On the expressivity of elementary linear logic: 
characterizing Ptime and 
an exponential time hierarchy

Patrick Baillot
LIP (UMR 5668 CNRS-ENS Lyon-INRIA-UCBL)
patrick.baillot@ens-lyon.fr

Abstract
Elementary linear logic is a simple variant of linear logic due to Girard 
and which characterizes in the proofs-as-programs approach the class of 
elementary functions, that is to say functions computable in time bounded 
by a tower of exponentials of fixed height. Other systems like light and 
soft linear logics have then been defined to characterize in a similar way 
the more interesting complexity class of polynomial time functions, but 
at the price of either a more complicated syntax or of more sophisticated 
encodings. These logical systems can serve as the basis of type systems 
for $\lambda$-calculus ensuring polynomial time complexity bounds on well-typed 
terms.

This paper aims at reviving interest in elementary linear logic by show-
ing that, despite its simplicity, it can capture smaller complexity classes 
than that of elementary functions. For that we carry a detailed analysis of 
its normalization procedure, and study the complexity of functions 
represented by a given type. We then show that by considering a slight 
variant of this system, with type fixpoints and free weakening (elementary 
affine logic with fixpoints) we can characterize the complexity of functions 
of type $!W \rightarrow !^{k+2}B$, where $W$ and $B$ are respectively types for binary 
words and booleans. The key point is a sharper study of the normaliza-
tion bounds. We characterize in this way the class $P$ of polynomial time 
predicates, and more generally the hierarchy of classes $k$-EXP, for $k \geq 0$, 
where $k$-EXP is the union of $\text{DTIME}(2^{n^k})$, for $i \geq 1$.

1 Introduction

Implicit computational complexity. This line of research promotes investiga-
tions to characterize classical complexity classes by programming languages 
or logics, without referring to explicit bounds on resources (time, space . . . ) but 
instead by restricting the primitives or the features of the languages. Various 
approaches have been used for that, primarily in logic and in functional pro-
gramming languages: restrictions of the comprehension scheme in second-order logic
[Lei94a, Lei02]; ramification in logic or in recursion theory [Lei94b, BC92]; read-only functional programs [Jon01]; variants of linear logic [Gir98, Laf04]...to name only a few.

Note that the programming disciplines induced by these systems are quite restrictive, but some of these characterizations have in a second step led to more flexible criteria for statically checking complexity bounds on programs: ramification and safe recursion have inspired the work on interpretation methods for complexity [BMM11] on the one-hand, and linear type systems for non-size-increasing computation [Hof03] on the other, which itself has led to typing methods for amortized complexity analysis [HJ06, HAH11].

**Linear logic.** The linear logic approach to implicit complexity fits in the proofs-as-programs paradigm. It stems from the observation that as duplication is controlled in linear logic by the modality !, weaker versions of this modality can define systems with a complexity-bounded normalization procedure: elementary linear logic (ELL) [Gir98, DJ03] characterizes in this way the class of Kalmar elementary functions (computable in time bounded by a tower of exponentials of fixed height), while light linear logic (LLL) [Gir98] and soft linear logic (SLL) [Laf04] characterize functions computable in polynomial time. These logical systems have then enabled the design of type systems for \( \lambda \)-calculus or functional languages ensuring that a well-typed program has a polynomial time complexity bound [BT09, GR07, BGM10].

Note that initially ELL did not seem as interesting as LLL and SLL since it corresponds to elementary complexity, which is not very relevant from a programming or a complexity theory point of view. However it has nice logical properties, a simpler language of formulas than LLL (no \( \boxdot \) modality) and allows for a more natural programming style than SLL. It has also been studied for its remarkable properties concerning \( \lambda \)-calculus optimal reduction [ACM04].

**Calibrating complexity.** One might regret a lack of homogeneity in the characterizations we have cited of complexity classes by variants of linear logic: indeed some common deterministic complexity classes like \( \text{EXP} \) have not yet been characterized, a different system is provided for each complexity class, and these various systems are not easy to compare.

By contrast, other methods in implicit complexity have defined frameworks that can be calibrated to delineate different complexity classes inside the (large) Kalmar elementary class:

- Jones considers in [Jon01] a read-only functional programming language and characterizes in it the classes \( k\text{-EXP} = \cup_k \text{DTIME}(2^{2^{2^k}}) \), for \( k \geq 0 \), by considering, for each \( k \), programs using only arguments of type-order at most \( k \);

- Leivant investigates in [Lei02] second-order logic with comprehension (quantifier elimination) restricted to various families of first-order formulas: the functions provably total in this logic with comprehension restricted to formulas of type-order at most \( k \) are precisely the functions of \( k\text{-EXP} \).

Note that even if these two frameworks fit in different computational approaches,
they both use as parameter the type-order (or implicational rank) of formulas.

**Contributions and overview.** A goal of the present work is to provide an analogous framework in linear logic, allowing to characterize in a single logic a hierarchy of complexity classes by calibrating a certain parameter. We will use elementary linear logic, which offers the advantage of simplicity. A key parameter in this system, as in LLL, is the number of nested modalities (!) and this will be the value calibrating our complexity bounds.

For technical reasons we will actually consider an extension of ELL obtained by adding to it type fixpoints. It had been observed from the beginning [Gir98] that this feature does not modify the dynamics of ELL and its complexity bounds. We will also allow unrestricted weakening, which is innocuous and common practice since [AR02]. The new system will be denoted $EAL_\mu$ (elementary affine logic with fixpoints). As is common in linear logic we will study normalization in the setting of *proof-nets*, a graphical representation of proofs.

In a first part of the paper (Sect. 3) we will actually revisit the study of complexity bounds for normalization in elementary linear logic sketched initially in [Gir98] but we will take here a special care on the explicit bounds. Given some types $W$ for binary words and $B$ for booleans, we will consider types of the form $W \to !^k W$ and $W \to !^k B$, where $!^k$ stands for a sequence of $k$ !s. Our detailed analysis of normalization will allow us to bound the complexity of functions represented by a given type $W \to !^k W$.

While [Gir98, DJ03] had characterized the complexity of functions represented by the union of types $\bigcup_k (N \to !^k N)$, where $N$ stands for unary integers, we will focus in the second part of the paper (Sect. 4) on characterizing the complexity of predicates represented by one type $!W \to !^k B$, where $k$ is fixed. For that we will improve both the argument for the normalization analysis, in order to derive a sharper bound, and the simulation of time-bounded Turing machines. This will lead to our main result: the predicates of type $!W \to !^{k+2} B$ for $k \geq 0$ in $EAL_\mu$ exactly correspond to the complexity class $k$-EXP, so in particular $P$ for $k = 0$ and $EXP$ for $k = 1$.

Note that a distinctive point of our characterization is that it does not rely on a restriction of a particular operation inside the proof-program, like the application of a function to an argument in [Jon01] or the comprehension rule in [Lei02]. Instead it only imposes a condition on the conclusion of the proof (or type of the program), that its to say on its interface. In this sense it is more modular than these previous characterizations. Observe also that our system is a second-order logic, as the one of [Lei02], but here comprehension is not restricted.

A preliminary short version [Bai11] of this work appeared in the proceedings of the 9th Asian Symposium on Programming Languages and Systems (APLAS’11). With respect to this conference version, the present paper adds the following aspects:

- a detailed account of the complexity bounds known for elementary logic prior to this paper (Sect. 3), which makes it more self-contained and the comparison with previous works clearer,
• complete proofs of all statements (some of them had been omitted in the conference version because of space constraints).

2 Definition of the logical system and of the complexity classes

We consider intuitionistic affine elementary logic with type fixpoints that we denote by \( EAL_\mu \). Actually for our purpose it is sufficient to consider its multiplicative fragment. The grammar of types is:

\[
A ::= \alpha | A \multimap A | !A | \forall \alpha.A | \mu \alpha.A
\]

Elementary affine logic \( EAL \) is obtained by considering the grammar without the \( \mu \alpha.A \) construction. We will write \( !^nA \) with \( n \in \mathbb{N} \) for \( !\ldots!A \) with \( n \) 's.

We will represent functions by proofs, but as often it will be convenient to use \( \lambda \)-calculus to denote the algorithmic content of proofs. For that we will consider an extension of \( \lambda \)-calculus with a \( \otimes \) construction:

\[
t, u ::= x | \lambda x.t | (t u) | t \otimes u | \text{let } t \text{ be } x \otimes y \text{ in } u
\]

Its reduction rule is obtained by the context-closure of the usual \( \beta \)-reduction rule and of the following one:

\[
\text{let } t_1 \otimes t_2 \text{ be } x \otimes y \text{ in } u \rightarrow u[t_1/x, t_2/y].
\]

The rules of \( EAL_\mu \) are now given on Fig. 1, as a sequent calculus decorated with \( \lambda \)-terms. 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)). Observe that some rules do not have any effect on the term \( (!, L_\mu, R_\mu, L_\forall, R_\forall) \); this is because we want to keep the term calculus as simple as possible, and the current calculus is anyway sufficient to represent the functions denoted by the proofs.

Observe that the formulas \( !A \multimap A \) and \( !A \multimap !A \) are not provable, which is the distinctive feature of elementary linear logic with respect to ordinary linear logic.

Finally, note that if we added the fixpoint rules to intuitionistic logic or linear logic cut elimination would not be normalizing anymore, but strong normalization does hold for \( EAL_\mu \) (see [Gir98]).

Let us denote the following types respectively for booleans, \( n \)-ary finite types, tally integers and binary words, which are adapted from system F:

\[
\begin{align*}
B &= \forall \alpha. \alpha \multimap \alpha \\
B^n &= \forall \alpha. \alpha \multimap \ldots \multimap \alpha, \text{ with } n + 1 \text{ occurrences of } \alpha \\
N &= \forall \alpha. (! (\alpha \multimap \alpha) \multimap ! (\alpha \multimap \alpha)) \\
W &= \forall \alpha. (! (\alpha \multimap \alpha) \multimap ! (\alpha \multimap \alpha) \multimap ! (\alpha \multimap \alpha)).
\end{align*}
\]
Axiom and Cut.

\[
\frac{x : A \vdash x : A}{Ax} \quad \frac{\Gamma \vdash t : A \quad \Delta, x : A \vdash u : B}{\Gamma, \Delta \vdash u[t/x] : B} \quad \text{Cut}
\]

Structural Rules.

\[
\frac{\Gamma \vdash t : A}{\Gamma, x : B \vdash t : A} \quad \text{Weak} \quad \frac{\Gamma, x_1 : !A, x_2 : !A \vdash t : B}{\Gamma, x : !A \vdash t[x/x_1, x/x_2] : B} \quad \text{Contr}
\]

Multiplicative Rules.

\[
\frac{\Gamma \vdash t : A \quad \Delta, x : B \vdash u : C}{\Gamma, \Delta, y : A \rightarrow B \vdash u[(y t)/x] : C} \quad L_{\rightarrow} \quad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x. t : A \rightarrow B} \quad R_{\rightarrow}
\]

\[
\frac{\Gamma, x_1 : A, x_2 : B \vdash t : C}{\Gamma, x : A \otimes B \vdash \text{let } x \text{ be } x_1 \otimes x_2 \text{ in } t : C} \quad L_{\otimes} \quad \frac{\Gamma \vdash t_1 : A \quad \Delta \vdash t_2 : B}{\Gamma, \Delta \vdash t_1 \otimes t_2 : A \otimes B} \quad R_{\otimes}
\]

Exponential Logical Rule.

\[
\frac{\Gamma \vdash t : A}{\Gamma \vdash !t : !A} \quad !
\]

Second Order Rules

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

Fixpoint Rules

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

Figure 1: The system $EAL_{\mu}$
Recall that these data-types admit some coercions [Gir98]. For \( W \) for instance, one can give a proof of type \( W \twoheadrightarrow !W \) which, as a \( \lambda \)-term, acts as an identity on the terms encoding binary words.

There is also a term \( \text{length} : W \twoheadrightarrow N \) which returns the length of a word, as a tally integer.

In this paper we want to characterize for a given system (\( EAL \) or \( EAL_\mu \)) and a type \( A \) of this system, the class of functions representable by a closed term of type \( A \). We are only considering for these characterizations first-order types with arity 1, that is to say types of the form \(!^nD_1 \twoheadrightarrow !^mD_2\), where \( D_1 \) and \( D_2 \) are data-types. We denote by \( F_S(A) \) this class of functions, where \( S \) is \( EAL \) or \( EAL_\mu \). If we choose for \( D_2 \) the type \( B \) we will actually characterize a class of predicates. The conditions on the integers \( n \) and \( m \) will be important because they will be a way to constrain the class of functions.

We extend this notation to a set \( \mathcal{A} \) of types, by defining \( F_S(\mathcal{A}) = \bigcup_{A \in \mathcal{A}} F_S(A) \).

**Complexity classes.** If \( F : \mathbb{N} \to \mathbb{N} \) is a proper complexity function we denote by \( \text{FDTIME}(F(n)) \) (resp. \( \text{DTIME}(F(n)) \)) the class of predicates on binary words (resp. functions on binary words) computable on a deterministic Turing machine in time \( O(F(n)) \). We denote: \( 2^n_0 = n, 2^n_{k+1} = 2^{2^n_k} \).

In this paper we will use the following complexity classes:

\[
\begin{align*}
P & = \bigcup_{i \in \mathbb{N}} \text{DTIME}(n^i) \\
\text{EXP} & = \bigcup_{i \in \mathbb{N}} \text{DTIME}(2^{n^i}) \\
k-\text{EXP} & = \bigcup_{i \in \mathbb{N}} \text{DTIME}(2^{n^i}), \text{ for } k \geq 0 \\
k-\text{FEXP} & = \bigcup_{i \in \mathbb{N}} \text{FDTIME}(2^{n^i}), \text{ for } k \geq 0 \\
\text{FELEM} & = \bigcup_{k \in \mathbb{N}} k-\text{FEXP}.
\end{align*}
\]

The class \( \text{FELEM} \) is called class of elementary functions, and is more commonly seen as a class of functions on integers. Note that we have \( P = 0-\text{EXP} \) and \( \text{EXP} = 1-\text{EXP} \).

### 3 Elementary affine logic and elementary complexity

In this Section we recall the main result of elementary affine logic: this system characterizes elementary time complexity. In order to be able to compare in the next Sections the new complexity bounds we will provide to the state-of-the-art, we recall the arguments sketched in [Gir98] to establish this complexity soundness result (Prop. 5). This proof is actually part of the folklore in the linear logic community, but we nevertheless give it again here because:

- to our knowledge it does not appear in print in full detail,
- this allows us to state results on upper bounds and lower bounds of functions for each level of a hierarchy of types (Proposition 12), while previous results were only stated for the union of this hierarchy [Gir98, DJ03].
3.1 Complexity bounds on cut elimination

In order to study cut elimination it is convenient to use proof-nets. The proof-nets considered here will use formulas of classical elementary linear logic with fixpoints:

\[ A ::= \alpha | \alpha^\perp | A \otimes A | A^\phi A | !A | ?A | A^{\mu} A | \forall A | \exists A \]

The connectives (modalities) \( ! \) and \( ? \) are called exponentials and \( \otimes / \wp \) are multiplicatives.

Formulas of \( EAL_\mu \) are translated in this grammar by using \( A \rightarrow B \equiv A^\perp \wp B \) and the usual linear logic De Morgan laws for linear negation:

\[
(A \otimes B)^\perp \equiv A^\perp \wp B^\perp, \quad (!A)^\perp \equiv ?A^\perp, \\
(\mu A)^\perp \equiv \wp A, \quad (?A)^\perp \equiv \exists A, \quad A^\perp \perp \equiv A.
\]

In order to handle weakening we will use proof-nets with polarities, following \[AR02\]. Note that proof-nets with polarities had been considered before, e.g. in \[Lam96\], but here we will follow the conventions and notations of \[AR02\].

The nodes are described on Fig. 2:

- nodes have ports which are positive (dark bullet) or negative (white bullet); an edge can link together either two positive or two negative ports; we say that an edge is positive (resp. negative) if it is connected to a positive (resp. negative) port;
- as the system is affine, the proof-nets use, beside the weakening w-node, also an h-node;
- two nodes \( \mu \) and \( \overline{\mu} \) are added, corresponding resp. to the fixpoint rules \( R_\mu \) and \( L_\mu \);
- each node \( ! \) or \( ? \) comes with a !-box, as shown on Fig. 3; two boxes are either disjoint or one is included in the other; the !-node is called the principal door of the box and the ?-nodes are its auxiliary doors.

One will translate an \( EAL_\mu \) proof \( \pi \) of conclusion \( x_1 : A_1, \ldots, x_n : A_n \vdash t : B \) by a graph \( \pi^* \) with conclusions \( A_1^\perp, \ldots, A_n^\perp, B \) where the edges of the \( A_i^\perp \) (\( 1 \leq i \leq n \)) are negative and the edge of \( B \) is positive. This translation is standard \[Gir87\] and we only describe a couple of illustrative cases:

- if \( \pi \) is obtained by an \( Ax \) rule, then \( \pi^* \) is an ax-node;
- if \( \pi \) is obtained by a \( Cut \) rule between \( \pi_1 \) and \( \pi_2 \), then \( \pi^* \) is obtained by linking \( \pi_1^* \) and \( \pi_2^* \) by a cut-node;
- if \( \pi \) is obtained by a \( ! \) rule on \( \pi_1 \), then \( \pi^* \) is obtained by applying a !-box on \( \pi_1^* \) (see Fig. 3),

7
On Fig. 2 below each node we have indicated the sequent-calculus rule it corresponds to. Note that the $h$-node is not used for this translation; it will only appear during normalisation.

Now, a graph $R$ is called a **proof-net** if there exists a proof $\pi$ such that $R = \pi^*$.

A **cut-node** between an $ax$-node (resp. $w$-node) and another node (resp. another node which is not an $ax$-node) is called an **axiom cut** (resp. a **weakening cut**). A cut between a $\otimes$-node and a $\varphi$-node is called a **multiplicative cut**. A cut between a $!$-node and a $?\text{-node}$ or $?\text{-node}$ (resp. a $?\text{-node}$) is called an **exponential cut** (resp. a **contraction cut**). **Quantifier cuts** and $\mu$-cuts are defined in an analogous way.

In a proof-net, a maximal tree with $?\text{-nodes}$, $ax$-nodes and $w$-nodes (of type $\text{? } A \perp$) as leaves, and $?\text{-nodes}$ as internal nodes, is called an **exponential tree**.

Now, given $R$, 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. When there is no ambiguity we will sometimes write simply $d$ instead of $d(R)$.

The **size** of $R$ at depth $i$, denoted $|R|_i$, is defined as follows: it is the sum of the nodes at depth $i$, counted with weight 3 in the case of nodes $\otimes$, $\varphi$ and $?\text{-nodes}$ (contraction) and with weight 1 otherwise. This difference of weights is needed for technical reasons, but note that $|R|_i$ is anyway linearly related to the number of nodes at depth $i$. We then denote $|R|_i = \Sigma_{j=i}^{d(R)} |R|_j$ and $|R| = |R|_0 +$, which is called the **size** of $R$.

In the sequel, we will sometimes need to take into account the size of a proof-net representing a binary word. It is easy to check that:

**Lemma 1** There exists a constant $a$ such that for any binary word $w$ of length $n$, $w$ can be represented by a proof-net $R_w$ with a unique conclusion of type $W$ and of size $|R_w| \leq a \cdot n$.

Note that a word $w$ can be represented by several proof-nets, differing only by irrelevant differences such as the order of contractions.

**Reduction.** We can describe the reduction procedure on proof-nets, which consists in eliminating cuts. It is defined by the rules given on Fig. 4, 5, 6, 7:

- **Fig. 5** shows the reduction of exponential cuts (contraction step and box-box step);
- **Fig. 6** shows the reduction of weakening cuts; notice that one of these steps introduces $h$-nodes;
- **Fig. 7** shows the reduction of cuts on $h$-nodes.

Observe that during a reduction step the depth of an edge does not change, hence the depth of the proof-net does not increase. This is called the **stratification property** [DJ03] and it is a key ingredient for the complexity properties. It is not valid in ordinary linear logic, and it comes from the fact that during reduction a $!\text{-box}$ is not opened and does not enter another box (see Fig. 5).

We say that an exponential cut $c$ is a **special cut** if the box $B$ corresponding to the $!$ node does not have any cut below its auxiliary doors. We have:
Figure 2: Nodes of the proof-nets.
Figure 3: !-box.

Figure 4: Reduction steps (1/4).
Figure 5: Reduction steps (2/4).

**Lemma 2** If a proof-net $R$ has no cut at depth strictly inferior to $i$, and only exponential cuts at depth $i$, then it has at least one special cut at depth $i$.

A proof of this Lemma can be found e.g. in [BP01].

Now, the following fact can be easily verified by examining each reduction step other than the contraction reduction step:

**Lemma 3** Let $R$ be a proof-net and $R'$ be obtained from $R$ by reducing a cut at depth $i$ which is not a contraction cut. Then we have $|R'|_i < |R|_i$ and $|R'| < |R|$.

Thus, if there were no contraction reduction steps, we would have a linear bound on the number of reduction steps. Now, in order to bound the number of steps in presence of contraction we need to carry on a more detailed study. But before that let us state a technical lemma that will be useful afterwards.

**Lemma 4** If $k \geq 0$ and $x \geq 1$ then we have:

$$5^{2^k} \leq 2^{3^x_{k+1}}.$$

**Proof**: We prove by induction on $k$ that the property holds for any $x \geq 1$:

- If $k = 0$, then we do have $5^x \leq (2^3)^x$. 

11
Figure 6: Reduction steps: weakening steps (3/4).
Figure 7: Reduction steps: $h$-steps (4/4).
• Assume the statement holds for \( k \); then we have:

\[
5^{2k+1} = 5^{2k} \leq 2^{3 \cdot 2^k}, \text{ by induction hypothesis on } k,
\]

\[
\leq 2^{2^{2^k}} \cdot 2^k, \text{ because } 3 \leq 2^x,
\]

\[
\leq 2^{2^{2^k}} = 2^{3^2}.
\]

So the statement holds for all \( k \). \( \square \)

Now the property we want to establish is the following one:

**Proposition 5** Let \( R \) be an EAL\( _\mu \) proof-net of depth \( d \). Then \( R \) can be reduced in a number of steps which is bounded by \( 2(d + 1)2^{d|R|} \).

Moreover the size \( |S| \) of any intermediary proof-net \( S \) obtained during the reduction is bounded by \( (d + 1)2^{d|R|} \).

**Remark 1** This property was first stated in [Gir98], but the bound sketched was of the form \( O(2^{d|R|}) \), for some function \( \phi \). In [DJ03] a bound of the form \( O(2^{d|R|}) \) was given on the number of cut elimination steps. In [BP01] a bound \( O(2^{P(R)}) \) for some polynomial \( P \) was given, but the evaluation process considered was not cut elimination but execution based on geometry of interaction.

To prove the bound on reduction of Prop. 5 we will consider a specific reduction strategy, adapted from [Gir98]:

- reduce the cuts level-by-level, that is to say first at depth \( i \) (round \( i \)) for \( i \) successively equal to 0, 1, \ldots, \( d \);
- at a given depth \( i \), proceed in two phases:
  - phase (a): reduce all cuts but the exponential cuts,
  - phase (b): reduce the exponential cuts, by repeatedly reducing a special cut.

Thanks to Lemma 2 this is a valid reduction strategy.

Let us denote by \( R^i \) the proof-net at the beginning of round \( i \) and \( R^{d+1} \) or \( R' \) the final proof-net. In order to bound the number of steps of this reduction strategy, the proof proceeds by, for \( i = 0 \) to \( d \):

- bounding the number of steps of round \( i \) by using \( |R^i|_i \),
- bounding the size increase, that is to say bounding \( |R^{i+1}|_{(i+1)+} \) by using \( |R^i|_{i+} \).

In order to prove these bounds it is convenient to use the notion of active nodes. We say a node of \( R \) which is either a contraction node or an auxiliary door of a box is active if it is above a cut node.

**Lemma 6** At each reduction step of phase (b), the number of active nodes decreases strictly.
Proof: Consider a reduction step of phase (b). It reduces a special cut \( c \). First, if it is a cut between the principal door of a box and the auxiliary door of another box (second case of Fig. 5), then an active node is removed during this step and we are done. Otherwise, if it is a cut between the principal door of a box \( \mathcal{B} \) and a contraction node (first case of Fig. 5) then one contraction node is removed, but \( k \) are created, where \( k \) is the number of auxiliary doors of \( \mathcal{B} \). However, as the cut is special there are not cuts below the auxiliary doors of \( \mathcal{B} \), therefore the contraction nodes created are not active and the claim holds.

Lemma 7 If \( S \) is the proof-net at the beginning of phase (b), the number of steps of this phase is bounded by \( |S|_i \).

Proof: Observe that the number of active nodes at depth \( i \) in a proof-net \( R \) is inferior or equal to \( |R|_i \). By Lemma 6 we can thus conclude that the number of steps of phase (b) is bounded by \( |S|_i \).

Now we can proceed with the proof of Prop. 5:

Proof: Let us first remark the following points:

1. during phase (a): no new node is introduced at depth superior or equal to \( (i + 1) \), so \( |R|_{(i+1)+} \) does not increase.

2. during phase (b): at each step the size \( |R|_{(i+1)+} \) can be at most doubled (in the case of a contraction step).

Recall that \( R^i \) is the proof-net at the beginning of round \( i \). By Lemmas 3 and 7 the number of steps of phase (a) and phase (b) are both bounded by \( |R^i| \). So the number of steps of round \( i \) is bounded by \( 2|R^i| \).

Now what is the size \( |R^{i+1}|_{(i+1)+} \) at the end of phase (b)? By the remark 2 above and as there are at most \( |R^i| \) steps in phase (b) we have:

\[
|R^{i+1}|_{(i+1)+} \leq 2|R^i| \cdot |R^i|_{(i+1)+} \\
\leq 2|R^i| \cdot 2|R^i|_{(i+1)+}, \text{ because } x \leq 2x, \\
\leq 2|R^{i+1}|_{(i+1)+} = 2|R^i|_{i+}.
\]

So as \( |R^{(0)}|_{0+} = |R| \), by induction on \( i \) we obtain:

for \( i \leq d \), \( |R^i|_{i+} \leq 2^{i|R|} \).

From that we deduce:

for \( i \leq d \), \( |R^i|_i \leq |R^i|_{i+} \leq 2^{i|R|} \leq 2_d^{i|R|} \).

Therefore as there are \((d + 1)\) rounds, the total number of steps is bounded by \( 2(d + 1) \cdot 2_d^{i|R|} \).

Now we want to prove the bound on the size of any intermediary proof-net \( S \). Note that we have already obtained the bound \( |R^i|_{i+} \leq 2^{i|R|} \). However during
the phase (b) of round \((i-1)\) for \(i \geq 1\) the size at depth \((i-1)\) can also increase, even though the number of active nodes decreases (Lemma 6). This can be the case with a contraction reduction step. We will thus more generally bound \(|R^i|_j\) for any \(i, j\), by the following lemma:

**Lemma 8** If \(0 \leq i \leq d\) and \(0 \leq j \leq d\) then we have:

\[
|R^i|_j \leq 2^{3|R|}.
\]

Moreover:

\[
|R^{d+1}|_j \leq 2^3|R|.
\]

**Proof:** of Lemma 8

Let us first prove the first statement, by induction on \(i\). If \(i = 0\) then the property is trivial.

Assume \(i \geq 1\) and that the property holds for \((i-1)\), and let us prove it for \(i\). We have already proven that \(|R^i|_{i+} \leq 2^{3|R|}\). So for \(j \geq i\) we have \(|R^i|_j \leq 2^{3|R|}\).

Let us now consider \(j = (i-1)\) which, as mentioned above, is the important case. The only case that can make the size at depth \((i-1)\) increase during phase (b) of round \((i-1)\) is the contraction reduction step. If \(k\) is the number of auxiliary doors of the duplicated box, then the new nodes created are \(k\) contraction nodes (of weight 3), \(k\) auxiliary doors (of weight 1), one ! node and one cut node (of weight 1). If \(S, S'\) are respectively the proof-nets before and after this step, then one can check that we have \(|S'|_{(i-1)} \leq 5 \cdot |S|_{(i-1)}\).

Moreover as the number of steps of phase (b) is bounded by the number of active nodes (Lemma 6), which itself is inferior or equal to \(|R^i|_{(i-1)} - 1\) (because there is at least one node which is not an active one) we have:

\[
|R^i|_{(i-1)} \leq 5 |R^{i-1}|_{(i-1)} - 1 \cdot |R^{i-1}|_{(i-1)}
\]

\[
\leq 2^{3|R|} - 1
\]

\[
\leq 2^3 - 1
\]

\[
\leq 2^3, \text{ by Lemma 4.}
\]

Now if \(j < (i-1)\), as during round \((i-1)\) cuts are reduced at depth \((i-1)\) the proof-net is unchanged at depth \(j\), therefore we have \(|R^i|_j = |R^{i-1}|_j \leq 2^3|R|\) by induction hypothesis on \((i-1)\). So the first statement is proven.

As to the second statement, consider round \(d\): the reduction takes place at depth \(d\); if after performing phase (a) there is an exponential cut, then this exponential cut involves a box at depth \(d\), which contradicts the fact that \(d\) is the depth of the proof-net. Therefore after phase (a) the proof-net is in its normal form \(R^{d+1}\). As during phase (a) the size at depth \(d\) could only decrease we conclude that \(|R^{d+1}|_d \leq |R^d|_d\), so by the first statement applied to \(i = j = d\) we conclude that \(|R^{d+1}|_d \leq 2^3|R|\).

Now, from Lemma 8 we deduce that for any \(0 \leq i \leq d+1\), we have \(|R^i| \leq \sum_{j=0}^{d} |R^i|_j \leq (d+1) \cdot 2^3|R|\). Finally, if \(S\) is an intermediary proof-net obtained
during the reduction of $S$, then there exists $i$ such that $S$ occurs during round $i$. The bound $(d + 1) \cdot 2^{3|R|}$ we have computed for $|R^i|$ is actually the largest size that could be reached during round $i$, so we have $|S| \leq (d + 1)2^{3|R|}$. This ends the proof of Prop. 5.

\[ \square \]

### 3.2 Extensional expressivity results

**Lemma 9** Let $q$ be a polynomial over one variable, with coefficients in $\mathbb{N}$. We have:

1. There exists a proof of $\mathbf{!N} \rightarrow \mathbf{!N}$ representing the function $q(n)$;
2. For any $k \geq 1$, there exists a proof of $\mathbf{!N} \rightarrow \mathbf{!1}^{k+1} \mathbf{N}$ representing the function $2^q(n)$.

**Proof:**

1. Recall that addition and multiplication on tally integers can both be represented by proofs of conclusions $\mathbf{N} \rightarrow \mathbf{N} \rightarrow \mathbf{N}$. It follows that any polynomial can be represented by a proof of conclusion $\mathbf{!N} \rightarrow \mathbf{!N}$.

2. By using the proof for multiplication one defines a proof representing the doubling function, $\text{double}$, with type $\mathbf{N} \rightarrow \mathbf{N}$. By iterating this function, that is to say with the proof corresponding to the term $\text{exp} = \lambda n. (n \ \text{double} \ \mathbf{1})$, of type $\mathbf{N} \rightarrow \mathbf{!N}$, one represents the function $2^n$.

By composing $k$ times $\text{EXP}$ we obtain a term $\text{exp}_k$ of type $\mathbf{N} \rightarrow \mathbf{!1}^{k+1} \mathbf{N}$ representing the function $2^k$. Finally by composing the term $\text{exp}_k$ with the term of type $\mathbf{!N} \rightarrow \mathbf{!N}$ representing $q(n)$, one gets a term of type $\mathbf{!N} \rightarrow \mathbf{!1}^{k+1} \mathbf{N}$ representing the function $2^q(n)$.

\[ \square \]

In [AR02] a simulation of polynomial time Turing machines in light affine logic is given. By the translation from light affine logic to elementary affine logic (basically replacing the $\S$ modality by $\mathbf{!}$), these constructions can be used in EAL. Let us denote by $\textbf{Config}_C$ the following type for configurations for Turing machines over a binary alphabet with one tape and $n$ states:

$$\textbf{Config}_C = \forall \alpha.!(\alpha \rightarrow \alpha) \rightarrow !(\alpha \rightarrow \alpha) \rightarrow !(\alpha \rightarrow \alpha \rightarrow (\alpha \otimes \alpha \otimes B^n)),$$

where $B^n$ is for representing the state of the machine.
Lemma 10 Let $M$ be a one-tape deterministic Turing machine over a binary alphabet. One can define terms:

\[
\begin{align*}
\text{init}_C & : W \to \text{Config}_C, \\
\text{step}_C & : \text{Config}_C \to \text{Config}_C, \\
\text{accept}_?_C & : \text{Config}_C \to \,!B, \\
\text{extract}_C & : \text{Config}_C \to W,
\end{align*}
\]

such that:

- given a binary word, $\text{init}_C$ produces the corresponding initial configuration of the machine,
- the term $\text{step}_C$ computes one step of the machine on a given configuration,
- given a configuration, the term $\text{accept}_?_C$ returns true (resp. false) if its state is accepting (resp. rejecting),
- the term $\text{extract}_C$ is used if the machine is used for computing a word (and not just a boolean result) and returns the word written on the tape.

Proof: [Sketch] The terms $\text{init}_C$ and $\text{extract}_C$ are easy, while $\text{step}_C$ is not: see [AR02], where these terms are defined for light affine logic. As to $\text{accept}_?_C$, it is defined by a case distinction on the component of type $B^n$ of the configuration. Note the $!$ modality on its codomain, which is due to the fact that in $\text{Config}_C$, $B^n$ is in the scope of a $!$ and that EAL does not admit the dereliction principle.

Proposition 11 (Extensional expressivity) Let $f$ be a function of $k$-FEXP for $k \geq 0$, then there exists an EAL proof of $!W \to !^{k+2}W$ representing it.

If $g$ is a predicate of $k$-EXP for $k \geq 0$, then there exists an EAL proof of $!W \to !^{k+3}B$ representing it.

Proof: Let us first consider the case of the function $f$ of $k$-FEXP. Let $M$ be a one-tape deterministic machine over a binary alphabet, of time $2^{g(n)}$, computing $f$. By Lemma 9, there exists a proof of type $!N \to !^{k+1}N$ representing the function $2^{g(n)}$. We represent the configurations of $M$ with the type $\text{Config}_C$ and we know by Lemma 10 that there are:

- a proof $\text{step}_C$ of conclusion $\text{Config}_C \to \text{Config}_C$ representing one step of execution on a configuration,
- a proof $\text{init}_C$ of conclusion $W \vdash \text{Config}_C$ for producing the initial configuration,
- a proof $\text{extract}_C$ of conclusion $\text{Config}_C \vdash W$ for extracting the result.
Now, using $\text{step}_C$ and iteration on $N$, one can define a proof of conclusion $N, !\text{Config}_C \vdash !\text{Config}_C$, which given an integer $n$ and a configuration $c$, will produce the configuration $c'$ obtained by $n$ computation steps starting from $c$.

Using the representation of $2^q(n)$, we also obtain a proof $\pi$ of $!N, !^{k+2}\text{Config}_C \vdash !^{k+2}\text{Config}_C$, which given an integer $n$ and a configuration $c$, produces the configuration $c'$ obtained by $2^q(n)$ computation steps starting from $c$.

Now, the requested proof is obtained by composing the following constructions:

- use duplication $!W \to !W \otimes !W$, length and the coercion $!N \to !^kW$, to obtain $!W \to !N \otimes !^{k+2}W$,
- compose it with $id \otimes \text{init}_C$ to obtain $!W \to !N \otimes !^{k+2}\text{Config}_C$,
- compose it with the proof $\pi$ described above, to get $!W \to !^{k+2}\text{Config}_C$,
- finally compose with $\text{extract}_C$ to obtain a proof of $!W \to !^{k+2}W$.

The resulting proof represents the computation of the machine $M$, hence the function $f$. For the case of the predicate $g$ of $k$-EXP we proceed in exactly the same way, but in the end instead of $\text{extract}_C$ of type $\text{Config}_C \to W$ we use $\text{accept}_C$ of type $\text{Config}_C \to !B$ and we thus obtain a proof of $!W \to !^{k+2}B$.

\[ \square \]

3.3 Complexity properties

**Proposition 12** We have:

\[ \mathcal{F}_{EAL}(!W \to !^kW) \subseteq (k + 1)\text{-EXP}, \]

for $k \geq 2$, \[ \mathcal{F}_{EAL}(!W \to !^kW) \supseteq (k - 2)\text{-EXP}, \]

and \[ \mathcal{F}_{EAL}(!W \to !^kB) \subseteq k\text{-EXP}, \]

for $k \geq 3$, \[ \mathcal{F}_{EAL}(!W \to !^kB) \supseteq (k - 3)\text{-EXP}. \]

These results also hold if we replace the types $!W \to !^kW$ (resp. $!W \to !^kB$) by $W \to !^kW$ (resp. $W \to !^kB$).

We used the types $!W \to !^kW$ and $!W \to !^kB$ in this Proposition in order to make easier the comparison with the improved results in the next Section (Theorem 14).

Observe the gap between the upper bound and the lower bound obtained both in the characterization of predicates and of functions. It does not allow for an exact characterization of the functions at a given level of the type hierarchy, even though it is sufficient for characterizing the class of functions of the full type hierarchy (see Prop. 13). In the next Section we will investigate how to refine these results in order to characterize exactly the functions at each type. This will be obtained by considering predicates in the system $EAL_\mu$ and by improving both the result on the upper bound and that on the lower bound.
Proof: [of Prop. 12]

The second and fourth inclusions (right-to-left inclusions) are consequences of Proposition 11.

Let us prove the first inclusion. W.l.o.g. we can consider a proof-net $R$ with two conclusions $\top W \vdash \bot W$. Call $f$ the function that it represents. We want to prove that $f$ belongs to $(k + 1)\text{-}\text{FEXP}$.

By Lemma 1 there exists $a$ such that any binary word $w$ of length $n$ can be represented by a proof-net $R_w$ of size $|R_w| \leq a \cdot n$. Take such a $R_w$, apply a box to it to obtain a proof-net of conclusion $\bot W$ and cut it with $R$; call $T$ the resulting proof-net of conclusion $\bot W$. As any proof-net of conclusion $W$ and without any cut at depths 0, 1 represents a word, it is sufficient in order to obtain the result $f(w)$ of the computation to reduce the cuts in $T$ from depth 0 to depth $(k + 1)$. Now, by the same method establishing Prop. 5 we get that: the reduction of $T$ up to depth $(k + 1)$ can be done in a number of steps bounded by $2(k + 2)2^{|T|}_{k+1}$ and with intermediate proof-nets of sizes bounded by $(k + 2)2^{|T|}_{k+1}$.

By definition of $T$ we know that there exists a constant $b$ such that $|T| \leq a \cdot n + b$. Therefore the reduction of $T$ up to depth $(k + 2)$ is done:

1. in a number of steps $O(2^{a\cdot n + b})$,
2. with intermediate proof-nets of sizes $O(2^{3a\cdot n+3b})$.

What is the time needed to simulate this reduction on a Turing machine? Obviously a cut elimination step in a proof-net cannot in general be simulated on a Turing machine in constant time. However, one can reasonably encode the proof-nets in such a way that there exists a polynomial $Q$ such that: any reduction step on a proof-net $R$ is simulated on the Turing machine in time $Q(|R|)$. Note that this accounts in particular for the necessity in the contraction reduction step to copy a part of the proof-net.

For the reduction of $T$ we have bounds $O(2^{|P_1(n)|}_{k+1})$ and $O(2^{|P_2(n)|}_{k+1})$ for the number of steps and the sizes of intermediate proof-nets, where $P_1$ and $P_2$ are polynomials. Therefore this reduction can be simulated in time bounded by $O(2^{|P_1(n)|}_{k+1} \cdot Q(2^{|P_2(n)|}_{k+1}))$, hence in time $O(2^{R(n)}_{k+1})$ for some polynomial $R$. So this shows that $f$ belongs to $(k + 1)\text{-}\text{FEXP}$.

As to the third inclusion, it is handled in a similar way as the first inclusion, except that in a proof-net of conclusion $\bot W \vdash \bot W$. It is sufficient to reduce the cuts up to depth $k$ in order to obtain the result. We thus get in the end a time bound of the form $O(2^{R(n)}_{k+1})$, and the predicate represented therefore belongs to $k\text{-}\text{FEXP}$.

Now, let us consider the last claim of the proposition, concerning the types $W \rightarrow \bot W$ and $W \rightarrow \bot$. The same proofs we used for the first and third inclusions actually also show that $\mathcal{F}_{\text{EAL}}(W \rightarrow \bot W) \subseteq (k + 1)\text{-}\text{FEXP}$ and $\mathcal{F}_{\text{EAL}}(W \rightarrow \bot) \subseteq k\text{-}\text{FEXP}$. As to the other inclusions just observe that if we have a proof of $\bot W \vdash \bot W$ (resp. $\bot W \vdash \bot$) representing a function $f$ then by cutting it with the coercion proof of conclusion $W \vdash \bot W$ we obtain a proof of $\bot W \vdash \bot W$ (resp. $\bot W \vdash \bot$) also representing $f$. \qed
Proposition 13 We have:

\[ \mathcal{F}_{EAL}(\bigcup_k (W \rightarrow !^k W)) = \text{FELEM}. \]

Actually it can be shown in a similar way that:

\[ \mathcal{F}_{EAL}(\bigcup_k (N \rightarrow !^k N)) = \text{FELEM}. \]

The latter statement is the result stated in [Gir98] and proved in [DJ03]. \textbf{Proof} (of Proposition 13)

By Proposition 12 we have:

\[ \bigcup_k \mathcal{F}_{EAL}(W \rightarrow !^k W) = \bigcup_k \text{k-FEXP} = \text{FELEM}. \]

\square

4 Characterization of the classes P and k-EXP

Now we can state the main result of this paper:

\textbf{Theorem 14} We consider the system \( EAL_\mu \). We have:

\[ \mathcal{F}_{EAL_\mu}(!W \rightarrow !^2 B) = P, \]
\[ \mathcal{F}_{EAL_\mu}(!W \rightarrow !^3 B) = \text{EXP}. \]

More generally, for any \( k \geq 0 \) we have:

\[ \mathcal{F}_{EAL_\mu}(!W \rightarrow !^{k+2} B) = \text{k-EXP}. \]

\textbf{Remark 2} 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. 4.2.

\textbf{Remark 3} With the type \( !W \rightarrow !B \) one characterizes only the constant functions.

Observe that as in [Jon01] we are characterizing here predicates and not general functions. We will come back to this point later, in Remark 5.

Note that with respect to Prop. 5, the improvement of Theorem 14 is of two degrees of exponentials for the upperbound (bounding \( \mathcal{F}_{EAL_\mu}(!W \rightarrow !^{k+2} B) \) by \( \text{k-EXP} \) instead of \( (k + 2)\text{-EXP} \)) \textit{and} of one degree of exponentials for the lower bound (\( \text{k-EXP} \) instead of \( (k - 1)\text{-EXP} \)). To obtain these results we will adapt essentially the same reduction strategy and methodology seen in the previous section, with the following modifications:

\begin{itemize}
  \item on the strategy: we will not perform the reduction until obtaining a normal form (proof-net without cut), but we will stop when we can extract the result;
\end{itemize}
on the methodology for obtaining the upper bound: we will use the assumption that the proof has a conclusion $\mathbf{W} \rightarrow !^{k+2} \mathbf{B}$ and we will make a finer analysis of the size increase;

on the methodology for obtaining the upper bound: we will take advantage of the fact that we are considering $EAL_\mu$ to use other data-types and improve in this way the simulation of time-bounded Turing machines.

4.1 Proof of the complexity soundness results

Let us state the key Lemma that we will use:

**Lemma 15 (Size bound)** Let $R$ be a proof-net with:

• only exponential and weakening 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:

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

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 same kind of bound as in Section 3.

**Proof:** We denote by $|R|_a$ the number of active nodes of $R$ at depth 0. Observe that we have $|R|_a + 1 \leq |R|_0$.

We will prove the following statement by induction on $k$:

$$|R'|_1 \leq (|R|_a + 1)^k \cdot |R|_1.$$

If $k = 0$ the result is trivial. Assume now the result valid for $k$ and consider $R$ with $k + 1$ exponential cuts at depth 0.

First let us consider the case where there is a weakening cut among these $k + 1$ cuts. We then reduce it persistently, following the weakening steps and $h$-steps of Fig. 6-7, and we end up with a proof-net $R'$ such that: $|R'|_a \leq |R|_a$, $|R'|_1 \leq |R|_1$ and $R'$ has $k$ cuts at depth 0. We can then apply the induction hypothesis to $R'$ and easily conclude.

Now, consider the case where we have $k + 1$ exponential cuts. By Lemma 2, $R$ admits a special cut $c$. We completely reduce the cut $c$, that is to say we reduce $c$ and hereditarily all the cuts of its exponential tree until performing box-box or axiom reduction steps. The increase of size at depth 1 is due to the duplications of the box $\mathbf{B}$, which is copied at most $|R|_a$ times. Note that no active node is created during these reduction steps, because $c$ is a special cut. We obtain in this way a proof-net $R'$ such that: $|R'|_1 \leq (|R|_a + 1) \cdot |R|_1$, $|R'|_a \leq |R|_a$ and $R'$ has $k$ cuts at depth 0.
Besides, by induction hypothesis we have that $R'$ can be reduced to $R''$ which is normal at depth 0 and:

$$|R''|_1 \leq (|R'|_a + 1)^k \cdot |R'|_1.$$  

Combining the inequalities we thus get:

$$|R''|_1 \leq (|R|_a + 1)^{k+1} \cdot |R|_1.$$  

We conclude by using the fact that $|R|_a + 1 \leq |R|_0$. 

We will need another result:

**Lemma 16 (Readback)** Let $R$ be a proof-net of conclusion $B$ which only has exponential cuts at depth 0.

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

**Proof**: We have $B = \forall \alpha. \alpha \rightarrow \alpha \rightarrow \alpha$. Consider the conclusion $B$ and the axiom $\alpha \perp, \alpha$ introducing the r.h.s. literal $\alpha$ of this $B$. This axiom node $N$ is at depth 0.

Consider the second conclusion $\alpha \perp$ of this axiom. Let us assume for a contradiction that it is above a cut formula $A$ and call $c'$ the corresponding cut. Now, if $A$ was of the form $!A'$ (resp. $?A'$) for some $A'$, then the formula $A$ would need to be introduced by a $!$-node (respectively by an auxiliary door of a box), and so $N$ would be included in a box, which would contradict the fact that it is at depth 0. Therefore $A$ is not of the form $!A'$ or $?A'$. So $c$ is is not an exponential cut, hence a contradiction.

So the second conclusion $\alpha \perp$ of the axiom is not above a cut; hence it must be above a conclusion. The only conclusion of the proof-net is $B$, and thus this $\alpha \perp$ literal is one of the two occurrences of $\alpha \perp$ in $\forall \alpha.(\alpha \perp \varphi \alpha \perp \varphi \alpha)$. If it is the leftmost (resp. rightmost) occurrence of $\alpha \perp$ then the result of the reduction will be true (resp. false). So we do not need to reduce $R$ to know the resulting value; this can be done in constant time. 

Finally we get:

**Proposition 17 (P soundness)** Let $R$ be a normal proof-net of conclusion $!W \vdash !B$. 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 time bounded by $P(n)$.

**Proof**: 

First, recall that by Lemma 1 for any binary word $w$ of length $n$, $w$ can be represented by a proof-net $R_w$ of size $|R_w| \leq a \cdot n$.

Now, let us examine the structure of $R$ at depth 0. If $?W \perp$ is obtained by a weakening it is trivial. Otherwise there is an integer $k \geq 1$ and a proof-net $S$ of conclusion $\vdash W \perp, \ldots, W \perp, !B$ with $k$ formulas $W \perp$ such that: $R$ is obtained from $S$ by applying a $!$-box and a certain number $k'$ of contraction rules on $?W \perp$ formulas.

Now let $T_w$ be a proof-net representing a word $w$, and let $T$ be the proof-net obtained by cutting $R$ with a box enclosing $R_w$. The proof-net $T$ can be
reduced in at most $2k^i$ steps (at depth 0) into a proof-net $T'$ consisting in a box containing $S$ cut with $k$ copies of $R_w$. Therefore:

$$|T'| \leq |S| + k \cdot |R_w| + k \leq |R| + k \cdot |R_w| + k.$$ 

Then, since $W = \forall \alpha. !(\alpha \rightarrow \alpha) \rightarrow !(\alpha \rightarrow \alpha) \rightarrow !(\alpha \rightarrow \alpha)$, by applying $k$ quantification reduction steps and $2k$ multiplicative reduction steps (at depth 1) we get a proof-net $T''$ with not cut at depth 0 and only exponential and weakening cuts at depth 1. Note that there are at most $3k$ exponential cuts at depth 1 and that $|T''| \leq |T'| \leq |R| + k \cdot |R_w| + k$.

Now by applying Lemma 15 to $T''$ at depth 1, we get that by reducing $T''$ at depth 1 we obtain $T^{(3)}$ with no cut at depths 0, 1 and such that $|T^{(3)}|_2 \leq |T''|_1 \cdot |T''|_2 \leq (|R| + k \cdot |R_w|)^{3k+1} + k$. The important point to notice here is that $(3k+1)$ does not depend on $n$.

Finally we perform on $T^{(3)}$, at depth 2, reduction of all cuts but exponential cuts. These only make the size decrease, by Lemma 3, and so the number of steps is bounded by $|T^{(3)}|_2$. We obtain in this way a proof-net $T^{(4)}$ of conclusion $1^2B$, which:

- does not have any cut at depths 0 and 1,
- only has exponential cuts at depth 2.

By Lemma 16, applied here at depth 2, the result can then be computed in constant time. So on the whole the computation has been carried out in a number of steps which is polynomial in $|R_w|$, hence polynomial in $n$.

Moreover, the size of each intermediary proof-net in the reduction sequence can be polynomially-bounded w.r.t. $n$ in a similar way as what has been done in the proof of Prop. 5. We deduce from that, as argued in the proof of Prop. 12, that each reduction step can be performed in polynomial time on a Turing machine, hence that the whole reduction is performed in polynomial time. □

**Proposition 18 (k-EXP soundness)** Let $R$ be a normal proof-net of conclusion $1^2W \vdash 1^{k+2}B$. 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 time bounded by $2^{P(n)}$.

**Proof**: We will proceed in a way generalizing the proof of Prop. 17, but we will need for that two intermediary lemmas. The first one is a small technical lemma, while the second one allows to reason about partial reduction up to a certain depth $i$.

**Lemma 19** We have the following bounds:

1. for $x \geq 1$: $2x \leq 2^x$,
2. for $x \geq 1$ and $j \geq 1$: $(2^x_j)^2 \leq 2^{2x_j}$. 


Proof : [of Lemma 19]
Consider statement (1). One can check that the function over \( \mathbb{R} \) defined by 
\[ x \rightarrow (2^x - 2x) \] is increasing from \( x \geq 1 \), and its value is 0 when \( x = 1 \), so it is 
positive for \( x \geq 1 \).

As to statement (2), one can prove it by induction on \( j \).

\[ \square \]

Lemma 20 Let \( R \) be a normal proof-net of conclusion \( \vdash \text{!}^{k+2} \text{B} \) and \( 2 \leq i \leq k + 2 \). Then there exists a polynomial \( Q \) such that:

consider \( S \) a proof-net obtained by cutting \( R \) with a proof-net representing a 
word of length \( n \); then one can in at most \( 2^{Q(n)} \) steps reduce \( S \) into a proof-net 
\( S' \) such that \( |S'|_{i+} \leq 2^{Q(n)} \) and with no cut at depth inferior or equal to \( i - 1 \), 
and only exponential cuts at depth \( i \).

Proof : [of Lemma 20]

We proceed by induction on \( i \).

In the case where \( i = 2 \): we proceed as in the proof of Prop. 17; the 
arguments used are still valid in the case of a proof-net of conclusion \( \vdash \text{!}^{k+2} \text{B} \) 
and so we obtain a polynomial bound on \( |S'|_{i+} \) and on the number of steps. So the 
property holds.

Now, assume the property is true for \( i \leq k + 1 \) and let us show it for \( i + 1 \).
By induction hypothesis one has obtained in \( 2^{Q(n)} \) steps a proof-net \( S' \) with 
\( |S'|_{i+} \leq 2^{Q(n)} \) and with no cut at depth inferior or equal to \( i - 1 \), and only 
exponential cuts at depth \( i \). We reduce the exponential cuts at depth \( i \) following the 
strategy of the proof of Prop. 5, that is to say by reducing special cuts. The 
number of steps performed at depth \( i \) is then bounded by \( |S'|_{i} \), and at each step 
the size of the proof-net at depth superior or equal to \( i + 1 \) at most doubles. Let 
us call \( S'' \) the resulting proof-net, which does not have any cut at depth inferior 
or equal to \( i \). We thus have:

\[
|S''|_{(i+1)+} \leq |S'|_{(i+1)+} \cdot 2^{Q(n)},
\]

Finally we reduce all non-exponential cuts at depth \( i + 1 \), which takes less 
than \( |S''|_{(i+1)} \) steps and makes \( |S''|_{(i+1)+} \) decrease. On the whole the number of 
steps performed is thus bounded by:

\[
2^{Q(n)} + |S'|_{i+} + |S''|_{(i+1)} \leq 2^{Q(n)} + 2^{Q(n)} + 2^{Q(n)}.
\]

We want to show that this sum is bounded by \( 2^{Q(n)} \).

By Lemma 19 (1) applied to \( x = Q(n) \) we have \( 2^{Q(n)} \leq 2^{Q(n)} \) and thus 
\( 2^{Q(n)} \leq 2^{Q(n)} = 2^{Q(n)} \). Moreover we have \( 2^{Q(n)} \leq 2^{Q(n)} \). We can assume that
Let us come back to the proof of Prop. 18. Call $S$ the proof-net obtained by cutting $R$ with a proof-net representing a word of length $n$. It has a single conclusion, of type $!k+2B$. Now, we apply Lemma 20 with $i = k + 2$ and thus obtain in at most $2^Q(n)$ reduction steps a proof-net $S'$ with no cut at depth inferior or equal to $k + 1$, and only exponential cuts at depth $k + 2$. Therefore the proof-net $S'$ consists of $k + 2$ boxes applied to a proof-net $T'$ such that: $T'$ has a single conclusion of type $B$ and only exponential cuts at depth 0. By Lemma 16 one can in constant time decide whether $T'$, and thus $S'$, evaluates to true or false.

Therefore, on the whole after at most $2^Q(n)$ reduction steps one can decide the result of the evaluation of $S$. Moreover as in the proof of Prop. 5 we could provide a size bound of the same order on the intermediary proof-nets in the reduction sequence and hence we conclude as in the proof of Prop. 12 that the evaluation can be done in time $O(2^{P(n)})$ on a Turing machine, for some polynomial $P$.

4.2 Proof of the extensional completeness results

To prove the extensional completeness results we will need another datatype using type fixpoints, that of Scott binary words:

$$WS = \mu\beta.\forall\alpha.(\beta \rightarrow \alpha) \rightarrow (\beta \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha).$$

Scott words have already been used in several works on implicit complexity [DLB06, BT10, RV10]. One can easily define terms for the basic operations on binary words over the type $WS$:

$$cons_0 = \lambda w.\lambda s_0.\lambda s_1.\lambda x.(s_0 w) : WS \rightarrow WS$$
$$cons_1 = \lambda w.\lambda s_0.\lambda s_1.\lambda x.(s_1 w) : WS \rightarrow WS$$
$$nil = \lambda s_0.\lambda s_1.\lambda x.x : WS$$
$$tail = \lambda w.(w id id nil) : WS \rightarrow WS$$

where $id = \lambda x.x$.

For this datatype we can define a term:

$$case : \forall\alpha.(WS \rightarrow \alpha) \rightarrow (WS \rightarrow \alpha) \rightarrow \alpha \rightarrow (WS \rightarrow \alpha)$$
$$case = \lambda F_0.\lambda F_1.\lambda a.\lambda w.(w F_0 F_1 a)$$
Remark 4 Actually Scott words can also be typed in elementary affine logic, without fixed points, e.g. with the following type:

\( \forall P . (U[P] \to U[P] \to P \to P) \),

where

\( U[P] = \forall X . ((X \to X \to P \to P) \to P) \).

However it is not clear if one could define a case function in this setting.

We define the following new type representing the configurations of a one-tape Turing machine over a binary alphabet, with \( n \) states:

\( \text{Configs} = W_S \otimes B \otimes W_S \otimes B^n \).

While \( \text{Config}_C \) was based on the Church encoding of words, \( \text{Configs} \) is based on their Scott encoding. Given an element of this type: the first component represents the left part of the tape, in reverse order; the second component represents the symbol scanned by the head; the third component represents the right part of the tape; the fourth part represents the current state.

Lemma 21 Let \( M \) be a one-tape deterministic Turing machine. One can define terms:

\[
\begin{align*}
\text{init}_S &: W_S \rightarrow \text{Configs}, \\
\text{step}_S &: \text{Configs} \rightarrow \text{Configs}, \\
\text{accept}?_S &: \text{Configs} \rightarrow B,
\end{align*}
\]

with the same property as the terms of Lemma 10. Recall in particular that given a configuration, the term \( \text{accept}?_S \) returns true (resp. false) if its state is accepting (resp. rejecting).

Proof : [Sketch]

The terms \( \text{init}_S \) and \( \text{accept}?_S \) are easy. The term \( \text{step}_S \) is actually simpler than \( \text{step}_C \); it can be constructed based on the transition function of \( M \), by doing a case distinction, using the term \( \text{case} \), as in [DLB06] (Sect.7, Lemma 4).

Note that the difference with Lemma 10 is the type of \( \text{accept}? \), which is here \( \text{Configs} \rightarrow B \) instead of \( \text{Config}_C \rightarrow !B \).

Proposition 22 (Extensional completeness) Let \( k \geq 0 \). Consider a function \( g \) representing a predicate of \( k\text{-EXP} \). Then there exists an \( \text{EAL}_m \) proof of conclusion \( !W \vdash !^{k+2}B \) representing \( g \).

Proof : We repeat the construction of the proof of the second statement of Prop. 11, with the following modifications:

- we use here \( \text{Configs} \) instead of \( \text{Config}_C \),
we use the terms init$_S$, step$_S$, accept?$_S$ of Lemma 21, instead of (respectively) init$_C$, step$_C$, accept?$_C$.

As accept?$_S$ has type Config$_S$ → B we obtain in the end a proof of type !W ⊢ !^{k+2}B instead of !W ⊢ !^{k+3}B. □

Finally, together the results of Propositions 17, 18 and 22 establish Theorem 14.

Remark 5 Observe that proofs of type !W ⊢ !^2W do not correspond to polynomial time functions. Indeed, one can easily define a proof wdouble of conclusion W → W which doubles the length of a word. By using iteration on words, applied here to this wdouble proof, one gets a proof wexp of conclusion W ⊢ !W, which, given a word of length n, produces a word of length $2^n$. Applying the ! rule one thus obtains a proof of !W ⊢ !^2W with the same behaviour.

To characterize the complexity class FP of polynomial time functions, one could use the type !W → !^2W$_S$, where W$_S$ is the type of Scott binary words. However a drawback of this characterization is that the proofs representing these functions could not be composed, because of the mismatch on the input and output types.

5 Conclusion and future work

Elementary linear logic was up to now considered as a simple variant of elementary linear logic with good structural properties but of limited interest for complexity. We have shown here that, provided one adds to it type fixpoints, it is expressive enough to characterize P, EXP and a time hierarchy inside the elementary class. As far as we know this is the first characterization of EXP and of the classes k-EXP within a variant of linear logic. An interesting feature of this approach is that this provides a single type system in which one characterizes different complexity classes with the same term calculus, simply by considering terms of different types.

Several questions remain open. Is it possible to obtain the same result without type fixpoint? Could one also characterize in this system LINSFACE or other space complexity classes in a way similar to [Lei02, Lei94b]? It would also be interesting to examine whether one could re-prove in this purely logical framework the classical hierarchy results, like P ̸= EXP, by carrying out a diagonalisation argument.

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. Thanks also to Christophe Raffalli, for useful discussions about the typing of Scott integers in system F.

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

28
References


