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

Patrick Baillot

CNRS, ENS de Lyon, Inria, UCBL, Université de Lyon. LIP

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 λ-calculus ensuring polynomial time complexity bounds on well-typed terms.

This paper aims at reviving interest in elementary linear logic by showing 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 $!\mathbf{W} \multimap !^{k+2}\mathbf{B}$, where $\mathbf{W}$ and $\mathbf{B}$ are respectively types for binary words and booleans. The key point is a sharper study of the normalization bounds. We characterize in this way the class P of polynomial time predicates, and more generally the hierarchy of classes $\mathsf{k\text{-}EXP}$, for $k \geq 0$, where $\mathsf{k\text{-}EXP}$ is the union of $\mathsf{DTIME}(2_k^{n^i})$, for $i \geq 1$.

Keywords: implicit computational complexity, linear logic, lambda-calculus, polynomial time complexity, type systems

1. Introduction

Implicit computational complexity. This line of research promotes investigations to characterize classical complexity classes by programming languages or logics, without refering to explicit bounds on resources (time, space . . .)

Email address: patrick.baillot@ens-lyon.fr (Patrick Baillot)

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 programming 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 λ-calculus or functional languages ensuring that a well-typed program has a polynomial time complexity bound [BT09, GRDR07, 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 § modality) and allows for a more natural programming style than SLL. It has also been studied for its remarkable properties concerning λ-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 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 $\mathsf{k\text{-}EXP} = \cup_i \mathsf{DTIME}(2_k^{n^i})$, 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-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 practise since [AR02]. The new system will be denoted EAL_μ (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 $\mathbf{W}$ for binary words and $\mathbf{B}$ for booleans, we will consider types of the form $!\mathbf{W} \multimap !^k\mathbf{W}$ and $!\mathbf{W} \multimap !^k\mathbf{B}$, 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 $!\mathbf{W} \multimap !^k\mathbf{W}$.

While [Gir98, DJ03] had characterized the complexity of functions represented by the *union* of types $\cup_k(\mathbf{N} \multimap !^k\mathbf{N})$, where $\mathbf{N}$ 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 $!\mathbf{W} \multimap !^k\mathbf{B}$, 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 $!\mathbf{W} \multimap !^{k+2}\mathbf{B}$ for $k \geq 0$ in EAL_μ *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).

Outline of the paper. In Sect. 2 we introduce the system of elementary affine logic that we will study. Then in Sect. 3 we recall the main results about this logic and bring slightly improved statements. Finally Sect. 4 is the core of the paper and establishes how each class k-EXP for $k \geq 0$ is characterized by a particular type of the logic.

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_{μ}. Actually for our purpose it is sufficient to consider its multiplicative fragment. The grammar of types is:

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

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

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

$$t, u ::= x \mid \lambda x.t \mid (t\ u) \mid t \otimes u \mid \mathrm{let}\, t\, \mathrm{be}\, x \otimes y\, \mathrm{in}\, u$$

In the term $\mathrm{let}\, t\, \mathrm{be}\, x \otimes y\, \mathrm{in}\, u$ the variables x and y are bound in u. We denote by $FV(t)$ the set of free variables in t.

We consider the reduction relation $\xrightarrow{1}$ obtained by the context-closure of the relation $\mapsto$ defined on Fig. 1. The rule $(\otimes)$ can be seen as a pattern-matching rule. The rules $(com1)$ and $(com2)$ are commutation rules, which are needed to make some redexes appear. The rule (gc) can be seen as a garbage-collection rule; it will be useful only later in the paper, when we will relate the reduction in this λ-calculus to graph rewriting.

We denote by $\rightarrow$ the reflexive and transitive closure of $\xrightarrow{1}$.

We will use sequents of the form $x_1 : B_1, \dots, x_n : B_n \vdash t : A$. If $\Gamma = x_1 : B_1, \dots, x_n : B_n$ then $!\Gamma$ will be the context $x_1 : !B_1, \dots, x_n : !B_n$.

The rules of EAL_{μ} are now given on Fig. 2, as a sequent calculus decorated with λ-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_{μ} and R_{μ}) and allowed for general weakening (rule (Weak)). Observe that some rules do not have any effect on the term (!, L_{μ}, R_{μ}, $L_{\forall}$, $R_{\forall}$): this is because we want to keep the term calculus

$$
\begin{array}{lcll}
((\lambda x.t)\ u) & \mapsto & t[u/x] & (\beta) \\
\mathrm{let}\, t_1 \otimes t_2 \,\mathrm{be}\, x \otimes y \,\mathrm{in}\, u & \mapsto & u[t_1/x, t_2/y] & (\otimes) \\
((\mathrm{let}\, t_1 \,\mathrm{be}\, x \otimes y \,\mathrm{in}\, t_2)\ t_3) & \mapsto & \mathrm{let}\, t_1 \,\mathrm{be}\, x \otimes y \,\mathrm{in}\, (t_2\ t_3) & (com1) \\
\mathrm{let}\, (\mathrm{let}\, t_1 \,\mathrm{be}\, x \otimes y \,\mathrm{in}\, t_2) \,\mathrm{be}\, x' \otimes y' \,\mathrm{in}\, t_3 & \mapsto & \mathrm{let}\, t_1 \,\mathrm{be}\, x \otimes y \,\mathrm{in}\, (\mathrm{let}\, t_2 \,\mathrm{be}\, x' \otimes y' \,\mathrm{in}\, t_3) & (com2) \\
\mathrm{let}\, t \,\mathrm{be}\, x \otimes y \,\mathrm{in}\, u & \mapsto & u \qquad \text{if } x, y \notin FV(u), & (gc)
\end{array}
$$

Figure 1: Lambda-calculus reduction rules

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.

A cut-elimination rewriting procedure can be defined on these sequent calculus proofs, see *e.g.* [DJ03]. We do not describe it here because anyway we will explain in detail cut-elimination on proof-nets in Section 3. Let us write $\pi \to \pi'$ if the sequent calculus proof π rewrites by cut-elimination steps to the proof π'. We have the following property (see also [DJ03]):

Proposition 1. *If π is a sequent calculus proof of conclusion $\Gamma \vdash t : A$ and π rewrites by cut-elimination to a proof π' without any cut, then π' has a conclusion of the form $\Gamma \vdash t' : A$, we have $t \to t'$ and t' is in normal form.*

Observe 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_μ (see [Gir98]).

Let us stress that we do not claim that EAL_μ with the rules of Fig. 2 is a good type system for λ-calculus. In particular it does not satisfy the subject-reduction property, though we have the weaker property of Prop. 1. A study of a type system derived from elementary affine logic and with good properties has been carried out in [CDLRDR08]. Here we are dealing with EAL_μ proofs, and the lambda-terms are merely a convenient notation to describe the algorithmic behaviour of these proofs.

Actually when we will study cut-elimination by means of proof-nets, for technical reasons we will need an extension of the lambda-calculus and of the sequent calculus. For that we add to the lambda-calculus a constant $\bullet$ and we consider the following extra sequent calculus rule (h):

$$\frac{}{\vdash \bullet : A}\ h$$

Let us call EAL_μ^+ the system obtained by adding (h) to EAL_μ.

Now let us consider the application of a function to an argument. If we have two proofs π_1 and π_2 with conclusions respectively of the form $\Gamma_1 \vdash t : !^n A \multimap !^m B$ and $\Gamma_2 \vdash u : A$, where $m, n \in \mathbb{N}$, then we can obtain from them a proof

π_3 of conclusion $\Gamma_1, !^n\Gamma_2 \vdash t\ u : !^m B$ by using a sequence of n (!) rules and the usual encoding of $\multimap$ elimination in sequent calculus, as shown on Fig. 3. We say that π_3 is obtained by applying π_1 to π_2.

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

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

Let us describe for each of these data-types the cut-free proofs representing values, by indicating the associated lambda-terms. We have:

- for $\mathbf{B}$: *true* and *false* are respectively the following terms:

$$\lambda x.\lambda y.x, \quad \lambda x.\lambda y.y,$$

- for $\mathbf{B}^n$: the i-th value, for $1 \leq i \leq n$ is the i-th projection:

$$\lambda x_1 \ldots \lambda x_n.x_i,$$

- for $\mathbf{N}$:

  the values are the Church integers, for $n \geq 0$:

$$\underline{n} = \lambda f.\lambda x.(f\ \ldots (f\ x)),\ \text{with } n \text{ occurrences of } f.$$

  Note that actually the integer 1 has a second lambda-term representation which is the identity: $\lambda f.f$.

- for $\mathbf{W}$: the binary word $[b_1, \ldots, b_n]$, where for $i \in [1, n]$, $b_i \in \{0, 1\}$ corresponds to the following lambda-calculus term:

$$\lambda s_0.\lambda s_1.\lambda x.(s_{b_1}\ (s_{b_2}\ (\ldots (s_{b_n}\ x) \ldots).$$

Note that these data-types admit some coercions [Gir98]. For $\mathbf{W}$ for instance, one can give a proof of type $\mathbf{W} \multimap !\mathbf{W}$ which, as a λ-term, acts as an identity on the terms encoding binary words.

There is also a term *length* : $\mathbf{W} \multimap \mathbf{N}$ which returns the length of a word, as a tally integer:

$$length = \lambda w.\lambda f.\lambda x.(w\ f\ f\ x)$$

In this paper we want to characterize for a given system $\mathcal{S}$ (EAL or EAL_μ) 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 types of the form $!^n D_1 \multimap !^m D_2$, where D_1 and D_2 are data-types.

Axiom and Cut.

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

Structural Rules.

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

Multiplicative Rules.

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

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

Exponential Logical Rule.

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

Second Order Rules

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

Fixpoint Rules

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

Figure 2: The system EAL_{μ}

$$\dfrac{\dfrac{\pi_1}{\Gamma_1 \vdash t : !^n A \multimap !^m B} \qquad \dfrac{\dfrac{\dfrac{\pi_2}{\Gamma_2 \vdash u : A}}{!^n\Gamma_2 \vdash u : !^n A}\ ! \qquad \dfrac{}{y : !^m B \vdash y : !^m B}\ Ax}{!^n\Gamma_2, z : !^n A \multimap !^m B \vdash z\ u : !^m B}\ L_{\multimap}}{\Gamma_1, !^n\Gamma_2 \vdash t\ u : !^m B}\ Cut$$

Figure 3: Application of a function to an argument

Definition 1. *Let n, m belong to $\mathbb{N}$. We say a function $f : \{0,1\}^\star \to \{0,1\}^\star$ is represented by an EAL_μ proof π_1 of conclusion $\vdash\, !^n\mathbf{W} \multimap\, !^m\mathbf{W}$ if:*

for any word $w \in \{0,1\}^\star$, if π_2 is a proof of conclusion $\vdash u : \mathbf{W}$ where u represents w, then π_1 applied to π_2 rewrites to a cut-free proof π_3 of conclusion $\vdash v :\, !^m\mathbf{W}$ where v represents $w' = f(w) \in \{0,1\}^\star$.

We say f is representable with the type $!^n\mathbf{W} \multimap\, !^m\mathbf{W}$ in EAL_μ if there exists an EAL_μ proof of this conclusion representing it. The class of these functions will be denoted as $\mathcal{F}_{EAL_\mu}(!^n\mathbf{W} \multimap\, !^m\mathbf{W})$.

We define in the same way the functions represented by an EAL_μ proof of conclusion $\vdash\, !^n\mathbf{W} \multimap\, !^m\mathbf{B}$ (resp. $\vdash\, !^n\mathbf{N} \multimap\, !^m\mathbf{N}$). The corresponding function class is denoted $\mathcal{F}_{EAL_\mu}(!^n\mathbf{W} \multimap\, !^m\mathbf{B})$ (resp. $\mathcal{F}_{EAL_\mu}(!^n\mathbf{N} \multimap\, !^m\mathbf{N})$).

We define similarly the functions classes representable in EAL (instead of EAL_μ): $\mathcal{F}_{EAL}(A)$, for $A = !^n\mathbf{W} \multimap\, !^m\mathbf{W}$, $!^n\mathbf{W} \multimap\, !^m\mathbf{B}$, or $!^n\mathbf{N} \multimap\, !^m\mathbf{N}$.

We also extend these notations to a set $\mathcal{A}$ of types, by defining $\mathcal{F}_\mathcal{S}(\mathcal{A}) = \cup_{A\in\mathcal{A}}\mathcal{F}_\mathcal{S}(A)$, for $\mathcal{S} = EAL$ or EAL_μ.

Complexity classes. We refer the reader to *e.g.* [AB09] for the definition of the notion of *proper complexity function*. If $F : \mathbb{N} \to \mathbb{N}$ is a proper complexity function we denote by $\mathsf{FDTIME}(F(n))$ (resp. $\mathsf{DTIME}(F(n))$) the class of functions on binary words (resp. predicates 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:

$$\begin{array}{rcl} \mathsf{P} & = & \cup_{i\in\mathbb{N}}\mathsf{DTIME}(n^i) \\ \mathsf{EXP} & = & \cup_{i\in\mathbb{N}}\mathsf{DTIME}(2^{n^i}) \\ \mathsf{k\text{-}EXP} & = & \cup_{i\in\mathbb{N}}\mathsf{DTIME}(2^{n^i}_k),\ \text{for } k \geq 0 \\ \mathsf{k\text{-}FEXP} & = & \cup_{i\in\mathbb{N}}\mathsf{FDTIME}(2^{n^i}_k),\ \text{for } k \geq 0 \\ \mathsf{FELEM} & = & \cup_{k\in\mathbb{N}}\mathsf{k\text{-}FEXP}. \end{array}$$

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

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] and developed in [DJ03]. So the techniques employed in this section are not new but the differences we bring with respect to [DJ03] are the following ones:

- we need to be careful with the handling of the weakening reduction steps (erasing) because we consider an affine system, and not a linear one;

- we provide a slightly improved upper bound for cut-elimination (see Prop. 7 and Remark 1);
- we consider functions over binary words (type $\mathbf{W}$) instead of tally integers (type $\mathbf{N}$); this is useful to do comparisons with standard time complexity classes;
- we provide lower bounds and upper bounds for each level of a hierarchy of types $(!\mathbf{W} \multimap !^k\mathbf{W})$ (Proposition 20), and not just for their union.

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:

$$A ::= \alpha \mid \alpha^{\perp} \mid A \otimes A \mid A \wp A \mid !A \mid ?A \mid \mu\alpha.A \mid \overline{\mu}\alpha.A \mid \forall\alpha.A \mid \exists\alpha.A$$

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

Formulas of EAL_μ are translated in this grammar by using $A \multimap B \equiv A^{\perp}\wp B$ and the usual linear logic De Morgan laws for linear negation:

$$\begin{array}{lll} (A \otimes B)^{\perp} \equiv A^{\perp}\wp B^{\perp}, & (!A)^{\perp} \equiv ?A^{\perp}, & \\ (\mu\alpha.A)^{\perp} \equiv \overline{\mu}\alpha.A^{\perp}, & (\forall\alpha.A)^{\perp} \equiv \exists\alpha.A^{\perp}, & A^{\perp\perp} \equiv A. \end{array}$$

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 notations of [AR02], though our garbage collection reductions rules will be different.

The nodes are described on Fig. 4:

- 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; ! and ? nodes can only be used with a !-*box* (see below);
- two nodes μ and $\overline{\mu}$ are added, corresponding resp. to the fixpoint rules R_μ and L_μ;
- each node ! or ? comes with a !-*box*, as shown on Fig. 5; 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 *auxilliary doors*;
- as the system is affine, the proof-nets use, beside the weakening w-node, also an h-node.

Definition 2. *We call proof-structure a typed graph with pending edges built from the nodes described above.*

Note that the h-node is not needed in linear logic or elementary linear logic, where weakening is restricted to formulas marked with a ? exponential. It will correspond to the rule (h) that we have defined in the EAL_μ^+ sequent calculus. We will explain later why this node is useful, when speaking about reduction.

One will translate an EAL_μ^+ proof π of conclusion $x_1 : A_1, \dots, x_n : A_n \vdash t : B$ for $n \geq 0$ by a proof-structure π^* with conclusions ${A_1}^\perp, \dots, {A_n}^\perp, B$ 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 π is obtained by an Ax rule, then π^* is an ax-node;

- if π is obtained by a Cut rule between π_1 and π_2, then π^* is obtained by linking ${\pi_1}^*$ and ${\pi_2}^*$ by a *cut*-node;

- if π is obtained by a !-rule on π_1, then π^* is obtained by applying a !-box on ${\pi_1}^*$ (see Fig. 5),

and so on. On Fig. 4 below each node we have indicated the sequent-calculus rule it corresponds to.

Now:

Definition 3. *A proof-structure R is called a proof-net (resp. net) if there exists an EAL_μ proof π (resp. an EAL_μ^+ proof π) such that $R = \pi^*$.*

As the only difference between EAL_μ^+ and EAL_μ is the (h) rule, a net R is a proof-net iff it does not contain any h node. In reality we are only interested in proof-nets, but nets will be a useful technical artifact for analyzing the reduction of proof-nets.

A net is *cut-free* if it does not contain any cut node.

We say that a node N in a proof-structure is *above* a node N' (or N' is *below* N) if there is a directed path from N to N'. Note that each node N is either above a *cut*-node or above a conclusion. In practice we will sometimes omit the orientation of edges when drawing proof-structures.

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 $⅋$-node is called a *multiplicative cut.* A cut between a !-node and a ?-node or $?c$-node (resp. a $?c$-node) is called an *exponential cut* (resp. a contraction cut). *Quantifier cuts* and *μ-cuts* are defined in an analogous way. A cut between an h-node and another node is called an *h-cut.*

Definition 4. *In a proof-structure, a maximal tree with ?-nodes, ax-nodes and w-nodes (of type $?A^\perp$) as leaves, and $?c$-nodes as internal nodes, is called an exponential tree.*

The following notion will be important in the sequel:

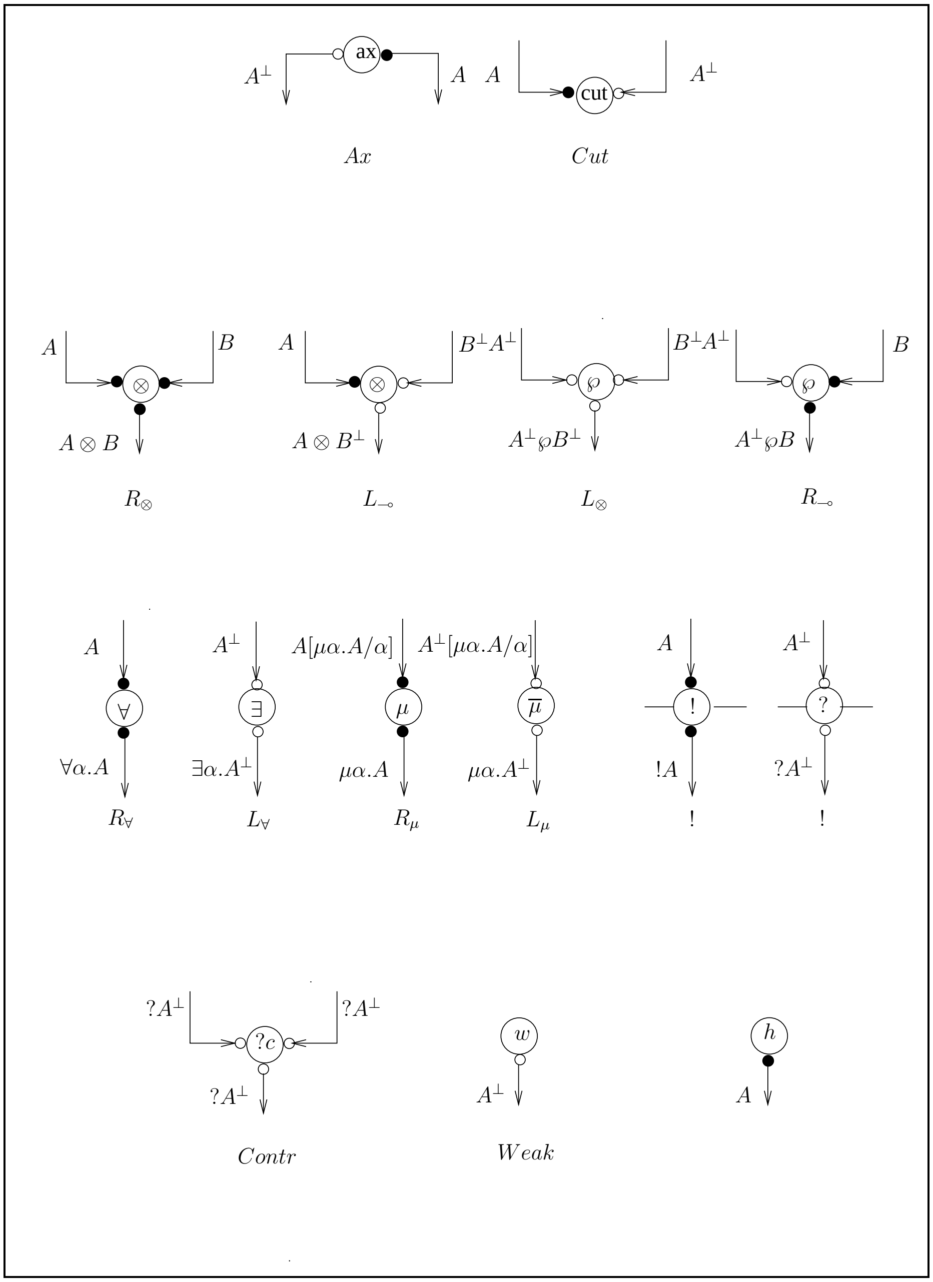


Figure 4: Nodes of the proof-structures.

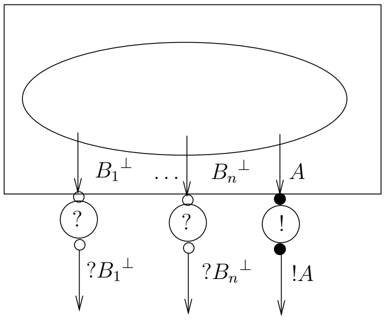


Figure 5: !-box.

Definition 5. *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 $d(R)$.

Definition 6. *The size of R at depth i, denoted $|R|_i$, is the sum of the nodes at depth i, counted with weight 3 in the case of nodes $\otimes$ and $\wp$, and with weight 1 otherwise.*

We denote $|R|_{i+} = \Sigma_{j=i}^{d(R)} |R|_j$ and $|R| = |R|_{0+}$, which is called the size of R.

This difference of weights between different kinds of nodes is needed for technical reasons (essentially to obtain forthcoming Lemma 5) but note that $|R|_i$ is anyway linearly related to the number of nodes at depth i.

Data types. The data-types defined in Sect. 2 are now written as follows:

$$\begin{array}{lcl}
\mathbf{B} & = & \forall\alpha.\alpha^\perp \wp \alpha^\perp \wp \alpha \\
\mathbf{B^n} & = & \forall\alpha.\alpha^\perp \wp \alpha^\perp \wp \dots \wp \alpha^\perp \wp \alpha, \text{ with } n \text{ occurrences of } \alpha^\perp \\
\mathbf{N} & = & \forall\alpha.?(\alpha \otimes \alpha^\perp) \wp !(\alpha^\perp \wp \alpha) \\
\mathbf{W} & = & \forall\alpha.?(\alpha \otimes \alpha^\perp) \wp ?(\alpha \otimes \alpha^\perp) \wp !(\alpha^\perp \wp \alpha)
\end{array}$$

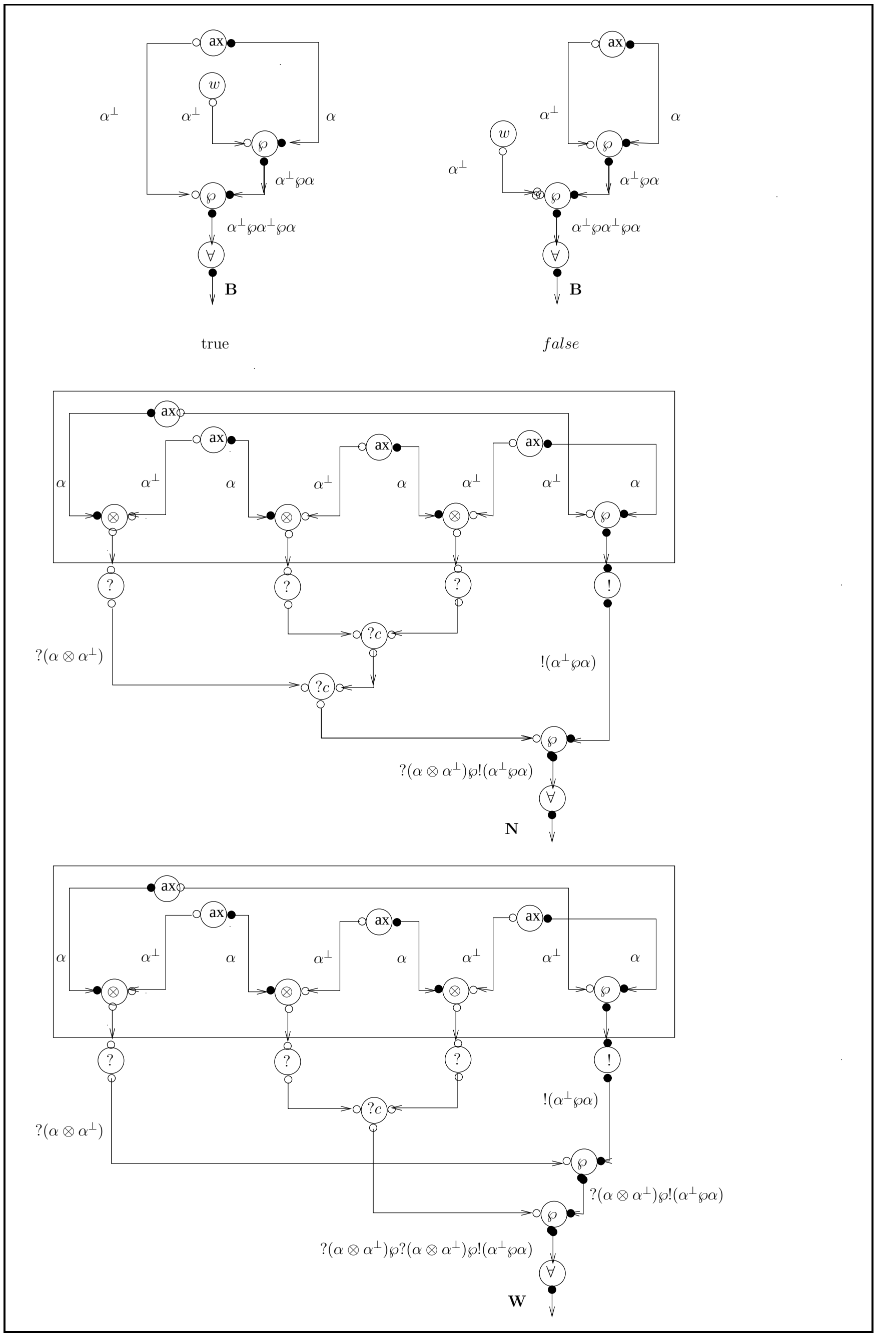
So for instance a tally integer is represented by a cut-free proof-net of conclusion $\mathbf{N}$ and a binary word by one of conclusion $\mathbf{W}$. We give on Fig. 6 the example of proof-nets representing respectively the booleans `true` and `false`, the tally integer 3, and the binary word $< 0, 1, 1 >$.

Now we observe that:

Lemma 2. *If R is a cut-free proof-net of conclusion $\mathbf{B}$ or $\mathbf{B}^n$ (resp. $\mathbf{N}$ or $\mathbf{W}$) then its depth is 0 (resp. at most 1).*

Proof : By definition cut-free proof-nets correspond to cut-free sequent calculus proofs, and one can check that the only cut-free sequent calculus proofs of $\mathbf{B}$ and $\mathbf{B}^n$ (resp. $\mathbf{N}$ and $\mathbf{W}$) have depth 0 (resp. depth inferior or equal to 1). □

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:



ax
w
$\alpha^\perp$
α
$\wp$
$\alpha^\perp \wp \alpha$
$\alpha^\perp \wp \alpha^\perp \wp \alpha$
$\forall$
B
true
false
$\otimes$
?
!
$?c$
$?(\alpha \otimes \alpha^\perp)$
$!(\alpha^\perp \wp \alpha)$
$?(\alpha \otimes \alpha^\perp) \wp !(\alpha^\perp \wp \alpha)$
N
$?(\alpha \otimes \alpha^\perp) \wp ?(\alpha \otimes \alpha^\perp) \wp !(\alpha^\perp \wp \alpha)$
W

Figure 6: Examples.

Lemma 3. *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* $\mathbf{W}$ *and of size $|R_w| \leq a \cdot n$.*

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

Reduction. We describe the reduction procedure on nets, which is defined by the rules given on Fig. 7, 8, 9, 10:

- Fig. 7 shows the reduction of cuts on axiom, multiplicative node, quantification node and fixpoint node;
- Fig. 8 shows the reduction of exponential cuts (contraction step and box-box step);
- Fig. 9 shows the reduction of cuts on weakening node; notice that one of these steps introduces an h-node;
- Fig. 10 shows *garbage collection* reduction steps.

If a net R reduces to R' then R' is also a net. We omit the proof of this claim, which can be carried out in a way similar to the setting of linear logic.

Examine on Fig. 9 the last case of reduction, dealing with a cut between a weakening node and the principal door of a box. In linear logic and elementary linear logic weakening is only possible on formulas of the form $?A$, so this is the only kind of weakening cut that can occur. This is why h-nodes are not necessary for reducing proof-nets for these systems.

In the present affine setting, with unrestricted weakening, we need to perform erasing with small steps. In addition to the weakening steps of Fig. 9 we also need the garbage collection reduction steps of Fig. 10 which progressively delete the irrelevant part of the net. Note that the garbage-collection rules do not correspond to the elimination of a cut node, and moreover the 4th reduction step of Fig. 10 introduces a new cut node.

We write $R \rightarrow R'$ if R reduces to R' by several (possibly 0) steps.

We say that a net R is *normal* if no reduction step can be applied. Observe that on the one-hand a net with no cut node is not necessarily normal, and that on the other hand a normal net might contain some cut nodes: for instance no reduction step can be applied to a cut between an h-node and a $\otimes$ node.

We will however show further in the paper (Prop. 14) that if we reduce a *proof-net* R to its normal form R', then R' does not contain any h-node nor cut. So in particular R' is a proof-net.

Observe that during a reduction step the depth of an edge does not change, hence the depth of the proof-structure 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 !-box is not opened and does not enter another box (see Fig. 8).

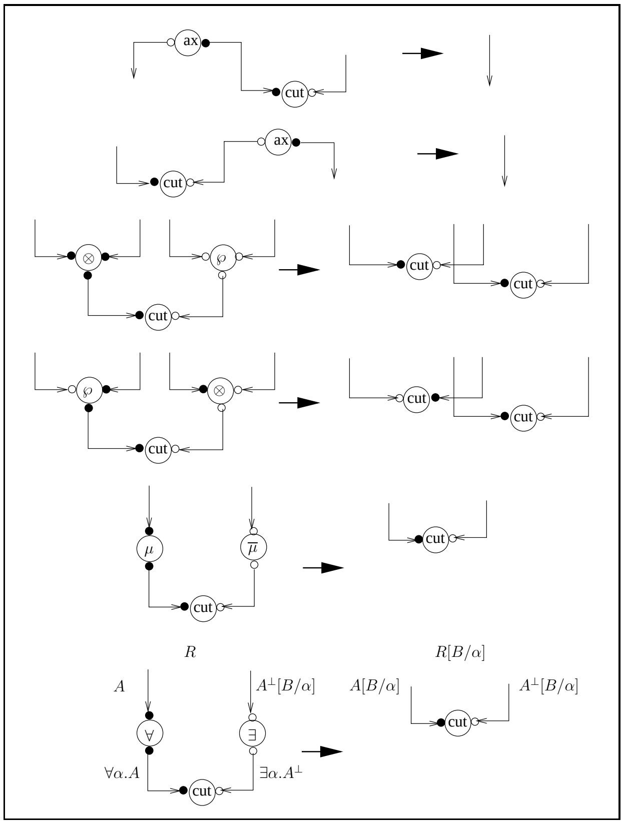


Figure 7: Reduction steps (1/4).

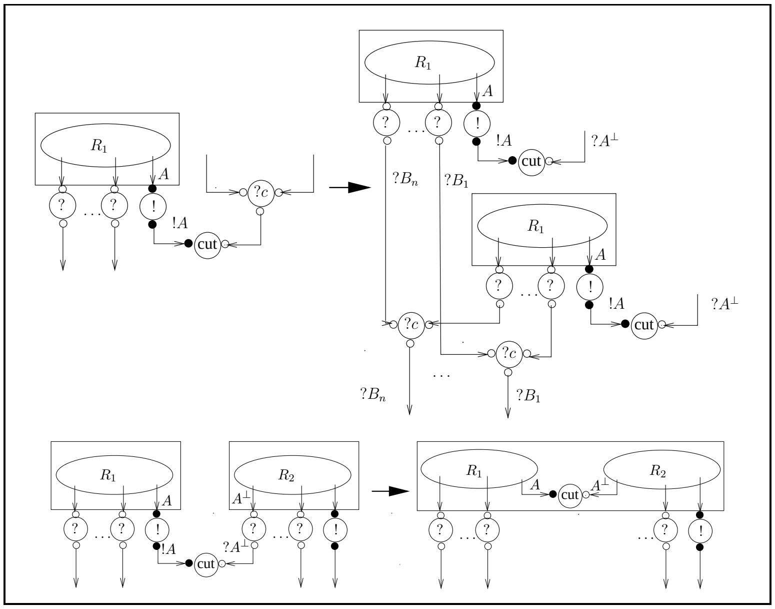


Figure 8: Reduction steps (2/4).

Bounds on reduction. The following notion is useful when defining reduction strategies:

Definition 7. *We say that an exponential cut c is a special cut if the box $\mathcal{B}$ corresponding to the ! node does not have any cut below its auxilliary doors.*

We have:

Lemma 4. *If a 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 5. *Let R be a net and R' be obtained from R by a reduction step at depth i which is not a contraction cut reduction step. 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.

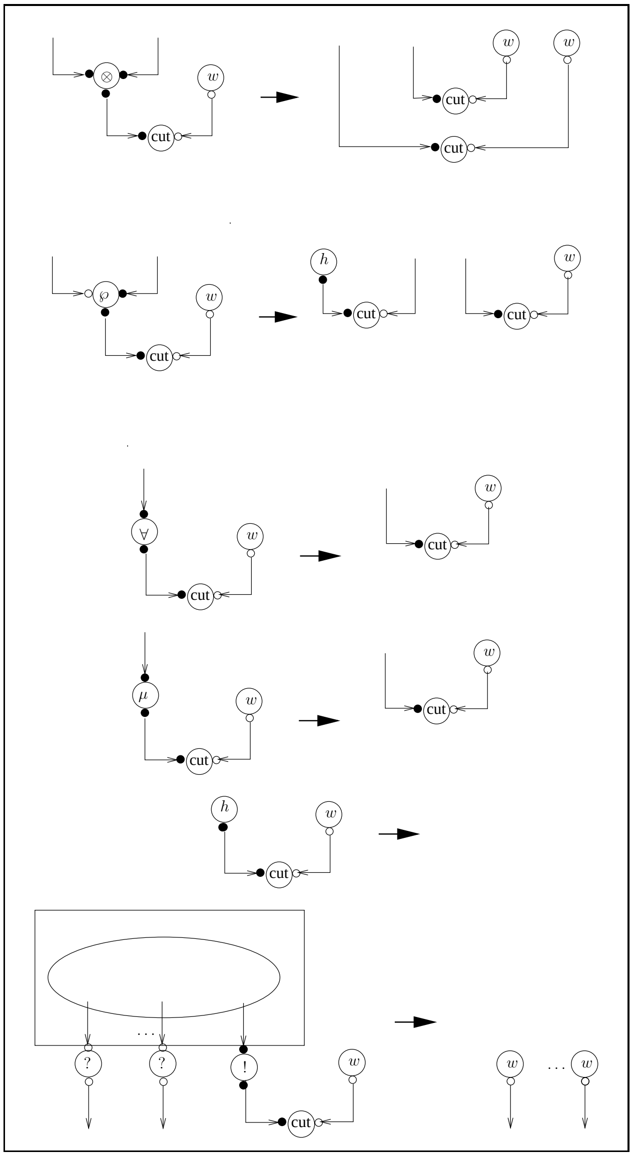


Figure 9: Reduction steps (3/4): weakening steps, denoted w_i for $1 \leq i \leq 6$.

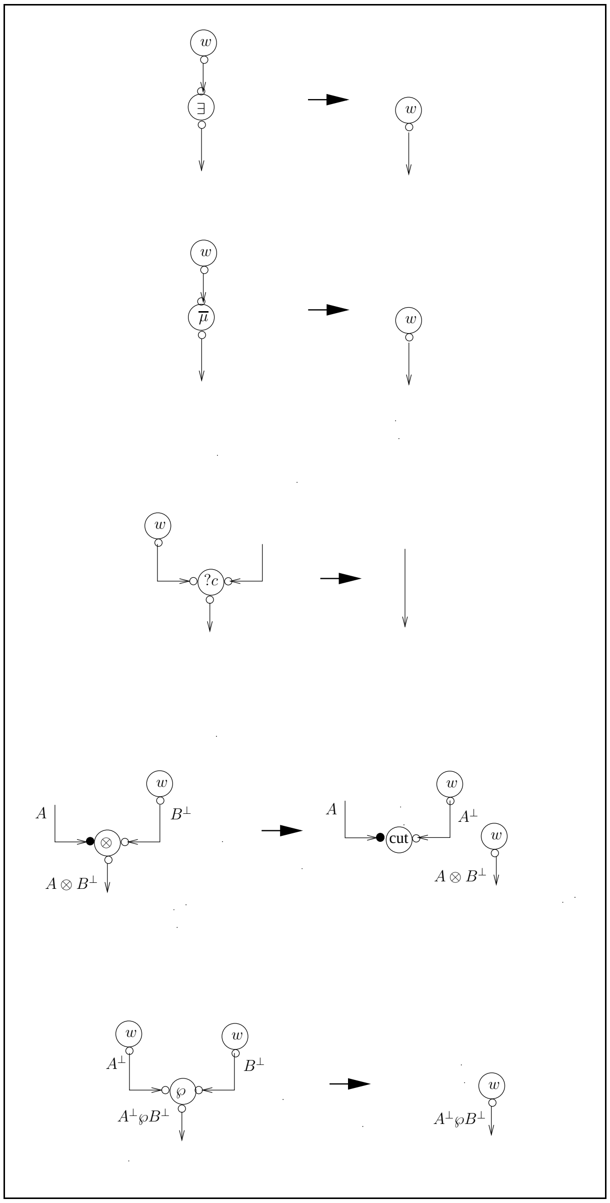


Figure 10: Reduction steps (4/4): garbage collection steps, denoted gc_i for $1 \leq i \leq 5$.

Lemma 6. *If $k \geq 0$ and $x \geq 1$ then we have:*

$$5^{2^x_k} \leq 2^{3x}_{k+1}.$$

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

- If $k = 0$, then we do have $5^x \leq (2^3)^x$.
- Assume the statement holds for k; then we have:

$$\begin{aligned} 5^{2^x_{k+1}} &= 5^{2^{2^x}_k} \leq 2^{3\cdot 2^x}_{k+1}, \text{ by induction hypothesis on } k,\\ &\leq 2^{2^{2x}\cdot 2^x}_{k+1}, \text{ because } 3 \leq 2^{2x},\\ &\leq 2^{2^{3x}}_{k+1} = 2^{3x}_{k+2}. \end{aligned}$$

So the statement holds for all k. □

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

Proposition 7. *Let R be an EAL^+_μ net of depth d. Then R can be reduced in a number of steps which is bounded by $2(d+1)2^{|R|}_d$.*

Moreover the size $|S|$ of any intermediary S obtained during the reduction is bounded by $(d+1)2^{3|R|}_d$.

The proof of Prop. 7 will be given further, after stating a few intermediary lemmas.

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

The refinement of the bound we give here with respect to [Gir98, DJ03] is due to a more precise analysis of the reduction.

To prove the bound on reduction of Prop. 7 we will consider a specific reduction strategy, adapted from [Gir98]:

- perform the reductions 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): perform all reductions but those of exponential cuts,
  - phase (b): reduce the exponential cuts, by repeatedly reducing a *special cut.*

Thanks to Lemma 4 this is a valid reduction strategy.

Let us denote by R^i the net at the beginning of round i and R^{d+1} or R' the final 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*.

Definition 8. *An active node in R is a contraction node or an auxilliary door ? of a box which is above a cut node.*

Lemma 8. *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 auxilliary door of another box (second case of Fig. 8), 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. 8) then one contraction node is removed, but k are created, where k is the number of auxilliary doors of $\mathcal{B}$. However, as the cut is special there are not cuts below the auxilliary doors of $\mathcal{B}$, therefore the contraction nodes created are not active and the claim holds. □

Lemma 9. *If S is the 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 net R is inferior or equal to $|R|_i$. By Lemma 8 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. 7:

Proof :[Prop. 7] 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 net at the beginning of round i. By Lemmas 5 and 9 the number of steps of phase (a) and phase (b) are both bounded by $|R^i|_i$. So the number of steps of round i is bounded by $2|R^i|_i$.

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

$$\begin{aligned}
|R^{i+1}|_{(i+1)+} &\leq 2^{|R^i|_i} \cdot |R^i|_{(i+1)+} \\
&\leq 2^{|R^i|_i} \cdot 2^{|R^i|_{(i+1)+}}, \text{ because } x \leq 2^x, \\
&\leq 2^{|R^i|_i + |R^i|_{(i+1)+}} = 2^{|R^i|_{i+}}.
\end{aligned}$$

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

$$\text{for } i \leq d, \quad |R^i|_{i+} \leq 2_i^{|R|}.$$

From that we deduce:

$$\text{for } i \leq d, \quad |R^i|_i \leq |R^i|_{i+} \leq 2_i^{|R|} \leq 2_d^{|R|}.$$

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

Now we want to prove the bound on the size of any intermediary 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 8). 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 10. *If $0 \leq i \leq d$ and $0 \leq j \leq d$ then we have:*

$$|R^i|_j \leq 2_i^{3|R|}.$$

Moreover:

$$|R^{d+1}|_j \leq 2_d^{3|R|}.$$

Proof :[of Lemma 10]

Let us start by proving 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 $|R^i|_{i+} \leq 2_i^{|R|}$. So for $j \geq i$ we have $|R^i|_j \leq 2_i^{|R|}$.

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 auxilliary doors of the duplicated box, then the new nodes created are k contraction nodes (of weight 3), k auxilliary 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 8), which itself is inferior or equal to $|R^{i-1}|_{(i-1)} - 1$ (because there is at least one node which is not an active one) we have:

$$\begin{array}{rcl} |R^i|_{(i-1)} & \leq & 5^{|R^{i-1}|_{(i-1)}-1} \cdot |R^{i-1}|_{(i-1)} \\ & \leq & 5^{|R^{i-1}|_{(i-1)}} \\ & \leq & 5^{2_{(i-1)}^{|R|}} \\ & \leq & 2_i^{3|R|}, \text{ by Lemma 6.} \end{array}$$

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 $|R^i|_j = |R^{(i-1)}|_j \leq 2_{i-1}^{3|R|}$ 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 $|R^{d+1}|_d \leq 2_d^{3|R|}$. □

Now, from Lemma 10 we deduce that for any $0 \leq i \leq d+1$, we have $|R^i| \leq \sum_{j=0}^{d} |R^i|_j \leq (d+1) \cdot 2_d^{3|R|}$. Finally, if S is an intermediary net obtained during the reduction of S, then there exists i such that S occurs during round i. The bound $(d+1) \cdot 2_d^{3|R|}$ we have computed for $|R^i|$ is actually the largest size that could be reached during round i, so we have $|S| \leq (d+1)2_d^{3|R|}$. This ends the proof of Prop. 7.

□

3.2. Computing with proof-nets.

In this subsection we will establish results on proof-net reduction and the relation between proof-nets and sequent calculus proofs, in order to show that we can use proof-nets for computation.

Proposition 7 has established that EAL_μ^+ nets reduction is weakly normalizing. By adapting the *standardization technique* of [Ter07] (see also [Maz06]) one can then show:

Theorem 11 (Strong normalization). *The reduction of EAL_μ^+ nets is strongly normalizing.*

Now, we want to show that the h-nodes that are created during the reduction of a proof-net eventually disappear. For that we first establish a technical lemma.

Lemma 12. *Let π be an EAL_μ proof of conclusion $x_1 : B_1, \dots, x_n : B_n \vdash t : A$, and $R = \pi^*$, of conclusions $B_1^\perp, \dots, B_n^\perp, A$. Let now S be the proof-structure obtained by a cut between R and the conclusion $A^\perp$ of a weakening node. Then S can be reduced into a proof-structure S' consisting of n weakening nodes of conclusions respectively $B_1^\perp, \dots, B_n^\perp$ (see Fig. 11).*

Proof :[Lemma 12] We proceed by induction on π. For that we examine the last rule of π. The two most interesting cases are those of $R_\multimap$ and $L_\multimap$ which we thus handle first. If π is obtained by:

- a rule $R_\multimap$ on a proof π_0: let $R_0 = \pi_0^*$, then R is obtained by adding a $\wp$ node to R_0 with one negative premise and one positive premise. By one step of reduction, S reduces to a proof-structure with one negative conclusion cut with an h-node, and its positive conclusion cut with a w node (see Fig. 12). By induction hypothesis on π_0, this proof-structure reduces to a proof-structure consisting in n weakening nodes, with $n-1$

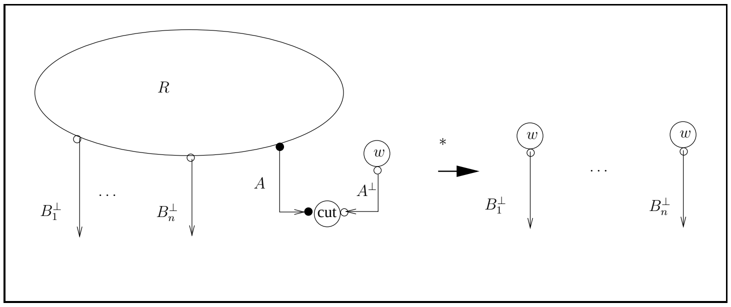


Figure 11: Statement of Lemma 12.

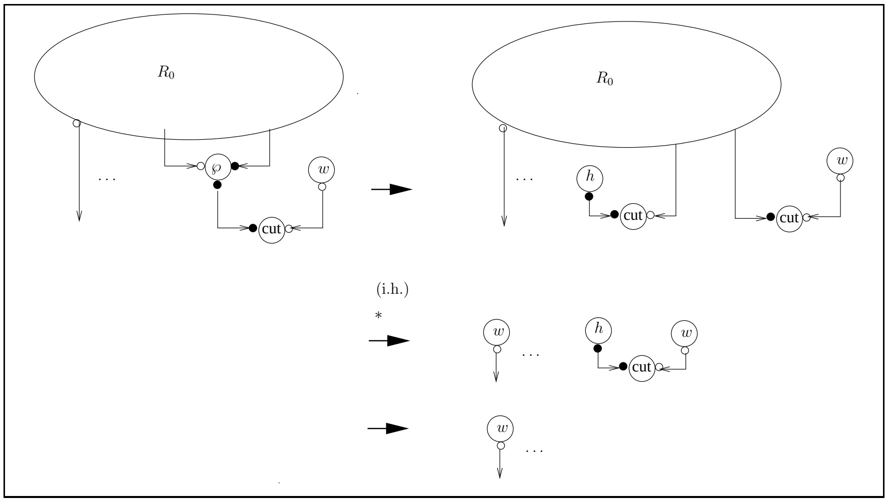


Figure 12: Case $R_{\multimap}$ of proof of Lemma 12.

among them with pending conclusion and the last one which is cut with an h-node. By applying one step of reduction to eliminate this cut (5th reduction step of Fig. 9), we finally obtain the expected proof-structure.

- a rule $L_{\multimap}$ on two proofs π_1 and π_2, with active formula $C \multimap D$: let $R_1 = \pi_1^*$ and $R_2 = \pi_2^*$, and denote by n_1 and n_2 their respective number of conclusions. The positive conclusion of the proof-net R_1 is typed with C, the positive conclusion of R_2 is typed with A, and R_2 has a negative conclusion typed with $D^\perp$. The proof-net R is obtained by linking R_1 and R_2 by an $\otimes$ node with positive premise C, negative premise $D^\perp$ and negative conclusion $C \otimes D^\perp$. The proof-structure S is obtained from R by cutting the positive conclusion A with the conclusion $A^\perp$ of a w-node. Now:

- by i.h. on π_2, S reduces to a proof-structure T consisting in: R_1 with a $\otimes$ node on its C conclusion and the $D^\perp$ conclusion of a w-node, together with $n_2 - 2$ w-nodes with pending conclusion.
- we apply a w / $\otimes$ garbage collection reduction rule (4th rule of Fig. 10) and obtain a proof-structure U consisting in R_1 cut with the conclusion of a w-node, and (n_2-1) w-nodes with pending conclusion;
- we apply the i.h. on R_1 and can conclude.

- a rule $R_\otimes$: we apply the first reduction step of Fig. 9 and then the i.h..
- a rule $L_\otimes$: we apply the i.h. and then the $w/\wp$ garbage collection reduction rule (5th rule of Fig. 10).
- a rule $R_\forall$ (resp. R_μ): we apply the 3rd (resp. 4th) reduction step of Fig. 9 and then the i.h..
- a rule $L_\forall$ (resp. L_μ): we apply the i.h. and then the corresponding garbage collection reduction rule (resp. 1st or 2nd rule of Fig. 10).
- a rule !: we apply the 6-th reduction step of Fig. 9.
- a rule *Contr*: we apply the i.h. and then the $w/?c$ garbage collection reduction rule (3rd rule of Fig. 10).
- a rule *weak*: we apply the i.h..
- a cut rule: we apply two times the i.h..
- a rule *Ax*: we apply an *ax* reduction step (first reduction step of Fig. 7).

This completes the proof. □

Now in order to state a confluence property for nets we actually need to consider a relaxed notion of equality between nets.

We say that two nets R and R' are *essentially equal* if:

- they have the same conclusions,
- they are equal put to the labeling of edges by formulas (types).

So two proof-nets are essentially equal if they have the same conclusions and are the same as *untyped* nets. Note that if two normal proof-nets of conclusion $\mathbf{B}$ (or $\mathbf{N}$, or $\mathbf{W}$) are essentially equal, then they are equal. Now we can state:

Proposition 13 (Confluence). *Assume that R is an EAL_μ^+ net and that $R \rightarrow R_1$ and $R \rightarrow R_2$. Then there exist R_1' and R_2' such that:*

- *$R_1 \rightarrow R_1'$ and $R_2 \rightarrow R_2'$,*
- *R_1' and R_2' are essentially equal.*

Proof :[Sketch] Note that we have not formally defined a notion of untyped net, but one can observe that the reduction does not need the types so it can be performed without them. If we prove the local confluence property then we can conclude by using the strong normalization property.

So let us examine local confluence. For that one needs to check the critical pairs of the reduction. The pairs coming from Fig. 7 and Fig. 8 do not raise any problem, essentially because we know that the reduction of ELL proof-nets is confluent. Among the critical pairs using at least one rule of either Fig. 9 or Fig. 10 only two raise problems:

- $\forall$ step of Fig. 7 and gc_1 step of Fig. 10: by applying a w_3 step of Fig. 9 one obtains two nets which are not in general equal, but essentially equal (this is why we defined this notion);
- negative $\otimes$ step of Fig. 7 and gc_4 step of Fig. 10: one can close the diagram by applying on one side (two times) Lemma 12, and on the other side step w_2 of Fig. 9, Lemma 12 (two times), and then step w_5 of Fig. 9.

Note that in the case of this second critical pair, for Lemma 12 we used the fact that nets can be sequentialized into EAL_μ^+ sequent calculus proofs. □

Now we have all the ingredients to prove:

Proposition 14. *If R is an EAL_μ proof-net and reduces to R' in normal form, then R is a proof-net, so in particular it does not contain any h-node.*

Proof : As we have seen with Theorem 11 that the reduction of nets is strongly normalizing, it is sufficient in order to prove the result to exhibit a strategy that erases all h-nodes created during the reduction. Observe that Lemma 12 defines a kind of big-step reduction for a weakening cut in a proof-net.

Consider a proof-net R. So it does not contain any h-node. Apply the following (non-deterministic) strategy to reduce R: repeat the following procedure until it is not possible anymore

- if there is at least one cut in the current proof-net S, choose a cut node N and:
  - if N is not a weakening cut, reduce it with the adequate reduction step of Fig. 7 or 8;
  - if N is a weakening cut: as S is a proof-net there exists a proof π such that $\pi^* = R$, and a sub-proof π_0 of π such that:
    - the last rule of π_0 is a cut on a A formula between two proofs π_1 of conclusion $\Gamma_1 \vdash t_1 : A$ and π_2 of conclusion $\Gamma_2, x : A \vdash t_2 : B$, and this cut rule corresponds to the node N;
    - the last rule of π_2 is a weakening rule on A.

    Let $R_1 = \pi_1^*$. Then R contains as sub-graph R_1 linked by the cut node N with the conclusion A of a w-node. Apply the big-step reduction of Lemma 12 to R_1 in order to reduce the cut N.

- if the current proof-net S does not contain any cut, then if one of the garbage collection step can be applied then trigger it.

  Observe that in all cases the resulting net S' does not contain any h-node.

Call R' the normal net obtained. This net does not contain any h-node, and so this concludes the proof. □

Remark 2. *In the rest of the paper we will follow the following garbage-collecting discipline when reducing proof-nets: whenever we reduce a weakening cut, we do a big-step weakening reduction as defined by Lemma 12. By this discipline, the reduction of a proof-net will always yield a proof-net.*

Moreover it turns out that proof-net reduction is consistent with lambda-term reduction in the following (weak) sense:

Proposition 15. *Let π be an EAL_μ proof of $\Gamma \vdash t : A$ and $R = \pi^*$. If R reduces to R' in normal form, then there exists an EAL_μ proof π' of conclusion $\Gamma \vdash t' : A$ such that:*

- *$t \to t'$ and t' is a normal form,*
- *we have $R' = \pi'^*$.*

Proof : To prove this property we will consider a new rewriting process $\rightsquigarrow$ on EAL_μ^+ sequent calculus proofs defined by:

- all ELL_μ sequent calculus cut-elimination steps are rewriting steps for $\rightsquigarrow$: this includes steps corresponding to all proof-structure reduction steps of Fig. 7 and 8, and the last reduction step of Fig. 9 ($w/!$ reduction step); if $\pi \rightsquigarrow \pi'$ by one of these steps and π has conclusion $\Gamma \vdash t : A$, then π' has a conclusion $\Gamma \vdash t' : A$ with $t \to t'$;
- new rewriting steps corresponding to the 5 first proof-structures reduction steps of Fig. 9 and the garbage collection reduction steps of Fig. 10:

$$\cfrac{\cfrac{\Gamma_1 \vdash t_1 : A_1 \quad \Gamma_2 \vdash t_2 : A_2}{\Gamma_1, \Gamma_2 \vdash t_1 \otimes t_2 : A_1 \otimes A_2}\, R_\otimes \qquad \cfrac{\Delta \vdash u : C}{z : A_1 \otimes A_2, \Delta \vdash u : C}\, \mathit{Weak}}{\Gamma_1, \Gamma_2, \Delta \vdash u : C}\, \mathit{Cut}$$

$$\rightsquigarrow$$

$$\cfrac{\Gamma_2 \vdash t_2 : A_2 \qquad \cfrac{\cfrac{\Gamma_1 \vdash t_1 : A_1 \quad \cfrac{\Delta \vdash u : C}{z_1 : A_1, \Delta \vdash u : C}\, \mathit{Weak}}{\Gamma_1, \Delta \vdash u : C}\, \mathit{Cut}}{z_2 : A_2, \Gamma_1, \Delta \vdash u : C}\, \mathit{Weak}}{\Gamma_1, \Gamma_2, \Delta \vdash u : C}\, \mathit{Cut}$$

$$\cfrac{\cfrac{\Gamma \vdash t : C}{\Gamma \vdash t : \forall\alpha.C}\, R_\forall \qquad \cfrac{\Delta \vdash u : A}{y : \forall\alpha.C, \Delta \vdash u : A}\, \mathit{Weak}}{\Gamma, \Delta \vdash u : A}\, \mathit{Cut} \quad \rightsquigarrow \quad \cfrac{\Gamma \vdash t : C \qquad \cfrac{\Delta \vdash u : A}{y : C, \Delta \vdash u : A}\, \mathit{Weak}}{\Gamma, \Delta \vdash u : A}\, \mathit{Cut}$$

similar rule for R_μ and $Weak$

$$\dfrac{\dfrac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \multimap B}\,R_{\multimap} \qquad \dfrac{\Delta \vdash u : C}{y : A \multimap B, \Delta \vdash u : C}\,Weak}{\Gamma, \Delta \vdash u : C}\,Cut$$

$$\rightsquigarrow$$

$$\dfrac{\dfrac{\dfrac{}{\vdash \bullet : A}\,h \qquad \Gamma, x : A \vdash t : B}{\Gamma \vdash t[\bullet/x] : B}\,Cut \qquad \dfrac{\Delta \vdash u : C}{z : B, \Delta \vdash u : C}\,Weak}{\Gamma, \Delta \vdash u : C}\,Cut$$

$$\dfrac{\dfrac{}{\vdash \bullet : A}\,h \qquad \dfrac{\Gamma \vdash t : B}{x : A, \Gamma \vdash t : B}\,Weak}{\Gamma \vdash t : B}\,Cut \quad \rightsquigarrow \quad \Gamma \vdash t : B$$

$$\dfrac{\dfrac{\Gamma \vdash t : B}{\Gamma, x : C[A/\alpha] \vdash t : B}\,Weak}{\Gamma, x : \forall \alpha.C \vdash t : B}\,L_{\forall} \quad \rightsquigarrow \quad \dfrac{\Gamma \vdash t : B}{\Gamma, x : \forall \alpha.C \vdash t : B}\,Weak$$

similar rule for L_μ and $Weak$

$$\dfrac{\dfrac{\dfrac{\pi_1}{\Gamma, x_1 :!A \vdash t : B}}{\Gamma, x_1 :!A, x_2 :!A \vdash t : B}\,Weak}{\Gamma, x :!A \vdash t[x/x_1] : B}\,Contr \quad \rightsquigarrow \quad \dfrac{\pi_1[x/x_1]}{\Gamma, x :!A \vdash t[x/x_1] : B}$$

$$\dfrac{\Gamma \vdash t : A \qquad \dfrac{\Delta \vdash u : C}{\Delta, x : B \vdash u : C}\,Weak}{\Gamma, \Delta, y : A \multimap B \vdash u : C}\,L_{\multimap} \quad \rightsquigarrow \quad \dfrac{\dfrac{\Gamma \vdash t : A \qquad \dfrac{\Delta \vdash u : C}{\Delta, z : A \vdash u : C}\,Weak}{\Gamma, \Delta \vdash u : C}\,Cut}{\Gamma, \Delta, y : A \multimap B \vdash u : C}\,Weak$$

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash t : C}{\Gamma, x_1 : A \vdash t : C}\,Weak}{\Gamma, x_1 : A, x_2 : B \vdash t : C}\,Weak}{\Gamma, x : A \otimes B \vdash \mathrm{let}\, x \,\mathrm{be}\, x_1 \otimes x_2 \,\mathrm{in}\, t : C}\,L_{\otimes} \quad \rightsquigarrow \quad \dfrac{\Gamma \vdash t : C}{\Gamma, x : A \otimes B \vdash t : C}\,Weak$$

Now, we consider EAL_μ^+ sequent calculus proofs up to the equivalence induced by commutation of rules, and write $\pi \rightsquigarrow \pi'$ if π' is obtained by rewriting a subproof of π with one of the rewriting rules described above. We also denote $\rightsquigarrow$ the transitive closure of this relation. Then:

1. by examining the various cases above, we observe that if $\pi \rightsquigarrow \pi'$ and π has conclusion $\Gamma \vdash t : A$, then π' has a conclusion $\Gamma \vdash t' : A$ with $t \to t'$;

(observe for instance that the (gc) lambda-calculus reduction rule of Fig. 1 is used for the last case of proof rewriting, corresponding to the 5th rule of Fig. 10)

2. we can see that if π is in normal form for $\rightsquigarrow$ and has conclusion $\Gamma \vdash t : A$, then t is in normal form for $\rightarrow$.
3. if π is an EAL_μ^+ proof, $R = \pi^\star$ and R reduces to R', then there exists an EAL_μ^+ proof π' such that $\pi \rightsquigarrow \pi'$ and $\pi'^\star = R'$; actually $\rightsquigarrow$ has precisely been defined for this purpose.

Now, in order to prove the statement consider π an EAL_μ proof of $\Gamma \vdash t : A$ and $R = \pi^*$. Assume also R reduces to R' in normal form. By the point 3. above we know that there exists π' an EAL_μ^+ proof such that $\pi'^* = R'$ and $\pi \rightsquigarrow \pi'$. Moreover π' is in normal form for $\rightsquigarrow$, because otherwise R' would not be in normal form. By Prop. 14 we know that R' does not contain any h-node, so π' is an EAL_μ proof. By the point 1. above, π' has a conclusion $\Gamma \vdash t' : A$ with $t \rightarrow t'$. Finally by point 2. t' is in normal form for $\rightarrow$. This concludes the proof. □

So in particular it follows from Prop. 15 that if π is a proof-net of conclusion $\vdash t : !^k D$, where $D = \mathbf{W}$ or another data-type, and $R = \pi^*$, then the reduction of R will compute the same value as the reduction of t.

To take advantage of proof-nets for computing, we want to define the application of a function to an argument with proof-nets as with sequent calculus. Consider R_1 and R_2 two proof-nets respectively of conclusions $\Gamma_1, ?^n A^\perp \wp !^m B$ and Γ_2, A. Let R_3 be the proof-net obtained by :

- applying n !-boxes to R_2 to obtain a proof-net of conclusion $?^n\Gamma_2, !^n A$;
- applying a $\otimes$ node to this latter proof-net and an axiom on $!^m B$ to obtain a proof-net R_2' of conclusion $?^n\Gamma_2, !^n A \otimes ?^m B^\perp, !^m B$;
- finally applying a *cut*-node to R_1 and R_2' to obtain R_3 with conclusion $\Gamma_1, ?^n\Gamma_2, !^m B$.

We say that R_3 is obtained by applying R_1 to R_2.

Just as for sequent calculus proofs we say that a function $f : \{0,1\}^\star \rightarrow \{0,1\}^\star$ is represented by an EAL_μ proof-net R_1 of conclusion $?^n\mathbf{W} \wp !^m\mathbf{W}$ if:

for any word $w \in \{0,1\}^\star$, if R_2 is a proof-net of conclusion $\mathbf{W}$ representing w, then R_1 applied to R_2 reduces to R_3 of conclusion $!^m\mathbf{W}$, and R_3 is obtained by applying m !-boxes to a proof-net R_3' representing $w' = f(w) \in \{0,1\}^\star$.

Thus the functions represented by EAL_μ proof-nets of conclusion $?^n\mathbf{W} \wp !^m\mathbf{W}$ are the functions of $\mathcal{F}_{EAL_\mu}(!^n\mathbf{W} \rightarrow !^m\mathbf{W})$.

We define similarly the functions represented by EAL_μ (resp. EAL) proof-nets of conclusions $?^n\mathbf{W}^\perp \wp !^m\mathbf{B}$ and $?^n\mathbf{N}^\perp \wp !^m\mathbf{N}$, and note that they correspond to the functions of $\mathcal{F}_{\mathcal{S}}(!^n\mathbf{W} \rightarrow !^m\mathbf{B})$ and $\mathcal{F}_{\mathcal{S}}(!^n\mathbf{N} \rightarrow !^m\mathbf{N})$ for $\mathcal{S} = EAL_\mu$ (resp. $\mathcal{S} = EAL$).

Proposition 16. *Let R be a proof-net of conclusion $\mathbf{B}$ (resp. $\mathbf{N}$ or $\mathbf{W}$) and R' be its normal form.*

Consider now S a proof-net obtained by reduction of R and which is normal at depth 0 (resp. depth inferior or equal to 1). Denote by $\overline{S}$ the restriction of S to nodes at depth 0 (resp. to nodes at depth inferior or equal to 1). Then we have $\overline{S} = R'$.

It follows that to obtain the normal form of a proof-net of conclusion $\mathbf{B}$ *(resp.* $\mathbf{N}$ *or* $\mathbf{W}$*) it is sufficient to reduce it completely at depth 0 (resp. depth inferior or equal to 1) and to forget the part of the graph which is not at depth 0 (resp. depth inferior or equal to 1).*

Proof : Consider the case of a proof-net R of conclusion $\mathbf{B}$ and S obtained by reduction of R which is normal at depth 0. Now if we reduce S to a normal form S', as we know that the reduction of a proof-net at depth $i+1$ does not change its part at depth $k \leq i$, the restrictions $\overline{S}$ and $\overline{S'}$ of the two proof-nets to nodes at depth 0 are equal: $\overline{S} = \overline{S'}$.

Moreover by uniqueness of the normal form we have $S' = R'$ and by Lemma 2 we know that R' has depth 0. Therefore $\overline{S'} = R'$ and we can conclude that $\overline{S} = \overline{S'} = R'$.

We proceed analogously in the case of a proof-net of conclusion $\mathbf{N}$ or $\mathbf{W}$. □

3.3. Extensional expressivity results

In this subsection we will show that some complexity classes are included in the classes defined by EAL types, and for that we will use a simulation of Turing machines.

Lemma 17. *Let q be a polynomial over one variable, with coefficients in $\mathbb{N}$. We have:*

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

Proof :

1. Addition and multiplication on tally integers can both be represented by proofs of conclusions $\mathbf{N} \multimap \mathbf{N} \multimap \mathbf{N}$, which are given in [DJ03] (Sect. 4.1 and 4.2) and correspond to the following λ-terms:

$$\begin{array}{ll} add = \lambda n.\lambda m.\lambda f.\lambda x.((n\ f)\ (m\ f\ x)) : & \mathbf{N} \multimap \mathbf{N} \multimap \mathbf{N}, \\ mult = \lambda n.\lambda m.\lambda f.(n\ (m\ f)) : & \mathbf{N} \multimap \mathbf{N} \multimap \mathbf{N}. \end{array}$$

   By composing these proofs, using one ! rule and several times the *Contr* rule, one can represent any polynomial by a proof of conclusion $!\mathbf{N} \multimap !\mathbf{N}$.
2. By applying the proof for multiplication to a proof representing the integer 2, one gets a proof representing the doubling function, *double*, with the following λ-term (where $\underline{2}$ is the Church integer for 2):

$$double = (mult\ \underline{2}) = (\lambda n.\lambda m.\lambda f.(n\ (m\ f)))\ \underline{2} : \quad \mathbf{N} \multimap \mathbf{N}.$$

By iterating this function, that is to say with the proof corresponding to the term $exp = \lambda n.(n\ double\ \underline{1})$, of type $\mathbf{N} \multimap !\mathbf{N}$, one represents the function 2^n.
By composing k times exp we obtain a term exp_k of type $\mathbf{N} \multimap !^k\mathbf{N}$ representing the function 2^n_k. Finally by composing the term exp_k with the term of type $!\mathbf{N} \multimap !\mathbf{N}$ representing $q(n)$, one gets a term of type $!\mathbf{N} \multimap !^{k+1}\mathbf{N}$ representing the function $2^{q(n)}_k$.

□

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 § modality by !), these constructions can be used in EAL. Let us denote by $\mathbf{Config_C}$ the following type for configurations for Turing machines over a binary alphabet with one tape and n states:

$$\mathbf{Config_C} = \forall\alpha.!(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha) \multimap !(\alpha \multimap \alpha \multimap (\alpha \otimes \alpha \otimes \mathbf{B^n})).$$

Elements of this type correspond to triples (l_1, l_2, s) where:

- l_1 and l_2 are two binary words representing respectively the left-hand-side (in reverse order) of the tape and the right-hand-side (in the right order), starting with the scanned symbol, of the tape;

- s represents the current state (described by an element of $\mathbf{B^n}$).

One could be tempted to use instead of this type the simpler one $T = \mathbf{W} \otimes \mathbf{W} \otimes \mathbf{B^n}$ but [AR02] shows that for any Turing machine it is possible to represent one step by a term $step_C : \mathbf{Config_C} \multimap \mathbf{Config_C}$, while it does not seem to be the case with T. Intuitively the difference between T and $\mathbf{Config_C}$ is that in elements of $\mathbf{Config_C}$ the representations of the two words l_1 and l_2 use the same bound variables s_0 and s_1.

Now we have:

Lemma 18. *Let $\mathcal{M}$ be a one-tape deterministic Turing machine over a binary alphabet. One can define terms:*

$$\begin{array}{rcl} init_C & : & \mathbf{W} \multimap \mathbf{Config_C}, \\ step_C & : & \mathbf{Config_C} \multimap \mathbf{Config_C}, \\ accept?_C & : & \mathbf{Config_C} \multimap !\mathbf{B}, \\ extract_C & : & \mathbf{Config_C} \multimap \mathbf{W}, \end{array}$$

such that:

- *given a binary word, $init_C$ produces the corresponding initial configuration of the machine,*

- *the term $step_C$ computes one step of the machine on a given configuration,*

- *given a configuration, the term $accept?_C$ returns true (resp. false) if its state is accepting (resp. rejecting),*

- *the term $extract_C$ 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 $init_C$ and $extract_C$ are easy, while $step_C$ is not: see [AR02], where these terms are defined for light affine logic. As to $accept?_C$, it is defined by a case distinction on the component of type $\mathbf{B^n}$ of the configuration. Note the ! modality on its codomain, which is due to the fact that in $\mathbf{Config_C}$, $\mathbf{B^n}$ is in the scope of a ! and that EAL does not admit the dereliction principle. □

Proposition 19 (Extensional expressivity). *Let f be a function of* k-FEXP *for $k \geq 0$, then there exists an EAL proof of $!\mathbf{W} \multimap !^{k+2}\mathbf{W}$ representing it.*

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

Proof : Let us first consider the case of the function f of k-FEXP. Let $\mathcal{M}$ be a one-tape deterministic machine over a binary alphabet, of time $2_k^{q(n)}$, computing f. By Lemma 17, there exists a proof of type $!\mathbf{N} \multimap !^{k+1}\mathbf{N}$ representing the function $2_k^{q(n)}$. We represent the configurations of $\mathcal{M}$ with the type $\mathbf{Config_C}$ and we know by Lemma 18 that there are:

- a proof $step_C$ of conclusion $\mathbf{Config_C} \multimap \mathbf{Config_C}$ representing one step of execution on a configuration.

- a proof $init_C$ of conclusion $\mathbf{W} \vdash \mathbf{Config_C}$ for producing the initial configuration,

- a proof $extract_C$ of conclusion $\mathbf{Config_C} \vdash \mathbf{W}$ for extracting the result.

Now, using $step_C$ and iteration on $\mathbf{N}$, one can define a proof of conclusion $\mathbf{N}, !\mathbf{Config_C} \vdash !\mathbf{Config_C}$, 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 $2_k^{q(n)}$ we also obtain a proof π of $!\mathbf{N}, !^{k+2}\mathbf{Config_C} \vdash !^{k+2}\mathbf{Config_C}$, which given an integer n and a configuration c, produces the configuration c' obtained by $2_k^{q(n)}$ computation steps starting from c.

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

- use duplication $!\mathbf{W} \multimap !\mathbf{W} \otimes !\mathbf{W}$, *length* and the coercion $\mathbf{W} \multimap !^{k+1}\mathbf{W}$ to obtain $!\mathbf{W} \multimap !\mathbf{N} \otimes !^{k+2}\mathbf{W}$,

- compose it with $id \otimes init_C$ to obtain $!\mathbf{W} \multimap !\mathbf{N} \otimes !^{k+2}\mathbf{Config_C}$,

- compose it with the proof π described above, to get $!\mathbf{W} \multimap !^{k+2}\mathbf{Config_C}$,

- finally compose with $extract_C$ to obtain a proof of $!\mathbf{W} \multimap !^{k+2}\mathbf{W}$.

The resulting proof represents the computation of the machine $\mathcal{M}$, 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 $extract_C$ of type $\mathbf{Config_C} \multimap \mathbf{W}$ we use $accept?_C$ of type $\mathbf{Config_C} \multimap !\mathbf{B}$ and we thus obtain a proof of $!\mathbf{W} \multimap !^{k+3}\mathbf{B}$.

□

3.4. Complexity properties

Now we can bring together the results of the previous subsections in order to provide some upper bounds and lower bounds on the classes of functions and predicates defined by EAL formulas.

Proposition 20. *We have:*

$$\begin{array}{rrcl} & \mathcal{F}_{EAL}(!\mathbf{W} \multimap !^k\mathbf{W}) & \subseteq & (\mathsf{k}+1)\text{-}\mathsf{FEXP}, \\ \textit{for } k \geq 2, & \mathcal{F}_{EAL}(!\mathbf{W} \multimap !^k\mathbf{W}) & \supseteq & (\mathsf{k}-2)\text{-}\mathsf{FEXP}, \\ \textit{and} & & & \\ & \mathcal{F}_{EAL}(!\mathbf{W} \multimap !^k\mathbf{B}) & \subseteq & \mathsf{k}\text{-}\mathsf{EXP}, \\ \textit{for } k \geq 3, & \mathcal{F}_{EAL}(!\mathbf{W} \multimap !^k\mathbf{B}) & \supseteq & (\mathsf{k}-3)\text{-}\mathsf{EXP}. \end{array}$$

These results also hold if we replace the types $!\mathbf{W} \multimap !^k\mathbf{W}$ *(resp.* $!\mathbf{W} \multimap !^k\mathbf{B}$*) by* $\mathbf{W} \multimap !^k\mathbf{W}$ *(resp.* $\mathbf{W} \multimap !^k\mathbf{B}$*).*

Proof : The second and fourth inclusions (right-to-left inclusions) are consequences of Proposition 19.

Let us prove the first inclusion. W.l.o.g. we can consider a proof-net R with two conclusions $?\mathbf{W}^{\perp}$, $!^k\mathbf{W}$. Call f the function that it represents. We want to prove that f belongs to $(\mathsf{k}+1)$-FEXP.

By Lemma 3 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 $!\mathbf{W}$ and cut it with R; call T the resulting proof-net of conclusion $!^k\mathbf{W}$. Thanks to Prop. 16 it is sufficient in order to obtain the result $f(w)$ of the computation to reduce T from depth 0 to depth $(k+1)$. Now, by the same method establishing Prop. 7 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 $(k+2)2^{3|T|}_{k+1}$.

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:

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

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 $(\mathsf{k}+1)$-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 $!^k\mathbf{B}$ it is sufficient to perform the reductions up to depth k in order to obtain the result, thanks to Prop. 16. We thus get in the end a time bound of the form $O(2^{R(n)}_{k})$, and the predicate represented therefore belongs to k-EXP.

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

We used in Prop. 20 the types $!\mathbf{W} \multimap !^k\mathbf{W}$ and $!\mathbf{W} \multimap !^k\mathbf{B}$, instead of types of the form $\mathbf{N} \multimap !^k\mathbf{N}$ as in [DJ03], in order to make easier the comparison with the improved results in the next Section (Theorem 22).

Observe in Prop. 20 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. 21). 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_μ and by improving both the result on the upper bound and that on the lower bound.

Proposition 21. *We have:*

$$\mathcal{F}_{EAL}(\cup_k(\mathbf{W} \multimap !^k\mathbf{W})) = \mathsf{FELEM}.$$

Actually it can be shown in a similar way that:

$$\mathcal{F}_{EAL}(\cup_k(\mathbf{N} \multimap !^k\mathbf{N})) = \mathsf{FELEM}.$$

The latter statement is the result stated in [Gir98] and proved in [DJ03].
Proof :[of Proposition 21]

By Proposition 20 we have:

$$\cup_k\mathcal{F}_{EAL}(\mathbf{W} \multimap !^k\mathbf{W}) = \cup_k\mathsf{k}\text{-}\mathsf{FEXP} = \mathsf{FELEM}.$$

□

4. Characterization of the classes P and k-EXP

This Section is devoted to the main result of this paper: while in Prop. 20 we only had lower bounds and upper bounds for the classes of predicates characterized by the type hierarchy $!\mathbf{W} \multimap !^k\mathbf{B}$, we will give here an exact characterization. This will be obtained at the price of switching from the system EAL to the system EAL_μ with type fixpoints. The reason why we need fixpoints will be explained in. Sect. 4.2.

Let us first state the result:

Theorem 22. *We consider the system EAL_μ. We have:*

$$\begin{array}{rcl} \mathcal{F}_{EAL_\mu}(!\mathbf{W} \multimap !^2\mathbf{B}) & = & \mathsf{P}, \\ \mathcal{F}_{EAL_\mu}(!\mathbf{W} \multimap !^3\mathbf{B}) & = & \mathsf{EXP}. \end{array}$$

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

$$\mathcal{F}_{EAL_\mu}(!\mathbf{W} \multimap !^{k+2}\mathbf{B}) = \mathsf{k\text{-}EXP}.$$

This theorem will be proved in the rest of this section.

Remark 3. *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.*

Remark 4. *With the type $!\mathbf{W} \multimap !\mathbf{B}$ 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 7.

Note that with respect to Prop. 7, the improvement of Theorem 22 is of two degrees of exponentials for the upperbound (bounding $\mathcal{F}_{EAL_\mu}(!\mathbf{W} \multimap !^{k+2}\mathbf{B})$ by $\mathsf{k\text{-}EXP}$ instead of $\mathsf{(k+2)\text{-}EXP}$) *and* of one degree of exponentials for the lower bound ($\mathsf{k\text{-}EXP}$ instead of $\mathsf{(k-1)\text{-}EXP}$)). To obtain these results we will adapt essentially the same reduction strategy and methodology seen in the previous section, with the following modifications:

- 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;
- on the methodology for obtaining the upper bound: we will use the assumption that the proof has a conclusion $!\mathbf{W} \multimap !^{k+2}\mathbf{B}$ and we will make a finer analysis of the size increase;
- on the methodology for obtaining the lower bound: we will take advantage of the fact that we are considering EAL_μ 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

We will now state our key Lemma. Essentially it shows that if we have a bound on the number of k of cuts of a proof-net R at depth 0, then we can give a polynomial bound on the size of the proof-net R' obtained after reduction at depth 0.

Remember that we follow the garbage-collection discipline of Remark 2 and thus the reduction of a proof-net yields a proof-net.

Lemma 23 (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 from R by performing all reductions at depth 0. Then we have:

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

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 and weakening 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 Lemma 12, 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 4, R admits a special cut c. Call $\mathcal{B}$ the box whose principal door (a !-node) is a premise of 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 $\mathcal{B}$, 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 $|R|_a + 1 \leq |R|_0$. □

Remark 5.

- *Observe that in any case we can apply Lemma 23 by bounding k by $|R|_0$, but then we basically recover the same kind of bound as in Section 3.*
- *In the statement of Lemma 23 we mentioned the proof-net R' obtained by reduction of R at depth 0: this expression makes sense because the reduction of cuts at a given depth is confluent.*

We will need another result:

Lemma 24 (Readback). *Let R be a proof-net of conclusion* $\mathbf{B}$ *which only has exponential cuts at depth 0.*

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

Moreover, one can in time $O(|R|_0)$ reduce R to its normal form, which represents either true or false.

Proof : We have $\mathbf{B} = \forall\alpha.\alpha^{\perp}\wp\alpha^{\perp}\wp\alpha$. Consider the conclusion $\mathbf{B}$ and the axiom $\alpha^{\perp}, \alpha$ introducing the r.h.s. litteral α of this $\mathbf{B}$. 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 auxilliary 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 $\mathbf{B}$, and thus this $\alpha^{\perp}$ litteral is one of the two occurrences of $\alpha^{\perp}$ in $\forall\alpha.(\alpha^{\perp}\wp\alpha^{\perp}\wp\alpha)$. 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. Therefore we have proved the first statement.

Let us now prove the second statement. W.l.o.g. let us assume for instance that we are in the situation where the occurrence of α in the conclusion $\forall\alpha.(\alpha^{\perp}\wp\alpha^{\perp}\wp\alpha)$ is linked by an axiom N to the leftmost occurrence of $\alpha^{\perp}$ (so the result of the reduction will be true). Consider the second occurrence of $\alpha^{\perp}$: it can have been introduced either by an axiom or by a weakening node. Assume for a contradiction that it has been introduced by an axiom N', and examine the second conclusion α of this axiom:

- by repeating the same argument as above, it cannot be above a cut node;
- therefore it must be above a conclusion.

But the only conclusion of the proof-net is $\forall\alpha.(\alpha^{\perp}\wp\alpha^{\perp}\wp\alpha)$ and the only occurrence of the litteral α in this conclusion is introduced by another axiom. Therefore this occurrence of $\alpha^{\perp}$ is introduced by a weakening node.

So we deduce from that that as a graph R consists of two parts R_1 and R_2: R_1 is the connected component containing the conclusion $\mathbf{B}$ and R_2 is the rest of the graph. The two graphs R_1 and R_2 are proof-structures and:

- we have seen that R_1 is a cut-free proof-net representing true or false;
- R_2 is a proof-structure with no conclusion and only exponential cuts at depth 0.

Now, we want to prove the following property by induction over the number of cuts of R_2 at depth 0:

(P): R can be reduced to R_1 in at most $|R_2|_0$ steps, which can be of two kinds: 6th step of Fig. 9 (w/ !-box), 3rd step of Fig. 10.

Consider first the case where R_2 has 0 cut at depth 0. A proof-structure with no conclusion and no cut at depth 0 is an empty proof-structure, so in this case we have $R = R_1$, hence R is already in normal form.

Now assume the i.h. for n and suppose that R_2 has $(n+1)$ cuts. We want to prove that one of these cuts is a cut between a w node and a !. For that consider the relation $\prec$ between cut nodes defined as follows:

Let c_1 be a cut between $!A_1$ / $?A_1^\perp$ and c_2 be a cut between $!A_2$ / $?A_2^\perp$;

$c \prec c'$ if there is a box $\mathcal{B}$ or an axiom N at depth 0 such that: $\mathcal{B}$ has an auxilliary door of conclusion $?A_1^\perp$ above the premise of c_1, and a principal door of conclusion $!A_2$ which is a premise of c_2; or the axiom N has a conclusion $?A_1^\perp$ above the premise of c_1, and the other conclusion $!A_2$ which is a premise of c_2.

Now, assume for a contradiction that the relation $\prec$ has a cycle, that is to say that there exists $n \geq 1$ and cuts $c_1, \ldots, c_n$ such that:

$$c_1 \prec c_2 \prec \cdots \prec c_n \prec c_1.$$

As R is a proof-net there exists a sequent calculus proof π such that $\pi^* = R$, and each c_i corresponds to an occurrence of cut rule in π. However we observe that because of the cycle, none of these cut rules has been applied first in π (reading the proof bottom-up). Hence a contradiction.

Therefore $\prec$ does not have any cycle, so as the number of cuts is finite $\prec$ admits a maximal element c_0. Call $!A_0$ / $?A_0^\perp$ the premises of c_0 and consider the exponential tree rooted at the $?A_0^\perp$ premise. Assume one of its leaves is an auxilliary door of a box $\mathcal{B}_0$ or the conclusion of an axiom node. Let us consider the case of the box, since that of the axiom is similar. Then $\mathcal{B}_0$ has a principal door, of conclusion say $!A'$. This principal door cannot be above a conclusion of R, since the only conclusion of the proof-net is $\mathbf{B}$. Therefore it must be above a cut c'. Hence $c_0 \prec c'$, which contradicts the maximality of c_0.

Therefore none of the leaves of the exponential tree of $?A_0^\perp$ is an auxilliary door of a box or a conclusion of an axiom, so all these leaves must be w nodes. By applying several times the third reduction step of Fig. 10 we obtain a w node as premise of c_0. We can thus reduce this cut by using the 6th reduction step of Fig. 9 and thus obtain R_2' with n cuts. Call $R' = R_1 \cup R_2'$ the new proof-net obtained. By i.h. R' satisfies (P) and so we can conclude that R satisfies (P).
□

Finally we get:

Proposition 25 (P soundness). *Let R be a normal proof-net of conclusion* $!\mathbf{W} \vdash !^2\mathbf{B}$. *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 reduced in time bounded by $P(n)$.

Proof :

First, recall that by Lemma 3 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 $?\mathbf{W}^\perp$ is obtained by a weakening it is trivial. Otherwise there is an integer $k \geq 1$ and a proof-net S of conclusion $\vdash \mathbf{W}^\perp, \ldots, \mathbf{W}^\perp, !\mathbf{B}$ with k formulas $\mathbf{W}^\perp$ such that: R is obtained from S by applying a !-box and a certain number k' of contraction rules on $?\mathbf{W}^\perp$ formulas.

Now let R_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 $2k'$ 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 $\mathbf{W} = \forall \alpha.?(\alpha \otimes \alpha^\perp)\wp?(\alpha \otimes \alpha^\perp)\wp!(\alpha^\perp \wp \alpha)$, 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 23 to T'' at depth 1, we get that by reducing T'' at depth 1 we obtain $T^{(3)}$ which is normal at depths 0, 1 and such that $|T^{(3)}|_2 \leq |T''|_1^{3k} \cdot |T''|_2 \leq (|R| + k \cdot |R_w|)^{3k+1} + k$. The important point to notice here is that $(3k+1)$ does not depend on n.

Finally we perform on $T^{(3)}$, at depth 2, all reductions but those of exponential cuts. These only make the size decrease, by Lemma 5, and so the number of steps is bounded by $|T^{(3)}|_2$. We obtain in this way a proof-net $T^{(4)}$ of conclusion $!^2\mathbf{B}$, which:

- is normal at depths 0 and 1,
- only has exponential cuts at depth 2.

So we have shown that T can be reduced to $T^{(4)}$ in a number of steps which is polynomially bounded with respect to R_w, hence with respect to n.

What about the time needed to reduce T to $T^{(4)}$ on a Turing machine? Note that in what precedes we have bounded the size of intermediary proof-nets obtained by reduction until reaching $T^{(4)}$. The bounds given are polynomially bounded by $|R_w|$, hence by n. As already argued in the proof of Prop. 20 there exists a polynomial Q such that: any reduction step on a proof-net R can be simulated on a Turing machine in time $Q(|R|)$. So together the bound on the size of intermediary proof-nets and the bound on the number of steps for reaching $T^{(4)}$ show that the reduction from T to $T^{(4)}$ can be performed in time polynomial in n.

Now, coming back to $T^{(4)}$, as it is a proof-net of conclusion $!^2\mathbf{B}$ which does not have any cut at depth inferior or equal to 1, and has only exponential cuts at depth 2, by Lemma 24, applied here at depth 2, $T^{(4)}$ can then be reduced to its normal form in time $O(|T^{(4)}|_2)$. Moreover recall that $|T^{(4)}|_2 \leq |T^{(3)}|_2$ which is polynomially bounded by n. So on the whole the computation has been carried out in time polynomial in n. □

Proposition 26 (k-EXP soundness). *Let R be a normal proof-net of conclusion $!\mathbf{W} \vdash !^{k+2}\mathbf{B}$. 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 reduced in time bounded by $2_k^{P(n)}$.

Proof : We will proceed in a way generalizing the proof of Prop. 25, 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 27. *We have the following bounds:*

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

Proof :[of Lemma 27]

Consider statement (1). One can check that the function over $\mathbb{R}$ defined by $x \to (2^x - 2x)$ 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. □

Lemma 28. *Let R be a normal proof-net of conclusion $!\mathbf{W} \vdash !^{k+2}\mathbf{B}$ 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 $2_{i-2}^{2Q(n)}$ steps reduce S into a proof-net S' such that $|S'|_{i+} \leq 2_{i-2}^{Q(n)}$ and which is normal at depth inferior or equal to $i-1$, and only has exponential cuts at depth i.

Proof :[of Lemma 28]

We proceed by induction on i.

In the case where $i = 2$: we proceed as in the proof of Prop. 25; the arguments used are still valid in the case of a proof-net of conclusion $!\mathbf{W} \vdash !^{k+2}\mathbf{B}$ 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 $2_{i-2}^{2Q(n)}$ steps a proof-net S' with $|S'|_{i+} \leq 2_{i-2}^{Q(n)}$ and which is normal at depth inferior or equal to $i-1$, and only has exponential cuts at depth i. We reduce the exponential cuts at depth i following the strategy of the proof of Prop. 7, 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 is normal at depth inferior or equal to i. We thus have:

$$\begin{aligned}
|S''|_{(i+1)+} &\leq |S'|_{(i+1)+} \cdot 2^{|S'|_i} \\
&\leq 2^{|S'|_{(i+1)+} + |S'|_i} \\
&= 2^{|S'|_{i+}} \\
&\leq 2^{2^{Q(n)}_{i-2}}, \text{ by induction hypothesis,} \\
&= 2^{Q(n)}_{i-1}
\end{aligned}$$

Finally we do all non-exponential reductions 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:

$$2^{2Q(n)}_{i-2} + |S'|_i + |S''|_{(i+1)} \leq 2^{2Q(n)}_{i-2} + 2^{Q(n)}_{i-2} + 2^{Q(n)}_{i-1}.$$

We want to show that this sum is bounded by $2^{2Q(n)}_{i-1}$.

By Lemma 27 (1) applied to $x = Q(n)$ we have $2Q(n) \leq 2^{Q(n)}$ and thus $2^{2Q(n)}_{i-2} \leq 2^{2^{Q(n)}}_{i-2} = 2^{Q(n)}_{i-1}$. Moreover we have $2^{Q(n)}_{i-2} \leq 2^{Q(n)}_{i-1}$. We can assume that $Q(n) \geq 2$ and the number of steps is bounded by:

$$\begin{aligned}
2^{2Q(n)}_{i-2} + 2^{Q(n)}_{i-2} + 2^{Q(n)}_{i-1} &\leq 3 \cdot 2^{Q(n)}_{i-1} \\
&\leq (2^{Q(n)}_{i-1})^2, \text{ because as } Q(n) \geq 3, 2^{Q(n)}_{i-1} \geq 3 \\
&\leq 2^{2Q(n)}_{i-1}, \text{ by Lemma 27 (2) with } x = Q(n).
\end{aligned}$$

The induction hypothesis is thus valid for $i+1$ and we are done. □

Let us come back to the proof of Prop. 26. 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 $!^{k+2}\mathbf{B}$. Now, we apply Lemma 28 with $i = k+2$ and thus obtain in at most $2^{2Q(n)}_k$ reduction steps a proof-net S' which is normal at depth inferior or equal to $k+1$, and with 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 $\mathbf{B}$ and only exponential cuts at depth 0. By Lemma 24 one can reduce T', and thus S', in time $O(|T'|_0)$.

Therefore, on the whole after at most $2^{2Q(n)}_k$ reduction steps one obtains the normal form of S. Moreover as in the proof of Prop. 7 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. 20 that the evaluation can be done in time $O(2^{P(n)}_k)$ on a Turing machine, for some polynomial P.

□

4.2. Proof of the extensional completeness results

We will now prove extensional completeness results. In order to improve the results of Prop. 20, that is to say to obtain a tighter match between the upper bounds and lower bounds for each type in the hierarchy $!\mathbf{W} \multimap !^k\mathbf{B}$, we need to simulate n-Exptime Turing machine execution with a type $!\mathbf{W} \multimap !^k\mathbf{B}$ with an integer k as small as possible. Using the type of configurations $\mathbf{Config_C}$ based on Church integers we had obtained in Prop. 19 a simulation of Ptime Turing machine execution with a type $!\mathbf{W} \multimap !^3\mathbf{B}$. Here by employing a new datatype using type fixpoint we will be able to define a new type for configurations which will allow us to simulate Ptime Turing machine execution with the type $!\mathbf{W} \multimap !^2\mathbf{B}$.

This new datatype is that of *Scott binary words*:

$$\mathbf{W_S} = \mu\beta.\forall\alpha.(\beta \multimap \alpha) \multimap (\beta \multimap \alpha) \multimap (\alpha \multimap \alpha).$$

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 $\mathbf{W_S}$:

$$\begin{array}{lcllcl} cons_0 & = & \lambda w.\lambda s_0.\lambda s_1.\lambda x.(s_0\ w) & : & \mathbf{W_S} \multimap \mathbf{W_S} \\ cons_1 & = & \lambda w.\lambda s_0.\lambda s_1.\lambda x.(s_1\ w) & : & \mathbf{W_S} \multimap \mathbf{W_S} \\ nil & = & \lambda s_0.\lambda s_1.\lambda x.x & : & \mathbf{W_S} \\ tail & = & \lambda w.(w\ id\ id\ nil) & : & \mathbf{W_S} \multimap \mathbf{W_S} \end{array}$$

where $id = \lambda x.x$.

For this datatype we can define a term:

$$\begin{array}{lcl} case & : & \forall\alpha.(\mathbf{W_S} \multimap \alpha) \multimap (\mathbf{W_S} \multimap \alpha) \multimap \alpha \multimap (\mathbf{W_S} \multimap \alpha) \\ case & = & \lambda F_0.\lambda F_1.\lambda a.\lambda w.(w\ F_0\ F_1\ a) \end{array}$$

Moreover we can define a conversion from Church binary words to Scott binary words:

$$convert_{CS} \quad = \quad \lambda w.(w\ cons_0\ cons_1\ nil) : \mathbf{W} \multimap !\mathbf{W_S}$$

Remark 6. *Actually Scott words can also be typed in elementary affine logic, without fixed points, e.g. with the following type:*

$$\forall P.(U[P] \multimap U[P] \multimap P \multimap P),$$

where

$$U[P] = \forall X.((X \multimap X \multimap P \multimap P) \multimap P).$$

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:

$$\mathbf{Config_S} = \mathbf{W_S} \otimes \mathbf{B} \otimes \mathbf{W_S} \otimes \mathbf{B^n}.$$

While $\mathbf{Config_C}$ was based on the Church encoding of words, $\mathbf{Config_S}$ 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 29. *Let $\mathcal{M}$ be a one-tape deterministic Turing machine. One can define terms:*

$$\begin{array}{rcl} init_S & : & \mathbf{W_S} \multimap \mathbf{Config_S}, \\ step_S & : & \mathbf{Config_S} \multimap \mathbf{Config_S}, \\ accept?_S & : & \mathbf{Config_S} \multimap \mathbf{B}, \end{array}$$

with the same property as the terms of Lemma 18. Recall in particular that given a configuration, the term accept?$_S$ returns true (resp. false) if its state is accepting (resp. rejecting).

Proof :

We use as syntactic sugar a m-ary version of the $\otimes$-elimination:

$$\mathrm{let}\, t\, \mathrm{be}\, x_1 \otimes \cdots \otimes x_m\, \mathrm{in}\, u.$$

We define:

$$\begin{array}{rcl} accept?_S & = & \lambda c.\mathrm{let}\, c\, \mathrm{be}\, x_1 \otimes x_2 \otimes x_3 \otimes x_4\, \mathrm{in}\, (x_4\ b_1 \ldots\ b_n), \\ & : & \mathbf{Config_S} \multimap \mathbf{B} \end{array}$$

where for $1 \leq i \leq n$, b_i is $true$ (resp. $false$) if the state q_i encoded by the ith element of type $\mathbf{B}^n$ is accepting (resp. non accepting) for the machine.

We also set:

$$\begin{array}{rcl} init_S & = & case\ t_1\ t_2\ t_3 \\ & : & \mathbf{W_S} \multimap \mathbf{Config_S} \end{array}$$

where:

$$\begin{array}{rclcl} t_1 & = & \lambda s.nil \otimes true \otimes s \otimes \overline{1} & : & \mathbf{W_S} \multimap \mathbf{Config_S} \\ t_2 & = & \lambda s.nil \otimes false \otimes s \otimes \overline{1} & : & \mathbf{W_S} \multimap \mathbf{Config_S} \\ t_3 & = & nil \otimes false \otimes nil \otimes \overline{1} & : & \mathbf{Config_S} \end{array}$$

and $\overline{1}$ is the first element of the type $\mathbf{B}^n$, which by convention is chosen to correspond to the initial state.

Note that the third argument of *case* here (t_3) is actually just a dummy argument because we do not really care about the behaviour of $init_S$ on the empty word.

As to the term $step_S$ it can be constructed based on the transition function of $\mathcal{M}$, by doing a case distinction, using the term *case*, as in [DLB06] (Sect.7, Lemma 4). The term $step_S$ obtained this way is actually simpler than $step_C$.

□

Note that the main difference with Lemma 18 is the type of *accept*?, which is here $\mathbf{Config_S} \multimap \mathbf{B}$ instead of $\mathbf{Config_C} \multimap !\mathbf{B}$.

Proposition 30 (Extensional completeness). *Let $k \geq 0$. Consider a function g representing a predicate of* k-EXP. *Then there exists an EAL_μ proof of conclusion $!\mathbf{W} \vdash !^{k+2}\mathbf{B}$ representing g.*

Proof :

We will proceed in a way similar to that of the proof of Prop. 19, but we will have to change some types and some minor steps.

Consider $\mathcal{M}$ a one-tape deterministic machine over a binary alphabet, of time $2_k^{q(n)}$, computing a predicate on binary words f. Recall that by Lemma 17 there exists a proof of type $!\mathbf{N} \multimap !^{k+1}\mathbf{N}$ representing the function $2_k^{q(n)}$. We represent the configurations of $\mathcal{M}$ with the type $\mathbf{Config_S}$ and by Lemma 29 we have:

- a proof $step_S$ of conclusion $\mathbf{Config_S} \multimap \mathbf{Config_S}$ representing one step of execution on a configuration.
- a proof $init_S$ of conclusion $\mathbf{W_S} \multimap \mathbf{Config_S}$ producing the initial configuration,
- a proof $accept?_S$ of conclusion $\mathbf{Config_S} \multimap \mathbf{B}$ for extracting the result.

By using $step_S$ and iteration on $\mathbf{N}$, one defines a proof of conclusion $\mathbf{N}, !\mathbf{Config_S} \vdash !\mathbf{Config_S}$, which given an integer n and a configuration c, produces the configuration c' obtained by n computation steps starting from c. With the representation of $2_k^{q(n)}$ we then obtain a proof π of $!\mathbf{N}, !^{k+2}\mathbf{Config_S} \vdash !^{k+2}\mathbf{Config_S}$, such that when it is given an integer n and a configuration c, it produces the configuration c' obtained by $2_k^{q(n)}$ computation steps starting from c.

Now, the proof for the simulation is obtained by composing the following constructions:

- use duplication $!\mathbf{W} \multimap !\mathbf{W} \otimes !\mathbf{W}$, $length : \mathbf{W} \multimap \mathbf{N}$, a coercion $\mathbf{W} \multimap !^k\mathbf{W}$ and $convert_{CS} : \mathbf{W} \multimap !\mathbf{W_S}$ to obtain a proof of $!\mathbf{W} \multimap !\mathbf{N} \otimes !^{k+2}\mathbf{W_S}$ which gives the length of the input as a tally integer, and a copy of the input as a Scott binary word,
- compose it with $id \otimes init_S$ to obtain $!\mathbf{W} \multimap !\mathbf{N} \otimes !^{k+2}\mathbf{Config_S}$,
- compose it with the proof π we have described above, to get a proof of $!\mathbf{W} \multimap !^{k+2}\mathbf{Config_S}$,
- finally compose with $accept?_S$ to obtain a proof of $!\mathbf{W} \multimap !^{k+2}\mathbf{B}$.

Note that in the last step we have used the fact that $accept?_S$ is of type $\mathbf{Config_S} \multimap \mathbf{B}$, and this is actually the main difference with the proof of Prop. 19.

The final proof obtained represents the predicate f. □

Finally, together the results of Propositions 25, 26 and 30 establish Theorem 22.

Remark 7. *Observe that proofs of type* $!\mathbf{W} \vdash !^2\mathbf{W}$ *do not correspond to polynomial time functions. Indeed, one can easily define a proof wdouble of conclusion* $\mathbf{W} \multimap \mathbf{W}$ *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* $\mathbf{W} \vdash !\mathbf{W}$*, which, given a word of length* n*, produces a word of length* 2^n*. Applying the* ! *rule one thus obtains a proof of* $!\mathbf{W} \vdash !^2\mathbf{W}$ *with the same behaviour.*

To characterize the complexity class FP *of polynomial time functions, one could use the type* $!\mathbf{W} \multimap !^2\mathbf{W_S}$*, where* $\mathbf{W_S}$ *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 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? Does the complexity bound depend on the reduction strategy? Could one also characterize in this system LINSPACE 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 $\neq$ 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. Many thanks also to Olivier Laurent, for important discussions about proof-nets, and to Christophe Raffalli for informations about the typing of Scott integers.

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

References

[AB09] S. Arora and B. Barak. *Computational Complexity - A Modern Approach.* Cambridge University Press, 2009.

[ACM04] A. Asperti, P. Coppola, and S. Martini. (Optimal) duplication is not elementary recursive. *Information and Computation*, 193:21–56, 2004.

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

[Bai11] P. Baillot. Elementary linear logic revisited for polynomial time and an exponential time hierarchy. In *Proceedings of 9th Asian Symposium on Programming Languages and Systems (APLAS 2011)*, volume 7078 of *LNCS*, pages 337–352. Springer, 2011.

[BC92] S. Bellantoni and S. Cook. New recursion-theoretic characterization of the polytime functions. *Computational Complexity*, 2:97–110, 1992.

[BGM10] P. Baillot, M. Gaboardi, and V. Mogbil. A polytime functional language from light linear logic. In *Proceedings of European Symposium on Programming (ESOP 2010)*, volume 6012 of *LNCS*, pages 104–124. Springer, 2010.

[BMM11] G. Bonfante, J.-Y. Marion, and J.-Y. Moyen. Quasi-interpretations: a way to control resources. *Theor. Comput. Sci.*, 412(25):2776–2796, 2011.

[BP01] P. Baillot and M. Pedicini. Elementary complexity and geometry of interaction. *Fundamenta Informaticae*, 45(1-2):1–31, 2001.

[BT09] P. Baillot and K. Terui. Light types for polynomial time computation in lambda calculus. *Inf. Comput.*, 207(1):41–62, 2009. (A preliminary conference version appeared in the proceedings of LICS'04).

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

[CDLRDR08] P. Coppola, U. Dal Lago, and S. Ronchi Della Rocca. Light logics and the call-by-value lambda calculus. *Logical Methods in Computer Science*, 4(4), 2008.

[DJ03] V. Danos and J.-B. Joinet. Linear logic & elementary time. *Information and Computation*, 183:123–137, 2003.

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

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

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

[GMRDR08] M. Gaboardi, J.-Y. Marion, and S. 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.

[GRDR07] M. Gaboardi and S. Ronchi Della Rocca. A soft type assignment system for lambda-calculus. In *Proceedings of Computer Science Logic (CSL 2007)*, volume 4646 of *LNCS*, pages 253–267. Springer, 2007.

[HAH11] J. Hoffmann, K. Aehlig, and M. Hofmann. Multivariate amortized resource analysis. In *Proceedings of Symposium on Principles of Programming Languages (POPL 2011)*, pages 357–370. ACM, 2011.

[HJ06] M. Hofmann and S. Jost. Type-based amortised heap-space analysis. In *Proceedings of European Symposium on Programming (ESOP 2006)*, volume 3924 of *LNCS*, pages 22–37. Springer, 2006.

[Hof03] M. Hofmann. Linear types and non-size-increasing polynomial time computation. *Inf. Comput.*, 183(1):57–85, 2003.

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

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

[Lam96] F. Lamarche. From proof nets to games. *Electr. Notes Theor. Comput. Sci.*, 3:107–119, 1996.

[Lei94a] D. Leivant. A foundational delineation of poly-time. *Information and Computation*, 110(2):391–420, 1994.

[Lei94b] D. Leivant. Predicative recurrence and computational complexity I: word recurrence and poly-time. In *Feasible Mathematics II*, pages 320–343. Birkhauser, 1994.

[Lei02] D. Leivant. Calibrating computational feasibility by abstraction rank. In *Proceedings LICS'02*, pages 345–353. IEEE Computer Society, 2002.

[Maz06] Damiano Mazza. Linear logic and polynomial time. *Mathematical Structures in Computer Science*, 16(6):947–988, 2006.

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

[Ter07] Kazushige Terui. Light affine lambda calculus and polynomial time strong normalization. *Arch. Math. Log.*, 46(3-4):253–280, 2007.