mathbf{ILAL}_{\multimap\otimes}$ formula A, there are proofs

$$\begin{array}{rcl} \pi_{+1} : PInt(A) & \vdash & PInt(A) \\ \pi_{+} : PInt(A), PInt(A) & \vdash & PInt(A) \\ \pi_{\times} : PInt(PInt(A)), !PInt(A) & \vdash & \S PInt(A) \end{array}$$

representing successor, addition and multiplication respectively.

Proof. We just give the underlying terms for π_{+1}, π_{+} and $\pi_{\times}$, which are

$$\begin{array}{rcl} M_{+1} & = & \lambda x.\lambda y.x((x_1 x)y) \\ M_{+} & = & \lambda x.\lambda y.(x_1 x)((x_2 x)y) \\ M_{\times} & = & x_1(\lambda x_1.M_{+})(\lambda x.\lambda y.y) \end{array}$$

respectively (remember the convention fixed in section 3 about the naming of free variables in typed lambda terms). □

The class of basic arithmetical functions is the smallest class satisfying the following constraints:

- The identity $id : \mathbb{N}^1 \to \mathbb{N}$ on natural numbers is a basic arithmetical function;
- For every $n \in \mathbb{N}$, the constant $n : \mathbb{N}^0 \to \mathbb{N}$ is a basic arithmetical function;
- If $f : \mathbb{N}^n \to \mathbb{N}$ and $g : \mathbb{N}^m \to \mathbb{N}$ are basic arithmetical functions, then $f + g : \mathbb{N}^{n+m} \to \mathbb{N}$, defined by

  $$(f+g)(x_1, \ldots, x_n, y_1, \ldots, y_m) = f(x_1, \ldots, x_n) + g(y_1, \ldots, y_m)$$

  is a basic arithmetical function;

- If $f : \mathbb{N}^n \to \mathbb{N}$ is a basic arithmetical function, then $\widetilde{f} : \mathbb{N}^{n+1} \to \mathbb{N}$ defined by

$$\widetilde{f}(x_1, \ldots, x_n, x) = x \cdot f(x_1, \ldots, x_n)$$

is a basic arithmetical function.

Lemma 3. For every formula A and for every basic arithmetical function $f : \mathbb{N}^n \to \mathbb{N}$, there is an $\mathbf{ILAL}_{\multimap\otimes}$ proof $\pi_f : A_1, \ldots, A_n \vdash \S^k PInt(A)$ representing f, where $A_1, \ldots, A_n$ all are integer formulae.

Proof. We proceed by induction on the definition of basic arithmetical functions f. The base cases are straightforward, so we can concentrate on the two inductive cases. If $f = g + h$, where $g : \mathbb{N}^n \to \mathbb{N}$ and $h : \mathbb{N}^m \to \mathbb{N}$, then by induction hypothesis, there must be proofs

$$\begin{array}{lcl} \pi_g & : & A_1, \ldots, A_n \vdash \S^k PInt(A) \\ \pi_h & : & B_1, \ldots, B_m \vdash \S^l PInt(A) \end{array}$$

representing g and h, respectively, where all the A_i and B_j are integer formulae. π_f will be

$$\frac{\dfrac{\pi_g : A_1, \ldots, A_n \vdash \S^k PInt(A)}{\S^l A_1, \ldots, \S^l A_n \vdash \S^{l+k} PInt(A)} \quad \rho}{\S^l A_1, \ldots, \S^l A_n, \S^k B_1, \ldots, \S^k B_m \vdash \S^{l+k} PInt(A)}$$

where ρ is

$$\frac{\dfrac{\pi_h : B_1, \ldots, B_m \vdash \S^l PInt(A)}{\S^k B_1, \ldots, \S^k B_m \vdash \S^{k+l} PInt(A)} \qquad \dfrac{\pi_+ : PInt(A), PInt(A) \vdash PInt(A)}{\S^{l+k} PInt(A), \S^{l+k} PInt(A) \vdash \S^{l+k} PInt(A)}}{\S^k B_1, \ldots, \S^k B_m, \S^{l+k} PInt(A) \vdash \S^{l+k} PInt(A)}$$

If $f = \widetilde{g}$, where $g : \mathbb{N}^n \to \mathbb{N}$, then by induction hypothesis there must be a proof

$$\pi_g : A_1, \ldots, A_n \vdash \S^k PInt(PInt(A))$$

representing g where all the A_i and B_j are integer formulae. The proof π_f will be

$$\frac{\pi_g : A_1, \ldots, A_n \vdash \S^k PInt(PInt(A)) \qquad \dfrac{\pi_\times : PInt(PInt(A)), !PInt(A) \vdash \S PInt(A)}{\S^k PInt(PInt(A)), \S^k !PInt(A) \vdash \S^{k+1} PInt(A)}}{A_1, \ldots, A_n, \S^k !PInt(A) \vdash \S^{k+1} PInt(A)}$$

This concludes the proof. □

Proposition 4. For every formula A and for every polynomial $f : \mathbb{N} \to \mathbb{N}$, there is an $\mathbf{ILAL}_{\multimap\otimes}$ proof $\pi_f : PInt(B) \vdash \S^k PInt(A)$ representing f.

Proof. Any polynomial $f : \mathbb{N} \to \mathbb{N}$ with integer coefficients can be written as

$$f(x) = \sum_{i=1}^{n} \prod_{j=1}^{m(i)} a_i^j$$

where a_i^j is either an integer constant or the indeterminate x. We can arrange all the constants in a sequence $a^{ca(1)}_{cp(1)}, \dots, a^{ca(p)}_{cp(p)}$ and all the x occurrences in another sequence $a^{ia(1)}_{ip(1)}, \dots, a^{ia(q)}_{ip(q)}$. Let m be $\sum_{i=1}^{n} m(i)$. The function $g : \mathbb{N}^m \to \mathbb{N}$ defined by

$$g(x^1_1, \dots, x^{m(1)}_1, \dots, x^1_n, \dots, x^{m(n)}_n) = \sum_{i=1}^{n} \prod_{j=1}^{m(i)} x^j_i$$

is a basic arithmetical function. So, by lemma 3, there is an $\mathbf{ILAL}_{\multimap\otimes}$ proof

$$\pi_g : A^1_1, \dots, A^{m(1)}_1, \dots, A^1_n, \dots, A^{m(n)}_n \vdash \S^k PInt(A)$$

encoding g. Now, the function f is obtained from g by:
(i) substituting to each $x^{ca(i)}_{cp(i)}$ $(1 \le i \le p)$ the constant $a^{ca(i)}_{cp(i)}$,
(ii) identifying all $x^{ia(i)}_{ip(i)}$ $(1 \le i \le q)$ into the same variable x.
An idea to define from π_g a proof π_f representing f is thus:

- for (i) to perform p cuts of π_g with proofs representing the integers $a^{ca(1)}_{cp(1)}, \dots, a^{ca(p)}_{cp(p)}$.
- for (ii) to cut the proof with another proof ρ which, intuitively, transforms an integer k into q copies of k.

The proof ρ can in fact be defined without using contraction, simply by applying the iteration scheme associated to an integer formula: the term underlying $\rho :\ PInt(A^{ia(1)}_{ip(1)} \otimes \dots \otimes A^{ia(q)}_{ip(q)}) \vdash \S(A^{ia(1)}_{ip(1)} \otimes \dots \otimes A^{ia(q)}_{ip(q)})$ is

$$x_1(M_{+1}, \dots, M_{+1})(M_0, \dots, M_0)$$

where

$$\begin{aligned} M_{+1} &= \lambda x.\lambda y.\lambda z.y((xy)z) \\ M_0 &= \lambda x.\lambda y.y. \end{aligned}$$

The proof π_f can then be defined as:

$$\dfrac{\rho \qquad \dfrac{\dfrac{\omega(a^{ca(p)}_{cp(p)}) : \vdash A^{ca(p)}_{cp(p)} \qquad \dfrac{\omega(a^{ca(1)}_{cp(1)}) : \vdash A^{ca(1)}_{cp(1)} \quad \pi_g : A^1_1, \dots, A^{m(n)}_n \vdash \S^k PInt(A)}{A^{ca(p)}_{cp(p)}, A^{ia(1)}_{ip(1)}, \dots, A^{ia(q)}_{ip(q)} \vdash \S^k PInt(A)}}{A^{ia(1)}_{ip(1)}, \dots, A^{ia(q)}_{ip(q)} \vdash \S^k PInt(A)}}{\S(A^{ia(1)}_{ip(1)} \otimes \dots \otimes A^{ia(q)}_{ip(q)}) \vdash \S^{k+1}(PInt(A))}}{PInt(A^{ia(1)}_{ip(1)} \otimes \dots \otimes A^{ia(q)}_{ip(q)}) \vdash \S^{k+1} PInt(A)}$$

For every i, the term underlying $\omega(a^{ca(i)}_{cp(i)}) :\vdash A^{ca(i)}_{cp(i)}$ is the $a^{ca(i)}_{cp(i)}$-th Church numeral. This concludes the proof. □

7. Linear Quantifiers and Fixpoints

We saw that $\mathbf{ILAL}_{\multimap}$ is not polytime complete while $\mathbf{ILAL}_{\multimap\otimes\forall\mu}$ is not polytime sound. We thus would like to consider an intermediate system enjoying both properties. An idea for that is to try to limit the power of quantifiers and fixpoints.

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

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

Fig. 7. Linear Quantifiers and Fixpoints

Observe that the counter-example of Proposition 2 in Section 4 used in a crucial way quantification with instantiation on formulae with modality §. Indeed note that rules $L_\forall$, L_μ, R_μ (see Figures 3, 4) used with formulae A containing modalities are responsible for **ILAL** not being reflective because, reading the proof bottom-up, they introduce new occurrences of modalities which can allow for new $P_!$ or $P_\S$ rules. A natural remedy is thus to restrict the use of $\forall$ and μ.

We say an $\mathbf{ILAL}_{\multimap\otimes\forall\mu}$ formula A is *linear* if it does not contain any instance of ! or §. We denote by $\mathscr{L}$ the class of **ILAL** linear formulae.

We want to replace rules $L_\forall$, L_μ and R_μ by rules that can be applied only in the case where A is a linear formula. For that we introduce two new connectives $\overline{\forall}$ and $\overline{\mu}$, defined by the rules in Figure 7.

Let us denote by $\mathbf{ILAL}_{\multimap\otimes\overline{\forall}\overline{\mu}}$ this new fragment. It can be verified that:

Proposition 5. The fragment $\mathbf{ILAL}_{\multimap\otimes\overline{\forall}\overline{\mu}}$ is stable by cut-elimination.

Observe that when read bottom-up the rules $L_{\overline{\forall}}$, $L_{\overline{\mu}}$, $R_{\overline{\mu}}$ do not introduce new occurrences of ! or §. It follows that the number of rules $P_!^1$, $P_!^2$ and $P_\S$ in a cut-free $\mathbf{ILAL}_{\multimap\otimes\overline{\forall}\overline{\mu}}$ proof is bounded by the number of occurrences of ! and § in its conclusion; therefore we have:

Fact 1. The fragment $\mathbf{ILAL}_{\multimap\otimes\overline{\forall}\overline{\mu}}$ is reflective.

By Theorem 1 we thus have:

Proposition 6. The system $\mathbf{ILAL}_{\multimap\otimes\overline{\forall}\overline{\mu}}$ is polytime sound.

In this section, we show that this fragment is also polytime complete.

A Turing Machine $\mathcal{M}$ is described by:

- A finite alphabet $\Sigma = \{a_1, \ldots, a_n\}$, where a_1 is considered as the blank symbol;
- A set $Q = \{q_1, \ldots, q_m\}$ of states, where q_1 is considered as the starting state;
- A transition function $\delta : Q \times \Sigma \to Q \times \Sigma \times \{\leftarrow, \rightarrow, \downarrow\}$.

A configuration for $\mathcal{M}$ is a quadruple in $\Sigma^* \times \Sigma \times \Sigma^* \times Q$. For example, if $\delta(q_i, a_j) = (q_l, a_k, \leftarrow)$, then $\mathcal{M}$ evolves from (sa_p, a_j, t, q_i) to $(s, a_p, a_k t, q_l)$ (and from $(\varepsilon, a_j, t, q_i)$ to $(\varepsilon, a_1, a_k t, q_l)$).

Linear quantifiers and fixpoints are enough to code a transition function. First of all, let us define the following formulae (parameterized on k):

$$\begin{aligned} PString_k(\alpha) &= \overline{\mu}\beta.\,\underbrace{(\beta \multimap \alpha) \multimap \ldots \multimap (\beta \multimap \alpha)}_{k\ \ times} \multimap \alpha \multimap \alpha \\ PChar_k(\alpha) &= \underbrace{\alpha \multimap \ldots \multimap \alpha}_{k\ \ times} \multimap \alpha \\ PState_k(\alpha) &= \underbrace{\alpha \multimap \ldots \multimap \alpha}_{k\ \ times} \multimap \alpha \\ SOString_k &= \overline{\forall}\alpha.PString_k(\alpha) \\ SOChar_k &= \overline{\forall}\alpha.PChar_k(\alpha) \\ SOState_k &= \overline{\forall}\alpha.PState_k(\alpha) \end{aligned}$$

For every $i \in \{1, \ldots, n\}$, the symbol a_i will be represented by

$$projection_i^n = \lambda x_1.\ldots\lambda x_n.x_i$$

which, seen as proof, has conclusion $SOChar_n$. Analogously, state q_i will be represented by $projection_i^m$. Counterparts of strings in Σ^* are defined by induction:

- The empty string ε is represented by $projection_{n+1}^{n+1}$.
- If $t \in \Sigma^*$ is represented by M, then, for every $i \in \{1, \ldots, n\}$, the string $a_i t$ is represented by

$$\lambda x_1.\ldots\lambda x_n \lambda x_{n+1}.x_i M$$

This encoding can be seen as a variant for lists of the Scott numeral representation (Wadsworth 1980).

The term $append_i^n : SOString_n \multimap SOString_n$ encodes the juxtaposition of a_i to the input string:

$$append_i^n = \lambda x.\lambda x_1.\ldots\lambda x_{n+1}.x_i x$$

Configurations become cut free proofs for

$$SOConfig_n^m = SOString_n \otimes SOChar_n \otimes SOString_n \otimes SOState_m$$

Lemma 4. For any transition function of a Turing machine there exists an $\mathbf{ILAL}_{\multimap\otimes\overline{\forall}\overline{\mu}}$ proof of $\vdash SOConfig_n^m \multimap SOConfig_n^m$ representing it.

Proof. Let us construct such a proof $step_{\mathcal{M}}$. The λ-term corresponding to $step_{\mathcal{M}}$ is

$$\lambda x.\mathbf{let}\ \ x\ \ \mathbf{be}\ \ (s, a, t, q)\ \ \mathbf{in}\ \ (q\ M_1\ \ldots\ M_m)(s, a, t) \tag{1}$$

where, for every i, term M_i has type $SOString_n \otimes SOChar_n \otimes SOString_n \multimap SOConfig_n^m$. Observe that in this proof the $\overline{\forall}$ quantifier of the $SOState_m$ type of q is instantiated with the type of M_i, $SOString_n \otimes SOChar_n \otimes SOString_n \multimap SOConfig_n^m$.

Now, M_i is itself given by:

$$\lambda x.\mathbf{let}\ \ x\ \ \mathbf{be}\ \ (s, a, t)\ \ \mathbf{in}\ \ (a\ N_i^1\ \ldots\ N_i^n)(s, t). \tag{2}$$

Each $N_i^j : SOString_n \otimes SOString_n \multimap SOConfig_n^m$ encodes the value of $\delta(q_i, a_j)$. Note that in the proof for M_i the $\overline{\forall}$ quantifier of the type $SOChar_n$ of a is instantiated with the type of the N_i^j.

$$\frac{\Gamma \vdash A}{\Gamma, 1 \vdash A}\ L_1 \qquad \frac{}{\vdash 1}\ R_1$$

Fig. 8. Rules for constant 1

$$\frac{\Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma, A \oplus B \vdash C}\ L_\oplus \qquad \frac{\Gamma \vdash A}{\Gamma \vdash A \oplus B}\ R^1_\oplus \qquad \frac{\Gamma \vdash A}{\Gamma \vdash B \oplus A}\ R^2_\oplus$$

Fig. 9. Rules for $\oplus$

Let us finally describe how the N_i^j are defined. If, as in the example above, $\delta(q_i, a_j) = (q_l, a_k, \leftarrow)$, N_i^j will be the term

$$\lambda x.\mathbf{let}\ \ x\ \ \mathbf{be}\ \ (s,t)\ \ \mathbf{in}\ \ (s\, P^1\ \ldots\ P^n\ R)t \tag{3}$$

where, for every r, term $P^r : \mathit{SOString}_n \multimap \mathit{SOString}_n \multimap \mathit{SOConfig}_n^m$ is

$$\lambda s.\lambda t.(s, \mathit{projection}_r^n, \mathit{append}_k^n\ t, \mathit{projection}_l^m)$$

and $R : \mathit{SOString}_n \multimap \mathit{SOConfig}_n^m$ is

$$\lambda t.(\mathit{projection}_{n+1}^{n+1}, \mathit{projection}_1^n, t, \mathit{projection}_l^m).$$

In the proof for N_i^j the quantifier $\overline{\forall}$ of the type $\mathit{SOString}_n$ of s is instantiated with the type of R. □

We can finally prove the following result:

Theorem 3. $f : \{0,1\}^* \to \{0,1\}^*$ is computable by a polynomial time Turing Machine iff f is uniformly encodable into $\mathbf{ILAL}_{\multimap\otimes\overline{\forall}\mu}$. Thus $\mathbf{ILAL}_{\multimap\otimes\overline{\forall}\mu}$ is polytime sound and complete.

Proof. Let $\mathcal{M}$ be a Turing Machine running in time $f : \mathbb{N} \to \mathbb{N}$. If f is a polynomial, then Proposition 4 gives us a proof $\pi_f : \mathit{PInt}(B) \vdash \S^k \mathit{PInt}(\mathit{SOConfig}_n^m)$ encoding f. Observe that using Lemma 4 the term underlying $\mathit{step}_{\mathcal{M}}$ can be typed by $\S^k!(\mathit{SOConfig}_n^m \multimap \mathit{SOConfig}_n^m)$. Putting these two ingredients together, we obtain a proof $\pi_{\mathcal{M}} : \mathit{PInt}(B) \otimes \S^{k+1}\mathit{SOConfig}_n^m \vdash \S^{k+1}\mathit{SOConfig}_n^m$, which is a uniform encoding for the function computed by $\mathcal{M}$. □

8. Additive Connective $\oplus$ and Fixpoints

Observe that we have used linear quantification in the previous section essentially to deal with case distinction. This can in fact also be done using another feature of linear logic: additive connectives, & and $\oplus$. In the intuitionistic setting we are considering, it is even enough for our purposes to consider the connective $\oplus$ only. We give the rules for $\oplus$ on Figure 9. We will also use the constant for the connective $\otimes$, denoted 1: the corresponding rules are given on Figure 8.

The term language is extended accordingly, with the following new productions:

$$M ::= 1 \mid \mathbf{inl}(t) \mid \mathbf{inr}(t) \mid \mathbf{case}\ M\ \ \mathbf{of}\ \ \mathbf{inl}(x) \Rightarrow M\, , \mathbf{inr}(x) \Rightarrow M$$

The fragment we are dealing with now is thus **ILAL** with $\multimap$, $\otimes$, 1, $\oplus$, $\overline{\mu}$, but no quantification. We show that the step function of a Turing Machine can be encoded in this fragment. Using the previous encoding of polynomials we will then be able to deduce that polytime Turing Machines can be simulated in this fragment. We use the connective $\oplus$ to define enumeration types and case distinction on those types (in particular conditional test with boolean type).

Let us define $ABool_k = 1 \oplus \ldots \oplus 1$ (with k components) for $k \geq 1$. This formula represents the k-ary boolean type and we denote its k normal proofs by $\underline{1}, \ldots, \underline{k}$. We use as short-hand term notation for the case distinction defined on $ABool_k$ using the previous rules:

$$\frac{\Gamma \vdash M_1 : B \quad \ldots \quad \Gamma \vdash M_k : B}{\Gamma, x : ABool_k \vdash \mathbf{case}\ \ x\ \ \mathbf{of}\ \ \underline{1} \Rightarrow M_1, \ldots, \underline{k} \Rightarrow M_k : B}$$

To simulate a Turing Machine $\mathcal{M}$ we set:

$$\begin{aligned} AChar_k &= ABool_k \\ AState_k &= ABool_k, \\ AString_k &= \overline{\mu}\alpha.(1 \oplus (AChar_k \otimes \alpha)) \\ AConfig_n^m &= AString_n \otimes AChar_n \otimes AString_n \otimes AState_m \end{aligned}$$

The empty string ε is represented by $\mathbf{inl}(1)$. The symbol a_i $(1 \leq i \leq n)$ is represented by $\underline{i}$ of conclusion $AChar_n$, and the state q_j $(j \leq m)$ by $\underline{j}$ of conclusion $AState_m$.

Then one can define proofs for:

$$\begin{aligned} cons &: AChar_n \otimes AString_n \multimap AString_n \\ pop &: AString_n \multimap AChar_n \otimes AString_n \end{aligned}$$

by

$$\begin{aligned} cons &= \lambda x.\mathbf{let}\ \ x\ \ \mathbf{be}\ \ (a, s)\ \ \mathbf{in}\ \ \mathbf{inr}(a, s) \\ pop &= \lambda s.\mathbf{case}\ \ s\ \ \mathbf{of}\ \ \mathbf{inl}(x) \Rightarrow (\underline{1}, \mathbf{inl}(1))\, , \mathbf{inr}(y) \Rightarrow y \end{aligned}$$

The proof *pop* applied to a non-empty string returns its head and tail; by convention it returns $(\underline{1}, \mathbf{inl}(1))$ when applied to the empty string.

Given the transition function δ of $\mathcal{M}$ we construct a proof $step_{\mathcal{M}} : AConfig_n^m \multimap AConfig_n^m$ implementing a step of execution of $\mathcal{M}$:

$$\begin{aligned} step_{\mathcal{M}} &= \lambda x.\mathbf{let}\ \ x\ \ \mathbf{be}\ \ (s, a, t, q)\ \ \mathbf{in} \\ &\qquad \mathbf{case}\ \ q\ \ \mathbf{of}\ \ (\ldots, \underline{i} \Rightarrow (\mathbf{case}\ \ a\ \ \mathbf{of}\ \ (\ldots, \underline{j} \Rightarrow M_{i,j}, \ldots), \ldots) \end{aligned}$$

where $M_{i,j}$ is defined according to the value of $\delta(q_i, a_j)$. For instance if $\delta(q_i, a_j) = (q_l, a_k, \leftarrow)$ then:

$$M_{i,j} = \mathbf{let}\ \ (pop\ s)\ \ \mathbf{be}\ \ (b, r)\ \ \mathbf{in}\ \ (r, b, (cons\ \underline{k}\ t), \underline{l})$$

Then, arguing as in the previous section we conclude that:

Proposition 7. The system $\mathbf{ILAL}_{\multimap\otimes\oplus\overline{\mu}}$ is polytime sound and complete.

9. Getting Rid of Second Order Quantification

Looking at the encoding of Turing machines from Section 7, we can can see that (linear) second order is used in a very restricted way there. In the proof $step_{\mathcal{M}}$, there are just three instances of $L_{\overline{\forall}}$ rule acting on $SOString_n$, $SOChar_n$ and $SOState_m$ respectively. In particular:

- $SOState_m$ is instantiated with formula $SOString_n \otimes SOChar_n \otimes SOString_n \multimap SOConfig_n^m$ (see (1) in the proof of Lemma 4 in Section 7);
- $SOChar_n$ is instantiated with formula $SOString_n \otimes SOString_n \multimap SOConfig_n^m$ (see (2));
- $SOString_n$ is instantiated with formula $SOString_n \multimap SOConfig_n^m$ (see (3)).

It would thus be sufficient in order to type the terms from (1), (2), (3) to replace the use of $\overline{\forall}$ by fixpoint constructions allowing to do three similar instantiations. This is what we are going to do here, obtaining in this way a suitable transition function in $\mathbf{ILAL}_{\multimap\otimes\overline{\mu}}$ whose associated term is the same one as that of Lemma 4.

Two formulae A and B are said to be *congruent*, written $A \approx B$, if B can be obtained from A by applying (zero or more times) the rule $\overline{\mu}\alpha.A \approx A[\overline{\mu}\alpha.A/\alpha]$. In other words, $\approx$ is the reflexive and transitive closure of the above (symmetric) rule.

Note that if $A \approx B$ then the identity term can be given type $A \multimap B$, and thus A and B are in particular isomorphic types.

We get the following:

Proposition 8. For every $k, h \in \mathbb{N}$, there are $\mathbf{ILAL}_{\multimap\otimes\overline{\mu}}$ formulae $FPState_k^h$, $FPChar_k^h$ and $FPString_k^h$ such that

$$\begin{array}{rcl} FPState_k^h & \approx & PState_h(FPString_k^h \otimes FPChar_k^h \otimes FPString_k^h \multimap FPConfig_k^h) \\ FPChar_k^h & \approx & PChar_k(FPString_k^h \otimes FPString_k^h \multimap FPConfig_k^h) \\ FPString_k^h & \approx & PString_k(FPString_k^h \multimap FPConfig_k^h) \\ \text{where } FPConfig_k^h & = & FPString_k^h \otimes FPChar_k^h \otimes FPString_k^h \otimes FPState_k^h \end{array}$$

Proof. Consider the following definitions:

$$\begin{array}{rcl} FPString_k(\alpha,\beta) & = & \overline{\mu}\gamma.PString_k(\gamma \multimap \gamma \otimes \beta \otimes \gamma \otimes \alpha) \\ FPChar_k(\alpha) & = & \overline{\mu}\beta.PChar_k(FPString_k(\alpha,\beta) \otimes FPString_k(\alpha,\beta) \multimap \\ & & FPString_k(\alpha,\beta) \otimes \beta \otimes FPString_k(\alpha,\beta) \otimes \alpha) \\ FPState_k^h & = & \overline{\mu}\alpha.PState_h(FPString_k(\alpha, FPChar_k(\alpha)) \otimes \\ & & FPChar_k(\alpha) \otimes FPString_k(\alpha, FPChar_k(\alpha)) \multimap \\ & & FPString_k(\alpha, FPChar_k(\alpha)) \otimes FPChar_k(\alpha) \otimes \\ & & FPString_k(\alpha, FPChar_k(\alpha)) \otimes \alpha \end{array}$$

$$\begin{aligned} \mathit{FPChar}_k^h &= \mathit{FPChar}_k(\mathit{FPState}_k^h) \\ \mathit{FPString}_k^h &= \mathit{FPString}_k(\mathit{FPState}_k^h, \mathit{FPChar}_k(\mathit{FPState}_k^h)) \end{aligned}$$

The thesis easily follows. □

Lemma 5. For any transition function of a Turing machine there exists an $\mathbf{ILAL}_{\multimap\otimes\overline{\mu}}$ proof of $\vdash \mathit{FPConfig}_n^m \multimap \mathit{FPConfig}_n^m$ representing it.

Proof. We can type in $\mathbf{ILAL}_{\multimap\otimes\overline{\mu}}$ the terms of (1), (2), (3) in Section 7, by taking advantage of the congruences given in Proposition 8. □

We then get as in Section 7:

Theorem 4. The system $\mathbf{ILAL}_{\multimap\otimes\overline{\mu}}$ is polytime sound and complete.

10. Discussion and Perspectives

Several works, such as (Marion & Moyen 2000, Marion 2001, Hofmann 2003) have advocated the study of *intensional* aspects of implicit computational complexity (ICC), that is to say the study of *algorithms* representable in a given ICC language, as opposed to *functions*. Note that this is not the main focus of the present paper, as we mainly studied functions representable in ICC systems and used encoding of Turing machines. Moreover the notion of uniform encodings we considered is somehow delicate from a programmer's point of view. It allows to consider different data structures for different inputs and outputs, so in a sense gives more flexibility. However, as it stands, finding the right data structure would still be part of the programming task, which might be a bit too much to ask. This in particular would be a difficulty for modular programming.

Nevertheless, our main point here was rather to stress that in the setting of light logics several interesting fragments or sublanguages are available. Before exploring intensional expressivity, one needs to study extensional expressivity, which is what we did here for **LAL** and its variants. This gives us some criteria for comparing logical fragments or languages. Once relevant languages with suitable extensional properties have been isolated, they can be used to suggest new constants or programming primitives, compatible with typing and preserving complexity properties. Note for example that the use of type fixpoints has enabled us to give simpler encodings of Turing machines than the original ones for second-order **LAL** (Asperti & Roversi 2002). We leave for future work a more complete study of the intensional aspects of the logical systems identified in this paper.

11. Conclusion

In this work we delineated the computational power of several fragments of Light Affine Logic with respect to a general notion of encoding of functions. The results are summarized in Figure 10, illustrating which fragments of **ILAL** are known to be respectively polytime sound or complete, or not. In particular we showed on the one hand that the purely implicative propositional fragment of **ILAL** is not polytime complete (under a further natural assumption on the encoding), and on the other hand that the extension with linear fixpoints is polytime complete and sound.

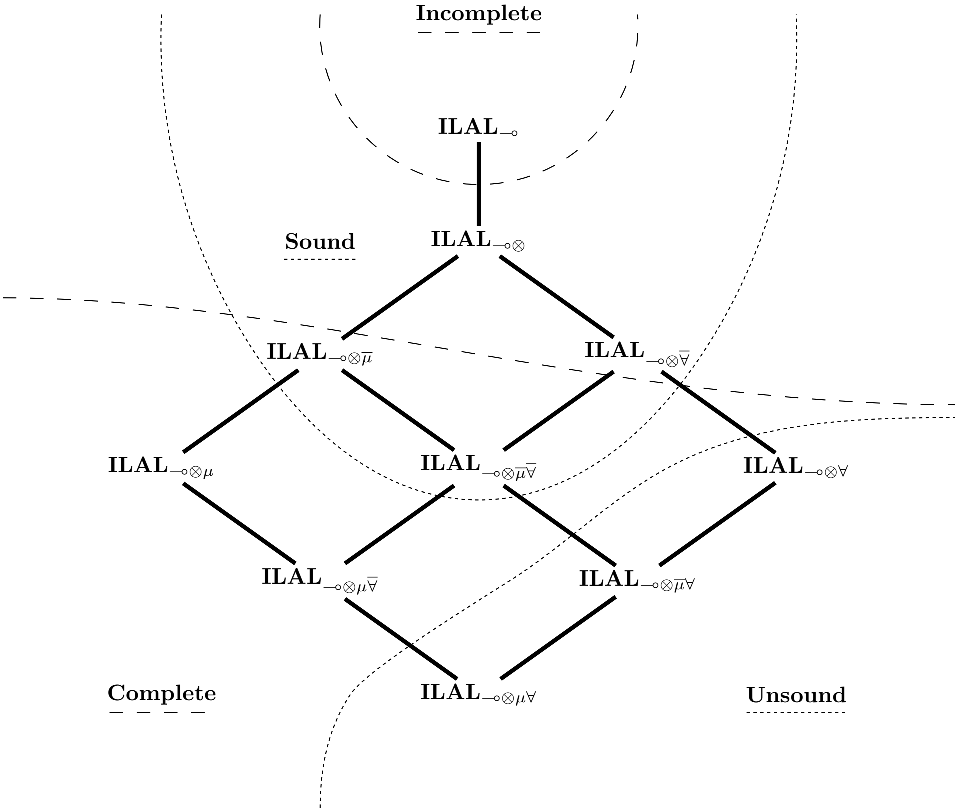

Fig. 10. Polytime soundness and completeness results for fragments of **ILAL** .

References

A. Asperti (1998). 'Light Affine Logic'. In *Proceedings of the 13th IEEE Syposium on Logic in Computer Science*, pp. 300–308.

A. Asperti & L. Roversi (2002). 'Intuitionistic Light Affine Logic'. *ACM Transactions on Computational Logic* **3(1)**:137–175.

P. Baillot & V. Mogbil (2004). 'Soft lambda-calculus: a language for polynomial time computation'. In *Proceedings of 7th International Conference on Foundations of Software Science and Computation Structures*, vol. 2987 of *LNCS*, pp. 29–41. Springer.

U. Dal Lago (2003). 'On the Expressive Power of Light Affine Logic'. In *Proceedings of 8th Italian Conference on Theoretical Computer Science*, vol. 2841 of *LNCS*, pp. 216–227. Springer.

U. Dal Lago & P. Baillot (2004). 'On Light Logics, Uniform Encodings and Polynomial Time'. Preliminary Version. Presented at the Workshop on Logics for Resources, Processes and Programs (affiliated with LICS and ICALP). 11 Pages.

S. Fortune, et al. (1983). 'The expressiveness of simple and second-order type structures'. *Journal of the ACM* **30**(1):151–185.

J.-Y. Girard (1987). 'Linear Logic'. *Theoretical Computer Science* **50**:1–102.

J.-Y. Girard (1998). 'Light Linear Logic'. *Information and Computation* **143(2)**:175–204.

M. Hofmann (2003). 'Linear types and non-size-increasing polynomial time computation.'. *Information and Computation* **183**(1):57–85.

Y. Lafont (2004). 'Soft Linear Logic and Polynomial Time'. *Theoretical Comper Science* **318**(1-2):163–181.

D. Leivant & J.-Y. Marion (1993). 'Lambda Calculus Characterizations of Poly-Time'. *Fundamenta Informaticae* **19**(1/2):167–184.

H. G. Mairson & K. Terui (2003). 'On the Computational Complexity of Cut-Elimination in Linear Logic'. In *Proceedings of 8th Italian Conference on Theoretical Computer Science*, vol. 2841 of *LNCS*, pp. 23–36. Springer.

J.-Y. Marion (2001). 'Complexité implicite des calculs de la théorie à la pratique'. Habilitation Thesis. Université Nancy 2.

J.-Y. Marion & J.-Y. Moyen (2000). 'Efficient First Order Functional Program Interpreter with Time Bound Certifications.'. In *Proceedings of 7th International Conference on Logic for Programming and Automated Reasoning*, vol. 1955 of *LNCS*, pp. 25–42. Springer.

A. Murawski & L. Ong (2000). 'Discreet Games, Light Affine Logic and PTIME Computation'. In *Proceedings of 14th Annual Conference of the European Association of Computer Science Logic*, vol. 1862 of *LNCS*, pp. 427–441.

A. Murawski & L. Ong (2004). 'On an interpretation of safe recursion in light affine logic'. *Theoretical Computer Science* **318**(1-2):197–223.

P. M. Neergard & H. G. Mairson (2002). 'LAL is Square: Representation and Expressiveness in Light Affine Logic'. Presented at the Fourth International Workshop on Implicit Computational Complexity.

L. Roversi (2000). 'Light Affine Logic as a Programming Language: a First Contribution'. *International Journal of Foundations of Computer Science* **11**(1):113 – 152.

R. Statman (1982). 'Completeness, Invariance, and λ-definability'. *Journal of Symbolic Logic* **47**(1):17–26.

K. Terui (2002). *Light logic and polynomial time computation*. Ph.D. thesis, Keio University.

K. Terui (2004). 'Light Affine Set Theory: A Naive Set Theory of Polynomial Time'. *Studia Logica* **77**(1):9–40.

C. Wadsworth (1980). 'Some unusual λ-calculus numeral systems'. In J. Seldin & J. Hindley (eds.), *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism.* Academic Press.