Characterizing Polynomial and Exponential Complexity Classes in Elementary Lambda-Calculus

Patrick Baillot, Erika de Benedetti, Simona Ronchi Della Rocca

Characterizing Polynomial and Exponential Complexity Classes in Elementary Lambda-Calculus*

Patrick Baillot[1], Erika De Benedetti[1,2], and Simona Ronchi Della Rocca[2]

[1] CNRS, ENS de Lyon, INRIA, UCBL, Université de Lyon, LIP - Lyon, France

[2] Università degli Studi di Torino, Dipartimento di Informatica - Torino, Italy

Abstract. In this paper an implicit characterization of the complexity classes `k-EXP` and `k-FEXP`, for $k \geq 0$, is given, by a type assignment system for a stratified λ-calculus, where types for programs are witnesses of the corresponding complexity class. Types are formulae of Elementary Linear Logic (`ELL`), and the hierarchy of complexity classes `k-EXP` is characterized by a hierarchy of types.

Keywords: Implicit computational complexity · Linear logic · Lambda-calculus

1 Introduction

Context. Early work on the study of complexity classes by means of programming languages has been carried out by Neil Jones [10,11], in particular using functional programming. The interest of these investigations is twofold: from the computational complexity point of view, they provide new characterizations of complexity classes, which abstract away from machine models; from the programming language point of view, they are a way to analyze the impact on complexity of various programming features (higher-order types, recursive definitions, read/write operations). This fits more generally in the research line of *implicit computational complexity* (ICC), whose goal is to study complexity classes without relying on explicit bounds on resources but instead by considering restrictions on programming languages and calculi. Seminal research in this direction has been carried out in the fields of recursion theory [4,13], λ-calculus [15] and linear logic [9]. These contributions usually exhibit a new specific language

* This work was supported by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).

or logic for each complexity class, for instance PTIME, PSPACE, LOGSPACE: let us call *monovalent* the characterizations of this kind. We think however that the field would benefit from some more uniform presentations, which would consist in both a general language and a family of static criteria on programs of this language, each of which characterizing a particular complexity class. We call such a setting a *polyvalent* characterization; we believe that this approach is more promising for providing insights on the relationships between complexity classes. Polyvalent characterizations of this nature have been given in [11,14], but their criteria used for reaching point (2) referred to the construction steps of the programs. Here we are interested in defining a polyvalent characterization where (2) is expressed by means of the program's type in a dedicated system.

Stratification and Linear Logic. An ubiquitous notion in implicit complexity is that of *stratification*, by which we informally designate here the fact of organizing computation into distinct strata. This intuition underlies several systems: ramified and safe recursion [13,4], in which data is organized into strata; stratified comprehension [14], where strata are used for quantification; variants of linear logic [9] where programs are divided into strata thanks to a modality. More recently stratification of data has been related fruitfully to type systems for non-interference [18].
The linear logic approach to ICC is based on the proofs-as-programs correspondence. This logic indeed provides a powerful system to analyse the duplication and sharing of arguments in functional computation: this is made possible by a specific logical connective for the duplication of arguments, the ! modality. As in functional computation the reuse of an argument can cause a complexity explosion, the idea is to use weak versions of ! to characterize complexity classes. This intuition is illustrated by elementary linear logic (ELL) [9,8], a simple variant of linear logic which provides a monovalent characterisation of elementary complexity, that is to say computation in time bounded by a tower of exponentials of fixed height. Other variants of linear logic provide characterizations of PTIME, but they use either a more complicated language [9] or a more specific programming discipline [12].

Contribution and Comparison. In [2] a polyvalent characterization in ELL proof-nets of the complexity classes $\texttt{k-EXP} = \cup_{i\in\mathbb{N}}\texttt{DTIME}(2_k^{n^i})$ for all $k \geq 0$ has been obtained. However this approach has some shortcomings:

1. The complexity soundness proof uses a partly semantic argument ([2] Lemma 3 p. 10) and so it does not provide a syntactic way to evaluate the programs with the given complexity bound.

2. The characterization is given for classes of predicates, and not for classes of functions. Moreover it is not so clear how to extend this result to functions because of the semantic argument mentioned above.
3. The language of proof-nets is not as standard and widespread as say that of λ-calculus.

In the present work, we wish to establish an analogous polyvalent characterization in the setting of λ-calculus, with a stronger complexity soundness result based on a concrete evaluation procedure. We think this could provide a more solid basis to explore other characterizations of this kind. In particular we define the $\lambda^!$-calculus, a variant of λ-calculus with explicit stratifications, which allows both to recover the results of [2] and to characterize also the function complexity classes k-FEXP, by two distinct hierarchies of types. In fact, the characterization obtained through a standard representation of data-types like in [2] does not account for some closure properties of the function classes k-FEXP, in particular composition, so we propose a new, maybe less natural, representation in order to grasp these properties. Our language makes it easier to define such non-standard representation.

Technical Approach. One could expect that the results of [2] might be extended to $\lambda^!$-calculus by considering a translation of terms into proof-nets. However it is not so straightforward: term reduction cannot be directly simulated by the evaluation procedure in [2], because (i) it follows a specific cut-elimination strategy and (ii) ultimately it uses a semantic argument. For this reason we give here a direct proof of the result in $\lambda^!$-calculus, which requires defining new measures on terms and is not a mere adaptation of the proof-net argument.

Related Works. The first results on ELL [9,8] as well as later works [19,6] have been carried out in the setting of proof-nets. Other syntaxes have then been explored. First, specific term calculi corresponding to the related system LLL and to ELL have been proposed [22,17,16]. Alternatively [5] used standard λ-calculus with a type system derived from ELL. The $\lambda^!$-calculus we use here has a syntax similar to e.g. [21,7], and our type system is inspired by [5].

Outline. In the following we first introduce the $\lambda^!$-calculus as an untyped calculus, delineate a notion of well-formed terms and study the complexity of the reduction of these terms (Sect. 2). We then define a type system inspired by ELL and exhibit two families of types corresponding respectively to the hierarchies k-EXP and k-FEXP for $k \geq 0$ (Sect. 3). Finally

we introduce a second characterization of this hierarchy, based on a non-standard data-type (Sect. 4). A conclusion follows.
A version of this work with a technical appendix containing detailed proofs is available as [3].

2 The $\lambda^!$-Calculus

2.1 Terms and Reduction

We use a calculus, $\lambda^!$-calculus, which adds to ordinary λ-calculus a ! modality and distinguishes two notions of λ-abstraction:

$$\mathtt{M}, \mathtt{N} ::= \mathtt{x} \mid \lambda\mathtt{x}.\mathtt{M} \mid \lambda^!\mathtt{x}.\mathtt{M} \mid \mathtt{M}\mathtt{N} \mid !\mathtt{M}$$

where $\mathtt{x}$ ranges over a countable set of term variables $\mathtt{Var}$. The usual notions of free variables is extended with $\mathrm{FV}(\lambda^!\mathtt{x}.\mathtt{M}) = \mathrm{FV}(\mathtt{M}) \setminus \{\mathtt{x}\}$, $\mathrm{FV}(!\mathtt{M}) = \mathrm{FV}(\mathtt{M})$. As usual, terms are considered modulo renaming of bound variables, and $=$ denotes the syntactic equality modulo this renaming.

Contexts. We consider the class of (one hole) contexts generated by the following grammar:

$$\mathcal{C} ::= \Box \mid \lambda\mathtt{x}.\mathcal{C} \mid \lambda^!\mathtt{x}.\mathcal{C} \mid \mathcal{C}\mathtt{M} \mid \mathtt{M}\mathcal{C} \mid !\mathcal{C}$$

As usual, capture of variables may occur. The *occurrence* of a term $\mathtt{N}$ in $\mathtt{M}$ is a context $\mathcal{C}$ such that $\mathtt{M} = \mathcal{C}[\mathtt{N}]$; in practice we simply write $\mathtt{N}$ for the occurrence if there is no ambiguity and call it a subterm of $\mathtt{M}$.

Depth. The *depth of the occurrence* $\mathcal{C}$ in $\mathtt{M}$, denoted by $\delta(\mathcal{C}, \mathtt{M})$, is the number of ! modalities surrounding the hole of $\mathcal{C}$ in $\mathtt{M}$.

Moreover, the *depth* $\delta(\mathtt{M})$ of a term $\mathtt{M}$ is the maximal nesting of ! in $\mathtt{M}$.

Example 1. $\mathtt{M} = !((\lambda\mathtt{x}.\mathtt{x})\ !!\mathtt{y}\ !\mathtt{y})$. Then $\delta(!((\lambda\mathtt{x}.\mathtt{x})\ !!\Box\ !\mathtt{y}), \mathtt{M}) = 3$ and $\delta(!((\lambda\mathtt{x}.\mathtt{x})\ !!\mathtt{y}\ !\Box), \mathtt{M}) = 2$; moreover, $\delta(\mathtt{M}) = 3$.

Dynamics. The reduction $\rightarrow$ is the contextual closure of the following rules:

$$(\lambda\mathtt{x}.\mathtt{M})\mathtt{N} \longrightarrow \mathtt{M}[\mathtt{N}/\mathtt{x}] \quad (\beta\text{-rule}) \qquad (\lambda^!\mathtt{x}.\mathtt{M})!\mathtt{N} \longrightarrow \mathtt{M}[\mathtt{N}/\mathtt{x}] \quad (!\text{-rule})$$

where $[\mathtt{N}/\mathtt{x}]$ denotes the capture free substitution of $\mathtt{x}$ by $\mathtt{N}$, whose definition is the obvious extension of the corresponding one for λ-calculus.

Observe that a term such as $(\lambda^!\mathtt{x}.\mathtt{M})\mathtt{P}$ is a redex only if $\mathtt{P} = !\mathtt{N}$ for some $\mathtt{N}$; the intuition behind these two kinds of redexes is that the abstraction λ expects an input at depth 0, while $\lambda^!$ expects an input at depth 1.

A *subterm at depth* i in $\mathtt{M}$ is an occurrence $\mathcal{C}$ in $\mathtt{M}$ such that $\delta(\mathcal{C}, \mathtt{M}) = i$; we denote by $\to_i$ the reduction of a redex occurring at depth i. As usual, $\overset{*}{\to}$ ($\overset{*}{\to}_i$) denotes the reflexive and transitive closure of $\to$ ($\to_i$). We say that a term is in *i-normal form* if it does not have any redex at depth less than or equal to i; then $\mathtt{M}$ is in normal form iff it is in $\delta(\mathtt{M})$-normal form. We denote as $\mathbf{nf}_i$ the set of terms in i-normal form.

We have a confluence property, whose proof is adapted from [20], taking into account the notion of depth:

Proposition 1.

(i) *Let* $\mathtt{M} \in \mathbf{nf}_i$ *and* $\mathtt{M} \to_j \mathtt{M}'$*, with* $j \geq i+1$*, then* $\mathtt{M}' \in \mathbf{nf}_i$*.*

(ii) [Confluence at fixed depth] *Let* $\mathtt{M} \to_i \mathtt{P}$ *and* $\mathtt{M} \to_i \mathtt{Q}$*, then there is a term* $\mathtt{N}$ *such that* $\mathtt{P} \overset{*}{\to}_i \mathtt{N}$ *and* $\mathtt{Q} \overset{*}{\to}_i \mathtt{N}$*.*

(iii) [Confluence] *Let* $\mathtt{M} \to \mathtt{P}$ *and* $\mathtt{M} \to \mathtt{Q}$*, then there is a term* $\mathtt{N}$ *such that* $\mathtt{P} \overset{*}{\to} \mathtt{N}$ *and* $\mathtt{Q} \overset{*}{\to} \mathtt{N}$*.*

We consider a specific subclass of terms, inspired by elementary linear logic (ELL) [9,17]:

Definition 1 (Well-formed Term). *A term* $\mathtt{M}$ *is well-formed (w.f.) if and only if, for any subterm* $\mathtt{N}$ *of* $\mathtt{M}$ *which is an abstraction, we have:*

1. *if* $\mathtt{N} = \lambda\mathtt{x}.\mathtt{P}$*, then* $\mathtt{x}$ *occurs at most once and at depth* 0 *in* $\mathtt{P}$*;*
2. *if* $\mathtt{N} = \lambda^!\mathtt{x}.\mathtt{P}$*, then* $\mathtt{x}$ *can only occur at depth* 1 *in* $\mathtt{P}$*.*

Example 2. $\lambda f.\lambda\mathtt{x}.f(f\mathtt{x})$, the standard representation of the Church integer 2, is not w.f.; its w.f. counterpart is $\lambda^! f.!(\lambda\mathtt{x}.f(f\mathtt{x}))$.

The motivation behind such definition is that the depth of subterms in a w.f. term does not change during reduction: if an abstraction expects an input at depth 0 (resp. 1), which is the case of λ (resp. $\lambda^!$), then the substitutions occur at depth 0 (resp. 1), as each occurrence of its bound variable is at depth 0 (resp. 1).

The class of w.f. terms is preserved by reduction and their depth does not increase during reduction:

Lemma 1. *If* $\mathtt{M}$ *is w.f. and* $\mathtt{M} \to \mathtt{M}'$*, then* $\mathtt{M}'$ *is w.f., and* $\delta(\mathtt{M}') \leq \delta(\mathtt{M})$*.*

From now on, we assume that all terms are well formed.

Sizes. In order to study the reduction, it is useful to examine the *size of* $\mathtt{M}$ *at depth* i, denoted by $|\mathtt{M}|_i$, defined as follows:

- If $\mathtt{M} = \mathtt{x}$, then $|\mathtt{x}|_0 = 1$ and $|\mathtt{x}|_i = 0$ for $i \geq 1$;
- If $\mathtt{M} = \lambda\mathtt{x}.\mathtt{N}$ or $\mathtt{M} = \lambda^!\mathtt{x}.\mathtt{N}$, then $|\mathtt{M}|_0 = |\mathtt{N}|_0 + 1$ and $|\mathtt{M}|_i = |\mathtt{N}|_i$ for $i \geq 1$;
- If $\mathtt{M} = \mathtt{NP}$, then $|\mathtt{M}|_0 = |\mathtt{N}|_0 + |\mathtt{P}|_0 + 1$ and $|\mathtt{M}|_i = |\mathtt{N}|_i + |\mathtt{P}|_i$ for $i \geq 1$;
- If $\mathtt{M} = !\mathtt{N}$, then $|\mathtt{M}|_0 = 0$ and $|\mathtt{M}|_{i+1} = |\mathtt{N}|_i$ for $i \geq 0$;

Let $\delta(\mathtt{M}) = d$; then $|\mathtt{M}|_{i+} = \sum_{j=i}^{d} |\mathtt{M}|_j$ and the *size of* $\mathtt{M}$ is $|\mathtt{M}| = \sum_{i=0}^{d} |\mathtt{M}|_i$. The definition is extended to contexts, where $|\Box|_i = 0$ for $i \geq 0$. We consider how the size of a term changes during reduction:

Lemma 2. *If* $\mathtt{M} \to_i \mathtt{M}'$*, then* $|\mathtt{M}'|_i \leq |\mathtt{M}|_i - 1$*, and* $|\mathtt{M}'|_j = |\mathtt{M}|_j$ *for* $j < i$*.*

Strategy. The fact that by Prop. 1.(i) reducing a redex does not create any redex at strictly lower depth suggests considering the following, non-deterministic, *level-by-level* reduction strategy: if the term is not in normal form reduce (non deterministically) a redex at depth i, where $i \geq 0$ is the minimal depth such that $\mathtt{M} \notin \mathbf{nf}_i$. A *level-by-level reduction sequence* is a reduction sequence following the level-by-level strategy. We say that a reduction sequence is *maximal* if either it is infinite, or if it finishes with a normal term.

Proposition 2. *Any reduction of a term by the level-by-level strategy terminates.*

It follows that a maximal level-by-level reduction sequence of a term $\mathtt{M}$ has the shape shown in (1), where $\rightsquigarrow_i$ denotes one reduction step according to the level-by-level strategy, performed at depth i. We call *round* i the subsequence of $\rightsquigarrow_i$ starting from $\mathtt{M}_i^1$. Note that, for all i and $j > i$, $\mathtt{M}_j^1 \in \mathbf{nf}_i$. We simply write $\rightsquigarrow$ when we do not refer to a particular depth.

$$\mathtt{M}_0^1 \rightsquigarrow_0 \ldots \rightsquigarrow_0 \mathtt{M}_0^{n_0} = \mathtt{M}_1^1 \rightsquigarrow_1 \ldots \rightsquigarrow_1 \mathtt{M}_1^{n_1} = \ldots = \mathtt{M}_\delta^1 \rightsquigarrow_\delta \ldots \rightsquigarrow_\delta \mathtt{M}_\delta^{n_\delta} \qquad (1)$$

In a particular case, namely in Lemma 3, we use a deterministic version of the level-by-level strategy, called *leftmost-by-level*, which proceeds at every level from left to right, taking into account the shape of the different redexes in our calculus. That is to say, it chooses at every step the leftmost subterm of the shape $\mathtt{MN}$, where $\mathtt{M}$ is an abstraction, and, in case it is already a redex it reduces it, in case it is of the shape $(\lambda^!\mathtt{x}.\mathtt{P})\mathtt{N}$, where $\mathtt{N} \neq !\mathtt{Q}$, for some $\mathtt{Q}$, then it looks for the next redex in $\mathtt{N}$. This corresponds to using the call-by-name discipline for β-redexes and the call-by-value for !-redexes [20].

$\mathtt{M} \Longrightarrow \mathtt{N}$ denotes that $\mathtt{N}$ is obtained from $\mathtt{M}$ by performing one reduction step according to the leftmost-by-level strategy. All the notations for $\to$ are extended to $\rightsquigarrow$ and $\Longrightarrow$ in a straightforward way.

2.2 Representation of Functions

In order to represent functions, we first need to encode data. For booleans we can use the familiar encoding $\mathtt{true} = \lambda\mathtt{x}.\lambda\mathtt{y}.\mathtt{x}$ and $\mathtt{false} = \lambda\mathtt{x}.\lambda\mathtt{y}.\mathtt{y}$. For tally integers, the usual encoding of Church integers does not give w.f. terms; instead, we use the following encodings for Church integers and Church binary words:

$$\begin{array}{ll} n \in \mathbb{N}, & \underline{n} = \lambda^! f.!(\lambda\mathtt{x}.f\ (f\ ...(f\ \mathtt{x})...)) \\ w \in \{0,1\}^\star, \quad w = \langle i_1, ..., i_n \rangle, & \underline{w} = \lambda^! f_0.\lambda^! f_1.!(\lambda\mathtt{x}.f_{i_1}\ (f_{i_2}\ ...(f_{i_n}\ \mathtt{x})...)) \end{array}$$

By abuse of notation we also denote by $\underline{1}$ the term $\lambda^! f.!f$. Observe that the terms encoding booleans are of depth 0, while those representing Church integers and Church binary words are of depth 1. We denote the length of a word $w \in \{0,1\}^\star$ by $\mathbf{length}(w)$.

We represent computation on a binary word by considering applications of the form $\mathtt{P}!\underline{w}$, with a ! modality on the argument, because the program should be able to duplicate its input. Concerning the form of the result, since we want to allow computation at arbitrary depth, we require the output to be of the form $!^k\mathtt{D}$, where $k \in \mathbb{N}$ and $\mathtt{D}$ is one of the data representations above.

We thus say that a function $f : \{0,1\}^\star \to \{\mathtt{true}, \mathtt{false}\}$ is represented by a term (*program*) $\mathtt{P}$ if $\mathtt{P}$ is a closed normal term and there exists $k \in \mathbb{N}$ such that, for any $w \in \{0,1\}^\star$ and $\mathtt{D} = f(w) \in \{\mathtt{true}, \mathtt{false}\}$ we have: $\mathtt{P}!\underline{w} \overset{*}{\to} !^k\mathtt{D}$. This definition can be adapted to functions with other domains and codomains.

2.3 Complexity of Reduction

We study the complexity of the reduction of terms of the form $\mathtt{P}!\underline{w}$. Actually it is useful to analyze the complexity of the reduction of such terms to their k-normal form, i.e. by reducing until depth k, for $k \in \mathbb{N}$. We define the notation 2_i^n in the following way: $2_0^x = x$ and $2_{i+1}^x = 2^{2_i^x}$.

Proposition 3. *Given a program* $\mathtt{P}$*, for any* $k \geq 2$*, there exists a polynomial* q *such that, for any* $w \in \{0,1\}^\star$*,* $\mathtt{P}!\underline{w} \overset{*}{\rightsquigarrow} \mathtt{M}_k^1 \in \mathbf{nf}_{k-1}$ *in at most* $2_{k-2}^{q(n)}$ *steps, and* $|\mathtt{M}_k^1| \leq 2_{k-2}^{q(n)}$*, where* $n = \mathbf{length}(w)$*. In particular, in the case where* $k = 2$ *we have a polynomial bound* $q(n)$*.*

In the rest of this section we prove Prop. 3.

Let $\mathtt{M} = \mathtt{P}!\underline{w}$ and consider a level-by-level reduction sequence of $\mathtt{M}$, using the notations of (1). By Lemma 2 we know that the number of steps at depth i is bounded by $|\mathtt{M}_i^1|$ and that there are $(d+1)$ rounds. In order to bound the total number of steps it is thus sufficient to bound $|\mathtt{M}_i^1|$ by means of $|\mathtt{M}|$:

Lemma 3 (Size-Growth). *If* $\mathtt{M} \overset{*}{\Longrightarrow}_i \mathtt{M}'$ *by* c *reduction steps, then* $|\mathtt{M}'| \leq |\mathtt{M}| \cdot (|\mathtt{M}|+1)^c$ $(0 \leq i \leq \delta(\mathtt{M}))$.

Proof (Prop. 3). We proceed by induction on $k \geq 2$. We assume that $\mathtt{P}$ is of the form $\lambda^!\mathtt{y}.\mathtt{Q}$ (otherwise $\mathtt{P}!\underline{w}$ is already a normal form).

- Case $k = 2$:
  We consider a level-by-level reduction sequence of $\mathtt{P}!\underline{w}$. We need to examine reduction at depths 0 and 1. At depth 0 we have $(\lambda^!\mathtt{y}.\mathtt{Q})!\underline{w} \to \mathtt{Q}[\underline{w}/\mathtt{y}] = \mathtt{M}_1^1$. Observe that $\mathtt{M}_1^1 \in \mathbf{nf}_0$ because the occurrences of $\mathtt{y}$ in $\mathtt{Q}$ are at depth 1; denote by b the number of occurrences of $\mathtt{y}$ in $\mathtt{Q}$, which does not depend on n.
  Since $|\mathtt{Q}[\underline{w}/\mathtt{y}]|_1 \leq |\mathtt{Q}|_1 + b \cdot |\underline{w}|_0$ and $|\underline{w}|_0 = 2$ (by definition of the encoding), we have that $|\mathtt{M}_1^1|_1 = |\mathtt{Q}[\underline{w}/\mathtt{y}]|_1 \leq |\mathtt{Q}|_1 + 2b$. Let c be $|\mathtt{Q}|_1 + 2b$, which does not depend on n: then, by Lemma 2, the number of steps at depth 1 is bounded by c. This proves the first part of the statement.
  Let $\mathtt{M}_2^1 \in \mathbf{nf}_1$ be the term obtained after reduction at depth 1. By Prop. 1.(ii) we have that $\mathtt{M}_1^1 \overset{*}{\Longrightarrow}_1 \mathtt{M}_2^1$ and by Lemma 2 this reduction is done in c' steps, where $c' \leq |\mathtt{M}_1^1|_1 \leq c$, so by Lemma 3 we have that $|\mathtt{M}_2^1| \leq |\mathtt{M}_1^1| \cdot (|\mathtt{M}_1^1|+1)^c$. Moreover $|\mathtt{M}_1^1| \leq |\mathtt{Q}| + b|\underline{w}|$, so it is polynomial in n, and the statement is proved for $k = 2$.
- Assume the property holds for k and let us prove it for $k+1$.
  By assumption $\mathtt{M}$ reduces to $\mathtt{M}_k^1$ in at most $2_{k-2}^{q(n)}$ steps and $|\mathtt{M}_k^1| \leq 2_{k-2}^{q(n)}$. Let $\mathtt{M}_k^1 \rightsquigarrow_k^* \mathtt{M}_{k+1}^1 \in \mathbf{nf}_k$. By Lemma 2 this reduction sequence has at most $|\mathtt{M}_k^1|_k$ steps, and $|\mathtt{M}_k^1|_k \leq |\mathtt{M}_k^1| \leq 2_{k-2}^{q(n)}$. So on the whole $\mathtt{M}$ reduces to $\mathtt{M}_{k+1}^1$ in at most $2 \cdot 2_{k-2}^{q(n)} \leq 2_{k-2}^{2q(n)}$ steps. Moreover by Prop. 1.(ii) we have that $\mathtt{M}_k^1 \overset{*}{\Longrightarrow} \mathtt{M}_{k+1}^1$ and by Lemma 2 and Lemma 3 we get

  $$|\mathtt{M}_{k+1}^1| \leq |\mathtt{M}_k^1| \cdot (|\mathtt{M}_k^1|+1)^{2_{k-2}^{q(n)}} \leq 2_{k-2}^{q(n)} \cdot (2_{k-2}^{2q(n)})^{2_{k-2}^{q(n)}} \leq 2_{k-2}^{q(n)} \cdot 2^{2_{k-2}^{3q(n)}} \leq 2_{k-1}^{q'(n)}$$

  for some polynomial $q'(n)$.

Approximations. From Prop. 3 we can easily derive a $2^{q(n)}_{k-2}$ bound on the number of *steps* of the reduction of $\mathtt{P}!\underline{w}$ not only to its $(k-1)$-normal form, but also to its k-normal form $\mathtt{M}^1_{k+1}$. Unfortunately this does not yield directly a *time* bound $O(2^{q(n)}_{k-2})$ for the simulation of this reduction on a Turing machine, because during round k the size of the term at depth $k+1$ could grow exponentially. However if we are only interested in the result at depth k, the subterms at depth $k+1$ are actually irrelevant. For this reason we introduce a notion of *approximation*, inspired by the semantics of stratified coherence spaces [1], which allows us to compute up to a certain depth k, while ignoring what happens at depth $k+1$.

We extend the calculus with a constant $*$; its sizes $|*|_i$ are defined as for variables. If $\mathtt{M}$ is a term and $i \in \mathbb{N}$, we define its *i-th approximation* $\overline{\mathtt{M}}^i$ by: $\overline{!\mathtt{M}}^0 = !*$, $\overline{!\mathtt{M}}^{i+1} = !\,\overline{\mathtt{M}}^i$, $\overline{\mathtt{x}}^i = \mathtt{x}$, and for all other constructions $\overline{(\cdot)}^i$ acts as identity, e.g. $\overline{\mathtt{MN}}^i = \overline{\mathtt{M}}^i\overline{\mathtt{N}}^i$.

So $\overline{\mathtt{M}}^i$ is obtained by replacing in $\mathtt{M}$ all subterms at depth $i+1$ by $*$. For instance we have $\overline{\underline{w}}^0 = \lambda^! f_0.\lambda^! f_1.!*$ and $\overline{\underline{w}}^{i+1} = \underline{w}$ for $i \geq 0$.

Lemma 4. *(i) Let $\mathtt{M} \to_j \mathtt{M}'$: if $j \leq i$ then $\overline{\mathtt{M}}^i \to_j \overline{\mathtt{M}'}^i$, otherwise $\overline{\mathtt{M}}^i = \overline{\mathtt{M}'}^i$.*
(ii) Let $\overline{\mathtt{M}}^i \to_i \overline{\mathtt{M}'}^i$: then $|\overline{\mathtt{M}'}^i| < |\overline{\mathtt{M}}^i|$.

Proposition 4. *Given a program $\mathtt{P}$, for any $k \geq 2$, there exists a polynomial q such that for any $w \in \{0,1\}^\star$, the reduction of $\overline{\mathtt{P}!\underline{w}}^k$ to its k-normal form can be computed in time $O(2^{q(n)}_{k-2})$ on a Turing machine, where $n = \mathbf{length}(w)$.*

Proof. Observe that $\overline{\mathtt{P}!\underline{w}}^k = \overline{\mathtt{P}}^k!\underline{w}$. By Prop. 3 and Lemma 4.(i), it reduces to its $(k-1)$-normal form $\overline{\mathtt{M}^1_k}^k$ in $O(2^{q(n)}_{k-2})$ steps and with intermediary terms of size $O(2^{q(n)}_{k-2})$. Now by Lemma 4.(ii) the reduction of $\overline{\mathtt{M}^1_k}^k$ at depth k is done in $O(2^{q(n)}_{k-2})$ steps and with intermediary terms of size $O(2^{q(n)}_{k-2})$. We can then conclude by using the fact that one reduction step in a term $\mathtt{M}$ can be simulated in time $p(|\mathtt{M}|)$ on a Turing machine, for a suitably chosen polynomial p.

3 Type System

We introduce a type assignment system for $\lambda^!$-calculus, based on ELL, such that all typed terms are also w.f. and the previous results are preserved.

Table 1. Derivation rules.

$$\frac{}{\Gamma, \mathtt{x} : \mathtt{A} \mid \Delta \mid \Theta \vdash \mathtt{x} : \mathtt{A}}\ (Ax^L) \qquad \frac{}{\Gamma \mid \Delta \mid \mathtt{x} : \sigma, \Theta \vdash \mathtt{x} : \sigma}\ (Ax^P)$$

$$\frac{\Gamma, \mathtt{x} : \mathtt{A} \mid \Delta \mid \Theta \vdash \mathtt{M} : \tau}{\Gamma \mid \Delta \mid \Theta \vdash \lambda \mathtt{x}.\mathtt{M} : \mathtt{A} \multimap \tau}\ (\multimap I^L) \qquad \frac{\Gamma \mid \Delta, \mathtt{x} : !\sigma \mid \Theta \vdash \mathtt{M} : \tau}{\Gamma \mid \Delta \mid \Theta \vdash \lambda^! \mathtt{x}.\mathtt{M} : !\sigma \multimap \tau}\ (\multimap I^I)$$

$$\frac{\Gamma_1 \mid \Delta \mid \Theta \vdash \mathtt{M} : \sigma \multimap \tau \quad \Gamma_2 \mid \Delta \mid \Theta \vdash \mathtt{N} : \sigma \quad \Gamma_1 \# \Gamma_2}{\Gamma_1, \Gamma_2 \mid \Delta \mid \Theta \vdash \mathtt{MN} : \tau}\ (\multimap E) \qquad \frac{\emptyset \mid \emptyset \mid \Theta' \vdash \mathtt{M} : \sigma}{\Gamma \mid !\Theta', \Delta \mid \Theta \vdash !\mathtt{M} : !\sigma}\ (!)$$

$$\frac{\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \mathtt{S} \quad \mathtt{a} \notin \mathrm{FTV}(\Gamma) \cup \mathrm{FTV}(\Delta) \cup \mathrm{FTV}(\Theta)}{\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \forall \mathtt{a}.\mathtt{S}}\ (\forall I) \qquad \frac{\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \forall \mathtt{a}.\mathtt{S}}{\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \mathtt{S}[\sigma/\mathtt{a}]}\ (\forall E)$$

$$\frac{\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \mathtt{S}[\mu \mathtt{a}.\mathtt{S}/\mathtt{a}]}{\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \mu \mathtt{a}.\mathtt{S}}\ (\mu I) \qquad \frac{\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \mu \mathtt{a}.\mathtt{S}}{\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \mathtt{S}[\mu \mathtt{a}.\mathtt{S}/\mathtt{a}]}\ (\mu E)$$

The set $\mathcal{T}$ of types are generated by the grammar

$$\begin{array}{lll} \mathtt{A} ::= \mathtt{a} \mid \mathtt{S} & & \text{(linear types)} \\ \mathtt{S} ::= \sigma \multimap \sigma \mid \forall \mathtt{a}.\mathtt{S} \mid \mu \mathtt{a}.\mathtt{S} & & \text{(strict linear types)} \\ \sigma ::= \mathtt{A} \mid !\sigma & & \text{(types)} \end{array}$$

where $\mathtt{a}$ ranges over a countable set of type variables. Observe that we consider both polymorphic types ($\forall \mathtt{a}.\mathtt{S}$) and type fixpoints ($\mu \mathtt{a}.\mathtt{S}$); the restriction of both abstractions to act on strict linear types is necessary for the subject reduction property.

A *basis* is a partial function from variables to types, with finite domain; given two bases Γ_1 and Γ_2, let $\Gamma_1 \# \Gamma_2$ iff $\mathrm{dom}(\Gamma_1) \cap \mathrm{dom}(\Gamma_2) = \emptyset$. Following the work of [5], we consider three different bases $\Gamma \mid \Delta \mid \Theta$, called respectively the *linear*, *modal* and *parking* basis, such that $\Gamma \# \Delta$, $\Gamma \# \Theta$ and $\Delta \# \Theta$. The premises in Γ assign to variables linear types, while the premises in Δ assign modal types.

The typing system proves statements of the shape $\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \sigma$, and derivations are denoted by Π, Σ. The rules are given in Table 1. Observe that, in rule $(\multimap E)$, $\mathtt{M}$ and $\mathtt{N}$ share variables in the modal and parking basis, but their linear bases must be disjoint. Note also that there is no axiom rule for variables in the modal basis, so the only way to introduce a variable in this basis is the (!) rule, moving variables from the parking to the modal basis. Finally, observe that there is no abstraction rule for variables in the parking basis: indeed parking variables only have

a "temporary" status, awaiting to be moved to the modal basis.
We say that a term $\mathtt{M}$ is *well-typed* iff there is a derivation $\Pi \triangleright \Gamma \mid \Delta \mid \emptyset \vdash \mathtt{M} : \sigma$ for some Γ, Δ, σ: indeed parking variables are only considered as an intermediary status before becoming modal variables. When all three bases are empty we denote the derivation by $\Pi \triangleright \vdash \mathtt{M} : \sigma$. The main difference w.r.t. the type system of [5] is the (!) rule: here we allow only the parking context to be non-empty, in order to ensure that typable terms are well formed: it is the key to obtain a $2_k^{\mathrm{poly}(n)}$ complexity bound for a specific k depending on the type, instead of just an elementary bound.

Both the type and depth of a term are preserved during reduction:

Theorem 1 (Subject Reduction). *$\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \sigma$ and $\mathtt{M} \to \mathtt{M}'$ imply $\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M}' : \sigma$.*

Proposition 5. *If a term is well-typed, then it is also well-formed.*

The proof comes easily from the following proposition:

Proposition 6 (Variables Depth). *Let $\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \sigma$. Then:*

- *if $\mathtt{x} \in \mathrm{dom}(\Gamma) \cup \mathrm{dom}(\Theta)$, then $\mathtt{x}$ can only occur at depth 0 in $\mathtt{M}$;*
- *if $\mathtt{x} \in \mathrm{dom}(\Delta)$, then $\mathtt{x}$ can only occur at depth 1 in $\mathtt{M}$.*

3.1 Datatypes

In section 2.2 we introduced w.f. terms encoding data, for which we now define the following types, adapted from system F, representing respectively booleans, Church tally integers and Church binary words:

$$\mathrm{B} = \forall \mathtt{a}.\mathtt{a} \multimap \mathtt{a} \multimap \mathtt{a} \qquad \mathrm{N} = \forall \mathtt{a}.!(\mathtt{a} \multimap \mathtt{a}) \multimap !(\mathtt{a} \multimap \mathtt{a})$$

$$\mathrm{W} = \forall \mathtt{a}.!(\mathtt{a} \multimap \mathtt{a}) \multimap !(\mathtt{a} \multimap \mathtt{a}) \multimap !(\mathtt{a} \multimap \mathtt{a})$$

We also use Scott binary words, defined inductively as

$$\widehat{\epsilon} \stackrel{\mathrm{def}}{=} \lambda f_0.\lambda f_1.\lambda \mathtt{x}.\mathtt{x} \qquad \widehat{0w} \stackrel{\mathrm{def}}{=} \lambda f_0.\lambda f_1.\lambda \mathtt{x}.f_0\widehat{w} \qquad \widehat{1w} \stackrel{\mathrm{def}}{=} \lambda f_0.\lambda f_1.\lambda \mathtt{x}.f_1\widehat{w}$$

having type $\mathrm{W}_S \stackrel{\mathrm{def}}{=} \mu \mathtt{b}.\forall \mathtt{a}.(\mathtt{b} \multimap \mathtt{a}) \multimap (\mathtt{b} \multimap \mathtt{a}) \multimap (\mathtt{a} \multimap \mathtt{a})$.

The following properties ensure that, given a datatype, every derivation having such type reduces to a term having the desired shape:

Proposition 7. *(i) If $\vdash \mathtt{M} :!^k\mathrm{B}$ for $k \geq 0$ and $\mathtt{M} \in \mathbf{nf}_k$, then either $\mathtt{M} =!^k \mathtt{true}$ or $\mathtt{M} =!^k \mathtt{false}$.*
(ii) If $\vdash \mathtt{M} :!^k\mathrm{W}_S$ for $k \geq 0$ and $\mathtt{M} \in \mathbf{nf}_k$, then $\mathtt{M} =!^k \widehat{w}$ for some $\widehat{w}$.

3.2 Complexity Soundness and Completeness

We are interested in giving a precise account of the hierarchy of classes characterized by this typed $\lambda^!$-calculus. Denote by $\mathtt{FDTIME}(F(n))$ and by $\mathtt{DTIME}(F(n))$ respectively the class of functions and the class of predicates on binary words computable on a deterministic Turing machine in time $O(F(n))$; the complexity classes we are interested in, for $k \geq 0$, are:

$$\mathtt{k\text{-}EXP} = \cup_{i\in\mathbb{N}}\mathtt{DTIME}(2_k^{n^i}) \quad \text{and} \quad \mathtt{k\text{-}FEXP} = \cup_{i\in\mathbb{N}}\mathtt{FDTIME}(2_k^{n^i}).$$

In particular, observe that $\mathtt{PTIME} = \cup_{i\in\mathbb{N}}\mathtt{DTIME}(n^i) = \mathtt{0\text{-}EXP}$ and $\mathtt{FPTIME} = \cup_{i\in\mathbb{N}}\mathtt{FDTIME}(n^i) = \mathtt{0\text{-}FEXP}$.

Soundness Let $\mathcal{F}(\sigma)$ denote the set of closed terms representing functions, to which type σ can be assigned: we prove that $\mathcal{F}(!\mathrm{W} \multimap !^{k+2}\mathrm{B}) \subseteq k\text{-}\mathtt{EXP}$ and $\mathcal{F}(!\mathrm{W} \multimap !^{k+2}\mathrm{W}_S) \subseteq k\text{-}\mathtt{FEXP}$.

Theorem 2 (Soundness). *Let* $\vdash \mathtt{P} : !\mathrm{W} \multimap !^{k+2}\mathrm{B}$ *where* $\mathtt{P}$ *is a program, and let* $\vdash \underline{w} : \mathrm{W}$ *where* $\mathbf{length}(w) = n$*; then the reduction* $\mathtt{P}!\underline{w} \xrightarrow{*} !^{k+2}\mathtt{D}$ *can be computed in time* $2_k^{p(n)}$*, where* $\mathtt{D}$ *is either* $\mathtt{true}$ *or* $\mathtt{false}$ *and* p *is a polynomial.*

Proof. Recall that a program $\mathtt{P}$ is a typed closed term in normal form: we denote by $\mathtt{M}'$ the normal form of $\mathtt{P}!\underline{w}$. By Prop. 4 we know that $\overline{\mathtt{P}!\underline{w}}^{k+2}$ can be reduced to a term $\mathtt{N}$ in $\mathbf{nf}_{k+2}$ in time $O(2_k^{p(n)})$ on a Turing machine, where $n = \mathbf{length}(w)$. Moreover by Lemma 4.(i) and Prop. 1.(iii) we have that $\overline{\mathtt{M}'}^{k+2} = \mathtt{N}$. Now, as $\mathtt{P}!\underline{w}$ has type $!^{k+2}\mathrm{B}$, by Theorem 1 the term $\mathtt{M}'$ is a closed term of type $!^{k+2}\mathrm{B}$ and, by Prop. 7.(i), it is equal to $!^{k+2}\mathtt{true}$ or $!^{k+2}\mathtt{false}$. Then $\mathtt{N} = \overline{\mathtt{M}'}^{k+2} = \mathtt{M}'$, so $\mathtt{P}!\underline{w}$ can be computed in time $O(2_k^{p(n)})$.

Complexity soundness can be proved for functions by a similar proof, in which Prop. 7.(ii) is used in order to read the output as a Scott word:

Theorem 3. *Let* $\vdash \mathtt{P} : !\mathrm{W} \multimap !^{k+2}\mathrm{W}_S$ *where* $\mathtt{P}$ *is a program, and let* $\vdash \underline{w} : \mathrm{W}$ *where* $\mathbf{length}(w) = n$*; then the reduction* $\mathtt{P}!\underline{w} \xrightarrow{*} !^{k+2}\widehat{w'}$ *can be computed in time* $2_k^{p(n)}$*, where* p *is a polynomial.*

Completeness We proved that $\mathcal{F}(!\mathrm{W} \multimap !^{k+2}\mathrm{B}) \subseteq k\text{-}\mathtt{EXP}$ and $\mathcal{F}(!\mathrm{W} \multimap !^{k+2}\mathrm{W}_S) \subseteq k\text{-}\mathtt{FEXP}$; now we want to strengthen this result by examining the converse inclusions. To do so we simulate $k\text{-}\mathtt{EXP}$ time bounded Turing machines, by an iteration, so as to prove the following results:

Theorem 4 (Extensional Completeness).

- *Let f be a binary predicate in k-*EXP*, for any $k \geq 0$; then there is a term* M *representing f such that* $\vdash \mathtt{M} : !\mathrm{W} \multimap !^{k+2}\mathrm{B}$.
- *Let g be a function on binary words in k-*FEXP*, for $k \geq 0$; then there is a term* M *representing g such that* $\vdash \mathtt{M} : !\mathrm{W} \multimap !^{k+2}\mathrm{W}_S$.

Note that this characterization, for $k = 0$, does not account for the fact that FPTIME is closed by composition: indeed, programs of type $!\mathrm{W} \multimap !^{k+2}\mathrm{W}_S$ cannot be composed, since we do not have any coercion from W_S to W. For this reason, we explore an alternative characterization.

4 Refining Types for an Alternative Characterization

Our aim is to take a pair $\langle n, w\rangle$ to represent the word w' such that:

$$w' = \begin{cases} w & \text{if } \mathbf{length}(w) \leq n, \\ \text{the prefix of } w \text{ of length } n & \text{otherwise.} \end{cases}$$

For this reason, we introduce a new data-type using the connective $\otimes$ defined by $\sigma \otimes \tau \stackrel{\text{def}}{=} \forall \mathtt{a}.((\sigma \multimap \tau \multimap \mathtt{a}) \multimap \mathtt{a})$ on types and the corresponding constructions on terms:

$$\begin{aligned} \mathtt{M}_1 \otimes \mathtt{M}_2 &\stackrel{\text{def}}{=} \lambda \mathtt{x}.\mathtt{x}\mathtt{M}_1\mathtt{M}_2 \\ \lambda(\mathtt{x}_1 \otimes \mathtt{x}_2).\mathtt{M} &\stackrel{\text{def}}{=} \lambda \mathtt{x}.(\mathtt{x}\lambda \mathtt{y}_1\mathtt{y}_2.\lambda \mathtt{z}.\mathtt{z}\mathtt{y}_1\mathtt{y}_2)\lambda \mathtt{x}_1\mathtt{x}_2.\mathtt{M} \\ \lambda^!(\mathtt{x}_1 \otimes \mathtt{x}_2).\mathtt{M} &\stackrel{\text{def}}{=} \lambda \mathtt{x}.(\mathtt{x}\lambda^! \mathtt{y}_1\mathtt{y}_2.\lambda \mathtt{z}.\mathtt{z}!\mathtt{y}_1!\mathtt{y}_2)\lambda^! \mathtt{x}_1\mathtt{x}_2.\mathtt{M} \end{aligned}$$

Note that we cannot define the abstraction in the usual way, i.e. $\lambda(\mathtt{x}_1 \otimes \mathtt{x}_2).\mathtt{M} \stackrel{\text{def}}{=} \lambda \mathtt{x}.\mathtt{x}(\lambda \mathtt{x}_1.\lambda \mathtt{x}_2.\mathtt{M})$, otherwise we could not type pairs in a uniform way; moreover, when applied to a pair, this term reduces to the usual one.
The associated reduction rules $(\lambda(\mathtt{x}_1 \otimes \mathtt{x}_2).\mathtt{N})(\mathtt{M}_1 \otimes \mathtt{M}_2) \to \mathtt{N}[\mathtt{M}_1/\mathtt{x}_1, \mathtt{M}_2/\mathtt{x}_2]$ and $(\lambda^!(\mathtt{x}_1 \otimes \mathtt{x}_2).\mathtt{N})(!\mathtt{M}_1 \otimes !\mathtt{M}_2) \to \mathtt{N}[\mathtt{M}_1/\mathtt{x}_1, \mathtt{M}_2/\mathtt{x}_2]$ are derivable.

We represent a pair $\langle n, w\rangle$ through a term $!\underline{n} \otimes !^2\widehat{w}$ of type $!\mathrm{N} \otimes !^2\mathrm{W}_S$, i.e. a combined data-type containing a Church integer $!\underline{n}$ and a Scott word $!^2\widehat{w}$: in practice, $\underline{n}$ is meant to represent the length of a list, whose content is described by $\widehat{w}$. In order to mantain this invariant, when computing on elements $!\underline{n} \otimes !^2\widehat{w}$ of this data-type, the property that the length of w is inferior or equal to n is preserved.

As before, we need to be able to extract the result, in this case a pair:

Proposition 8. *If* $\vdash \mathtt{M} : !^k\mathrm{N} \otimes !^{k+1}\mathrm{W}_S$ *for* $k \geq 0$ *and* $\mathtt{M} \in \mathbf{nf}_{k+1}$*, then there exists* $m \in \mathbb{N}$ *and* $w \in \{0,1\}^\star$ *such that* $\mathtt{M} =_{!^k} \underline{m} \otimes !^{k+1}\widehat{w}$*.*

Then we are able to prove both soundness and completeness results:

Theorem 5. *Let* $\vdash \mathtt{P} : (!\mathrm{N} \otimes !^2\mathrm{W}_S) \multimap (!^{k+1}\mathrm{N} \otimes !^{k+2}\mathrm{W}_S)$ *where* $\mathtt{P}$ *is a program, then for any* $\underline{m}$ *and* $\widehat{w}$ *the reduction of* $\mathtt{P}(!\underline{m} \otimes !^2\widehat{w})$ *to its normal form can be computed in time* $2_k^{p(n)}$*, where* p *is a polynomial and* $n = m + \mathbf{length}(w)$*.*

Theorem 6. *Let* f *be a function on binary words in* k-FEXP*, for* $k \geq 0$*; then there is a term* $\mathtt{M}$ *representing* f *such that* $\vdash \mathtt{M} : (!\mathrm{N} \otimes !^2\mathrm{W}_S) \multimap (!^{k+1}\mathrm{N} \otimes !^{k+2}\mathrm{W}_S)$*.*

Observe that we are able to compose two terms having type $(!\mathrm{N} \otimes !^2\mathrm{W}_S) \multimap (!\mathrm{N} \otimes !^2\mathrm{W}_S)$, so to illustrate the fact that FPTIME is closed by composition; moreover, if $f \in$ FPTIME and $g \in k$-FEXP, then we can compose terms representing them, which shows that $g \circ f \in k$-FEXP.

While the previous characterization of k-FEXP in Section 3.2 offers the advantage of simplicity, because it uses classical data-types (Church and Scott binary words), this second characterization offers a better account of the closure properties of these complexity classes, at the price of a slightly more involved representation of words.

5 Conclusions

We have shown how the concept of !-stratification coming from linear logic can be fruitfully employed in λ-calculus and characterize the hierarchies k-EXP and k-FEXP, including the classes PTIME and FPTIME. A nice aspect of our system with respect to former polyvalent characterizations [11,14] is that the complexity bound can be deduced by looking only at the interface of the program (its type) without refering to the constructions steps. In our proofs we have carefully distinguished the respective roles played by syntactic ingredients (well-formedness) and typing ingredients. This has allowed us to illustrate how types can provide two different characterizations of the class k-FEXP, based on the use of different data-types. We believe that the separation between syntactic and typing arguments can facilitate the possible future usage of our calculus with other type systems. As future work it would be challenging to investigate if similar characterizations could be obtained for other hierarchies, like possibly space hierarchies.

References

1. Baillot, P.: Stratified coherence spaces: a denotational semantics for light linear logic. Theor. Comput. Sci. 318(1-2), 29–55 (2004)
2. Baillot, P.: Elementary linear logic revisited for polynomial time and an exponential time hierarchy. In: Yang, H. (ed.) APLAS. Lecture Notes in Computer Science, vol. 7078, pp. 337–352. Springer (2011)
3. Baillot, P., De Benedetti, E., Ronchi Della Rocca, S.: Characterizing polynomial and exponential complexity classes in elementary lambda-calculus. Tech. rep. (2014), `http://hal.archives-ouvertes.fr/hal-01015171`, 31 pages
4. Bellantoni, S., Cook, S.A.: A new recursion-theoretic characterization of the polytime functions. Computational Complexity 2, 97–110 (1992)
5. Coppola, P., Dal Lago, U., Ronchi Della Rocca, S.: Light logics and the call-by-value lambda-calculus. Logical Methods in Computer Science 4(4) (2008)
6. Dal Lago, U.: Context semantics, linear logic, and computational complexity. ACM Trans. Comput. Log. 10(4) (2009)
7. Dal Lago, U., Masini, A., Zorzi, M.: Quantum implicit computational complexity. Theor. Comput. Sci. 411(2), 377–409 (2010)
8. Danos, V., Joinet, J.B.: Linear logic and elementary time. Inf. Comput. 183(1), 123–137 (2003)
9. Girard, J.Y.: Light linear logic. Inf. Comput. 143(2), 175–204 (1998)
10. Jones, N.D.: Computability and complexity - from a programming perspective. Foundations of computing series, MIT Press (1997)
11. Jones, N.D.: The expressive power of higher-order types or, life without cons. J. Funct. Program. 11(1), 5–94 (2001)
12. Lafont, Y.: Soft linear logic and polynomial time. Theor. Comput. Sci. 318(1-2), 163–180 (2004)
13. Leivant, D.: Predicative recurrence and computational complexity I: word recurrence and poly-time. In: Feasible Mathematics II, pp. 320–343. Birkhauser (1994)
14. Leivant, D.: Calibrating computational feasibility by abstraction rank. In: LICS. pp. 345–. IEEE Computer Society (2002)
15. Leivant, D., Marion, J.Y.: Lambda-calculus characterizations of poly-time. Fundam. Inform. 19(1/2), 167–184 (1993)
16. Madet, A.: Implicit Complexity in Concurrent Lambda-Calculi. Ph.D. thesis, Université Paris 7 (December 2012), http://tel.archives-ouvertes.fr/tel-00794977
17. Madet, A., Amadio, R.M.: An elementary affine lambda-calculus with multithreading and side effects. In: Ong, C.H.L. (ed.) TLCA. Lecture Notes in Computer Science, vol. 6690, pp. 138–152. Springer (2011)
18. Marion, J.Y.: A type system for complexity flow analysis. In: LICS. pp. 123–132. IEEE Computer Society (2011)
19. Mazza, D.: Linear logic and polynomial time. Mathematical Structures in Computer Science 16(6), 947–988 (2006)
20. Ronchi Della Rocca, S., Paolini, L.: The Parametric Lambda-Calculus: a Metamodel for Computation. Texts in Theoretical Computer Science, Springer, Berlin (2004), `http://www.springer.com/sgw/cda/frontpage/0,,5-40356-72-14202886-0,00.html`
21. Ronchi Della Rocca, S., Roversi, L.: Lambda-calculus and intuitionistic linear logic. Studia Logica 59(3), 417–448 (1997)
22. Terui, K.: Light affine lambda-calculus and polynomial time strong normalization. Arch. Math. Log. 46(3-4), 253–280 (2007)