

A P-Time Completeness Proof for Light Logics

Luca Roversi

Institut de Mathématiques de Luminy
UPR 9016 – 163, avenue de Luminy – Case 907
13288 Marseille Cedex 9 – France
`rovers@iml.univ-mrs.fr`

Abstract. We explain why the original proofs of P-Time completeness for Light Affine Logic and Light Linear Logic can not work, and we fully develop a working one.

Keywords: Light Affine/Linear Logics, P-Time completeness, Programming with feasible functions.

1 Introduction

The aim of this work is twofold. On one side, it develops in full details the proof that Light Affine Logic (LAL) [1] is P-Time complete. This means showing that a P-Time Turing machine can be encoded as a derivation of LAL, which is a smart simplification of Light Linear Logic (LLL) [3]. The simplification consists of allowing the unconstrained use of weakening. This does not affect the complexity of the cut elimination for LAL. It remains bound by a polynomial in the dimension of the derivation. On the other side, this work introduces a very compact paradigmatic functional language Λ_{LA} for programming with the derivations of LAL, which represent feasible functions.

The development of the proof of P-Time completeness of LAL is not merely a programming exercise with an exotic functional notation. Many readers might get to this conclusion just recalling that the P-Time completeness of LAL was claimed to hold in [1], where the hints for proving it say to follow what Girard does in [3]. However, following Girard, one gets stuck. The problem is that the derivation of LLL, encoding the transition function of the Turing machine being represented, does not correspond to an iterable program, which we call *t_fun* for short. Let us see why this happens. Firstly, recall that an *iteration principle* can be derived in LLL: it requires that the iterated function has a type $\tau \multimap \tau$ for some “light linear” type τ . Secondly, the iteration principle serves for encoding the Turing machine: *t_fun* is iterated on the starting configuration at most as many times as the bound given by the polynomial. Assume now **config** be the name of the type for the representation in LAL of the configurations (tape, state, head position) of a given Turing machine. Following [3], *t_fun* can not be written with a type different from **config** \multimap §**config**, where § is one of the two modalities “!” and “§” of LLL. So, *t_fun* can not be argument of the iteration principle, and nothing works, neither in LLL, nor in LAL.

The solution to the problem is to change the representation of the configurations. To see how, we use an example. Suppose we want to represent a configuration \mathcal{C} of a Turing machine such that the tape is $10 \star 10$, the state is s_i , and the head is on the cell containing \star . The derivation used in [3] to encode \mathcal{C} corresponds to the following tuple of terms of System F [4]:

$$[\lambda o z s x . o(zx), \lambda o z s x . s(z(ox)), state_i] , \quad (1.1)$$

which becomes:

$$\lambda O Z S . \S(\lambda x . \bar{!}O(\bar{!}Z x)) \otimes \lambda O Z S . \S(\lambda x . \bar{!}S(\bar{!}Z(\bar{!}O x))) \otimes state_i \quad (1.2)$$

in the language Λ_{LA} we shall introduce. The term $\lambda o z s x . o(zx)$ encodes the tape to the left of the head in reversed order, $\lambda o z s x . s(z(ox))$ is the part of tape from the head position to its right, and $state_i$ is some encoding of the state s_i . On the contrary, our encoding of \mathcal{C} corresponds to the term:

$$\lambda o o' z z' s s' x x' . [o(zx), s'(z'(o'x')), state_i] \quad (1.3)$$

of System F, which becomes:

$$\lambda O O' Z Z' S S' . \S(\lambda x x' . \bar{!}O(\bar{!}Z x) \otimes \bar{!}S'(\bar{!}Z'(\bar{!}O' x'))) \otimes state_i \quad (1.4)$$

in Λ_{LA} . The difference between the two choices is evident. The old one separates the components of the tape in two different λ -abstractions, while the new one keeps them merged into a single λ -body. We are now in the position to have a good intuition about why (1.2) can not work. Firstly, recall that in LAL there is the \S -box constructor “ \S ”. Secondly, recall that LAL does not allow any box opening. We can only merge the borders of two boxes, but we can never drop one border completely: this is the key point for proving the complexity bound! Any encoding t_fun of the transition function working on (1.2) needs to access the components of (1.2). This can be accomplished by a t_fun with a \S -box which border must be merged with all those in (1.2). The lack of dereliction does not allow to get rid of such a \S -box in t_fun : it gets recorded in the co-domain of the type of t_fun , which can only be **config** \multimap \S **config**. This problem disappears if t_fun is written for manipulating (1.4). Indeed, the \S -box of t_fun used to access the components of (1.4) is the same as the one needed to build the new configuration. Encoding the configurations as in (1.4) we get:

Theorem. Any P-Time Turing machine with a polynomial $p(x)$ of maximal non null degree ϑ , bounding its computational complexity, can be encoded in a term M of Λ_{LA} , such that M has the Linear Affine Logic formula **tape** \multimap $\S^{\vartheta+5}$ **tape** as its type, for some suitable type **tape**.

Contents: Section 2 introduces the language Λ_{LA} . Section 3 recalls Intuitionistic Light Affine Logic and decorates its derivations with Λ_{LA} . Section 4 defines the encoding of the P-Time Turing machine in the typable fragment of Λ_{LA} , mainly focusing on the encoding of the transition function. The representation in Λ_{LA} of all the remaining details are demanded to Appendix A, and B. Section 5 recalls the justification about writing this work. Section 6 concludes the work.

Acknowledgments: Many thanks to Yves Lafont for his fundamental help about how to terminate this work. Thanks also to the anonymous referees whose suggestions helped me to eliminate some inaccuracies. The work has been partially supported by the TMR-Marie Curie Grant, contract n. ERBFMBICT972805.

2 The Functional Language

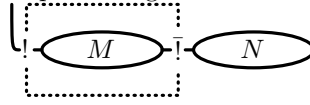
Syntax. Let $c, h, j, o, s, x, y, w, z$ range over the set of *linear* names \mathbf{T}_{vars} , and J, O, X, Y, Z range over the set of *exponential* identifiers $!\mathbf{T}_{\text{vars}}$. Let also χ be ranging over $\mathbf{T}_{\text{vars}} \cup !\mathbf{T}_{\text{vars}}$. The set of *patterns* is generated by:

$$R ::= \mathbf{T}_{\text{vars}} \mid !\mathbf{T}_{\text{vars}} \mid R \otimes R$$

and is ranged over by φ . The set Λ of the functional terms is given by:

$$M, N, P, Q ::= \mathbf{T}_{\text{vars}} \cup !\mathbf{T}_{\text{vars}} \mid \lambda\varphi.M \mid MN \mid P \otimes Q \mid !M \mid \bar{!}M \mid \S M \mid \bar{\S}M$$

For any pattern $\chi_1 \otimes \dots \otimes \chi_n$, the set $\mathbf{FV}(\chi_1 \otimes \dots \otimes \chi_n)$ of its free variables is $\{\chi_1, \dots, \chi_n\}$. As usual, λ binds the variables of M so that $\mathbf{FV}(\lambda\varphi.M)$ is $\mathbf{FV}(M) \setminus \mathbf{FV}(\varphi)$. The free variable sets of all the remaining terms are obvious as the constructors $\otimes, !, \S, \bar{!}$, and $\bar{\S}$ do not bind variables. Both $!$ and \S build $!$ -boxes and \S -boxes, respectively, being M the *body*. The term constructor $\bar{!}$ can mark one of the entry points of both $!$ -boxes and \S -boxes, while $\bar{\S}$ can mark only those of \S -boxes. An idea about what we mean by “entry point” can be given pictorially. Assume $!M$ be a $!$ -box, having a single entry point with a closed N , plugged into it. The figure representing this situation is:



where the dashed line stands for the ideal box containing M , $!$ is its exit, $\bar{!}$ is its entry point, and N has its root plugged into the entry point of $!M$.

The elements of Λ are considered up to the usual α -equivalence. It allows the renaming of the bound variables of a term M . For example, $!(\lambda x.(\bar{!}y) x)$ and $!(\lambda X.(\bar{!}y) X)$ are each other α -equivalent.

The substitution of M for χ in N is denoted by $N\{^M \downarrow_\chi\}$. It is the obvious extension to Λ of the capture-free substitution of terms for variables, defined for the λ -Calculus [2]. For example, $X\{^x \downarrow_X\}$ yields x .

It can be observed that in the definition of the substitution the existence of two sets of variables is overlooked. Both the dynamics on the terms of Λ (introduced below), and the way we give a type to them with the formulas of Intuitionistic Light Linear Logic (introduced in Section 3) will establish a substitution policy about how correctly substitute terms for variables, as follows: the substitution $N\{^M \downarrow_\chi\}$ will become equivalent to a *partial* substitution that behaves as usual *only* in one of the two, mutually exclusive, cases:

- if χ is an exponential identifier, then M must be either a $!$ -box, or an exponential identifier;

– if χ is a linear identifier, then M can be any term.

Otherwise, the substitution is undefined. For example, $X\{x \downarrow_X\}$ will not work.

The substitutions can be generalized to $\{M_1 \downarrow_{\chi_1} \cdots M_n \downarrow_{\chi_n}\}$, meaning the simultaneous replacement of M_i for χ_i , for every $1 \leq i \leq n$.

The notation $M[N_1, \dots, N_n]$ means that M may contain N_1, \dots, N_n as its sub-terms. In particular, $!M[\bar{!}N]$ means that $!M$ may contain $\bar{!}N$ as its sub-term. Notice that, under the definition of Λ , the notation $!M[\bar{!}N]$ is ambiguous: if we let M be $(x \bar{!}X \bar{!}Y)$, then $!M$ can be written both as $!M[\bar{!}X]$ and $!M[\bar{!}Y]$.

We shall use \equiv as syntactic coincidence.

Finally, a relation *unpack* on \otimes -tuples of terms is defined:

$$\begin{aligned} \text{unpack}(M_1 \otimes \dots \otimes M_m, P_1 \otimes \dots \otimes P_m \{^{N_{j_1}} \downarrow_{X_{j_1}} \dots ^{N_{j_n}} \downarrow_{X_{j_n}}\}) \text{ iff} \\ \{j_1, \dots, j_n\} \subseteq \{1, \dots, m\} \text{ where } 1 \leq k \neq i \leq n \text{ implies } j_k \neq j_i, \\ \{k_1, \dots, k_{m-n}\} \text{ is } \{1, \dots, m\} \setminus \{j_1, \dots, j_n\}, \\ M_{j_i} \equiv !Q_{j_i}[\bar{!}N_{j_i}] \text{ with } N_{j_i} \neq Z \text{ for any } Z, \\ \text{if } M_{k_l} \text{ is a !-box } !Q_{k_l}[\bar{!}N_{k_l}], \text{ then } N_{k_l} \equiv Z \text{ for some } Z, \\ P_{j_i} \equiv !Q_{j_i}[\bar{!}X_{j_i}], P_{k_l} \equiv M_{k_l}. \end{aligned}$$

Dynamics. The rewriting system \rightsquigarrow on Λ is the contextual closure of the union of two rewriting relations \triangleright_β and \triangleright_d on $\Lambda \times \Lambda$. The first is:

$$\begin{aligned} (\lambda\chi_1 \otimes \dots \otimes \chi_m.M)M_1 \otimes \dots \otimes M_m \triangleright_\beta \\ (\lambda X_{j_1} \otimes \dots \otimes X_{j_n}.M\{\dots ^{P_{j_i}} \downarrow_{X_{j_i}} \dots ^{P_{k_l}} \downarrow_{X_{k_l}} \dots\})N_{j_1} \otimes \dots \otimes N_{j_n} \\ \text{if } \text{unpack}(M_1 \otimes \dots \otimes M_m, P_1 \otimes \dots \otimes P_m \{^{N_{j_1}} \downarrow_{X_{j_1}} \dots ^{N_{j_n}} \downarrow_{X_{j_n}}\}) . \end{aligned}$$

The relation *unpack* formalizes the idea that an exponential variable can duplicate both exponential variables and !-boxes, but nothing else. On the other hand, every term can replace a linear variable. An example of \triangleright_β -reduction is:

$$(\lambda X \otimes x.M) (!(\lambda y.(\bar{!}(wz))y) \otimes (w'z')) \triangleright_\beta (\lambda Y.M\{!(\lambda y.(\bar{!}Y)y) \downarrow_X w'z' \downarrow_x\})(wz) .$$

No problem arises replacing x by $w'z'$. The part of $!(\lambda y.(\bar{!}(wz))y)$ that can be duplicated by X , possibly occurring more than once in M , is only $!(\lambda y.(\bar{!}Y)y)$. So wz is kept as a single argument after the reduction.

The second rewriting relation *merges* the borders of two boxes:

$$\bar{\diamond} \diamond M \triangleright_d M \text{ with } \diamond \in \{!, \S\} .$$

The α -equivalence must be used to avoid variable clashes when rewriting terms, so that linear (respectively exponential) variables rename linear (respectively exponential) variables.

The reflexive, and transitive closure of \rightsquigarrow on Λ is \rightsquigarrow^* .

Finally, the pair $\langle \Lambda, \rightsquigarrow \rangle$ is the functional language Λ_{LA} .

3 Intuitionistic Light Affine Logic

This section recalls the sequent calculus of Intuitionistic Light Affine Logic, and decorates its inference rules by the terms of Λ_{LA} . For a correct decoration we need some adjustments of the logical formulas that the sequent calculus can derive, with respect to [1].

Logical Formulas. Let α, β, γ range over the set of *linear* identifiers \mathbf{F}_{vars} , and let δ range over the set of *exponential* identifiers $!\mathbf{F}_{\text{vars}}$. Let also ς range over $\mathbf{F}_{\text{vars}} \cup !\mathbf{F}_{\text{vars}}$. The language of the formulas of Intuitionistic Light Linear Logic is generated by:

$$\begin{aligned} \rho, \sigma, \tau &::= L \mid E \\ L &::= \mathbf{F}_{\text{vars}} \mid \rho \multimap \sigma \mid \sigma \otimes \tau \mid \S\tau \mid \forall\varsigma.L \\ E &::= !\mathbf{F}_{\text{vars}} \mid !\rho \mid \forall\varsigma.E \end{aligned}$$

As usual, \forall is a binder: the free variables of $\forall\varsigma_1 \dots \varsigma_n. \tau$ are $\mathbf{FV}(\tau) \setminus \{\varsigma_1 \dots \varsigma_n\}$ with $\mathbf{FV}(\tau)$ having the obvious inductive definition. The formulas are taken up to α -equivalence. The *linear* formulas are those generated by the grammar with L as its start symbol. The *exponentials* start from E .

Any *linear* formula τ not having \otimes as its principal operator is thought of as a *degenerate* tensor, namely a tuple with a single element.

A *basic set of assumptions* is a set of pairs $\{\chi_1 : \sigma_1, \dots, \chi_n : \sigma_n\}$ such that:

1. Every χ_i is an *exponential* (respectively *linear*) term variable if, and only if, σ_i is an *exponential* (respectively *linear*) formula;
2. $\{\chi_1 : \sigma_1, \dots, \chi_n : \sigma_n\}$ is a function with finite domain $\{\chi_1, \dots, \chi_n\}$. Namely, if $i \neq j$ then $\chi_i \neq \chi_j$.

An *extended set of assumptions* is a basic set containing also pairs $\wp : \sigma$, and satisfying some constraints. Assume \wp be $\chi_1 \otimes \dots \otimes \chi_m$. Then:

1. σ must be $\sigma_1 \otimes \dots \otimes \sigma_p$, with $p \geq m$;
2. $\{\chi_1 : \tau_1, \dots, \chi_m : \tau_m\}$ is a basic set of assumptions, where every τ_i is a, possibly degenerate, tensor of formulas in $\{\sigma_1, \dots, \sigma_p\}$.

For example, $\{X : \delta, y : \beta\}$ is a legal extended set. $\{X \otimes x : \delta, y : \beta\}$ is not.

From now on, by “assumptions” we mean “extended set of assumptions”. Meta-variables for ranging over the assumptions are Γ and Δ .

The *substitutions* on formulas replace *linear* (respectively *exponential*) formulas for *linear* (respectively *exponential*) variables. The simultaneous substitution of $\tau_1 \dots \tau_n$ for $\varsigma_1 \dots \varsigma_n$ is denoted by $\{\tau_1 \downarrow_{\varsigma_1} \dots \tau_n \downarrow_{\varsigma_n}\}$.

Logical Rules. We recall the sequent calculus for Intuitionistic Light Affine Logic [1] by decorating it with the terms of Λ_{LA} . The judgments have form:

$\Gamma \vdash M : \tau$, where Γ is a set of assumptions, M is a term of Λ_{LA} , and τ is a formula. The decorated system is:

$$\begin{array}{c}
(\text{Ax}) \frac{}{\chi : \tau \vdash \chi : \tau} \quad (\text{Cut}) \frac{\Gamma \vdash M : \sigma \quad \Delta, \chi : \sigma \vdash N : \tau}{\Gamma, \Delta \vdash N \{^M \downarrow_{\chi}\} : \tau} \\
\\
(\text{W}) \frac{\Gamma \vdash M : \tau}{\Gamma, \chi : \sigma \vdash M : \tau} \quad (\text{C}) \frac{\Gamma, X : !\sigma, Y : !\sigma \vdash M : \tau}{\Gamma, Z : !\sigma \vdash M \{^Z \downarrow_X \downarrow_Y\} : \tau} \\
\\
(-\circ_l) \frac{\Gamma \vdash M : \sigma \quad \Delta, \chi : \tau \vdash N : \rho}{\Gamma, \Delta, x : \sigma \multimap \tau \vdash N \{^x M \downarrow_{\chi}\} : \rho} \quad (-\circ_r) \frac{\Gamma, \wp : \tau_1 \otimes \dots \otimes \tau_n \vdash M : \tau}{\Gamma \vdash \lambda_{\wp}. M : \tau_1 \otimes \dots \otimes \tau_n \multimap \tau} \\
\\
(\otimes_l) \frac{\Gamma, \chi_1 : \tau_1, \chi_2 : \tau_2 \vdash M : \tau}{\Gamma, \chi_1 \otimes \chi_2 : \tau_1 \otimes \tau_2 \vdash M : \tau} \quad (\otimes_r) \frac{\Gamma \vdash M : \tau \quad \Delta \vdash N : \sigma}{\Gamma, \Delta \vdash M \otimes N : \tau \otimes \sigma} \\
\\
(!) \frac{\dots \chi_i : \sigma_i \dots \vdash M : \tau \quad 0 \leq i \leq n \leq 1}{\dots X_i : !\sigma_i \dots \vdash !M \{ \dots \overline{!}X_i \downarrow_{\chi_i} \dots \} : !\tau} \\
\\
(\S) \frac{\dots \chi_i : \tau_i \dots \chi'_j : \sigma_j \dots \vdash M : \tau \quad 0 \leq i \leq m \quad 0 \leq j \leq n}{\dots X_i : !\tau_i \dots x_j : \S \sigma_j \dots \vdash \S M \{ \dots \overline{!}X_i \downarrow_{\chi_i} \dots \overline{\S}x_j \downarrow_{\chi'_j} \dots \} : \S \tau} \\
\\
(\forall_l) \frac{\Gamma, \chi : \{\tau \downarrow_{\varsigma}\} \sigma \vdash M : \rho}{\Gamma, \chi : \forall \varsigma. \sigma \vdash M : \rho} \quad (\forall_r) \frac{\Gamma \vdash M : \sigma \quad \varsigma \notin \text{FV}(\Gamma)}{\Gamma \vdash M : \forall \varsigma. \sigma}
\end{array}$$

Observe that (!)-rule can have at most one assumption. So the notation $!M[\overline{!}N]$ can not be anymore ambiguous.

Observe that Λ_{LA} gives a very parsimonious representation of the derivations. The contraction is left implicit, allowing multiple occurrences of the same exponential variable. The pattern matching avoids the use of any *let*-like binder that would require some commuting conversions in \rightsquigarrow . The representation of the boxes is much more compact than that used in [1, 5, 6], and this prevents the need of a lot of commuting conversions. Somebody might object about the (relative) complexity of the \triangleright_{β} -reduction. It is the side effect of the lack of explicit encoding for the (C)-rule. Assuming we have it, we could redefine \triangleright_{β} simply as:

$$(\lambda \chi_1 \otimes \dots \otimes \chi_m. M) M_1 \otimes \dots \otimes M_m \triangleright_{\beta} M \{^{M_1} \downarrow_{\chi_1} \dots \downarrow_{\chi_m} \} .$$

In this case, the duplication of every M_i with form $!P_i[\overline{!}Q_i]$ as many times as the occurrences of every χ_i should be filtered by the explicit contraction. The aim: forbidding the duplication of any Q_i different from both an exponential variable and a !-box. Moreover, the explicit term for contraction would induce commuting conversions. Hence, we would pay in term of more reductions, and

more constructs. So, the detailed encoding of the P-Time Turing machines in Appendix A, and B would get (much) more unreadable.

The language Λ_{LA} does not encode explicitly the II order quantification. This only means that it has less reduction steps than Asperti's original functional notation to speak about the derivations of LAL [1]. So, the computational complexity is preserved.

Splitting the logical formulas into linear and exponential ones is a consequence of splitting the set of assumptions for the derivations. The logic, however, is unchanged: as soon as the term decorations are forgotten, any difference on the logical formulas can be forgotten as well.

4 Encoding P-Time Turing Machines

The encoding morally divides into two parts that we call *quantitative* and *qualitative*¹. The quantitative part is relative to the representation of the polynomial which bounds the computational complexity. The qualitative one is about encoding the transition function of the machine being encoded.

The quantitative part gets the representation of the initial tape as input. Its main tasks are: calculating the integer \mathcal{B} which bounds the number of computational steps, and using \mathcal{B} for iterating the representation t_fun of the transition function. The qualitative part takes a *configuration* as input, namely a copy of the representation of a tape and a state. The qualitative part implements the transition function t_fun which shifts the head of the machine on the tape, according to the actual state and to the last character read.

We assume to encode a machine $\langle \Sigma, \mathcal{S}, \mathcal{F} \rangle$ where Σ is the *input* tape alphabet $\{0, 1\}$, \mathcal{S} is the set of states with cardinality m , \mathcal{F} is the transition function $(\Sigma \cup \{\star\}) \times \mathcal{S} \longrightarrow \Sigma \times \mathcal{S} \times \{L, R\}$. The symbol \star stands for the “blank” cell, while L and R are the directions of the head moves. The head is supposed to write a symbol into the last read cell of the tape, before moving. Among the states in \mathcal{S} , we distinguish the initial one S_0 . The alphabet $\Sigma \cup \{\star\}$ is ranged over by ζ , the set of states by S , and $\{L, R\}$ by μ .

Configurations. A configuration is determined by a tape, a position of the head on it, and a state. The representation we choose is:

$$\lambda OO' ZZ' JJ'. \S (\lambda x x'. (\chi_1 (\dots (\chi_p \bar{x}) \dots) \otimes \chi'_1 (\dots (\chi'_q \bar{x}') \dots)) \otimes state_i) ,$$

where $\chi_i \in \{\bar{1}O, \bar{1}Z, \bar{1}J\}$ with $1 \leq i \leq p$, $\chi'_j \in \{\bar{1}O', \bar{1}Z', \bar{1}J'\}$ with $1 \leq j \leq q$, being $p, q \geq 0$. The type **config** of any term *config* like the one here above is:

$$\begin{aligned} \mathbf{config} &\stackrel{\text{def}}{=} \forall \alpha. !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap \\ &\quad !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap \\ &\quad !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap \\ &\quad \S (\alpha \multimap \alpha \multimap (\alpha \otimes \alpha \otimes \mathbf{state})) , \end{aligned}$$

¹ Lafont suggested this terminology.

where **state** $\stackrel{\text{def}}{=} \forall \alpha \beta. (\overbrace{(\alpha \multimap \beta) \otimes \dots \otimes (\alpha \multimap \beta)}^m \otimes \alpha) \multimap \beta$ is the type of the terms $state_i$, each one representing an element of \mathcal{S} , which, recall, has cardinality m .

Example 1. The configuration where the head is on \star of the tape:

$$10 \star 10 \ , \quad (4.1)$$

and the actual state is S_i , is encoded with:

$$\lambda OO' ZZ' JJ'. \S((\lambda xx'. \bar{1}Z(\bar{1}O \ x) \otimes \bar{1}J'(\bar{1}O'(\bar{1}Z' \ x')))) \otimes state_i) \ . \quad (4.2)$$

The leftmost component of the tensor in the body of the λ -abstraction is the part of the tape to the left of the head, in reversed order. The cell read by the head, and the part of the tape to its right is the central component of the tensor.

States. The term $state_i$ is:

$$\lambda x_0 \otimes \dots \otimes x_{m-1} \otimes v. x_i \ v \quad (0 \leq i \leq m-1) \ .$$

Every $state_i$ is designed to extract a row from an array. The parameter x_i stands for the row, realized by a *closed* term. The parameter v stands for the variables that the rows of the array would share additively, if they were not closed terms. Namely, $state_i$ is the i^{th} projection for the representation of an additive tuple with Intuitionistic Light Affine Logic. We can summarize as follows the behavior of $state_i$ on a tuple with two elements. Assume you want to encode a pair containing two typable terms M and N of Λ_{LA} , of which only one between them will be used in the computation. Suppose also x_1, \dots, x_n be *all* the *linear* free variables *common* to both M and N . Then $M \otimes N$ is not a legal term. It can not be typed because any x_j here above occurs twice in it, every x_j requiring an exponential type. This contrasts with the effective use of x_j we are going to do: since we assume to use *either* M , *or* N , every x_j is eventually used linearly. Like in [1], the pair is represented as the triple:

$$(\lambda x_1 \otimes \dots \otimes x_n. M) \otimes (\lambda x_1 \otimes \dots \otimes x_n. N) \otimes (x_1 \otimes \dots \otimes x_n) \ .$$

The leftmost component M is extracted by means of a projection that applies $\lambda x_1 \otimes \dots \otimes x_n. M$ to $x_1 \otimes \dots \otimes x_n$. The rightmost component N is obtained analogously, by a projection that applies $\lambda x_1 \otimes \dots \otimes x_n. N$ to $x_1 \otimes \dots \otimes x_n$. Both every $state_i$, and the array, to which $state_i$ is applied, generalize the projections, and the pair of the example here above.

Starting Configurations. Any *starting configuration* $config_0$ has form:

$$\lambda OO' ZZ' JJ'. \S((\lambda xx'. x \otimes \chi'_1(\dots(\chi'_q \ x') \dots)) \otimes state_0) \ ,$$

where every χ'_j ranges over $\{\bar{1}O', \bar{1}Z'\}$. Namely, the tape has only input characters on it, and the head is on its leftmost input symbol: the part of the tape to the left of the tape is empty. The term $state_0$ encodes S_0 .

4.1 The Quantitative Part

The quantitative part calculates the bound on the length of the computation of the P-Time Turing machine encoded. We only state the main representation theorem. More details are in Appendix A. Assume $p(x)$ be the polynomial $\sum_{i=0}^{\vartheta} a_i x^i$, associated to our Turing machine, with maximal non null degree ϑ . Then, $p(x)$ can be encoded by P in Λ_{LA} with type $\mathbf{int} \multimap \xi^{\vartheta+3} \mathbf{int}$, where:

$$\mathbf{int} \stackrel{\text{def}}{=} \forall \alpha. !(\alpha \multimap \alpha) \multimap \xi(\alpha \multimap \alpha) .$$

Of course, $p(n) = m$, if, and only if, $P\bar{n}$ evaluates to \bar{m} , where, for every $n \geq 0$:

$$\bar{n} \stackrel{\text{def}}{=} \lambda X. \xi(\lambda y. \underbrace{\bar{!}X \dots \bar{!}X}_n y) : \mathbf{int} .$$

4.2 The Qualitative Part

The qualitative part implements the transition function of the P-Time Turing machine being encoded.

Some preliminary definitions are worth giving:

$$\begin{aligned} I &\stackrel{\text{def}}{=} \lambda x. x : \mathbf{i}_\alpha \\ \pi_1 &\stackrel{\text{def}}{=} \lambda x \otimes y. x : \mathbf{bool}_\alpha \\ \pi_2 &\stackrel{\text{def}}{=} \lambda x \otimes y. y : \mathbf{bool}_\alpha \\ \Pi_{111} &\stackrel{\text{def}}{=} \lambda(((x \otimes y) \otimes w) \otimes z) \otimes x'. x x' : \mathbf{a_bool}_{\alpha, \beta} \\ \Pi_{112} &\stackrel{\text{def}}{=} \lambda(((x \otimes y) \otimes w) \otimes z) \otimes x'. y x' : \mathbf{a_bool}_{\alpha, \beta} \\ \Pi_{12} &\stackrel{\text{def}}{=} \lambda(((x \otimes y) \otimes w) \otimes z) \otimes x'. w x' : \mathbf{a_bool}_{\alpha, \beta} \\ \Pi_2 &\stackrel{\text{def}}{=} \lambda(((x \otimes y) \otimes w) \otimes z) \otimes x'. z x' : \mathbf{a_bool}_{\alpha, \beta} \\ \mathbf{i}_\alpha &\stackrel{\text{def}}{=} \alpha \multimap \alpha \\ \mathbf{bool}_\alpha &\stackrel{\text{def}}{=} (\alpha \otimes \alpha) \multimap \alpha \\ \mathbf{a_bool}_{\alpha, \beta} &\stackrel{\text{def}}{=} (((\alpha \multimap \beta) \otimes (\alpha \multimap \beta)) \otimes (\alpha \multimap \beta)) \otimes (\alpha \multimap \beta) \otimes \alpha \multimap \beta . \end{aligned}$$

I is the identity and π_1, π_2 the projections on usual booleans, while $\Pi_{111}, \Pi_{112}, \Pi_{12}$, and Π_2 are projections analogous to the states.

Moreover, let \mathbf{ozj} abbreviate $o \otimes o' \otimes z \otimes z' \otimes j \otimes j'$, and let \mathbf{OZJ} stand for $\bar{!}O \otimes \bar{!}O' \otimes \bar{!}Z \otimes \bar{!}Z' \otimes \bar{!}J \otimes \bar{!}J'$. The respective types are: $\alpha \stackrel{\text{def}}{=} \mathbf{i}_\alpha \otimes \mathbf{i}_\alpha \otimes \mathbf{i}_\alpha \otimes \mathbf{i}_\alpha \otimes \mathbf{i}_\alpha \otimes \mathbf{i}_\alpha$, and $!\alpha \stackrel{\text{def}}{=} !\mathbf{i}_\alpha \otimes !\mathbf{i}_\alpha \otimes !\mathbf{i}_\alpha \otimes !\mathbf{i}_\alpha \otimes !\mathbf{i}_\alpha \otimes !\mathbf{i}_\alpha$.

We are now ready for introducing the main terms.

$(\Pi_{111} \otimes \bar{!}O)$, the obtained element would be Q_{i1} , while $(\Pi_{12} \otimes \bar{!}J)$ would extract Q_{i3} . Finally, every Q_{ij} in *table* is one among the two shifting terms here below:

$$\begin{aligned} \mathit{left}_{ij} &\stackrel{\text{def}}{=} \lambda \mathbf{ozj} . \lambda h_l t_l t_r . (t_l \otimes h_l (\chi_{ij}^l t_r)) \otimes \mathit{state}_{ij}^l \\ \mathit{right}_{ij} &\stackrel{\text{def}}{=} \lambda \mathbf{ozj} . \lambda h_l t_l t_r . (\chi_{ij}^r (h_l t_l) \otimes t_r) \otimes \mathit{state}_{ij}^r , \end{aligned}$$

where χ_{ij}^l ranges over $\{o', z', j'\}$, and χ_{ij}^r over $\{o, z, j\}$. Every *left* and *right* term describes what to do when moving the head leftward or rightward, respectively. Of course, the form of *table* must be defined to satisfy the obvious link between the encoding and the machine encoded. The link is: $\mathcal{F}\langle \zeta, S \rangle \mapsto \langle \zeta', S', \mu \rangle$ iff the term h_r^l (*state* (*table* \mathbf{ozjz})) of *comp* \rightsquigarrow -reduces to Q , where: if $\mu \equiv L$, then $Q \equiv \mathit{left}[\chi^l, \mathit{state}']$, if $\mu \equiv R$, then $Q \equiv \mathit{right}[\chi^r, \mathit{state}']$, if $\zeta \equiv 1$, then $h_r^l \equiv \Pi_{111}$, if $\zeta \equiv 0$, then $h_r^l \equiv \Pi_{112}$, if $\zeta \equiv \star$, then $h_r^l \equiv \Pi_{12}$, *state* encodes S , *state'* encodes S' , if $\zeta' \equiv 1$, then both $\chi^l \equiv o$ and $\chi^r \equiv o'$, if $\zeta' \equiv 0$, then both $\chi^l \equiv z$ and $\chi^r \equiv z'$, if $\zeta' \equiv \star$, then both $\chi^l \equiv j$ and $\chi^r \equiv j'$.

For those who want to check that the compulsory requirement about *t_fun* is satisfied, namely, that *t_fun* has an *iterable* type $\mathbf{config} \multimap \mathbf{config}$, we give some hints about the intermediate typing: $\mathit{step} : \mathbf{step}_{\alpha, \beta}$, $\mathit{base} : \mathbf{base}_{\alpha, \beta}$, $\mathit{comp} : \mathbf{comp}$, $\mathit{table} : \mathbf{table}_{\alpha}$, $\mathit{left}_{ij} : \mathbf{shift}_{\alpha}$, and $\mathit{right}_{ij} : \mathbf{shift}_{\alpha}$, where:

$$\begin{aligned} \mathbf{step}_{\alpha, \beta} &\stackrel{\text{def}}{=} \mathbf{a_bool}_{\alpha, \beta} \multimap \mathbf{i}_{\alpha} \multimap \sigma_{\alpha, \beta} \multimap \sigma_{\alpha, \beta} \\ \mathbf{base}_{\alpha, \beta} &\stackrel{\text{def}}{=} \mathbf{a_bool}_{\alpha, \beta} \multimap \alpha \multimap \sigma_{\alpha, \beta} \\ \mathbf{comp}_{\alpha} &\stackrel{\text{def}}{=} \alpha \multimap ((\sigma_{\alpha, \tau_{\alpha}} \otimes \sigma_{\alpha, \tau_{\alpha}}) \otimes \mathbf{state}) \multimap (\alpha \otimes \alpha \otimes \mathbf{state}) \\ \mathbf{table}_{\alpha} &\stackrel{\text{def}}{=} \alpha \multimap \underbrace{\mathbf{row}_{\alpha} \otimes \dots \otimes \mathbf{row}_{\alpha}}_m \otimes \alpha \\ \mathbf{row}_{\alpha} &\stackrel{\text{def}}{=} \alpha \multimap (((\mathbf{shift}_{\alpha} \otimes \mathbf{shift}_{\alpha}) \otimes \mathbf{shift}_{\alpha}) \otimes \mathbf{shift}_{\alpha}) \otimes \alpha \\ \mathbf{shift}_{\alpha} &\stackrel{\text{def}}{=} \alpha \multimap \tau_{\alpha} \\ \sigma_{\alpha, \beta} &\stackrel{\text{def}}{=} ((\mathbf{a_bool}_{\alpha, \beta} \otimes \mathbf{i}_{\alpha}) \otimes \alpha) \\ \tau_{\alpha} &\stackrel{\text{def}}{=} (\alpha \multimap \alpha) \multimap \alpha \multimap \alpha \multimap ((\alpha \otimes \alpha) \otimes \mathbf{state}) . \end{aligned}$$

It may help also saying that the projections Π_{111} , Π_{112} , Π_{12} , Π_2 are used in *t_fun* with the types $\mathbf{a_bool}_{\alpha, \tau_{\alpha}}$ here above, because they serve as actual parameter for replacing h_r^l in *comp*.

4.3 Gluing all Together

The whole encoding of the P-Time Turing machine with polynomial $p(x)$ of maximal non null degree ϑ is a term with type $\mathbf{tape} \multimap \S^{\vartheta+5} \mathbf{tape}$, where:

$$\mathbf{tape} \stackrel{\text{def}}{=} \forall \alpha . !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha \multimap \alpha) .$$

The general scheme defining a term representing a tape is:

$$\lambda OO'ZZ'.\S(\lambda xx'.\chi_1(\dots(\chi_p x)\dots) \otimes \chi'_1(\dots(\chi'_q x')\dots)) ,$$

where $\chi_i \in \{\bar{1}O, \bar{1}Z\}$ with $1 \leq i \leq p$, $\chi'_j \in \{\bar{1}O', \bar{1}Z'\}$ with $1 \leq j \leq q$, and $p, q \geq 0$.

Appendix B has the details about the whole encoding of the Turing machine. The relation between such encoding and the Turing machine is the obvious one: if the Turing machine yields a tape t_o from an input tape t_i on the alphabet $\{0, 1\}$, the same relation holds between the encodings of t_i and t_o thanks to the term in Appendix B. Its main tasks are: feeding the quantitative part with an integer, obtained from the initial tape, and feeding the qualitative part by the starting configuration, namely the initial tape and the initial state.

5 On the Obvious Encoding

In [1], P-Time completeness of LAL is claimed to hold by saying that it can be proved following [3], where the proof of P-Time completeness for Light Linear Logic is sketched. The proof in [3] is developed on the obvious representation of configurations. For example, the tape $10 \star 10$ with the head reading \star in the state s_i , would be:

$$\lambda OZJ.\S(\lambda x.\bar{1}Z(\bar{1}O x)) \otimes \lambda OZJ.\S(\lambda x.\bar{1}J(\bar{1}O(\bar{1}Z x))) \otimes state_i : \mathbf{config}' , \quad (5.1)$$

where $state_i$ encodes s_i , and:

$$\begin{aligned} \mathbf{config}' &\stackrel{\text{def}}{=} \mathbf{tape}' \otimes \mathbf{tape}' \otimes \mathbf{state}' \\ \mathbf{tape}' &\stackrel{\text{def}}{=} \forall \alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap \S(\alpha \multimap \alpha) , \end{aligned}$$

for some suitable type \mathbf{state}' .

Any transition function t_fun' working on (5.1) needs to access the bodies of the \S -boxes of the λ -abstractions for producing a new configuration. This can be done *only* by endowing t_fun' with a \S -box as well, which border can be merged with those in (5.1). Recall, indeed, that the boxes can not be opened in LAL, but only merged. So, the use of a \S -box in t_fun' gets recorded in the co-domain of its type: $\mathbf{config}' \multimap \S \mathbf{config}'$. This type does not allow to iterate t_fun' . As can be seen in the definition of *iter* in Appendix A, the iteration works only on terms with coinciding domain and co-domains. Since the encoding of the P-Time Turing machines in LLL rests on iterating t_fun' , we get stuck.

Our, more “parallel”, representation of the configurations allows to get an encoding which of the P-TIME Turing machines in LAL. The differences between the sequent calculi of LLL and LAL make Λ_{LLA} useless for verifying directly with it that our encoding works on LLL as well. However, we do not see any reasons why the principles our encoding rests on could not be used successfully on LLL. The interested reader could try to follow our encoding idea on a Proof Nets language for LLL. The Proof Nets would avoid useless syntax overheads, caused by the high number of rules defining LLL.

6 Conclusions

This paper has been written as consequence of an attempt to study Light Affine Logic as a programming language, using Curry-Howard principles. The final aim is writing a compact language for Light Affine Logic, which is automatically typable, and P-Time complete. The study of the completeness led to the need of writing down all the details about the encoding of a P-Time Turing machine in Light Affine Logic, following [3]. Something went wrong, so the proof about P-Time completeness of Light Affine Logic was still waiting to be worked out correctly. This is what we have just done, also contributing with introducing a very compact language to program feasible functions.

References

1. A. Asperti. Light Affine Logic. In *Proceedings of Symposium on Logic in Computer Science LICS'98*, 1998.
2. H.P. Barendregt. *The Lambda Calculus*. North-Holland, second edition, 1984.
3. J.-Y. Girard. Light Linear Logic. *Information and Computation*, 143:175 – 204, 1998.
4. J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*. Cambridge University Press, 1989.
5. L. Roversi. Concrete syntax for intuitionistic light affine logic with polymorphic type assignment. In *Sixth Italian Conference on Theoretical Computer Science*. World Scientific, 9 – 11 November (Prato – Italy) 1998.
6. L. Roversi. A polymorphic language which is typable and poly-step. In *Advances in Computing Science – ASIAN'98*, volume LNCS 1538. Springer-Verlag, 8 – 10 December (Manila – The Philippines) 1998.

A Details on the Quantitative Part

Let denote $\tau \otimes \dots \otimes \tau$, with n elements, by τ_n . Let $p(x)$ be the polynomial $\sum_{i=0}^{\vartheta} a_i x^i$ describing the computational bound of the Turing machine being encoded. Let also $\kappa = \frac{\vartheta(\vartheta+1)}{2}$. The term P encoding $p(x)$ is defined as:

$$\begin{aligned}
 & \lambda x. \bar{\S}((\lambda y_0^1 \otimes \dots \otimes y_0^i \otimes \dots \otimes y_{i-1}^i \otimes \dots \otimes y_0^{\vartheta} \otimes \dots \otimes y_{\vartheta-1}^{\vartheta} \cdot \\
 & \quad \text{sum_int}_{\vartheta+1}^{\vartheta+2} \bar{\S}^1(\text{coerc_int}^{\vartheta,0} \bar{\S}^1 \langle \langle a_0 x^0 \rangle \rangle_{y^0}) \\
 & \quad \vdots \\
 & \quad \otimes \bar{\S}^{i+1}(\text{coerc_int}^{\vartheta-i,0} \bar{\S}^{i+1} \langle \langle a_i x^i \rangle \rangle_{y^i}) \\
 & \quad \vdots \\
 & \quad \otimes \bar{\S}^{\vartheta+1}(\text{coerc_int}^{0,0} \bar{\S}^{\vartheta+1} \langle \langle a_{\vartheta} x^{\vartheta} \rangle \rangle_{y^{\vartheta}}) \\
 &) \bar{\S}(\text{tuple_int}_{\kappa} x) : \mathbf{int} \multimap \bar{\S}^{\vartheta+3} \mathbf{int}
 \end{aligned}$$

where:

$$\begin{aligned} \langle\langle ax^0 \rangle\rangle_y &\mapsto \mathit{coerc_int}^{0,0} \bar{a} : \mathfrak{!int} \\ \langle\langle ax^n \rangle\rangle_y &\mapsto \mathit{mult_int}^n \langle y^{n-1} \rangle (\mathit{coerc_int}^{n-1,1} \bar{a}) : \mathfrak{!}^{n+1}\mathit{int} \quad (n \geq 1), \end{aligned}$$

being $y_1 \dots y_n$ the free variables of $\langle\langle ax^n \rangle\rangle_y$, and:

$$\begin{aligned} \langle x^0 \rangle &\mapsto \mathit{coerc_int}^{0,0} x_0 : \mathfrak{!int} \\ \langle x^n \rangle &\mapsto \mathit{mult_int}^n \langle x^{n-1} \rangle (\mathit{coerc_int}^{n-1,1} x_n) : \mathfrak{!}^{n+1}\mathit{int} \quad (n \geq 1) \end{aligned}$$

being $x_1 \dots x_n$ the free variables of $\langle x^n \rangle$, and:

$$\begin{aligned} \bar{0} &\stackrel{\text{def}}{=} \lambda X. \mathfrak{!}(\lambda x. x) : \mathit{int} \\ \bar{1} &\stackrel{\text{def}}{=} \lambda X. \mathfrak{!}(\lambda x. \bar{!}X x) : \mathit{int} \\ \mathit{sum_int}_n &\stackrel{\text{def}}{=} \lambda x_1 \otimes \dots \otimes x_n. X. \mathfrak{!}(\lambda y. \mathfrak{!}(\lambda x. X)(\dots(\mathfrak{!}(x_n X) y) \dots)) : \mathit{int}_n \multimap \mathit{int} \\ \mathit{sum_int}_n^p &\stackrel{\text{def}}{=} \lambda x_1 \otimes \dots \otimes x_n. \mathfrak{!}^p(\mathit{sum_int}_n \mathfrak{!}^p x_1 \otimes \dots \otimes \mathfrak{!}^p x_n) : (\mathfrak{!}^p \mathit{int})_n \multimap \mathfrak{!}^p \mathit{int} \\ \mathit{succ_int} &\stackrel{\text{def}}{=} \lambda x. \mathit{sum_int}_2 \bar{1} \otimes x : \mathit{int} \multimap \mathit{int} \\ \mathit{succ_int}^{p,q} &\stackrel{\text{def}}{=} \lambda x. \mathfrak{!}^p(\mathfrak{!}^q(\mathit{succ_int} \mathfrak{!}^q(\mathfrak{!}^p x))) : \mathfrak{!}^{p+q} \mathit{int} \multimap \mathfrak{!}^{p+q} \mathit{int} \\ \bar{0}^{p,q} &\stackrel{\text{def}}{=} \mathfrak{!}^{p,q} \bar{0} : \mathfrak{!}^{p,q} \mathit{int} \\ \mathit{coerc_int}^{p,q} &\stackrel{\text{def}}{=} \lambda x. \mathfrak{!}(\mathfrak{!}(\lambda x. \mathfrak{!}(\mathit{succ_int}^{p,q} \bar{0}^{p,q})) : \mathit{int} \multimap \mathfrak{!}^{p+1,q} \mathit{int} \\ \mathit{iter} &\stackrel{\text{def}}{=} \lambda x X y. \mathfrak{!}(\mathfrak{!}(\lambda x. X)(\mathfrak{!} y)) : \mathit{int} \multimap (\tau \multimap \tau) \multimap \mathfrak{!} \tau \multimap \mathfrak{!} \tau \\ \mathit{iter}^p &\stackrel{\text{def}}{=} \lambda x y z. \mathfrak{!}^p(\mathit{iter} \mathfrak{!}^p x \mathfrak{!}^p y \mathfrak{!}^p z) : \mathfrak{!}^p \mathit{int} \multimap \mathfrak{!}^p(\tau \multimap \tau) \multimap \mathfrak{!}^{p+1} \tau \multimap \mathfrak{!}^{p+1} \tau \\ \mathit{mult_int} &\stackrel{\text{def}}{=} \lambda x X. \mathit{iter} x \mathfrak{!}(\lambda y. \mathit{sum_int} \bar{!}X y) \bar{0}^{1,0} : \mathit{int} \multimap \mathit{int} \multimap \mathfrak{!} \mathit{int} \\ \mathit{mult_int}^p &\stackrel{\text{def}}{=} \lambda x y. \mathfrak{!}^p(\mathit{mult_int} \mathfrak{!}^p x \mathfrak{!}^p y) : \mathfrak{!}^p \mathit{int} \multimap \mathfrak{!}^p \mathit{int} \multimap \mathfrak{!}^{p+1} \mathit{int} \\ \mathit{tuple_int}_n &\stackrel{\text{def}}{=} \lambda x. \mathfrak{!}(\mathfrak{!}(\lambda x_1 \otimes \dots \otimes x_n. \mathit{succ_int} x_1 \otimes \dots \otimes \mathit{succ_int} x_n) \bar{0}_n) : \\ &\quad \mathit{int} \multimap \mathfrak{!}(\mathit{int}_n) \end{aligned}$$

where $p, q \geq 0$ and $n \geq 1$.

B Details on Gluing all Together: the Turing Machine

Let $P : \mathit{int} \multimap \mathfrak{!}^{\vartheta+3} \mathit{int}$ be the term encoding the quantitative polynomial of degree ϑ , and T the term encoding the table which the transition function \mathcal{F} rests on. The term encoding the Turing machine is defined as:

$$\begin{aligned} &\lambda t. \mathit{config2tape}^{\vartheta+5} (\mathfrak{!}((\lambda t_1 \otimes t_2. \mathit{iter}^{\vartheta+3} (P(\mathit{tape2int} t_1)) \\ &\quad (\mathfrak{!}^{\vartheta+3} \mathfrak{!} T) \\ &\quad (\mathit{starting_config}^{\vartheta+3} t_2) \\ &\quad) \mathfrak{!}(\mathit{dbl_tape} t))) : \mathit{tape} \multimap \mathfrak{!}^{\vartheta+5} \mathit{tape} \end{aligned}$$

