

# On Light Logics, Uniform Encodings and Polynomial Time

Ugo Dal Lago\*  
dallago@cs.unibo.it

Patrick Baillot†  
pb@lipn.univ-paris13.fr

June 7, 2004

## Abstract

We investigate on the extensional expressive power of Light Affine Logic, analysing the class of uniformly representable functions for various fragments of the logic. We give evidence on the incompleteness (for polynomial time) of the propositional fragment. Following previous work, we show that second order leads to polytime unsoundness. We then introduce simple constraints on second order quantification and least fixpoints, proving the obtained fragment 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 logics, which have been defined precisely for capturing relevant function classes, namely complexity classes.

Light Linear Logic (**LLL**, [6]) has been proposed by Girard as a variant of Linear Logic (**LL**, [5]) characterizing the class **FP** of deterministic polynomial time functions. It has been later simplified by Asperti into Light Affine Logic (**LAL**, [1]). The limited computational power of **LLL/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 [10, 11, 12, 14]. Another line of research in that direction is Lafont's Soft Linear Logic (**SLL**, [7]) 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 box is linked to the modalities). Recently, Mairson and Neergaard [9] 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 [8] for various subsystems of **LL**.

An important point is that the encodings in [6, 2] 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.

---

\*Dipartimento di Scienze dell'Informazione, Università di Bologna, Italy.

†LIPN-CNRS, Université Paris-Nord, France.

This question is all the more sensible as **LAL** and **SLL** are compatible with another feature: fixpoints. Indeed, least fixpoints of formulae were from the beginning one of the intuitions underlying the definition of **LLL** (see the introduction of [6]). They are also *definable* in Light Set Theory (**LST**, see [6, 14]), 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 [3].

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 [4], 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. A uniform encoding  $\mathcal{E}(f)$  of  $f : (\{0, 1\}^*)^n \rightarrow \{0, 1\}^*$  into a logic consists of:

- A proof  $\pi$  with conclusion  $A_1, \dots, A_n \vdash B$ , (where  $A_1, \dots, A_n, B$  can be different);
- For every  $i \in \{1, \dots, n\}$ , a suitable correspondence  $\Phi_i$  between elements of  $\{0, 1\}^*$  and cut-free proofs having conclusions  $\vdash A_i$ . These correspondences and their inverses must be logspace computable;
- A correspondence  $\Psi$  between elements of  $\{0, 1\}^*$  and cut-free proofs having conclusion  $\vdash B$ .

This correspondence and its inverse must be logspace computable.

Clearly, the following diagram should commute

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

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

In this paper, we proceed on the way opened in [4], extending this investigation to various fragments of light affine logic. First of all, we give some evidence on the incompleteness (for polynomial time) of the propositional fragment. Then we introduce simple constraints on second order quantification and least fixpoints, proving the obtained fragments to be polytime sound and complete.

## 2 Syntax

Following existing literature, we will use an intuitionistic variant of **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  $\alpha$  ranges over a set  $\mathcal{L}$  of *atoms*. *Sequents* have the form  $A_1, \dots, A_n \vdash B$ , where  $A_1, \dots, A_n, B$  are all formulae. We will study various fragments of **ILAL**. The core will be **ILAL** $_{\multimap}$ , and is reported in figure 1. To this core, we can add other connectives obtaining more powerful logics. For example, we can add tensor ( $\otimes$ , see figure 4) and second order quantification ( $\forall$ , see figure 3). Another interesting connective that can be added to the logic is the least fixpoint operator ( $\mu$ , see figure 4). In this way, we can build several fragments of Intuitionistic Light Affine Logic, such as **ILAL** $_{\multimap \otimes \forall}$  or **ILAL** $_{\multimap \otimes \forall \mu}$ . An **ILAL** *proof* is simply a tree whose nodes are labelled with sequents according to **ILAL** rules. A proof  $\pi$  having conclusion  $\Gamma \vdash A$  is sometimes denoted as  $\pi : \Gamma \vdash A$ .

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

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

**Identity and Cut.**

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

**Structural Rules.**

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

**Implicative Logical Rules.**

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

**Exponential Logical Rules.**

$$\frac{A \vdash B}{!A \vdash !B} P^! \quad \frac{\vdash A}{\vdash !A} P^! \quad \frac{\Gamma, \Delta \vdash A}{!\Gamma, \S \Delta \vdash \S A} P^{\S}$$

Figure 1: Implicative Intuitionistic Light Affine Logic, **ILAL**<sub>∞</sub>.

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

Figure 2: Tensor Logical Rules

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

Figure 3: Second Order Rules

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

Figure 4: Fixpoint Rules

In this setting, rules for  $!$ ,  $\S$ ,  $\forall$  and  $\mu$  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  $A_1, \dots, A_n \vdash B$  types term  $M$ , then free variables appearing in  $M$  will be  $x_1, \dots, x_n$ . Most results about **ILAL** are traditionally given on proof-nets, which are handy in studying the dynamics of proofs. Nevertheless, we chose to present **ILAL** as a sequent calculus, in order to save space. 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 problem convert the proofs into proof-nets if one wants to examine the normalization issues.

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

It is easy to check that this definition of box-depth is equivalent to the one traditionally given on **ILAL** proof-nets [2].

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. Any **ILAL** reflective fragment is polytime sound, as a direct consequence of

**Theorem 1 (ILAL normalization complexity, [2])** *Normalization of an **ILAL** proof  $\pi$  takes polynomial time, the exponent of the polynomial depending only on  $\partial(\pi)$ .*

### 3 The Full Case

Let's start with the fragment **ILAL** $_{\rightarrow \otimes \forall}$ . We know from [2] that this fragment is polytime complete. In spite of that, it is not reflective due to rule  $L_\forall$ , which can be used to build proofs with fixed conclusion but unbounded box-depth. Indeed, **ILAL** $_{\rightarrow \otimes \forall}$  is polytime unsound if the full power of second-order quantification is exploited [4]:

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

The question is: can we restrict **ILAL** $_{\rightarrow \otimes \forall}$  to reach a polytime sound and complete system? This is the main subject of this paper. The solution which has been pursued in [4] 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 touching coding schemes.

### 4 **ILAL** $_{\rightarrow}$ and Polynomial Time

In this section, we will prove that, under reasonable assumptions on the encodings, **ILAL** $_{\rightarrow}$  is not polytime complete. **ILAL** $_{\rightarrow}$  can be seen as a type assignment system for pure lambda-calculus. If a pure lambda-term  $M$  can be typed by an **ILAL** $_{\rightarrow}$  proof, then it is simply-typable. Moreover, if  $M$  can be typed by a cut-free **ILAL** $_{\rightarrow}$  proof, then it is necessarily a  $\beta$ -normal form, but can possibly contain  $\eta$ -redexes. An encoding of  $f$  into **ILAL** $_{\rightarrow}$  is said to be *extensional* if all the correspondences  $\Phi_1, \dots, \Phi_n, \Psi$  map distinct elements of  $\{0, 1\}^*$  to **ILAL** $_{\rightarrow}$  proofs whose underlying lambda-terms are distinct and  $\eta$ -free.

Now, we can recall a theorem by Statman:

**Theorem 2 (Finite Completeness Theorem, [13])** *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

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

And now we can get

**Proposition 2** *equality is not extensionally encodable into  $\mathbf{ILAL}_{\rightarrow}$ .*

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

$$\begin{aligned} M & : A \rightarrow B \rightarrow C \\ N & : C \\ L & : C \\ N & \neq_{\beta\eta} L \end{aligned}$$

where  $N$  and  $L$  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  $B$ :

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

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

$$\begin{aligned} MP(s)Q(s) & \rightarrow_{\beta}^* N \\ MP(s)Q(t) & \rightarrow_{\beta}^* L \end{aligned}$$

From the extensionality hypothesis, we know that both  $N$  and  $L$  are  $\eta$ -free. Now, call  $\mathcal{M}$  the model  $\mathcal{M}(N)$  obtained by theorem 2 applied to the term  $N$ . 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,  $MP(s)Q(s) =_{\beta\eta} N$  and  $MP(t)Q(s) =_{\beta\eta} L$ ; so, by soundness we have:

$$\begin{aligned} \mathcal{M} & \models MP(s)Q(s) = N \\ \mathcal{M} & \models MP(t)Q(s) = L. \end{aligned}$$

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

$$\mathcal{M} \models N = L.$$

By Theorem 2 this implies  $N =_{\beta\eta} L$ , hence a contradiction.  $\square$

## 5 Polynomials and $\mathbf{ILAL}_{\rightarrow\otimes}$

Throughout the paper when speaking of polynomials we will mean polynomials with positive integer coefficients. In this section, we will prove that all polynomials can be represented into  $\mathbf{ILAL}_{\rightarrow\otimes}$  using a Church-style encoding for numerals.

For every  $\mathbf{ILAL}_{\rightarrow\otimes}$  formula  $A$ ,  $\text{Int}(A)$  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  $A$ ,  $\text{Int}(A)$  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:

$$B ::= \text{Int}(A) \mid !B \mid \S B$$

where  $A$  ranges over  $\mathbf{ILAL}_{\rightarrow\otimes}$  formulas.

**Lemma 1** *For every  $\mathbf{ILAL}_{\rightarrow\otimes}$  formula  $A$ , there are proofs*

$$\begin{aligned} \pi_{+1} : \text{Int}(A) & \vdash \text{Int}(A) \\ \pi_{+} : \text{Int}(A), \text{Int}(A) & \vdash \text{Int}(A) \\ \pi_{\times} : \text{Int}(\text{Int}(A)), !\text{Int}(A) & \vdash \S \text{Int}(A) \end{aligned}$$

*representing successor, addition and multiplication respectively.*

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

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

respectively.  $\square$

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

- The identity on natural numbers is a basic arithmetical function;
- All constants are basic arithmetical functions;
- If  $f : \mathbb{N}^n \rightarrow \mathbb{N}$  and  $g : \mathbb{N}^m \rightarrow \mathbb{N}$  are basic arithmetical functions, then  $f + g : \mathbb{N}^{n+m} \rightarrow \mathbb{N}$ , defined by

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

is a basic arithmetical function;

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

$$\tilde{f}(x_1, \dots, x_n, x) = f(x_1, \dots, x_n) x$$

is a basic arithmetical function.

**Lemma 2** *For every formula  $A$  and for every basic arithmetical function  $f : \mathbb{N}^n \rightarrow \mathbb{N}$ , there is an  $\mathbf{ILAL}_{\rightarrow \otimes}$  proof  $\pi_f : A_1, \dots, A_n \vdash \S^k \text{Int}(A)$  representing  $f$ , where  $A_1, \dots, A_n$  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  $g : \mathbb{N}^n \rightarrow \mathbb{N}$  and  $h : \mathbb{N}^m \rightarrow \mathbb{N}$ , then by induction hypothesis, there must be proofs

$$\begin{aligned} \pi_g &: A_1, \dots, A_n \vdash \S^k \text{Int}(A) \\ \pi_h &: B_1, \dots, B_m \vdash \S^l \text{Int}(A) \end{aligned}$$

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

$$\frac{\frac{\pi_g : A_1, \dots, A_n \vdash \S^k \text{Int}(A)}{\S^l A_1, \dots, \S^l A_n \vdash \S^{l+k} \text{Int}(A)} \quad \rho}{\S^l A_1, \dots, \S^l A_n, \S^k B_1, \dots, \S^k B_m \vdash \S^{l+k} \text{Int}(A)}$$

where  $\rho$  is

$$\frac{\frac{\pi_h : B_1, \dots, B_m \vdash \S^l \text{Int}(A)}{\S^k B_1, \dots, \S^k B_m \vdash \S^{k+l} \text{Int}(A)} \quad \frac{\pi_+ : \text{Int}(A), \text{Int}(A) \vdash \text{Int}(A)}{\S^{l+k} \text{Int}(A), \S^{l+k} \text{Int}(A) \vdash \S^{l+k} \text{Int}(A)}}{\S^k B_1, \dots, \S^k B_m, \S^{l+k} \text{Int}(A) \vdash \S^{l+k} \text{Int}(A)}$$

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

$$\pi_g : A_1, \dots, A_n \vdash \S^k \text{Int}(\text{Int}(A))$$

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

$$\frac{\pi_g : A_1, \dots, A_n \vdash \S^k \text{Int}(\text{Int}(A)) \quad \frac{\pi_\times : \text{Int}(\text{Int}(A)), \text{Int}(A) \vdash \S \text{Int}(A)}{\S^k \text{Int}(\text{Int}(A)), \S^k \text{Int}(\text{Int}(A)) \vdash \S^{k+1} \text{Int}(A)}}{A_1, \dots, A_n, \S^k \text{Int}(\text{Int}(A)) \vdash \S^{k+1} \text{Int}(A)}$$

This concludes the proof.  $\square$

**Proposition 3** For every formula  $A$  and for every polynomial  $f : \mathbb{N} \rightarrow \mathbb{N}$ , there is an **ILAL** $_{-\otimes}$  proof  $\pi_f : \text{Int}(B) \vdash \S^k \text{Int}(A)$  representing  $f$ .

**Proof.** Any polynomial  $f : \mathbb{N} \rightarrow \mathbb{N}$  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  $a_{cp(1)}^{ca(1)}, \dots, a_{cp(p)}^{ca(p)}$  and all the  $x$  occurrences in another sequence  $a_{ip(1)}^{ia(1)}, \dots, a_{ip(q)}^{ia(q)}$ . Let  $m$  be  $\sum_{i=1}^n m(i)$ . The function  $g : \mathbb{N}^m \rightarrow \mathbb{N}$  defined by

$$g(x_1^1, \dots, x_1^{m(1)}, \dots, x_n^1, \dots, x_n^{m(n)}) = \sum_{i=1}^n \prod_{j=1}^{m(i)} x_i^j$$

is a basic arithmetical function. So, by lemma 2, there is an **ILAL** $_{-\otimes}$  proof

$$\pi_g : A_1^1, \dots, A_1^{m(1)}, \dots, A_n^1, \dots, A_n^{m(n)} \vdash \S^k \text{Int}(A)$$

encoding  $g$ . Now, the function  $f$  is obtained from  $g$  by:

- (i) substituting to each  $x_{cp(i)}^{ca(i)}$  ( $1 \leq i \leq p$ ) the constant  $x_{cp(i)}^{ca(i)}$ ,
- (ii) identifying all  $x_{ip(i)}^{ia(i)}$  ( $1 \leq i \leq q$ ) into the same variable  $x$ .

An idea to define from  $\pi_g$  a proof  $\pi_f$  representing  $f$  is thus:

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

The proof  $\rho$  can in fact be defined without using contraction, simply by applying the iteration scheme associated to an integer formula: the term underlying  $\rho : \vdash \text{Int}(A_{ip(1)}^{ia(1)} \otimes \dots \otimes A_{ip(q)}^{ia(q)}) \vdash \S(A_{ip(1)}^{ia(1)} \otimes \dots \otimes A_{ip(q)}^{ia(q)})$  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  $\pi_f$  can then be defined as:

$$\frac{\frac{\omega(a_{cp(1)}^{ca(1)}) : \vdash A_{cp(1)}^{ca(1)} \quad \pi_g : A_1^1, \dots, A_n^{m(n)} \vdash \S^k \text{Int}(A)}{\omega(a_{cp(p)}^{ca(p)}) : \vdash A_{cp(p)}^{ca(p)} \quad A_{cp(p)}^{ca(p)}, A_{ip(1)}^{ia(1)}, \dots, A_{ip(q)}^{ia(q)} \vdash \S^k \text{Int}(A)}}{\frac{A_{ip(1)}^{ia(1)}, \dots, A_{ip(q)}^{ia(q)} \vdash \S^k \text{Int}(A)}{\S(A_{ip(1)}^{ia(1)} \otimes \dots \otimes A_{ip(q)}^{ia(q)}) \vdash \S^{k+1}(\text{Int}(A))}}{\rho \quad \frac{\S(A_{ip(1)}^{ia(1)} \otimes \dots \otimes A_{ip(q)}^{ia(q)}) \vdash \S^{k+1}(\text{Int}(A))}{\text{Int}(A_{ip(1)}^{ia(1)} \otimes \dots \otimes A_{ip(q)}^{ia(q)}) \vdash \S^{k+1} \text{Int}(A)}}$$

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

$$\boxed{
\begin{array}{cc}
\frac{\vdash \Gamma, A[C/\alpha] \vdash B \quad C \in \mathcal{L}}{\Gamma, \bar{\forall}\alpha.A \vdash B} L_{\bar{\forall}} & \frac{\Gamma \vdash A \quad \alpha \notin FV(\Gamma)}{\Gamma \vdash \bar{\forall}\alpha.A} R_{\bar{\forall}} \\
\frac{\vdash \Gamma, A[\bar{\mu}\alpha.A/\alpha] \vdash B \quad A \in \mathcal{L}}{\Gamma, \bar{\mu}\alpha.A \vdash B} L_{\bar{\mu}} & \frac{\Gamma \vdash A[\bar{\mu}\alpha.A/\alpha] \quad A \in \mathcal{L}}{\Gamma \vdash \bar{\mu}\alpha.A} R_{\bar{\mu}}
\end{array}
}$$

Figure 5: Linear Quantifiers and Fixpoints

## 6 Linear Quantifiers and Fixpoints

We say an  $\mathbf{ILAL}_{\rightarrow \otimes \forall \mu}$  formula  $A$  is *linear* if it does not contain any instance of  $!$  or  $\S$ . We denote by  $\mathcal{L}$  the class of  $\mathbf{ILAL}$  linear formulae. Suppose that in rule  $L_{\forall}$ , the formula  $C$  must be linear, with similar restrictions applying to  $L_{\mu}$  and  $R_{\mu}$ . In other words, suppose we replace  $\forall$  and  $\mu$  by two new operators  $\bar{\forall}$  and  $\bar{\mu}$ , which are subject to the rules in figure 5. Let us denote by  $\mathbf{ILAL}_{\rightarrow \otimes \bar{\forall} \bar{\mu}}$  this new fragment. Observe that when read bottom-up the rules  $L_{\bar{\forall}}$ ,  $L_{\bar{\mu}}$ ,  $R_{\bar{\mu}}$  do not introduce new occurrences of  $!$  or  $\S$ . It follows that the number of rules  $P_1^1$ ,  $P_1^2$  and  $P_3$  in a cut-free  $\mathbf{ILAL}_{\rightarrow \otimes \bar{\forall} \bar{\mu}}$  proof is bounded by the number of occurrences of  $!$  and  $\S$  in its conclusion; therefore we have:

**Fact 1** *The fragment  $\mathbf{ILAL}_{\rightarrow \otimes \bar{\forall} \bar{\mu}}$  is reflective.*

By Theorem 1 we thus have:

**Proposition 4** *The system  $\mathbf{ILAL}_{\rightarrow \otimes \bar{\forall} \bar{\mu}}$  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  $\Sigma = \{a_1, \dots, a_n\}$ , where  $a_1$  is considered as the blank symbol;
- A set  $Q = \{q_1, \dots, q_m\}$  of states, where  $q_1$  is considered as the starting state;
- A transition function  $\delta : Q \times \Sigma \rightarrow Q \times \Sigma \times \{\leftarrow, \rightarrow, \downarrow\}$ .

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  $k$ ):

$$\begin{aligned}
String(k) &= \bar{\forall}\alpha.\bar{\mu}\beta.\underbrace{(\beta \multimap \alpha) \multimap \dots \multimap (\beta \multimap \alpha)}_{k \text{ times}} \multimap \alpha \multimap \alpha \\
Character(k) &= \bar{\forall}\alpha.\underbrace{\alpha \multimap \dots \multimap \alpha}_{k \text{ times}} \multimap \alpha \\
State(k) &= \bar{\forall}\alpha.\underbrace{\alpha \multimap \dots \multimap \alpha}_{k \text{ times}} \multimap \alpha
\end{aligned}$$

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

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

which, seen as proof, has conclusion  $Character(n)$ . Analogously, state  $q_i$  will be represented by  $projection_i^n$ . Counterparts of strings in  $\Sigma^*$  are defined by induction:

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

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

$append_i^n : String(k) \multimap String(k)$  encodes the juxtaposition of  $a_i$  to the input string:

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

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

Figure 6: Rules for constant 1

$$\boxed{\frac{\Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma, A \oplus B \vdash C} L_{\oplus} \quad \frac{\Gamma \vdash A}{\Gamma \vdash A \oplus B} R_{\oplus}^1 \quad \frac{\Gamma \vdash A}{\Gamma \vdash B \oplus A} R_{\oplus}^2}$$

Figure 7: Rules for  $\oplus$

Configurations become cut free proofs for

$$\text{Config}(\mathcal{M}) = \text{String}(n) \otimes \text{Character}(n) \otimes \text{String}(n) \otimes \text{State}(m)$$

A proof  $\text{step}(\mathcal{M})$  with conclusion  $\text{Config}(\mathcal{M}) \multimap \text{Config}(\mathcal{M})$  encoding the transition function  $\mathcal{M}$  can be built easily. The  $\lambda$ -term corresponding to  $\text{step}(\mathcal{M})$  is

$$\lambda x. \text{let } x \text{ be } (s, a, t, q) \text{ in } (q M_1 \dots M_m)(s, a, t)$$

where, for every  $i$ , term  $M_i : \text{String}(n) \otimes \text{Character}(n) \otimes \text{String}(n) \multimap \text{Config}(\mathcal{M})$  has the form:

$$\lambda x. \text{let } x \text{ be } (s, a, t) \text{ in } (a N_i^1 \dots N_i^n)(s, t).$$

Each  $N_i^j : \text{String}(n) \otimes \text{String}(n) \multimap \text{Config}(\mathcal{M})$  encodes the value of  $\delta(q_i, a_j)$ . If, as in the example above,  $\delta(q_i, a_j) = (q_i, a_k, \leftarrow)$ ,  $N_i^j$  will be the term

$$\lambda x. \text{let } x \text{ be } (s, t) \text{ in } (s P^1 \dots P^n R) t$$

where, for every  $r$ , term  $P^r$  is

$$\lambda s. \lambda t. (s, \text{projection}_r^n, \text{append}_i^k t, \text{projection}_j^l)$$

and  $R$  is

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

We can finally prove the following result:

**Proposition 5**  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$  is computable by a polynomial time Turing Machine iff  $f$  is uniformly encodable into  $\mathbf{ILAL}_{\multimap \otimes \bar{\vee} \bar{\mu}}$ .

**Proof.** Let  $\mathcal{M}$  be a Turing Machine running in time  $f : \mathbb{N} \rightarrow \mathbb{N}$ . If  $f$  is a polynomial, then proposition 3 gives us a proof  $\pi_f : \text{Int}(B) \vdash \S^k \text{Int}(\text{Config}(\mathcal{M}))$  encoding  $f$ . Observe that the term underlying  $\text{step}(\mathcal{M})$  can be typed by  $\S^k!(\text{Config}(\mathcal{M}) \multimap \text{Config}(\mathcal{M}))$ . Putting these two ingredients together, we obtain a proof  $\pi_{\mathcal{M}} : \text{Int}(B) \otimes \S^{k+1} \text{Config}(\mathcal{M}) \vdash \S^{k+1} \text{Config}(\mathcal{M})$ , which is a uniform encoding for the function computed by  $\mathcal{M}$ .  $\square$

## 7 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 7. We will also use the constant for the connective  $\otimes$ , denoted 1: the corresponding rules are given on Figure 6.

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

$$\frac{\Gamma \vdash M_1 : B \quad \dots \quad \Gamma \vdash M_k : B}{\Gamma, x : Bool(k) \vdash \mathbf{case} \ x \ \mathbf{of} \ \underline{1} \Rightarrow M_1, \dots, \underline{k} \Rightarrow M_k : B}$$

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

$$\begin{aligned} Character(k) &= Bool(k) \\ State(k) &= Bool(k), \\ String(k) &= \bar{\mu}\alpha.(1 \oplus (Character(k) \otimes \alpha)) \\ Config(\mathcal{M}) &= String(n) \otimes Char(n) \otimes String(n) \otimes State(m) \end{aligned}$$

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

Then one can define proofs for:

$$\begin{aligned} cons &: Character(n) \otimes String(n) \multimap String(n) \\ pop &: String(n) \multimap Char(n) \otimes String(n) \end{aligned}$$

by

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

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

Given the transition function  $\delta$  of  $\mathcal{M}$  we construct a proof  $step(\mathcal{M}) : Config(\mathcal{M}) \multimap Config(\mathcal{M})$  implementing a step of execution of  $\mathcal{M}$ :

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

where  $M_{i,j}$  is defined according to the value of  $\delta(q_i, a_j)$  (for more readability we used a  $n$ -ary generalized notation for  $\otimes$  constructions). For instance if  $\delta(q_i, a_j) = (q_l, a_k, \leftarrow)$  then:

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

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

**Proposition 6**  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$  is computable by a polynomial time Turing Machine iff  $f$  is uniformly encodable into **ILAL** $_{\multimap, \otimes, \bar{\mu}}$ .

## 8 Conclusion and Further Directions

In this work we delineated the expressive power of several fragments of Light Affine Logic. Using a fairly liberal notion of an encoding, we illustrated 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 two possible extensions, using linear fixpoints and either linear quantification or the plus connective, are polytime complete (as well as polytime sound). We are currently investigating on the expressive power of the fragment with fixpoints and without quantification nor additives.

## References

- [1] Andrea Asperti. Light affine logic. In *Proceedings of the 13th IEEE Symposium on Logic in Computer Science*, pages 300–308, 1998.
- [2] Andrea Asperti and Luca Roversi. Intuitionistic light affine logic. *ACM Transactions on Computational Logic*, 3(1):137–175, 2002.
- [3] P. Baillot and V. Mogbil. Soft lambda-calculus: a language for polynomial time computation. In *Proceedings of FOSSACS'04*, volume 2987 of *LNCS*, pages 29–41. Springer, 2004.
- [4] Ugo Dal Lago. On the expressive power of light affine logic. In Carlo Blundo and Cosimo Laneve, editors, *Eight Italian Conference on Theoretical Computer Science, Proceedings*, volume 2841 of *Lecture Notes in Computer Science*, pages 216–227. Springer, 2003.
- [5] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.
- [6] Jean-Yves Girard. Light linear logic. *Information and Computation*, 143(2):175–204, 1998.
- [7] Yves Lafont. Soft linear logic and polynomial time. *Theoretical Computer Science*, 318(1-2):163–181, 2004.
- [8] 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.
- [9] Peter Møller Neergard and Harry G. Mairson. LAL is square: Representation and expressiveness in light affine logic. Presented at the Fourth International Workshop on Implicit Computational Complexity, 2002.
- [10] Andrzej Murawski and Luke Ong. Can safe recursion be interpreted in light logic? Presented at the Second International Workshop on Implicit Computational Complexity, 2000.
- [11] Andrzej Murawski and Luke Ong. Discreet games, light affine logic and PTIME computation. In *Proceedings of 14th Annual Conference of the European Association of Computer Science Logic*, pages 427–441, 2000.
- [12] Luca Roversi. Light affine logic as a programming language: a first contribution. *International Journal of Foundations of Computer Science*, 11(1):113 – 152, 2000.
- [13] Richard Statman. Completeness, invariance, and  $\lambda$ -definability. *Journal of Symbolic Logic*, 47(1):17–26, 1982.
- [14] Kazushige Terui. *Light logic and polynomial time computation*. PhD thesis, Keio University, 2002.