# On Elementary Linear Logic and polynomial time (Extended Abstract)

Patrick Baillot*
ENS Lyon, Université de Lyon, LIP (UMR 5668 CNRS-ENSL-INRIA-UCBL)

## 1 Introduction

Linear logic (LL) [Gir87] has been used in implicit computational complexity to characterize various complexity classes within the proofs-as-programs approach. This can then be the basis of type systems to guarantee complexity properties on lambda-calculus.

As duplication is controlled in LL by the connective ! the key idea of this line of work is to consider variants of this system with a weaker version of the !, so as to restrict the computational complexity of the reduction procedure. However this results in different systems for characterizing the various complexity classes: Light linear logic [Gir98, AR02] and Soft linear logic [Laf04] for the class FP of polynomial time functions, Elementary linear logic [Gir98, DJ03] for elementary functions, the type system $STA_B$ [GMR08] for $PSPACE$ ... By contrast in [Jon01] Jones characterizes in a common functional language (a *read-only* language) a hierarchy of complexity classes, each class being obtained by restricting the order of arguments allowed in the program. We would like to characterize in a similar way a hierarchy of complexity classes by a single logic, by considering various types.

The system of Elementary linear logic offers the advantage of simplicity and for instance its proof-nets and its geometry of interaction models are arguably easier to analyze than those of LL or light linear logic. In the present work we show how by considering various types in Elementary linear logic extended with type fixpoints we can characterize the classes P, EXP, and more generally the hierarchy of classes $k$-EXP for $k \geq 1$.

## 2 Characterization of the classes P and $k$-EXP

We consider intuitionistic affine elementary logic with type fixpoints (see [DLB06]). Actually for our purpose it is sufficient to consider its multiplicative fragment. The grammar of types is:

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

The rules of this system, $IEAL_\mu$, are given on Fig. 1. This system only differs from the intuitionistic version of elementary linear logic without additive connectives [Gir98, DJ03] by the fact that we have added the fixpoint construction (rules $L_\mu$ and $R_\mu$) and allowed for general weakening (rule (Weak)).

Let us denote the following types respectively for booleans, $n$-ary finite types, tally integers and binary

*This work was partially supported by project ANR-08-BLANC-0211-01 "COMPLICE".

**Axiom and Cut**.

$$\frac{}{A \vdash A}\ Ax \qquad \frac{\Gamma \vdash A \quad \Delta, A \vdash B}{\Gamma, \Delta \vdash B}\ Cut$$

**Structural Rules**.

$$\frac{\Gamma \vdash A}{\Gamma, B \vdash A}\ Weak \qquad \frac{\Gamma, !A, !A \vdash B}{\Gamma, !A \vdash B}\ Contr$$

**Multiplicative Rules**.

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

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

**Exponential Logical Rule**.

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

**Second Order Rules**

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

**Fixpoint Rules**

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

Figure 1: The system $IEAL_\mu$

words:

$$\begin{array}{rcl} B & = & \forall\alpha.\alpha \multimap \alpha \multimap \alpha \\ B^n & = & \forall\alpha.\alpha \multimap \ldots \multimap \alpha,\ \text{with } n+1 \text{ occurrences of } \alpha \\ N & = & \forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \\ W & = & \forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \end{array}$$

We denote: $2^n_0 = n$, $2^n_{k+1} = 2^{2^n_k}$. Recall that elementary functions are the functions computable on a Turing machine in time $O(2^n_k)$, for some $k$.

The standard results of elementary linear logic [DJ03] still hold in the setting of $EAL_\mu$:

- for any integer $d$, any proof of $N \multimap !^d N$ represents an elementary function (*complexity soundness*);
- for any elementary function $f : \mathbb{N} \to \mathbb{N}$ there exists an integer $d$ and a proof of $N \multimap !^d N$ representing it (*extensional completeness*).

Now, by considering alternative types we can delineate more complexity classes:

**Theorem 1** *We consider the system $EAL_\mu$.*

- *The functions representable by proofs of $!W \multimap !^2 B$ (resp. $!W \multimap !^3 B$) are exactly the class P (resp. EXP);*
- *More generally, for any $k \geq 0$, the functions representable by proofs of $!W \multimap !^{k+2} B$ are exactly the class k-EXP.*

where:

$$\begin{array}{rcl} k\text{-EXP} & = & \cup_{i\in\mathbb{N}} DTIME(2^{n^i}_k), \\ \text{EXP} & = & \text{1-EXP}. \end{array}$$

**Remark 1** *Note that we do not use fixpoints in the final types involved. However, technically speaking the fixpoints are used in the proofs of completeness, in order to simulate polynomial time (resp. k-exponential time) Turing machines, as we will see in Sect. 3.2.*

## 3 Proof sketch

We outline here the proof of the first claim of Theorem 1, showing that $!W \multimap !^2 B$ characterizes P. The characterizations of the other classes can then be obtained in a similar way.

### 3.1 Complexity soundness

In order to prove the complexity bound we study the cut elimination process and take advantage of the assumption that the conclusion of the proof is of type $!W \multimap !^2 B$ in order to derive a sharper bound. For that, as usual in linear logic it is convenient to use *proof-nets* to analyse cut elimination as proof-net reduction.

We do not recall in this extended abstract the definition of elementary linear logic proof-nets and their reduction steps but refer the reader to *e.g.* [Bai08]. The fixpoint rules $L_\mu$ and $R_\mu$ can be handled by two new nodes $\mu$ and $\overline{\mu}$ and with a reduction rule similar to that of multiplicative nodes (see [BCDL11]).

Now, given a proof-net $R$, the depth of a node is the number of exponential boxes containing it, and the depth $d(R)$ of $R$ is the maximal depth of its nodes. We denote by $|R|_i$ the number of nodes at depth $i$ and by $|R|$ the total number of nodes.

Now we can state the key Lemma:

**Lemma 2 (Size bound)** *Let $R$ be a proof-net with:*

- *only exponential cuts at depth 0,*
- *$k$ cuts at depth 0.*

*Let $R'$ be the proof-net obtained by reducing $R$ at depth 0. Then we have:*

$$|R'|_1 \leq |R|_0^k . |R|_1 .$$

So if we have a bound on the number $k$ of cuts we obtain a polynomial bound on the size of the proof-net $R'$ after reduction at depth 0. In any case we can bound $k$ by $|R|_0$, but then we basically recover the usual exponential bound [Gir98].

We will need another result:

**Lemma 3 (Readback)** *Let $R$ be a proof-net of conclusion $!B$ with:*

- *no cut at depth 0,*
- *only exponential cuts at depth 1.*

*Given $R$, one can in constant time decide whether it reduces to true or false.*

Finally we get:

**Proposition 4 (PTIME soundness)** *Let $R$ be a normal proof-net of conclusion $!W \vdash !^2B$. Then there exists a polynomial $P$ such that:*

*any proof-net obtained by cutting $R$ with a proof-net representing a word of length $n$ can be evaluated in a number of steps bounded by $P(n)$.*

**Proof :**[sketch] Reduce cuts at depths 0, 1 and non-exponential cuts at depth 2. One can check that this can be done in a polynomial number of steps by using Lemma 2. Then extract the result using the readback Lemma 3. □

## 3.2 Extensional completeness

The following simulation of polynomial time Turing machines is quite standard. The only trick is on the choice of the type to represent the configurations.

Any polynomial on one variable can be represented in $EAL_\mu$ (or $EAL$) with a proof of $!N \multimap !N$. Moreover we have a proof $length : W \multimap N$.

Using type fixpoints we can define the following type $W_S$ for Scott binary words [DLB06, BT10, RV10] and $Configs_S$ for the configurations of a one-tape Turing over a binary alphabet:

$$\begin{aligned} W_S &= \mu\beta.\forall\alpha.(\beta \multimap \alpha) \multimap (\beta \multimap \alpha) \multimap (\alpha \multimap \alpha) \\ Configs_S &= W_S \otimes W_S \otimes B^n . \end{aligned}$$

We also have proofs respectively for computing the initial configuration from a word, and for testing whether a configuration is accepting :

$$\begin{aligned} init :& \quad W \vdash Configs_S \\ accept? :& \quad Configs_S \vdash B \end{aligned}$$

We consider a Turing Machine $\mathcal{M}$ with running time bounded by a polynomial $q$. One can give a proof representing its step function:

$$step : Configs_S \vdash Configs_S .$$

Then we can simulate the execution of $\mathcal{M}$ in the following way:

iterate *step* $q(|w|)$ times on input $(init(w))$ so as to get:

$$!W \vdash !^2 Configs_S .$$

Composing this proof with *accept*? we get:

$$!W \vdash !^2 B .$$

**Remark 2** *Without type fixpoints we can define using second-order a type $Config$ based on Church integers (following [AR02]). However the problem is then that the corresponding term accept? has type $Config \vdash !B$. So we obtain for the simulation in the end the type $!W \vdash !^3 B$, which is not what we need ...*

**Acknowledgements** . The author would like to thank Jean-Yves Girard whose initial question, whether P could be characterized in elementary linear logic, triggered this work.

## References

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

[Bai08] P. Baillot. *Linear Logic, Types and Implicit Computational Complexity.* PhD thesis, Université Paris 13, March 2008. Habilitation à diriger les recherches. Available from the author's webpage.

[BCDL11] P. Baillot, P. Coppola, and U. Dal Lago. Light Logics and Optimal Reduction: Completeness and Complexity. *Information and Computation*, 209:118–142, 2011.

[BT10] Aloïs Brunel and Kazushige Terui. Church => scott = ptime: an application of resource sensitive realizability. In *Proceedings International Workshop on Developments in Implicit Computational complExity (DICE 2010)*, volume 23 of *EPTCS*, pages 31–46, 2010.

[DJ03] Vincent Danos and Jean-Baptiste Joinet. Linear logic & elementary time. *Information and Compution*, 183:123–137, 2003.

[DLB06] Ugo Dal Lago and Patrick Baillot. Light affine logic, uniform encodings and polynomial time. *Mathematical Structures in Computer Science*, 16(4):713–733, 2006.

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

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

[GMR08] Marco Gaboardi, Jean-Yves Marion, and Simona Ronchi Della Rocca. A logical account of pspace. In *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2008)*. ACM, 2008.

[Jon01] Neil D. Jones. The expressive power of higher-order types or, life without cons. *J. Funct. Program.*, 11(1):5–94, 2001.

[Laf04] Yves Lafont. Soft linear logic and polynomial time. *Theoret. Comput. Sci.*, 318(1–2):163–180, 2004.

[RV10] Luca Roversi and Luca Vercelli. Safe Recursion on Notation into a Light Logic by Levels. In *Proceedings of International Workshop on Developments in Implicit Computational complexity (DICE 2010)*, volume 23 of *EPTCS*, pages 63 – 77, 2010.