Information and Computation ••• (••••) •••–•••

Contents lists available at ScienceDirect

Information and Computation

www.elsevier.com/locate/yinco

Characterizing polynomial and exponential complexity classes in elementary lambda-calculus ☆

Patrick Baillot [a,*], Erika De Benedetti [b], Simona Ronchi Della Rocca [b]

[a] Univ Lyon, CNRS, ENS de Lyon, Université Claude-Bernard Lyon 1, LIP UMR5668, F-69342, Lyon Cedex 07, France
[b] Università degli Studi di Torino, Dipartimento di Informatica, Torino, Italy

ARTICLE INFO

Article history:
Received 1 July 2015
Available online xxxx

Keywords:
Implicit computational complexity
Linear logic
Lambda-calculus

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.

1. Introduction

Inside the research field of *implicit computational complexity* (ICC), whose goal is to study complexity classes without relying on a particular computational model, we focus on the problem of studying complexity classes by means of programming languages and calculi. Early work on this topic has been carried out by Neil Jones [1,2], 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). Seminal research in this direction has been carried out in the fields of recursion theory [3,4], λ-calculus [5] and linear logic [6]. These contributions usually exhibit a new specific language 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 [2,7], but their criteria used for reaching different classes referred to the construction steps of the programs. Here we are interested in defining a polyvalent characterization where computational classes are expressed by means of the program's type in a dedicated system.

Along these lines, we give a polyvalent characterization of the complexity classes $\texttt{k-EXP} = \cup_{i\in\mathbb{N}}\texttt{DTIME}(2_k^{n^i})$ and $\texttt{k-FEXP} = \cup_{i\in\mathbb{N}}\texttt{FDTIME}(2_k^{n^i})$ for all $k \geq 0$, where $\texttt{DTIME}(F(n))$ and $\texttt{FDTIME}(F(n))$ denote respectively the class of predicates and the class of functions on binary words computable on a deterministic Turing machine in time $O(F(n))$, and 2_k^n is n

☆ 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), and by the ANR Project ELICA ANR-14-CE25-0005.

* Corresponding author.
E-mail addresses: patrick.baillot@ens-lyon.fr (P. Baillot), debenede@di.unito.it (E. De Benedetti), ronchi@di.unito.it (S. Ronchi Della Rocca).

https://doi.org/10.1016/j.ic.2018.05.005
0890-5401/

if $k = 0$, $2^{2^n_{k-1}}$ otherwise. The language we consider is the $\lambda^!$-calculus, an extension of λ-calculus enriched with a !-modality, whose role is to introduce in the language a notion of *stratification*, and we consider the subset of all terms satisfying a given well-formedness condition. The stratification of the computation then allows to obtain an elementary complexity bound when applying a well-formed program to a data. Namely, we define the k-approximation of a computation, consisting in computing up to the k-stratum, ignoring the higher strata of the term. Then we prove that the k-approximation of the computation can be performed in time bounded by a tower of 2's of height k, whose exponent depends on the size of the supplied data. Nevertheless, in order to characterize complexity classes we must consider both a complete computation and the shape of the result. For these reasons we introduce a type assignment system, inspired by ELL [6], such that all well-typed terms are also well-formed and functions are characterized by their type: the result is a sound and complete characterization of k-EXP and k-FEXP, for $k \geq 0$, and in particular each hierarchy of complexity classes corresponds to a hierarchy of types depending on k. However, the latter characterization of functions is a bit disappointing, in the sense that it is does not account for the compositional closure of the complexity class FPTIME; in order to overcome this issue, we propose a different, maybe less natural typing for functions, which allows to compose programs whose interface (type) corresponds to the FPTIME class. Finally, we show how such newly defined type can be employed fruitfully in order to characterize other complexity classes.

Our result clearly starts from [8], where the first author obtained a polyvalent characterization in ELL proof-nets of the complexity classes $\text{k-EXP} = \cup_{i\in\mathbb{N}}\text{DTIME}(2^{n^i}_k)$ for all $k \geq 0$. We were motivated by three observations. First of all, the language of proof-nets is not as standard and widespread as say that of λ-calculus. Moreover, the complexity soundness proof uses a partly semantic argument ([8] Lemma 3 p. 10) and so it does not provide a syntactic way to evaluate the programs with the given complexity bound. Last but not least, the characterization is given for classes of predicates, but not for classes of functions, and it is not so clear how to extend this result to functions because of the method used for the proof.

We would like to put into evidence that the results of [8] cannot be trivially extended to λ-calculus by considering a straightforward translation of λ-terms into proof-nets, since cut-elimination cannot be directly simulated by reduction: indeed, the proof of the complexity bound in the case of proof-nets follows a specific cut-elimination strategy; moreover, as said above it uses a partly semantic argument. For such reasons we need to define some new measures on terms, which are not adapted from proof-nets, in order to give a direct proof of the analogous result in $\lambda^!$-calculus: we believe that this λ-calculus-based approach may be of help in revealing the underlying working principles of ELL and in building characterizations of other complexity classes.

A nice aspect of our system, with respect to former polyvalent characterizations [2,7], is the fact that the complexity bound can be deduced by looking only at the interface of the program, namely its type, without referring to its constructions steps. Moreover, we distinguish the respective roles played by the syntactic aspect (well-formedness) and the typing; this allows us to illustrate how types can provide two different characterizations of the class k-FEXP, based only on the use of different datatypes, and it could prove to facilitate the possible future usage of such elementary $\lambda^!$-calculus with other typing systems.

Related work The first results on ELL [6,9] as well as later works [10,11] 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 [12–14]. Alternatively [15] used standard λ-calculus with a type system derived from ELL. The $\lambda^!$-calculus we use here has a syntax similar to e.g. [16,17], and our type system is inspired by [15].

A preliminary conference version of the present work appeared in [18]. With respect to this short version the present paper contains:

- full proofs which could not be included in [18], like for instance that on the complexity bound on the number of reduction steps (Proposition 17) which makes use of a specific reduction strategy (*leftmost-by-level* strategy);
- a detailed treatment of the characterization of FPTIME with alternative data-types and which is closed under composition (Sect. 6); it was only briefly described in the conference version;
- characterizations of additional complexity classes, polyFEXP and NP (Sect. 6.1), which are new with respect to the previous paper.

Outline The paper is organized as follows. In Section 2 the definition of $\lambda^!$-calculus is given, together with some of its properties. Section 3 supplies the representation of functions and a first complexity result. In Section 4 the type assignment system is introduced, and in Section 5 the polyvalent characterization of complexity classes is defined, based on types. Section 6 proposes an alternative characterization allowing composition. Section 7 contains some final considerations.

2. The elementary λ-calculus

Syntax We introduce an elementary λ-calculus, which adds to ordinary λ-calculus a !-modality and distinguishes between two notions of λ-abstraction. The !-modality is reminiscent of the same notion in linear logic and it is used for organizing subterms into strata. The language of terms is defined by the following grammar:

Please cite this article in press as: P. Baillot et al., Characterizing polynomial and exponential complexity classes in elementary lambda-calculus, Inf. Comput. (2018), https://doi.org/10.1016/j.ic.2018.05.005

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

where, as usual, $\mathtt{x}$ ranges over a countable set of term variables $\mathtt{Var}$. The set of terms is denoted by $\Lambda^!$. The language itself is not new [16,17]; however, in the sequel we will focus our attention on a strict subset of terms, for which a specific condition of well-formedness holds.

As usual, terms are considered up to α-equivalence and the symbol $=$ denotes the syntactic equality modulo such renaming. Moreover, we assume the so-called *variable convention*, that is, all bound variables are different from the free variables, so that capture of free variables is prevented after substitution.

The usual notions of free variables, substitution and number of occurrences hold as for pure λ-calculus, with the following extensions: $\mathrm{FV}(\lambda^!\mathtt{x}.\mathtt{M}) = \mathrm{FV}(\mathtt{M}) \setminus \{\mathtt{x}\}$ and $\mathrm{FV}(!\mathtt{M}) = \mathrm{FV}(\mathtt{M})$ for free variables, $(\lambda^!\mathtt{y}.\mathtt{M})[\mathtt{N}/\mathtt{x}] = \lambda^!\mathtt{y}.(\mathtt{M}[\mathtt{N}/\mathtt{x}])$ and $(!\mathtt{M})\{\mathtt{N}/\mathtt{x}\} = !\mathtt{M}\{\mathtt{N}/\mathtt{x}\}$ for the substitution of $\mathtt{N}$ to $\mathtt{x}$ in $!\mathtt{M}$, $n_0(\mathtt{x}, \lambda^!\mathtt{y}.\mathtt{M}) = n_0(\mathtt{x}, \mathtt{M})$ and $n_0(\mathtt{x}, !\mathtt{M}) = n_0(\mathtt{x}, \mathtt{M})$ for the number of free occurrences of $\mathtt{x}$ in $\mathtt{M}$. In the following, we use the notation $!^i$ as a shorthand for $\underbrace{!\ldots!}_{i}$.

We consider the class of *term contexts*, with possibly several holes, generated by the following grammar:

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

where the symbol $\Box$ represents the *hole* in the term context. A context containing a single hole is a *simple context*.

Let $\mathcal{C}$ be a context having n holes, for $n \geq 1$: then $\mathcal{C}[\mathtt{N}_1]\ldots[\mathtt{N}_n]$ is the term obtained by plugging the term $\mathtt{N}_j$ into the j-th hole of the context ($1 \leq j \leq n$), where the holes are assumed to be ordered from left to right. If $n = 1$, then the case is that of a simple context. Observe that, as usual, capture of variables may occur.

We define an *occurrence* of a term $\mathtt{N}$ in $\mathtt{M}$ as a simple context $\mathcal{C}$ such that $\mathtt{M} = \mathcal{C}[\mathtt{N}]$; in practice, we will simply write $\mathtt{N}$ for the occurrence, if there is no ambiguity, and call it a subterm of $\mathtt{M}$.

Reductions The reduction $\rightarrow$ is the contextual closure of the following rewriting rules, which are naturally induced by the two notions of abstraction in the language:

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

Observe that a term of the shape $(\lambda^!\mathtt{x}.\mathtt{M})\mathtt{P}$ is a redex only if $\mathtt{P} = !\mathtt{N}$ for some term $\mathtt{N}$; otherwise such application, also called a *block*, cannot be reduced. A redex $(\lambda\mathtt{x}.\mathtt{M})\mathtt{N}$ or $(\lambda^!\mathtt{x}.\mathtt{M})!\mathtt{N}$ is *erasing* if $\mathtt{x} \notin \mathrm{FV}(\mathtt{M})$.

Depth In elementary λ-calculus we need a measure of stratification, in order to study how the computation evolves over different strata; such a measure is supplied by the notion of depth, which is clearly inspired by light logics.

Let $\mathtt{M}$ be a term, and let $\mathcal{C}$ be a simple context such that $\mathtt{M} = \mathcal{C}[\mathtt{N}]$ for some term $\mathtt{N}$. The *depth of a simple context* $\mathcal{C}$, denoted by $\delta(\mathcal{C})$, is defined by induction on $\mathcal{C}$ as follows:

$$\delta(\Box) = 0; \qquad \delta(\lambda\mathtt{x}.\mathcal{C}) = \delta(\mathcal{C}); \qquad \delta(\lambda^!\mathtt{x}.\mathcal{C}) = \delta(\mathcal{C});$$
$$\delta(\mathcal{C}\mathtt{Q}) = \delta(\mathcal{C}); \qquad \delta(\mathtt{P}\mathcal{C}) = \delta(\mathcal{C}); \qquad \delta(!\mathcal{C}) = \delta(\mathcal{C}) + 1.$$

Observe that, intuitively, the depth of an occurrence $\mathcal{C}$ is the number of !s enclosing the hole of the simple context $\mathcal{C}$. We introduce now a notation, for speaking about the occurrences of subterms occurring t a given depth.

- $\mathcal{C}[\_]_{i_1}\ldots[\_]_{i_n}$ denotes a context having $n \geq 0$ holes, the j-th hole being at depth $i_j \in \mathbb{N}$; moreover, $\mathcal{C}[\mathtt{N}_1]_{i_1}\ldots[\mathtt{N}_n]_{i_n}$ is the term obtained by plugging the term $\mathtt{N}_j$ into the j-th hole of the context ($1 \leq j \leq n$).
- $\mathcal{C}\lfloor \_, \ldots, \_ \rfloor_i$, where the symbol _ occurs $n \geq 0$ times, denotes a context such that $\Box$ is its only subterm at depth i, and it occurs n times at depth i in it. Moreover, $\mathcal{C}\lfloor \mathtt{N}_1, \ldots, \mathtt{N}_n \rfloor_i$ is the term obtained by filling the context $\mathcal{C}\lfloor \_, \ldots, \_ \rfloor_i$ with the terms $\mathtt{N}_1, \ldots, \mathtt{N}_n$, where holes are intended to be ordered from left to right.

Observe that, given a term $\mathtt{M}$ and an integer i, there is a unique context $\mathcal{C}$ such that $\mathtt{M} = \mathcal{C}\lfloor \mathtt{N}_1, \ldots, \mathtt{N}_n \rfloor_i$.

Example 1. Let $\mathtt{M} = (\lambda^!\mathtt{x}.!(\mathtt{xx})!(\lambda^!\mathtt{z}.!\mathtt{z}))!\mathtt{I}$, where $\mathtt{I} = \lambda\mathtt{x}.\mathtt{x}$ is the identity function: then $\mathtt{M} = \mathcal{C}\lfloor \mathtt{xx}, \lambda^!\mathtt{z}.!\mathtt{z}, \mathtt{I} \rfloor_1$, where $\mathcal{C} = (\lambda^!\mathtt{x}.!\Box!\Box)!\Box$.

Note that any other representation of $\mathtt{M}$ through a context of depth 1 does not satisfy the constraints of the definition.

Informally, we often write that a term $\mathtt{N}$ *is at depth* i in $\mathtt{M}$, for some $i \geq 0$, whenever there is a context $\mathcal{C}$ such that $\mathtt{M} = \mathcal{C}[\mathtt{N}]_i$. Moreover, the *depth of a term* $\mathtt{M}$, denoted by $\delta(\mathtt{M})$ is defined as follows:

$$\delta(\mathtt{x}) = 0; \qquad \delta(\lambda\mathtt{x}.\mathtt{P}) = \delta(\mathtt{P}); \qquad \delta(\lambda^!\mathtt{x}.\mathtt{P}) = \delta(\mathtt{P});$$
$$\delta(\mathtt{PQ}) = \max\{\delta(\mathtt{P}), \delta(\mathtt{Q})\}; \qquad \delta(!\mathtt{P}) = \delta(\mathtt{P}) + 1.$$

The depth of a term $\mathtt{M}$ can also be thought of as the maximal nesting of !s in $\mathtt{M}$, that is, the maximum depth of all simple contexts $\mathcal{C}$ such that $\mathtt{M} = \mathcal{C}[\mathtt{N}]$ for some subterm $\mathtt{N}$ of $\mathtt{M}$.

Please cite this article in press as: P. Baillot et al., Characterizing polynomial and exponential complexity classes in elementary lambda-calculus, Inf. Comput. (2018), https://doi.org/10.1016/j.ic.2018.05.005

Example 2. Let $\mathtt{M}$ be the term $!((\lambda \mathtt{x}.\mathtt{x})\, !!_{\mathtt{y}}\, !\mathtt{y})$: then we have $\delta(!((\lambda \mathtt{x}.\mathtt{x})\, !!\Box\, !\mathtt{y})) = 3$ and $\delta(!((\lambda \mathtt{x}.\mathtt{x})\, !!_{\mathtt{y}}\, !\Box)) = 2$; moreover, $\delta(\mathtt{M}) = 3$.

Well-formed terms A term $\mathtt{M}$ is *well-formed* iff:

- for any subterm $\lambda \mathtt{x}.\mathtt{N}$ of $\mathtt{M}$, $\mathtt{x}$ occurs at most once and at depth 0 in $\mathtt{N}$;
- for any subterm $\lambda^{!}\mathtt{x}.\mathtt{N}$ of $\mathtt{M}$, $\mathtt{x}$ occurs any number of times and only at depth 1 in $\mathtt{N}$.

This notion is clearly inspired by ELL. Note that the abstraction λ is a (affine) linear one.

Consider a well-formed term $\lambda \mathtt{x}.\mathtt{M}$ (resp. $\lambda^{!}\mathtt{x}.\mathtt{M}$); by the definition of multi-hole context, if $\mathtt{x} \in \mathrm{FV}(\mathtt{M})$ then there is a context $\mathcal{C}$ such that $\mathtt{M} = \mathcal{C}[\mathtt{x}]_0$ (resp. $\mathtt{M} = \mathcal{C}[\mathtt{x}]_1 \ldots [\mathtt{x}]_1$). Moreover, in the first case $\mathtt{M}\{\mathtt{N}/\mathtt{x}\} = \mathcal{C}[\mathtt{N}]_0$, while in the second case $\mathtt{M}\{\mathtt{N}/\mathtt{x}\} = \mathcal{C}[\mathtt{N}]_1 \ldots [\mathtt{N}]_1$, for every term $\mathtt{N}$.

Example 3.

- $\lambda f.\lambda \mathtt{x}.f(f\mathtt{x})$ is not well-formed, because f is bound by a λ and it occurs twice; this term represents the Church integer 2 in ordinary λ-calculus.
- $\lambda^{!} f.!(\lambda \mathtt{x}.f(f\mathtt{x}))$ is well-formed: note that it is obtained from the term above by replacing the λ binding f by a $\lambda^{!}$ and by adding a ! in such a way that the occurrences of f are at depth 1.

It is easy to check that to be well-formed is preserved by substitution.

Property 4. Let $\mathtt{M}, \mathtt{N}$ be well formed; then $\mathtt{M}\{\mathtt{N}/\mathtt{x}\}$ is well-formed.

The motivation behind the definition of well-formed terms is that in a well-formed term, during a reduction, the depth of a subterm occurrence is inherited by its descendants.

Descendant of a subterm after a reduction step Let $\mathtt{M} \to \mathtt{M}'$, so $\mathtt{M} = \mathcal{C}[(\lambda \mathtt{x}.\mathtt{P})\mathtt{Q}]$ ($\mathtt{M} = \mathcal{C}[(\lambda^{!}\mathtt{x}.\mathtt{P})!\mathtt{Q}]$) and $\mathtt{M}' = \mathcal{C}[\mathtt{R}\{\mathtt{Q}/\mathtt{x}\}]$, where $\mathcal{C}$ is a simple context.

- Every occurrence of a subterm $\mathtt{N}$ in $\mathcal{C}$ is a descendent in $\mathtt{M}'$ of the same occurrence in $\mathtt{M}$.
- Let $\mathtt{M} = \mathcal{C}[(\lambda \mathtt{x}.\mathcal{C}_1[\mathtt{N}])\mathtt{Q}]$ ($\mathcal{C}[(\lambda^{!}\mathtt{x}.\mathcal{C}_1[\mathtt{N}])!\mathtt{Q}]$) and $\mathtt{M}' = \mathcal{C}[\mathcal{C}_1[\mathtt{N}]\{\mathtt{Q}/\mathtt{x}\}]$. If $\mathtt{N}\{\mathtt{Q}/\mathtt{x}\} = \mathtt{N}$, then the occurrence of $\mathtt{N}$ in $\mathtt{M}'$ defined by $\mathcal{C}[\mathcal{C}_1\{\mathtt{Q}/\mathtt{x}\}]$ is a descendant of the occurrence of $\mathtt{N}$ in $\mathtt{M}$ defined by $\mathcal{C}[(\lambda \mathtt{x}.\mathcal{C}_1)\mathtt{Q}]$ ($\mathcal{C}[(\lambda^{!}\mathtt{x}.\mathcal{C}_1)!\mathtt{Q}]$).
- Let $\mathtt{M} = \mathcal{C}[(\lambda \mathtt{x}.\mathtt{P})\mathcal{C}_2[\mathtt{N}]]$ ($\mathtt{M} = \mathcal{C}[(\lambda^{!}\mathtt{x}.\mathtt{P})!\mathcal{C}_2[\mathtt{N}]]$) and $\mathtt{M}' = \mathcal{C}[\mathtt{R}\{\mathcal{C}_2[\mathtt{N}]/\mathtt{x}\}]$; then, any occurrence of a subterm of $\mathtt{M}'$ defined by $\mathcal{C}[\mathcal{C}'_2]$, for some simple context $\mathcal{C}'_2$ such that $\mathcal{C}[\mathtt{R}\{\mathcal{C}_2[\mathtt{N}]/\mathtt{x}\}] = \mathcal{C}[\mathcal{C}'_2[\mathtt{N}]]$, is a descendant of the occurrence of $\mathtt{N}$ in $\mathtt{M}$ defined by $\mathcal{C}[(\lambda \mathtt{x}.\mathtt{P})\mathcal{C}_2]$.

A more standard definition of descendant, based on the notion of labeled reduction, can be found in [19], def. 4.2.

Example 5. Let $\mathtt{M} = (\lambda \mathtt{x}.\mathtt{x}\mathtt{x}\mathtt{y})(\lambda \mathtt{z}.\mathtt{z})$ and $\mathtt{M}' = (\lambda \mathtt{z}.\mathtt{z})(\lambda \mathtt{z}.\mathtt{z})\mathtt{y}$. Then both the occurrences of $(\lambda \mathtt{z}.\mathtt{z})$ in $\mathtt{M}'$ defined by $\Box(\lambda \mathtt{z}.\mathtt{z})\mathtt{y}$ and $(\lambda \mathtt{z}.\mathtt{z})\Box\mathtt{y}$ are descendant of the occurrence of $(\lambda \mathtt{z}.\mathtt{z})$ in $\mathtt{M}$ defined by $(\lambda \mathtt{x}.\mathtt{x}\mathtt{x}\mathtt{y})\Box$. Moreover $(\lambda \mathtt{z}.\mathtt{z})(\lambda \mathtt{z}.\mathtt{z})\Box$ in $\mathtt{M}'$ is a descendant of $(\lambda \mathtt{x}.\mathtt{x}\mathtt{x}\Box)(\lambda \mathtt{z}.\mathtt{z})$ in $\mathtt{M}$. Observe that, in the case $(\lambda \mathtt{y}.\mathtt{x})(\lambda \mathtt{z}.\mathtt{z}) \to \mathtt{x}$, the subterm occurrence $(\lambda \mathtt{y}.\mathtt{x})\Box$ in the redex has no descendant in the reduct.

Theorem 6. *Let $\mathtt{M}$ be well-formed, and let $\mathtt{M} \to \mathtt{M}'$.*

i. *$\mathtt{M}'$ is a well-formed term.*
ii. *The depth of a subterm occurrence in $\mathtt{M}$ and the depth of its descendants in $\mathtt{M}'$ are the same.*
iii. *$\delta(\mathtt{M}') \leq \delta(\mathtt{M})$.*

Proof.

i. From Property 4.
ii. Let $\mathtt{M} = \mathcal{C}[\mathtt{R}]$, where $\mathtt{R}$ is either a β-redex or a !-redex; the proof follows by induction on the definition of descendant. The complete proof is boring but easy, since it follows from the observation that since a λ-abstraction expects an input at depth 0 and its bound variable is at depth 0, the substitutions occur at depth 0; on the contrary, a $\lambda^{!}$-abstraction expects an input at depth 1 and its bound variable occurs at depth 1, so the substitutions occur at depth 1.
iii. If $\mathtt{M} = \mathcal{C}[\mathtt{R}]$ and $\mathtt{R}$ is either $(\lambda \mathtt{x}.\mathtt{P})\mathtt{Q}$ or $(\lambda^{!}\mathtt{x}.\mathtt{P})!\mathtt{Q}$, where $\mathtt{x} \notin \mathrm{FV}(\mathtt{R})$, then $\mathtt{M}' = \mathcal{C}[\mathtt{P}]$; so by definition $\delta(\mathtt{M}') \leq \delta(\mathtt{M})$. Otherwise $\mathtt{x} \in \mathrm{FV}(\mathtt{R})$ and by point ii. of this theorem the depth of every subterm occurrence in $\mathtt{M}$ and of its descendants in $\mathtt{M}'$ is the same, so $\delta(\mathtt{M}') = \delta(\mathtt{M})$, since the depth of term is the maximum between the depth of all its subterms.

Please cite this article in press as: P. Baillot et al., Characterizing polynomial and exponential complexity classes in elementary lambda-calculus, Inf. Comput. (2018), https://doi.org/10.1016/j.ic.2018.05.005

From this point onwards, we only consider the subclass of well-formed elementary λ-terms and we refer to it as $\lambda^!$-*calculus*.

Stratified reduction If $\mathtt{R}$ is a *redex at depth* i in $\mathtt{M}$, then there is a simple context $\mathcal{C}$ such that $\mathtt{M} = \mathcal{C}[\mathtt{R}]_i$ and $\delta(\mathcal{C}) = i$, where $\mathtt{R}$ is either a β or !-redex. Intuitively, $\to_i$ denotes a reduction of a redex at depth i; the notation $\overset{*}{\to}$ ($\overset{*}{\to}_i$) stands for the reflexive and transitive closure of $\to$ ($\to_i$).

In order to study the complexity of the reduction, it is necessary to define a measure of terms which is also based on their depth. The *size of* $\mathtt{M}$ *at depth* i, denoted by $|\mathtt{M}|_i$, is defined by induction on $\mathtt{M}$ as follows:

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

Let $\delta(\mathtt{M}) = d$; then the size of $\mathtt{M}$ *from depth* i is $|\mathtt{M}|_{i+} = \sum_{j=i}^{d} |\mathtt{M}|_j$, while the *size of* $\mathtt{M}$ is $|\mathtt{M}| = \sum_{i=0}^{d} |\mathtt{M}|_i$.

Note that in the previous definition we use the notion of size in a not standard way, since we measure the number of symbols different from !, being ! the measure of the depth. The definition above is extended naturally to contexts by imposing $|\Box|_i = 0$ for every $i \geq 0$. Observe that $|\mathcal{C}[\mathtt{N}_1]_{i_1}\ldots[\mathtt{N}_n]_{i_n}|_j = |\mathcal{C}|_j + a_1 + \ldots + a_n$, where $a_k = 0$ if $j < i_k$, $a_k = |\mathtt{N}_k|_{j-i_k}$ otherwise, for $1 \leq k \leq n$: indeed we need to take into account the (possible) increase of depth of each term $\mathtt{N}_k$ when plugged into the k-th hole of the context.

It is interesting to examine how the size at all depths lower than or equal to i changes, after reducing a redex at depth i:

Lemma 7. *If* $\mathtt{M} \to_i \mathtt{M}'$, *then* $|\mathtt{M}'|_i < |\mathtt{M}|_i$ *and* $|\mathtt{M}'|_j = |\mathtt{M}|_j$ *for* $j < i$.

Proof. If $\mathtt{M} \to_i \mathtt{M}'$ by reducing an erasing redex, then the proof is trivial. Otherwise, let $\mathtt{M} = \mathcal{C}[\mathtt{R}]$, where $\mathtt{R}$ is either a β-redex or a !-redex occurring at depth i: the proof follows easily by induction on $\mathcal{C}$.

We denote by $\mathbf{nf}_i$ the set of terms in i-*normal form*, where informally, a term is in i-normal form if and only if it does not contain any redex at depth less than or equal to i. As a consequence, $\mathtt{M}$ is in normal form if and only if it is in $\delta(\mathtt{M})$-normal form.

Confluence The set of well-formed terms can be coherently considered as a calculus: in fact the $\lambda^!$-calculus enjoys a confluence property with respect to the stratified reduction.

Lemma 8 *(Confluence).*

i. 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* $\mathtt{Q} \overset{*}{\to}_i \mathtt{N}$.
ii. 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* $\mathtt{Q} \overset{*}{\to} \mathtt{N}$.
iii. Let $\mathtt{M} \in \mathbf{nf}_i$ *and* $\mathtt{M} \to_j \mathtt{M}'$, *with* $j \geq i + 1$, *then* $\mathtt{M}' \in \mathbf{nf}_i$.

Proof.

i. The proof follows the one given in [20], which can be adapted taking into account the two shape of redexes and the notion of depth.
ii. The proof follows easily from point (i) of the current Lemma.
iii. The proof follows by induction on $\mathcal{C}$, such that $\mathtt{M} = \mathcal{C}[\mathtt{R}]$, being $\mathtt{R}$ the reduced redex.

Level-by-level reduction strategy By Lemma 8.iii, reducing a redex at a given depth does not create any redex at strictly lower depth. Such property of $\lambda^!$-calculus suggests that we consider the following non-deterministic *level-by-level reduction strategy*: if the term is not in normal form, then 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 it finishes with a normal term.

Lemma 9. *Any reduction of a term* $\mathtt{M}$ *by the level-by-level strategy terminates.*

Proof. Let $\mathtt{M}$ be any $\lambda^!$-term and let $\delta = \delta(\mathtt{M})$. By Lemma 6 we know that, in order to reduce $\mathtt{M}$, we must reduce it to a δ-normal form; then it is sufficient to show that any maximal level-by-level reduction sequence s of $\mathtt{M}$ contains an i-normal form, for any $i \leq \delta$, and the proof follows by choosing $i = \delta$.

Please cite this article in press as: P. Baillot et al., Characterizing polynomial and exponential complexity classes in elementary lambda-calculus, Inf. Comput. (2018), https://doi.org/10.1016/j.ic.2018.05.005

$$\mathtt{M}_0^1 \rightsquigarrow_0 \ldots \rightsquigarrow_0 \mathtt{M}_0^{n_0} = \mathtt{M}_1^1 \rightsquigarrow_1 \ldots \rightsquigarrow_1 \mathtt{M}_1^{n_1} \ldots \mathtt{M}_i^1 \rightsquigarrow_i \ldots \rightsquigarrow_i \mathtt{M}_i^{n_i} = \mathtt{M}_{i+1}^1 \ldots \rightsquigarrow_\delta \mathtt{M}_\delta^{n_\delta} = \mathtt{M}_{\delta+1}^1$$

Fig. 1. General shape of a level-by-level reduction sequence.

Note that a maximal level-by-level reduction sequence has the shape shown in Fig. 1, where $\rightsquigarrow_i$ denotes one reduction step at depth i according to the level-by-level strategy; we use simply $\rightsquigarrow$ when we do not refer to a particular depth. In the figure, $\mathtt{M}_i \rightsquigarrow_i$ denotes the term in $\mathbf{nf}_i$ reached after performing all the reduction steps at depth i, which are a finite number n_i, by Lemma 9.

Leftmost-by-level reduction strategy In particular cases, we use a deterministic version of the level-by-level strategy, called *leftmost*-by-level strategy, choosing at every step the leftmost redex which is not under the scope of a λ-abstraction.

A formal account of the leftmost-by-level strategy can be given through a notion of stratified evaluation context, where the stratified evaluation context of $\mathtt{M}$ at depth i, for $i \leq \delta(\mathtt{M})$, is denoted by $\mathcal{E}_i^{\mathtt{M}}$ and it is a simple context defined by induction on i:

- $\mathcal{E}_0^{\mathtt{M}}$ is defined inductively as

$\Box$	if $\mathtt{M} = (\lambda \mathtt{x}.\mathtt{P})\mathtt{Q}$ or $\mathtt{M} = (\lambda^! \mathtt{x}.\mathtt{P})!\mathtt{Q}$
$(\lambda^! \mathtt{x}.\mathtt{P})\mathcal{E}_0^{\mathtt{Q}}$	if $\mathtt{M} = (\lambda^! \mathtt{x}.\mathtt{P})\mathtt{Q}$ and $\mathtt{Q} \neq !\mathtt{Q}'$
$(\lambda^! \mathtt{x}.\mathcal{E}_0^{\mathtt{P}})\mathtt{Q}$	if $\mathtt{M} = (\lambda^! \mathtt{x}.\mathtt{P})\mathtt{Q}$, $\mathtt{Q} \in \mathbf{nf}_0$ and $\mathtt{Q} \neq !\mathtt{Q}'$
$\lambda \mathtt{x}.\mathcal{E}_0^{\mathtt{N}}$	if $\mathtt{M} = \lambda \mathtt{x}.\mathtt{N}$
$\lambda^! \mathtt{x}.\mathcal{E}_0^{\mathtt{N}}$	if $\mathtt{M} = \lambda^! \mathtt{x}.\mathtt{N}$
$\mathtt{N}\mathcal{E}_0^{\mathtt{Q}}$	if $\mathtt{M} = \mathtt{NQ}$, $\mathtt{N}$ is not an abstraction and $\mathtt{N} \in \mathbf{nf}_0$
$\mathcal{E}_0^{\mathtt{N}}\mathtt{Q}$	if $\mathtt{M} = \mathtt{NQ}$ and $\mathtt{N}$ is not an abstraction
undefined	in any other case

- Let $\mathtt{M} = \mathcal{C}\lfloor \mathtt{N}_1, \ldots, \mathtt{N}_n \rfloor_i$, and let $\mathtt{N}_k$ $(1 \leq k \leq n)$ be the leftmost subterm of $\mathtt{M}$ such that $\mathtt{N}_k \notin \mathbf{nf}_0$: then $\mathcal{E}_i^{\mathtt{M}} = \mathcal{C}\lfloor \mathtt{N}_1, \ldots, \mathtt{N}_{k-1}, \mathcal{E}_0^{\mathtt{N}_k}, \ldots \mathtt{N}_n \rfloor_i$.

Note that the evaluation context takes into account the fact that we have two kinds of redexes. At a given depth, it considers the leftmost application $\mathtt{MN}$ such that $\mathtt{M}$ is an abstraction: in case of a λ-abstraction, or in case of a $\lambda^!$-abstraction applied to $\mathtt{N} = !\mathtt{P}$, for some $\mathtt{P}$, this is the leftmost redex, otherwise it looks for the leftmost redex in the term $\mathtt{N}$. In practice, this corresponds to using a call-by-name discipline for β-redexes and a sort of call-by-value discipline for !-redexes [20]. So the evaluation context of a term $\mathtt{M}$ is:

$$\mathcal{E}^{\mathtt{M}} = \begin{cases} \mathcal{E}_i^{\mathtt{M}} & \text{where } i \text{ is the least } i \text{ such that } \mathcal{E}_i^{\mathtt{M}} \text{ is defined, if any;} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

We are now able to give a formal definition of the leftmost-by-level strategy. Let $\mathtt{M} \Longrightarrow \mathtt{M}'$ denote the fact that $\mathtt{M}'$ is obtained from $\mathtt{M}$ by one reduction step according to the leftmost-by-level strategy, then $\mathtt{M} \Longrightarrow \mathtt{M}'$ if there are an evaluation context $\mathcal{E}^{\mathtt{M}}$, a redex $\mathtt{N}$ and a term $\mathtt{N}'$ such that $\mathtt{M} = \mathcal{E}^{\mathtt{M}}[\mathtt{N}]$, $\mathtt{M}' = \mathcal{E}^{\mathtt{M}}[\mathtt{N}']$ and $\mathtt{N} \to \mathtt{N}'$. All notations for $\to$ are extended to $\rightsquigarrow$ and $\Longrightarrow$ in a straightforward way.

The importance of this strategy is evident from the next theorem.

Theorem 10 *(Size-growth). If $\mathtt{M} \overset{*}{\Longrightarrow}_i \mathtt{M}'$ by c reduction steps, then $|\mathtt{M}'| \leq |\mathtt{M}| \cdot (|\mathtt{M}| + 1)^c$ $(0 \leq i \leq \delta(\mathtt{M}))$.*

The proof is quite technical, and needs some more definitions and lemmas; it is given in the next subsection.

2.1. Proof of Theorem 10

Active points In order to find a sharp bound for the increase of size at depth greater than or equal to i, we need to identify a measure at depth i which decreases with every reduction at depth $i-1$, for every $i > 0$; clearly the size at depth i as given in the paragraph about stratified reduction does not have such a property, since a !-redex can make the size at depth i grow quadratically.

Let $\mathtt{M} \Longrightarrow_i \mathtt{M}'$, which represents the reduction of a redex at depth i following the leftmost-by-level strategy. The measure, denoted by $\lceil \mathtt{M} \rceil_i$, satisfying the previous constraint is the maximum number of potential duplications of a subterm at depth i during the reduction of $\mathtt{M}$, called *active points*. Note that such measure is *dynamic*, in the sense that it accounts for the number of occurrences of variables bound by a $\lambda^!$-abstraction which could be replaced during the reduction: therefore, this measure depends strictly on the evaluation strategy.

First we need to define the *maximum number of $\lambda^!$-bound occurrences at depth i* in $\mathtt{M}$, denoted by $\circ_i(\mathtt{M})$, for $i > 0$, as follows:

- If $i = 1$, then
  - $\mathtt{M} = \mathtt{x}$ implies $\circ_1(\mathtt{M}) = 0$;
  - $\mathtt{M} = \lambda\mathtt{x}.\mathtt{P}$ implies $\circ_1(\mathtt{M}) = \circ_1(\mathtt{R})$;
  - $\mathtt{M} = \lambda^!\mathtt{x}.\mathtt{P}$ implies $\circ_1(\mathtt{M}) = \max\{n_0(\mathtt{x}, \mathtt{R}), \circ_1(\mathtt{R})\}$;
  - $\mathtt{M} = \mathtt{NP}$ implies $\circ_1(\mathtt{M}) = \max\{\circ_1(\mathtt{N}), \circ_1(\mathtt{R})\}$;
  - $\mathtt{M} = !\mathtt{N}$ implies $\circ_1(\mathtt{M}) = 0$
- If $i > 1$, then $\mathtt{M} = \mathcal{C}\lfloor \mathtt{N}_1, ..., \mathtt{N}_m \rfloor_{i-1}$ implies $\circ_i(\mathtt{M}) = \max_{j=1}^{m} \circ_1(\mathtt{N}_j)$.

The number of *active points* of $\mathtt{M}$ at depth i, denoted by $\lceil \mathtt{M} \rceil_i$, is defined by induction on $\mathcal{E}^{\mathtt{M}}$ as follows:

- If $\mathcal{E}^{\mathtt{M}} = \mathcal{E}_0^{\mathtt{M}}$, then
  - $\mathcal{E}_0^{\mathtt{M}} = \Box$ implies $\lceil \mathtt{M} \rceil_1 = \circ_1(\mathtt{M})$;
  - $\mathcal{E}_0^{\mathtt{M}} = \lambda\mathtt{x}.\mathcal{E}_0^{\mathtt{N}}$ implies $\lceil \mathtt{M} \rceil_1 = \lceil \mathtt{N} \rceil_1$;
  - $\mathcal{E}_0^{\mathtt{M}} = \lambda^!\mathtt{x}.\mathcal{E}_0^{\mathtt{N}}$ implies $\lceil \mathtt{M} \rceil_1 = \lceil \mathtt{N} \rceil_1$;
  - $\mathcal{E}_0^{\mathtt{M}} = \mathtt{P}\mathcal{E}_0^{\mathtt{N}}$ implies $\lceil \mathtt{M} \rceil_1 = \lceil \mathtt{N} \rceil_1$;
  - $\mathcal{E}_0^{\mathtt{M}} = \mathcal{E}_0^{\mathtt{P}}\mathtt{N}$ implies $\lceil \mathtt{M} \rceil_1 = \max\{\lceil \mathtt{R} \rceil_1, \circ_1(\mathtt{N})\}$;
  - $\mathcal{E}_0^{\mathtt{M}} = (\lambda^!\mathtt{x}.\mathtt{P})\mathcal{E}_0^{\mathtt{N}}$ implies $\lceil \mathtt{M} \rceil_1 = \max\{\circ_1(\lambda^!\mathtt{x}.\mathtt{R}), \lceil \mathtt{N} \rceil_1\}$;
  - $\mathcal{E}_0^{\mathtt{M}} = (\lambda^!\mathtt{x}.\mathcal{E}_0^{\mathtt{P}})\mathtt{N}$ implies $\lceil \mathtt{M} \rceil_1 = \lceil \mathtt{R} \rceil_1$.
- If $\mathcal{E}^{\mathtt{M}} = \mathcal{E}_i^{\mathtt{M}}$ for some $i > 0$ and $\mathtt{M} = \mathcal{C}\lfloor \mathtt{N}_1, ..., \mathtt{N}_m \rfloor_i$, then $\lceil \mathtt{M} \rceil_{i+1} = \max_{j=1}^{m} \lceil \mathtt{N}_j \rceil_1$.
- If $\mathcal{E}_i^{\mathtt{M}}$ is undefined for some $i \geq 0$, then $\lceil \mathtt{M} \rceil_{i+1} = 0$.

Observe that the number of active points is undefined at depth 0: indeed there are no active points at depth 0, because all variables bound by a $\lambda^!$-abstraction occur at depth greater than 0. Moreover, it is easy to see that $\lceil \mathtt{M} \rceil_1 \leq |\mathtt{M}|_1$ for any $\mathtt{M}$, since the number of $\lambda^!$-bound occurrences is always bounded by the total number of occurrences of variables in $\mathtt{M}$. Furthermore, $\lceil \mathtt{M} \rceil_1 \leq |\mathtt{M}|_1 \leq |\mathtt{M}|$.

Note that, when $\mathcal{E}_0^{\mathtt{M}} = (\lambda^!\mathtt{x}.\mathtt{P})\mathcal{E}_0^{\mathtt{N}}$, it means that at this point of the computation we don't know yet whether $\mathtt{M}$ is a $!$-redex or a block; therefore we take into consideration the number of active points of $\mathtt{N}$, which will later be discarded in case $\mathtt{M}$ proves to be a block.

Example 11. Consider the term $\mathtt{M} = \lambda^!\mathtt{z}.(\lambda^!\mathtt{y}.(\lambda^!\mathtt{x}.!(\mathtt{zxx}))!(\mathtt{zyy}))!\mathtt{z}$: then $\circ_1(\mathtt{M}) = 3$, while $\mathcal{E}_0^{\mathtt{M}} = \lambda^!\mathtt{z}.\Box$ and $\lceil \mathtt{M} \rceil_1 = 2$.

Our main goal now is to show that the number of active points does not increase during a reduction. To do so, we first prove that the number of active points of a term $\mathtt{M}$ at depth i is bounded by $\circ_i(\mathtt{M})$; moreover, reducing a redex $\mathtt{M}$ to $\mathtt{M}'$ implies that the number of $\lambda^!$-bound occurrences does not increase:

Lemma 12.

i. If $\mathtt{x}$ *occurs at most once and at depth* 0 *in* $\mathtt{P}$*, then* $\circ_1(\mathtt{R}\{\mathtt{Q}/\mathtt{x}\}) \leq \max\{\circ_1(\mathtt{R}), \circ_1(\mathtt{Q})\}$*; if* $\mathtt{x}$ *occurs at depth* 1 *in* $\mathtt{P}$*, then* $\circ_1(\mathtt{R}\{\mathtt{Q}/\mathtt{x}\}) \leq \circ_1(\mathtt{R})$.
ii. $\lceil \mathtt{M} \rceil_{i+1} \leq \circ_{i+1}(\mathtt{M})$ *for all* $\mathtt{M}$*,* $i \geq 0$.
iii. Let $\mathtt{M} = (\lambda\mathtt{x}.\mathtt{P})\mathtt{Q}$ *or* $\mathtt{M} = (\lambda^!\mathtt{x}.\mathtt{P})!\mathtt{Q}$*: then* $\circ_1(\mathtt{R}\{\mathtt{Q}/\mathtt{x}\}) \leq \lceil \mathtt{M} \rceil_1$.

Proof.

i. Easy: in the first case, $\mathtt{Q}$ is copied only once and so the number of $\lambda^!$-bound occurrences is left untouched; in the second case $\mathtt{Q}$ might be copied multiple times, but since all $\lambda^!$-bound occurrences of $\mathtt{Q}$ are at depth greater than 1, their number does not count at depth 1.
ii. Easy, by induction on $\mathcal{E}^{\mathtt{M}}$.
iii. Easy, by the definition of bounded occurrences and active points.

The previous results help showing that the number of active points does not increase while reducing a term by the leftmost-by-level strategy:

Lemma 13. $\mathtt{M} \Longrightarrow \mathtt{M}'$ *implies* $\lceil \mathtt{M}' \rceil_{i+1} \leq \lceil \mathtt{M} \rceil_{i+1}$*, for every* $i \geq 0$.

Proof. Easy, by induction on $\mathcal{E}_0^{\mathtt{M}}$.

Example 14. Consider again the term $\mathtt{M} = \lambda^!\mathtt{z}.(\lambda^!\mathtt{y}.(\lambda^!\mathtt{x}.!(\mathtt{zxx}))!(\mathtt{zyy}))!\mathtt{z}$: since $\mathcal{E}_0^{\mathtt{M}} = \lambda^!\mathtt{z}.\Box$, we have $\mathtt{M} \Longrightarrow \mathtt{M}' = \lambda^!\mathtt{z}.(\lambda^!\mathtt{x}.!(\mathtt{zxx}))!(\mathtt{zzz})$ and $\lceil \mathtt{M} \rceil_1 = 2$. Moreover, $\mathcal{E}_0^{\mathtt{M}'} = \lambda^!\mathtt{z}.\Box$, so $\lceil \mathtt{M}' \rceil_1 = 2$.

Please cite this article in press as: P. Baillot et al., Characterizing polynomial and exponential complexity classes in elementary lambda-calculus, Inf. Comput. (2018), https://doi.org/10.1016/j.ic.2018.05.005

As a further step, we study how the size of the whole term changes at depths higher than n, when performing a $\Longrightarrow_n$ step:

Lemma 15. *$\mathtt{M} \Longrightarrow_n \mathtt{N}$ implies $|\mathtt{N}|_i \leq \lceil \mathtt{M} \rceil_{n+1} \cdot |\mathtt{M}|_i + |\mathtt{M}|_i$ for every $i > n$.*

Proof. We proceed by induction on n, and then by induction on $\mathcal{E}_n^{\mathtt{M}}$.

All these results allow us to easily obtain a sort of stratified version of the size-growth result when performing a reduction step at depth n, where the size of the reduced term at depth i is bounded by a function of the size of the initial term at depth i and of the number of its active points at depth $n+1$:

Lemma 16. *$\mathtt{M} \overset{*}{\Longrightarrow}_n \mathtt{M}'$ in k reduction steps implies $|\mathtt{M}'|_i \leq |\mathtt{M}|_i \cdot (\lceil \mathtt{M} \rceil_{n+1} + 1)^k$, for every $i > n$.*

Proof. Let $\mathtt{M} = \mathtt{M}_1 \Longrightarrow_n \mathtt{M}_2 \Longrightarrow_n \ldots \Longrightarrow_n \mathtt{M}_k = \mathtt{M}'$: we proceed by induction on k.

Let $k = 2$, so $\mathtt{M} \Longrightarrow_n \mathtt{M}'$ and $\lceil \mathtt{M} \rceil_{n+1} \leq |\mathtt{M}|_{n+1}$: then $|\mathtt{M}'|_i \leq |\mathtt{M}|_i \cdot (\lceil \mathtt{M} \rceil_{n+1} + 1)$ by Lemma 15.

Now let $\mathtt{M} \overset{*}{\Longrightarrow}_n \mathtt{M}_h$ in $h-1$ reduction steps, for some $h > 2$. By inductive hypothesis $|\mathtt{M}_h|_i \leq |\mathtt{M}|_i \cdot (\lceil \mathtt{M} \rceil_{n+1} + 1)^{h-1}$, for every $i > n$. If $\mathtt{M}_h \Longrightarrow_n \mathtt{M}_{h+1}$, then

$$\begin{aligned}
|\mathtt{M}_{h+1}|_i &\leq |\mathtt{M}_h|_i \cdot (\lceil \mathtt{M}_h \rceil_{n+1} + 1) && \text{by Lemma 15} \\
&\leq |\mathtt{M}_h|_i \cdot (\lceil \mathtt{M} \rceil_{n+1} + 1) && \text{by Lemma 13} \\
&\leq |\mathtt{M}|_i \cdot (\lceil \mathtt{M} \rceil_{n+1} + 1)^{h-1} \cdot (\lceil \mathtt{M} \rceil_{n+1} + 1) && \text{by inductive hypothesis} \\
&= |\mathtt{M}|_i \cdot (\lceil \mathtt{M} \rceil_{n+1} + 1)^h.
\end{aligned}$$

The proof of Theorem 10 is a direct corollary of the previous lemma. In fact, as a consequence of the definition of active points we know that $\lceil \mathtt{M} \rceil_{i+1} \leq |\mathtt{M}|$. Then, by Lemma 16, $|\mathtt{M}'|_j = |\mathtt{M}|_j$ for $j < i$, so $|\mathtt{M}'|_j \leq |\mathtt{M}|_j \cdot (\lceil \mathtt{M} \rceil_{i+1} + 1)^c$ also holds for $j \leq i$: then $|\mathtt{M}'| = \sum_{j=0}^{\delta(\mathtt{M}')} |\mathtt{M}'|_j \leq \sum_{j=0}^{\delta(\mathtt{M})} |\mathtt{M}|_j \cdot (\lceil \mathtt{M} \rceil_{i+1} + 1) \leq |\mathtt{M}| \cdot (\lceil \mathtt{M} \rceil_{i+1} + 1)^c$.

3. Computing with elementary lambda-calculus

Representation of functions Since our primary aim is representing functions, we first need to encode data in order to represent input and output of such functions.

The representation of boolean values is given by the familiar encoding $\mathtt{true} \overset{\text{def}}{=} \lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{x}$ and $\mathtt{false} \overset{\text{def}}{=} \lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{y}$, while the representation of tally integers and binary words cannot be given by the usual Church encoding, since the terms corresponding to Church integers and words do not satisfy the well-formedness condition, that is, neither are terms of $\lambda^!$-calculus. In order to overcome this issue, we use the following encodings for Church integers and Church binary words:

$$\begin{array}{llll}
n \in \mathbb{N}, & & \underline{n} & = \lambda^! \mathtt{f}.!(\lambda \mathtt{x}.\mathtt{f}\,(\mathtt{f}\,\ldots(\mathtt{f}\,\mathtt{x})\ldots)) \\
w \in \{0,1\}^\star, & w = \langle i_1, \ldots, i_n \rangle, & \underline{w} & = \lambda^! \mathtt{f}_0.\lambda^! \mathtt{f}_1.!(\lambda \mathtt{x}.\mathtt{f}_{i_1}\,(\mathtt{f}_{i_2}\,\ldots(\mathtt{f}_{i_n}\,\mathtt{x})\ldots))
\end{array}$$

where, by abuse of notation, we also denote by $\underline{1}$ the term $\lambda^! \mathtt{f}.!\mathtt{f}$.

As is well known, Church integers are convenient for computing iteration in λ-calculus, but not so much for computing the predecessor, because this needs to be done by iteration. Similarly Church words are not well-suited for computing the tail of a word, nor for defining a function on words by case distinction on its first bit. For this reason we will also consider a second encoding of binary words. These are called Scott words and allow for an easy representation of basic operations such as computing the tail of the word or representing a function on words by case distinction

Scott binary words are given by:

$$\widehat{\epsilon} \overset{\text{def}}{=} \lambda \mathtt{f}_0.\lambda \mathtt{f}_1.\lambda \mathtt{x}.\mathtt{x} \qquad \widehat{0w} \overset{\text{def}}{=} \lambda \mathtt{f}_0.\lambda \mathtt{f}_1.\lambda \mathtt{x}.\mathtt{f}_0 \widehat{w} \qquad \widehat{1w} \overset{\text{def}}{=} \lambda \mathtt{f}_0.\lambda \mathtt{f}_1.\lambda \mathtt{x}.\mathtt{f}_1 \widehat{w}$$

We will describe the basic operations on Scott binary word in Sect. 5, after we have defined a type system.

Observe that the terms encoding booleans and Scott binary words 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 **length**(w).

Representation of the computation Let $\mathtt{P}$ be a closed $\lambda^!$-term in normal form, which we call a *program*. We represent the computation of a program on a binary word $\underline{w}$, by considering applications of the form $\mathtt{P}!\underline{w}$, where the argument is at depth 1 because we want the program to be able to duplicate its input if needed. Concerning the shape of the result, since we want to allow computation at arbitrary depth, we require the output to be of the form $!^k\mathcal{D}$, where $k \in \mathbb{N}$ and $\mathcal{D}$ is one of the data representations above. We thus say that a function $f : \{0,1\}^\star \to \{0,1\}^\star$ is represented by a program $\mathtt{P}$ if

there exists $k \in \mathbb{N}$ such that, for any $w \in \{0,1\}^\star$, $\mathtt{P}!\underline{w} \xrightarrow{*} !^k\mathcal{D}$ where $\mathcal{D} = \widehat{f(w)}$. Note that here we are using the Church binary word representation for the input and the Scott binary word representation for the output; however, this definition can be adapted to functions with other domains and codomains: in particular the codomain can be simply $\{0,1\}$, which is represented by $\{\mathtt{true}, \mathtt{false}\}$.

Complexity bound We will analyze the complexity of the reduction of terms of the shape $\mathtt{P}!\underline{w}$ to their k-normal form, that is of reducing up to depth k, for $k \in \mathbb{N}$; the proof is based on Theorem 10.

Let 2_i^n be defined inductively as $2_0^x = x$ and $2_{i+1}^x = 2^{2_i^x}$.

Proposition 17. *Let $\mathtt{P}$ be a program; 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, if $k = 2$ then the bound is polynomial in n.*

Proof. We proceed by induction on k.

Without loss of generality, we can assume $\mathtt{P}$ to be a $\lambda^!$-abstraction of the form $\lambda^!\mathtt{y}.\mathtt{Q}$; otherwise $\mathtt{P}!\underline{w}$ is a normal form and the property holds.

Let $k = 2$ and consider a level-by-level reduction sequence of $\mathtt{M}_0^1 = \mathtt{P}!\underline{w}$: we examine round 0 and round 1 of the reduction sequence, following the notation of Fig. 1.

At round 0, the only reduction is $(\lambda^!\mathtt{y}.\mathtt{Q})!\underline{w} \to \mathtt{Q}\{\underline{w}/\mathtt{y}\} = \mathtt{M}_1^1$. Observe that $\mathtt{M}_1^1 \in \mathbf{nf}_0$ because the occurrences of $\mathtt{y}$ in $\mathtt{Q}$ are at depth 1.

Let us now consider round 1. Let $b = n_0(\mathtt{y}, \mathtt{Q})$, which is independent of n: then $|\mathtt{Q}\{\underline{w}/\mathtt{y}\}|_1 \leq |\mathtt{Q}|_1 + b \cdot |\underline{w}|_0$ and, by definition of $\underline{w}$, $|\underline{w}|_0 = 2$. Therefore, $|\mathtt{M}_1^1|_1 = |\mathtt{Q}\{\underline{w}/\mathtt{y}\}|_1 \leq |\mathtt{Q}|_1 + 2b$. Let c be the constant $|\mathtt{Q}|_1 + 2b$, which is again independent of n: by Lemma 7, the number of steps at the end of round 1 is bounded by c, so the part of the statement concerning the number of steps holds.

Now let $\mathtt{M}_2^1 \in \mathbf{nf}_1$ be the term obtained at the end of round 1. By Lemma 8.i (confluence) $\mathtt{M}_1^1 \overset{*}{\Longrightarrow}_1 \mathtt{M}_2^1$ and by Lemma 7 such reduction is performed in c' steps, where $c' \leq |\mathtt{M}_1^1|_1 \leq c$; therefore $|\mathtt{M}_2^1| \leq |\mathtt{M}_1^1| \cdot (|\mathtt{M}_1^1| + 1)^c$ by Theorem 10. Moreover $|\mathtt{M}_1^1| \leq |\mathtt{Q}| + b|\underline{w}|$, so the size is polynomial in n.

Let us assume the property holds for k: we prove that it holds also for $k+1$. By inductive hypothesis, $\mathtt{M}$ 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 $\mathtt{M}_k^1 \overset{*}{\rightsquigarrow}_k \mathtt{M}_{k+1}^1 \in \mathbf{nf}_k$.

By Lemma 7 the reduction sequence has at most $|\mathtt{M}_k^1|_k$ steps and $|\mathtt{M}_k^1|_k \leq |\mathtt{M}_k^1| \leq 2_{k-2}^{q(n)}$, therefore $\mathtt{M}$ 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 Lemma 8.i (depth-wise confluence) $\mathtt{M}_k^1 \overset{*}{\Longrightarrow} \mathtt{M}_{k+1}^1$ and by Lemma 7 and Theorem 10 we obtain

$$|\mathtt{M}_{k+1}^0| \leq |\mathtt{M}_k^1| \cdot (|\mathtt{M}_k^1| + 1)^{2_{k-2}^{q(n)}} \leq 2_{k-2}^{q(n)} \cdot (2_{k-2}^{2q(n)})^{2_{k-2}^{q(n)}} \leq 2_{k-2}^{q(n)} \cdot 2^{2_{k-2}^{3q(n)}} \leq 2_{k-1}^{q'(n)}$$

for some polynomial $q'(n)$: therefore the statement holds for $k+1$.

Approximations From Proposition 17 we can easily derive a $2_{k-2}^{q(n)}$ 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 $\mathtt{M}_{k+1}^1$. Unfortunately, this does not yield directly a *time* bound $O(2_{k-2}^{q(n)})$ for the simulation of this reduction on a Turing machine: indeed, even though the size of the final term of round k fulfills the desired bound, the same does not hold for intermediate terms of round k, for which the size at depth $k+1$ can grow exponentially.

Nonetheless, since we are only interested in the result of the computation at depth k, the size of the subterms at depth $k+1$ is actually irrelevant. In order to overcome such issue we introduce a notion of *approximation*, which allows to compute up to a certain depth k while ignoring the higher strata of the term: such notion is inspired by the semantics of stratified coherence spaces [21], where partial information is managed with the aim of obtaining more and more refinements of the actual computation.

In order to accommodate approximations, we extend the calculus with a constant $*$, whose sizes are $|*|_0 = 1$ and $|*|_{i+1} = 0$ for every $i \geq 0$. The definition is then extended to term contexts in the following way.

The *i-th approximation* of a context $\mathcal{C}$ for $i \in \mathbb{N}$, denoted by $\overline{\mathcal{C}}^i$, is defined by induction on $\mathcal{C}$ as follows:

$$\overline{!\mathcal{C}}^0 = !* \qquad \overline{!\mathcal{C}}^{i+1} = !\overline{\mathcal{C}}^i \qquad \overline{\mathtt{x}}^i = \mathtt{x} \qquad \overline{\Box}^i = \Box$$

$$\overline{\mathcal{C}\mathcal{C}'}^i = \overline{\mathcal{C}}^i\overline{\mathcal{C}'}^i \qquad \overline{\lambda\mathtt{x}.\mathcal{C}}^i = \lambda\mathtt{x}.\overline{\mathcal{C}}^i \qquad \overline{\lambda^!\mathtt{x}.\mathcal{C}}^i = \lambda^!\mathtt{x}.\overline{\mathcal{C}}^i$$

Observe that $\overline{\mathcal{C}\lfloor \mathtt{N}_1 \ldots \mathtt{N}_n \rfloor_j}^i = \overline{\mathcal{C}}^i \lfloor \overline{\mathtt{N}_1}^{i-j} \ldots \overline{\mathtt{N}_n}^{i-j} \rfloor_j$ if $j \leq i$; otherwise $\overline{\mathcal{C}\lfloor \mathtt{N}_1 \ldots \mathtt{N}_n \rfloor_j}^i = \overline{\mathcal{C}}^i$, since all subterms filling the holes at depth $j > i$ do not count when the approximation at depth i is considered. In practice, when considering the i-th approximation of a term $\mathtt{M}$, $\overline{\mathtt{M}}^i$ is obtained by replacing all subterms of $\mathtt{M}$ at depth $i+1$ by the constant $*$.

Please cite this article in press as: P. Baillot et al., Characterizing polynomial and exponential complexity classes in elementary lambda-calculus, Inf. Comput. (2018), https://doi.org/10.1016/j.ic.2018.05.005

Example 18. For every Church binary word $\underline{w}$, the approximations at depth 0 and $i+1$ are respectively $\overline{\underline{w}}^0 = \lambda^! f_0.\lambda^! f_1.!*$ and $\overline{\underline{w}}^{i+1} = \underline{w}$, for $i \geq 0$.

First we examine the effect of approximation on substitutions, reductions and contexts respectively:

Lemma 19 *(Approximation of substitution).*

i. If the occurrence of $\mathtt{x}$ *in* $\mathtt{M}$ *is at depth 0, then* $\overline{\mathtt{M}\{\mathtt{N}/\mathtt{x}\}}^i = \overline{\mathtt{M}}^i\{\overline{\mathtt{N}}^i/\mathtt{x}\}$.
ii. If all occurrences of $\mathtt{x}$ *in* $\mathtt{M}$ *are at depth 1, then* $\overline{\mathtt{M}\{\mathtt{N}/\mathtt{x}\}}^0 = \overline{\mathtt{M}}^0$, $\overline{\mathtt{M}\{\mathtt{N}/\mathtt{x}\}}^{i+1} = \overline{\mathtt{M}}^{i+1}\{\overline{\mathtt{N}}^i/\mathtt{x}\}$, *for* $i \geq 0$.

Proof.

i. Let $\mathtt{M} = \mathcal{C}[\mathtt{x}]_0$: then $\overline{\mathtt{M}\{\mathtt{N}/\mathtt{x}\}}^i = \overline{\mathcal{C}[\mathtt{N}]_0}^i = \overline{\mathcal{C}}^i[\overline{\mathtt{N}}^i]_0 = \overline{\mathtt{M}}^i\{\overline{\mathtt{N}}^i/\mathtt{x}\}$ for every $i \geq 0$.
ii. Let $\mathtt{M} = \mathcal{C}[\mathtt{x}]_1 \ldots [\mathtt{x}]_1$: then $\overline{\mathtt{M}\{\mathtt{N}/\mathtt{x}\}}^0 = \overline{\mathcal{C}[\mathtt{N}]_1 \ldots [\mathtt{N}]_1}^0 = \overline{\mathcal{C}}^0 = \overline{\mathtt{M}}^0$ and $\overline{\mathtt{M}\{\mathtt{N}/\mathtt{x}\}}^{i+1} = \overline{\mathcal{C}[\mathtt{N}]_1 \ldots [\mathtt{N}]_1}^{i+1} = \overline{\mathcal{C}}^{i+1}[\overline{\mathtt{N}}^i]_1 \ldots [\overline{\mathtt{N}}^i]_1 = \overline{\mathcal{C}}^{i+1}\{\overline{\mathtt{N}}^i/\mathtt{x}\}$ for every $i \geq 0$.

Lemma 20 *(Approximation of reduction).* $\overline{(\lambda \mathtt{x}.\mathtt{M})\mathtt{N}}^i \to \overline{\mathtt{M}\{\mathtt{N}/\mathtt{x}\}}^i$ *and* $\overline{(\lambda^! \mathtt{x}.\mathtt{M})!\mathtt{N}}^i \to \overline{\mathtt{M}\{\mathtt{N}/\mathtt{x}\}}^i$ *for* $i \geq 0$.

Proof. If the redex is erasing the proof is trivial.

Otherwise, the proof follows easily by Lemma 19: in the first case, by definition of $\lambda^!$-terms we know that $\mathtt{M} = \mathcal{C}[\mathtt{x}]_0$, so $\overline{(\lambda \mathtt{x}.\mathtt{M})\mathtt{N}}^i = (\lambda \mathtt{x}.\overline{\mathcal{C}[\mathtt{x}]_0}^i)\overline{\mathtt{N}}^i \to \overline{\mathcal{C}}^i[\overline{\mathtt{N}}^i]_0 = \overline{\mathtt{M}}^i[\overline{\mathtt{N}}^i/\mathtt{x}] = \overline{\mathtt{M}[\mathtt{N}/\mathtt{x}]}^i$, in the second case, since $\mathtt{M} = \mathcal{C}\lfloor \mathtt{x} \ldots \mathtt{x} \rfloor_1$ by definition, $\overline{(\lambda^! \mathtt{x}.\mathtt{M})!\mathtt{N}}^0 = (\lambda^! \mathtt{x}.\overline{\mathtt{M}}^0)!* \to \overline{\mathtt{M}}^0 = \overline{\mathtt{M}[\mathtt{N}/\mathtt{x}]}^0$ and $\overline{(\lambda^! \mathtt{x}.\mathtt{M})!\mathtt{N}}^{i+1} = (\lambda^! \mathtt{x}.\overline{\mathcal{C}\lfloor \mathtt{x} \ldots \mathtt{x} \rfloor_1}^{i+1})\overline{\mathtt{N}}^i \to \overline{\mathcal{C}}^{i+1}\lfloor \overline{\mathtt{N}}^i \ldots \overline{\mathtt{N}}^i \rfloor_1 = \overline{\mathtt{M}}^{i+1}[\overline{\mathtt{N}}^i/\mathtt{x}] = \overline{\mathtt{M}[\mathtt{N}/\mathtt{x}]}^{i+1}$ for every $i \leq 0$.

Lemma 21 *(Approximation and depth).*

i. $\mathtt{M} \to_j \mathtt{M}'$ *implies* $\overline{\mathtt{M}}^i \to_j \overline{\mathtt{M}'}^i$ *if* $j \leq i$, $\overline{\mathtt{M}}^i = \overline{\mathtt{M}'}^i$ *otherwise.*
ii. $\overline{\mathtt{M}}^i \to_i \overline{\mathtt{M}'}^i$ *implies* $|\overline{\mathtt{M}'}^i| < |\overline{\mathtt{M}}^i|$.

Proof.

i. As $\mathtt{M} \to \mathtt{M}'$ by one step at depth j, we have that $\mathtt{M} = \mathcal{C}[\mathtt{P}]_j$ and $\mathtt{M}' = \mathcal{C}[\mathtt{P}']_j$where $\mathtt{P}$ is a redex and $\mathtt{P}'$ is its contractum. Therefore by Lemma 19 $\overline{\mathtt{M}}^i = \overline{\mathtt{M}'}^i$ if $i+1 \leq j$, otherwise $\overline{\mathtt{M}}^i = \overline{\mathcal{C}}^i[\overline{\mathtt{R}}^{i-j}]_j \to \overline{\mathcal{C}}^i[\overline{\mathtt{R}'}^{i-j}]_j = \overline{\mathtt{M}'}^i$ by using Lemma 20, where the reductions takes place at depth j.
ii. Let $\overline{\mathtt{M}}^i \to \overline{\mathtt{M}'}^i$ by one reduction step at depth i. By Lemma 7 $|\overline{\mathtt{M}'}^i|_j \leq |\overline{\mathtt{M}}^i|_j$ for every $j < i$ and $|\overline{\mathtt{M}'}^i|_i < |\overline{\mathtt{M}}^i|_i$; now we examine the size of $\overline{\mathtt{M}}^i$ at depth $i+1$.
We know that $\overline{\mathtt{M}}^i = \mathcal{C}[\mathtt{P}]_i$ and $\overline{\mathtt{M}'}^i = \mathcal{C}[\mathtt{P}']_i$, where $\mathtt{P}$ is a redex and $\mathtt{P}'$ is its contractum:
– if $\mathtt{P}$ is a β-redex, then $|\overline{\mathtt{M}'}^i|_{i+1} < |\overline{\mathtt{M}}^i|_{i+1}$;
– if $\mathtt{P}$ is a !-redex, then $\overline{\mathtt{M}}^i = \mathcal{C}[\mathtt{P}]_i$ and $\mathtt{P} = (\lambda^! \mathtt{x}.\mathtt{N})!*$; moreover, all occurrences of $\mathtt{x}$ in $\mathtt{N}$ are at depth 1, so they occur at depth $i+1$ in $\overline{\mathtt{M}}^i$. Since any subterm of $\overline{\mathtt{M}}^i$ at depth $i+1$ is the constant $*$, there are no occurrences of $\mathtt{x}$ in $\mathtt{N}$: therefore $\mathtt{P}' = \mathtt{N}$ and $|\overline{\mathtt{M}'}^i|_{i+1} < |\overline{\mathtt{M}}^i|_{i+1}$.
If $j \geq i+2$, then $|\overline{\mathtt{M}'}^i|_j = 0 = |\overline{\mathtt{M}}^i|_j$ and so the statement follows.

Finally, with the aid of approximants, we are able to prove the time bound for the computation of the application $\mathtt{P}!\underline{w}$ on a Turing machine:

Proposition 22. *Let* $\mathtt{P}$ *be a program; for any* $k \geq 2$, *there exists a polynomial* q *such that for any* $w \in \{0,1\}^\star$, *the reduction of* $\overline{\mathtt{R}!\underline{w}}^k$ *to its* k*-normal form can be computed in time* $O(2_{k-2}^{q(n)})$ *on a Turing machine, where* $n = \mathbf{length}(w)$.

Proof. Observe that $\overline{\mathtt{R}!\underline{w}}^k = \overline{\mathtt{R}}^k!\underline{w}$. By Proposition 17 and Lemma 21.i, $\overline{\mathtt{R}}^k!\underline{w}$ reduces to its $(k-1)$-normal form $\overline{\mathtt{M}_k^1}^k$ in $O(2_{k-2}^{q(n)})$ steps and with intermediary terms of size $O(2_{k-2}^{q(n)})$. By Lemma 21.ii, the reduction of $\overline{\mathtt{M}_k^1}^k$ at depth k is done in $O(2_{k-2}^{q(n)})$ steps and with intermediary terms of size $O(2_{k-2}^{q(n)})$: therefore, we can 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 [12], for a suitably chosen polynomial p.

Please cite this article in press as: P. Baillot et al., Characterizing polynomial and exponential complexity classes in elementary lambda-calculus, Inf. Comput. (2018), https://doi.org/10.1016/j.ic.2018.05.005

Table 1
Derivation rules for typed $\lambda^!$-calculus.

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

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

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

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

$$\frac{\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M}:\forall \mathtt{a}.\mathtt{S}}{\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M}:\mathtt{S}\{\mathtt{S}'/\mathtt{a}\}}\ (\forall E) \qquad \frac{\emptyset \mid \emptyset \mid \Theta' \vdash \mathtt{M}:\sigma}{\Gamma\, !\Theta', \Delta \mid \Theta \vdash !\mathtt{M}:!\sigma}\ (!)$$

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

4. A type assignment system for $\lambda^!$-calculus

In the previous section we have seen complexity bounds on the reduction of $\lambda^!$-calculus programs. We will now introduce a type system for terms of this calculus, with several motivations. The first motivation, which is standard, is that we want to be able to specify the domain and codomain of a program (e.g. data-types such as Church binary words, booleans ...). The second motivation, which is a little bit less standard, is that we wish to ensure statically that the evaluation of a program fed with an input will not end up with a deadlock, and typically that it will reduce to a value, for instance a boolean or a word (in Church or Scott representation). Finally the third reason is that we would like our type system to provide us statically some information about the depth at which computation will be performed. Typically Proposition 22 has provided us a time bound for computation until depth k, but to use it in practice we need to know what is the depth up to which reduction needs to be performed so as to obtain as result a value. The types will provide this piece if information.

To reach these goals we introduce a type assignment system for $\lambda^!$-calculus based on ELL, such that all typed terms are also well-formed and therefore all previous results are preserved in the typed setting. In particular the information about the depth at which computation needs to performed will be given by the nesting of ! modalities in the type of the program.

In this section we will define the type system and prove some general properties such as subject-reduction. In the following sections we will then use the types to characterize complexity classes.

An elementary typing system The set $\mathcal{T}$ of types is generated by the following 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 Var. Types of the shape $!\sigma$ are called *modal*.

Observe that, among linear types, we distinguish the subclass of *strict* linear types, also featuring polymorphic types ($\forall \mathtt{a}.\mathtt{S}$) and type fixpoints ($\mu \mathtt{a}.\mathtt{S}$). Such a subclass contains all non-atomic linear types and is closed under substitution.

A *basis* is a partial function from variables to types, with finite domain, ranging over Γ, Δ, Θ. As usual, the domain of a basis Γ is denoted by $\mathrm{dom}(\Gamma)$ and it represents the set of variables for which there exists a mapping in Γ, that is, $\mathrm{dom}(\Gamma) = \{\mathtt{x} \mid \Gamma(\mathtt{x}) \text{ is defined}\}$. Taking inspiration from [15], we consider three different bases $\Gamma \mid \Delta \mid \Theta$ having pairwise disjoint domains, called respectively *linear*, *modal* and *parking* basis, such that Γ maps term variables to linear types, Δ maps term variables to modal types and Θ maps term variables to types.

The rules of the type assignment system are given in Table 1, where $\Gamma_1 \# \Gamma_2$ stands for $\mathrm{dom}(\Gamma_1) \cap \mathrm{dom}(\Gamma_2) = \emptyset$, while $\mathrm{FTV}(\sigma)$ denotes the set of free variables of σ and $\mathrm{FTV}(\Gamma) = \cup_{\mathtt{x} \in \mathrm{dom}\Gamma} \mathrm{FTV}(\Gamma(\mathtt{x}))$.

The typing system proves statements of the shape $\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M}:\sigma$; derivations are ranged over by Π, Σ, Φ, so $\Pi \triangleright \Gamma \mid \Delta \mid \Theta \vdash \mathtt{M}:\sigma$ identifies a particular derivation proving the statement $\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M}:\sigma$. When all three bases are empty, we write the derivation as $\Pi \triangleright \vdash \mathtt{M}:\sigma$.

We say that a term $\mathtt{M}$ is *well-typed* if and only if there is a derivation $\Pi \triangleright \Gamma \mid \Delta \mid \emptyset \vdash \mathtt{M}:\sigma$ for some Γ, Δ, σ: indeed, variables in the parking context are considered as having an intermediary status, since they eventually have to shift to the modal context in order for the term to be well-typed. Some comments about the rules are in order:

- rule $(\multimap E)$ deals with the linear basis in multiplicative way, while in additive way with the two other basis;
- there is no axiom rule for introducing variables in the modal basis, but a variable can be introduced in the parking basis and then moved to the modal basis by applying rule (!);
- there is no abstraction rule for variables in the parking basis, thus underlining the fact that parking variables only have a "temporary" status;

Please cite this article in press as: P. Baillot et al., Characterizing polynomial and exponential complexity classes in elementary lambda-calculus, Inf. Comput. (2018), https://doi.org/10.1016/j.ic.2018.05.005

$$\dfrac{\dfrac{\Pi \triangleright \Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \mathtt{S}}{\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \forall \mathtt{a}.\mathtt{S}}\ (\forall I)}{\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \mathtt{S}\{\mathtt{S}'/\mathtt{a}\}}\ (\forall E) \quad \rightsquigarrow \quad \Pi\{\mathtt{S}'/\mathtt{a}\} \triangleright \Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \mathtt{S}\{\mathtt{S}'/\mathtt{a}\}$$

Fig. 2. Erasing of a ∀-detour.

- with respect to the system of [15], rule (!) is restricted so that both the linear and modal bases in the premise are empty; such restriction guarantees that typed terms are also well-formed, so that a stratified complexity bound can be obtained, for a specific k depending on the type, instead of the usual rough elementary bound;
- rules (μI), (μE), $(\forall I)$ and $(\forall E)$, also referred to as *non-constructive* rules, are the only rules which do not contribute to the syntactic construction of the term.

By observing the shape of the rules, it is easy to see that new mappings of variables to types can be added to any basis by means of implicit weakening:

Lemma 23 *(Weakening). If* $\Gamma_1 \mid \Delta_1 \mid \Theta_1 \vdash \mathtt{M} : \sigma$*, then there is a derivation* $\Gamma_1, \Gamma_2 \mid \Delta_1, \Delta_2 \mid \Theta_1, \Theta_2 \vdash \mathtt{M} : \sigma$*, for every* $\Gamma_2, \Delta_2, \Theta_2$ *disjoint from each other and from* $\Gamma_1, \Delta_1, \Theta_1$*.*

Proof. Easy, by induction on Π.

Properties of the system As usual, in order to prove the subject reduction property we need a substitution lemma. Here such lemma is split into three points, one for each basis in which the substituted variable may occur:

Lemma 24 *(Substitution). Let* $\Gamma_1 \# \Gamma_2$*.*

i) If $\Pi \triangleright \Gamma_1, \mathtt{x} : A \mid \Delta \mid \Theta \vdash \mathtt{M} : \tau$ *and* $\Sigma \triangleright \Gamma_2 \mid \Delta \mid \Theta \vdash \mathtt{N} : A$*, then there is* $\Phi \triangleright \Gamma_1, \Gamma_2 \mid \Delta \mid \Theta \vdash \mathtt{M}\{\mathtt{N}/\mathtt{x}\} : \tau$*.*
ii) If $\Pi \triangleright \Gamma_1 \mid \Delta \mid \Theta, \mathtt{x} : \sigma \vdash \mathtt{M} : \tau$ *and* $\Sigma \triangleright \emptyset \mid \Delta \mid \Theta \vdash \mathtt{N} : \sigma$*, then there is* $\Phi \triangleright \Gamma_1 \mid \Delta \mid \Theta \vdash \mathtt{M}\{\mathtt{N}/\mathtt{x}\} : \tau$*.*
iii) If $\Pi \triangleright \Gamma_1 \mid \Delta, \mathtt{x} : !\sigma \mid \Theta \vdash \mathtt{M} : \tau$ *and* $\Sigma \triangleright \Gamma_2 \mid \Delta \mid \Theta \vdash !\mathtt{N} : !\sigma$*, then there is* $\Phi \triangleright \Gamma_1, \Gamma_2 \mid \Delta \mid \Theta \vdash \mathtt{M}\{\mathtt{N}/\mathtt{x}\} : \tau$*.*

Proof. All three points follow by induction on Π.

i) Easy.
ii) Easy.
iii) Observe that $\Sigma \triangleright \Gamma_2 \mid \Delta \mid \Theta \vdash !\mathtt{N} : !\sigma$ ends with an application of rule (!) to $\Sigma' \triangleright \emptyset \mid \emptyset \mid \Theta'' \vdash \mathtt{N} : \sigma$, where $\Delta = !\Theta'', \Delta''$ for some Δ''.
Consider the case where Π ends with an application of rule (!); then either $\mathtt{x} \notin \mathrm{FV}(\mathtt{R})$ and Π is

$$\dfrac{\Pi' \triangleright \emptyset \mid \emptyset \mid \Theta' \vdash \mathtt{P} : \rho}{\Gamma_1 \mid \Delta, \mathtt{x} : !\sigma \mid \Theta \vdash !\mathtt{P} : !\rho}\ (!)$$

where $\mathtt{M} = !\mathtt{P}$, $\tau = !\rho$ and $\Delta = !\Theta', \Delta'$, so Φ is

$$\dfrac{\Pi' \triangleright \emptyset \mid \emptyset \mid \Theta' \vdash \mathtt{P} : \rho}{\Gamma_1, \Gamma_2 \mid \Delta \mid \Theta \vdash !\mathtt{P} : !\rho}\ (!)$$

or $\mathtt{x} \in \mathrm{FV}(\mathtt{R})$ and Π is

$$\dfrac{\Pi' \triangleright \emptyset \mid \emptyset \mid \Theta', \mathtt{x} : \sigma \vdash \mathtt{P} : \rho}{\Gamma_1 \mid \Delta, \mathtt{x} : !\sigma \mid \Theta \vdash !\mathtt{P} : !\rho}\ (!)$$

where $\mathtt{M} = !\mathtt{P}$, $\tau = !\rho$ and $\Delta = !\Theta', \Delta'$. By Lemma 23 we can build $\Pi'' \triangleright \emptyset \mid \emptyset \mid \Theta''', \mathtt{x} : \sigma \vdash \mathtt{P} : \rho$ and $\Sigma'' \triangleright \emptyset \mid \emptyset \mid \Theta''' \vdash \mathtt{N} : \sigma$ such that $\Theta''' = \Theta', \Theta''$ and $\Delta = !\Theta''', \Delta'''$ for some context Δ'''. Then $\Phi \triangleright \Gamma_1, \Gamma_2 \mid \Delta \mid \Theta \vdash !(\mathtt{R}\{\mathtt{N}/\mathtt{x}\}) : !\rho$ is obtained by applying point (ii) of the current Lemma to Π'', followed by one application of rule (!).

In the spirit of formula-as-types approach, subject reduction is related to *detour elimination*. As usual, a detour occurs in a derivation whenever the application of a rule introducing a connective is immediately followed by an application of a rule eliminating the same connective. Here we have three kinds of detours, namely the ∀-detour, the μ-detour and the $\multimap$-detour. ∀ and μ detours in a derivation can be erased without altering the structure of the subject, following the procedures illustrated in Figs. 2 and 3 respectively.

A derivation Π is said to be *clean* if it does not contain any detour.

The subject reduction corresponds to a sequence of detour eliminations.

Theorem 25 *(Subject reduction).* $\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \sigma$ *and* $\mathtt{M} \to \mathtt{M}'$ *imply* $\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M}' : \sigma$*.*

Please cite this article in press as: P. Baillot et al., Characterizing polynomial and exponential complexity classes in elementary lambda-calculus, Inf. Comput. (2018), https://doi.org/10.1016/j.ic.2018.05.005

$$\dfrac{\dfrac{\Pi \triangleright \Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \mathtt{S}\{\mu\mathtt{a}.\mathtt{S}/\mathtt{a}\}}{\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \mu\mathtt{a}.\mathtt{S}}\ (\mu I)}{\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \mathtt{S}\{\mu\mathtt{a}.\mathtt{S}/\mathtt{a}\}}\ (\mu E) \quad \rightsquigarrow \quad \Pi \triangleright \Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \mathtt{S}\{\mu\mathtt{a}.\mathtt{S}/\mathtt{a}\}$$

Fig. 3. Erasing of a μ-detour.

Proof. Let $\mathtt{M} = \mathcal{C}[\mathtt{R}]$, where either $\mathtt{R} = (\lambda\mathtt{x}.\mathtt{P})\mathtt{Q}$ or $(\lambda^!\mathtt{x}.\mathtt{P})!\mathtt{Q}$: we proceed by induction on $\mathcal{C}$.

Let $\mathcal{C} = \Box$, so $\mathtt{M} = \mathtt{R}$ and $\mathtt{M}' = \mathtt{R}\{\mathtt{Q}/\mathtt{x}\}$. If $\mathtt{M}$ is a β-redex, then $\Pi \triangleright \Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \sigma$ is of the shape:

$$\dfrac{\dfrac{\dfrac{\Gamma', \mathtt{x} : \mathtt{A}'' \mid \Delta \mid \Theta \vdash \mathtt{P} : \sigma''}{\Gamma' \mid \Delta \mid \Theta \vdash \lambda\mathtt{x}.\mathtt{P} : \mathtt{A}'' \multimap \sigma''}\ (\multimap I^L)}{\Gamma' \mid \Delta \mid \Theta \vdash \lambda\mathtt{x}.\mathtt{P} : \mathtt{A}' \multimap \sigma'}\ \delta \qquad \Gamma'' \mid \Delta \mid \Theta \vdash \mathtt{N} : \mathtt{A}'}{\Gamma', \Gamma'' \mid \Delta \mid \Theta \vdash (\lambda\mathtt{x}.\mathtt{P})\mathtt{N} : \sigma'}\ (\multimap E)$$

Otherwise, if $\mathtt{M}$ is a !-redex, then Π is

$$\dfrac{\dfrac{\dfrac{\Gamma' \mid \Delta, \mathtt{x} : \tau'' \mid \Theta \vdash \mathtt{P} : \sigma''}{\Gamma' \mid \Delta \mid \Theta \vdash \lambda^!\mathtt{x}.\mathtt{P} : \tau'' \multimap \sigma''}\ (\multimap I^L)}{\Gamma' \mid \Delta \mid \Theta \vdash \lambda^!\mathtt{x}.\mathtt{P} : \tau' \multimap \sigma'}\ \delta \qquad \Gamma'' \mid \Delta \mid \Theta \vdash !\mathtt{N} : \tau'}{\Gamma', \Gamma'' \mid \Delta \mid \Theta \vdash (\lambda^!\mathtt{x}.\mathtt{P})!\mathtt{N} : \sigma'}\ (\multimap E)$$

In both cases, δ is a (possibly empty) sequence of non-constructive rules: we prove, by induction on the length of δ, that all its rules can be erased by a sequence of detour eliminations. If δ is empty, then the proof is trivial. Otherwise, the last applied rule must be an elimination rule. Let $\delta = \delta_1\delta_2$, where δ_2 is composed by elimination rules and δ_1 ends with an introduction rule. Then the last applied rule of δ_1 and the first of δ_2 are necessarily an introduction and an elimination of the same connective, so they form a detour that can be eliminated. By iterating the procedure we obtain, in case of a β-redex:

$$\dfrac{\dfrac{\Gamma', \mathtt{x} : \mathtt{A}' \mid \Delta \mid \Theta \vdash \mathtt{P} : \sigma'}{\Gamma' \mid \Delta \mid \Theta \vdash \lambda\mathtt{x}.\mathtt{P} : \mathtt{A}' \multimap \sigma'}\ (\multimap I^L) \qquad \Gamma'' \mid \Delta \mid \Theta \vdash \mathtt{N} : \mathtt{A}'}{\Sigma \triangleright \Gamma', \Gamma'' \mid \Delta \mid \Theta \vdash (\lambda\mathtt{x}.\mathtt{P})\mathtt{N} : \sigma'}\ (\multimap E)$$

Then by Lemma 24 there is $\Pi' \triangleright \Gamma', \Gamma'' \mid \Delta \mid \Theta \vdash \mathtt{R}\{\mathtt{N}/\mathtt{x}\} : \sigma'$. The case of a !-redex is similar.

All the other cases follow easily by induction.

Furthermore we examine the depth of the free occurrences of a variable in a typed $\lambda^!$-term. The following property, similar to the result given in Theorem 6 for the untyped calculus, essentially depends on the restriction imposed on the premise of rule (!), which is responsible for the fact that all free variables are either at depth 0 or 1, so effectively guaranteeing that the depth of a (sub)term does not increase:

Lemma 26. *Let* $\Pi \triangleright \Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \sigma$ *and* $\mathtt{x} \in FV(\mathtt{M})$*:*

- $\mathtt{x} \in dom(\Gamma) \cup dom(\Theta)$ *if and only if* $\mathtt{x}$ *occurs at depth* 0 *in* $\mathtt{M}$*,*
- $\mathtt{x} \in dom(\Delta)$ *if and only if* $\mathtt{x}$ *occurs at depth* 1 *in* $\mathtt{M}$*.*

Proof. By induction on Π.

Finally, by exploiting the results of Theorem 25 and Lemma 26, we are able to show that the typed terms are exactly those of $\lambda^!$-calculus:

Lemma 27. *If a term is well-typed, then it is also well-formed.*

Proof. Consider a derivation Π typing $\mathtt{M}$. For each subderivation of Π of the shape

$$\dfrac{\Gamma, \mathtt{x} : \mathtt{A} \mid \Delta \mid \Theta \vdash \mathtt{P} : \sigma}{\Gamma \mid \Delta \mid \Theta \vdash \lambda\mathtt{x}.\mathtt{P} : \mathtt{A} \multimap \sigma}\ (\multimap I^L) \quad \text{or} \quad \dfrac{\Gamma \mid \Delta, \mathtt{y} :!\tau \mid \Theta \vdash \mathtt{Q} : \sigma}{\Gamma \mid \Delta \mid \Theta \vdash \lambda^!\mathtt{y}.\mathtt{Q} :!\tau \multimap \sigma}\ (\multimap I^!)$$

by Lemma 26, $\mathtt{x}$ occurs at most once at depth 0 in $\mathtt{P}$, while all occurrences of $\mathtt{y}$ (if any) are at depth 1 in $\mathtt{Q}$: this corresponds exactly to the notion of well-formed terms.

Finally we now prove the functoriality of the ! modality, which will be useful in the following sections:

Please cite this article in press as: P. Baillot et al., Characterizing polynomial and exponential complexity classes in elementary lambda-calculus, Inf. Comput. (2018), https://doi.org/10.1016/j.ic.2018.05.005

Proposition 28. *Let* $\vdash \mathtt{M} : \sigma_1 \multimap \ldots \multimap \sigma_n \multimap \tau$ *for some* $n > 0$*: then there is a term* $\mathtt{M}_{k,\tau}$ *such that* $\vdash \mathtt{M}_{k,\tau} : !^k\sigma_1 \multimap \ldots \multimap !^k\sigma_n \multimap !^k\tau$*, and* $\mathtt{M}_{k,\tau}\,!^k\mathtt{P}_1 \ldots !^k\mathtt{P}_n$ *and* $!^k(\mathtt{M}\mathtt{P}_1 \ldots \mathtt{P}_n)$ *have the same normal form, for all closed terms* $\mathtt{P}_i$ $(1 \leq i \leq n)$*, for any* $k \geq 1$*.*

Proof. Note that the type τ is mentioned as subscript in the notation $\mathtt{M}_{k,\tau}$, because the decomposition of the type as $\sigma_1 \multimap \ldots \multimap \sigma_n \multimap \tau$ is unique only up to the choice of τ. For instance if $\vdash \mathtt{M} : \mathtt{a} \multimap (\mathtt{a} \multimap \mathtt{a})$, then we have $\vdash \mathtt{M}_{1,\mathtt{a}} : !\mathtt{a} \multimap !\mathtt{a} \multimap !\mathtt{a}$ and $\vdash \mathtt{M}_{1,\mathtt{a}\multimap\mathtt{a}} : !\mathtt{a} \multimap !(\mathtt{a} \multimap \mathtt{a})$.

Now we proceed with the proof, by induction on k.

Let $k = 1$. By applying n times rule $(\multimap E)$ to $\vdash \mathtt{M} : \sigma_1 \multimap \ldots \multimap \sigma_n \multimap \tau$ and to the parking axioms $\emptyset \mid \emptyset \mid \mathtt{x}_i : \sigma_i \vdash \mathtt{x}_i : \sigma_i$ $(1 \leq i \leq n)$, we obtain the proof $\emptyset \mid \emptyset \mid \mathtt{x}_1 : \sigma_1, \ldots, \mathtt{x}_n : \sigma_n \vdash \mathtt{M}\mathtt{x}_1 \ldots \mathtt{x}_n : \tau$. By applying rule (!) to such derivation, followed by n applications of rule $(\multimap I^!)$, we obtain $\vdash \lambda^!\mathtt{x}_1 \ldots \mathtt{x}_n.!(\mathtt{M}\mathtt{x}_1 \ldots \mathtt{x}_n) : !\sigma_1 \multimap \ldots !\sigma_n \multimap !\tau$. Observe that $(\lambda^!\mathtt{x}_1 \ldots \mathtt{x}_n.!(\mathtt{M}\mathtt{x}_1 \ldots \mathtt{x}_n))!\mathtt{P}_1 \ldots !\mathtt{P}_n$ and $!(\mathtt{M}\mathtt{P}_1 \ldots \mathtt{P}_n)$ have the same normal form, since the former can be easily reduced to the latter by n reduction steps:

$$\begin{array}{lcl}(\lambda^!\mathtt{x}_1 \ldots \mathtt{x}_n.!(\mathtt{M}\mathtt{x}_1 \ldots \mathtt{x}_n))!\mathtt{P}_1 \ldots !\mathtt{P}_n & \to & (\lambda^!\mathtt{x}_2 \ldots \mathtt{x}_n.!(\mathtt{M}\mathtt{P}_1\mathtt{x}_2 \ldots \mathtt{x}_n))!\mathtt{P}_2 \ldots !\mathtt{P}_n \\ & \overset{*}{\to} & (\lambda^!\mathtt{x}_n.!(\mathtt{M}\mathtt{P}_1\mathtt{P}_2 \ldots \mathtt{P}_{n-1}\mathtt{x}_n))!\mathtt{P}_n \\ & \to & !(\mathtt{M}\mathtt{P}_1 \ldots \mathtt{P}_n)\end{array}$$

The inductive case follows easily.

From this point onwards, for any given $\mathtt{M}$ of type $\sigma_1 \multimap \ldots \multimap \sigma_n \multimap \tau$, if the choice of τ is clear from the context we will simply denote by $\mathtt{M}_k$ the term $\mathtt{M}_{k,\tau}$ with the type $!^k\sigma_1 \multimap \ldots \multimap !^k\sigma_n \multimap !^k\tau$ defined by Proposition 28.

5. Characterization of complexity classes

Now we want to use our type system to characterize some complexity classes by types. We will for that take advantage of the representations of booleans, integers, words that we have introduced in Sect. 3 and for which we will now define corresponding data-types. Church words will be useful because they allow for iteration, and Scott words both because they allow to define functions by case distinction, and because they are represented by terms of depth 0. For readers familiar with safe recursion [3] one analogy one can have in mind is that Church words can be thought of as normal arguments and Scott words as safe arguments.

Types of data and pairs In Section 3 we introduced some $\lambda^!$-terms encoding data. For such terms we define the following *datatypes*, adapted from System F, representing respectively booleans, Church tally integers, Church binary words and Scott binary words:

$$\begin{array}{lcl}\mathtt{B} & \overset{\text{def}}{=} & \forall\mathtt{a}.\mathtt{a} \multimap \mathtt{a} \multimap \mathtt{a} \\ \mathtt{N} & \overset{\text{def}}{=} & \forall\mathtt{a}.!(\mathtt{a} \multimap \mathtt{a}) \multimap !(\mathtt{a} \multimap \mathtt{a}) \\ \mathtt{W} & \overset{\text{def}}{=} & \forall\mathtt{a}.!(\mathtt{a} \multimap \mathtt{a}) \multimap !(\mathtt{a} \multimap \mathtt{a}) \multimap !(\mathtt{a} \multimap \mathtt{a}) \\ \mathtt{W}_\mathtt{S} & \overset{\text{def}}{=} & \mu\mathtt{b}.\forall\mathtt{a}.(\mathtt{b} \multimap \mathtt{a}) \multimap (\mathtt{b} \multimap \mathtt{a}) \multimap (\mathtt{a} \multimap \mathtt{a})\end{array}$$

Note that Scott binary words have already been used in the literature for the study of light logics [22] [23] [24].

As usual, it is possible to define the connective $\otimes$ for pairs by second order as $\sigma \otimes \tau \overset{\text{def}}{=} \forall\mathtt{a}.(\sigma \multimap \tau \multimap \mathtt{a}) \multimap \mathtt{a}$, for which the following syntactic sugar on terms is specified:

$$\begin{array}{lcll}\mathtt{M}_1 \otimes \mathtt{M}_2 & \overset{\text{def}}{=} & \lambda\mathtt{x}.\mathtt{x}\mathtt{M}_1\mathtt{M}_2 & : \ \sigma_1 \otimes \sigma_2 \\ \lambda(\mathtt{x}_1 \otimes \mathtt{x}_2).\mathtt{M} & \overset{\text{def}}{=} & \lambda\mathtt{x}.(\mathtt{x}\lambda\mathtt{y}_1.\lambda\mathtt{y}_2.\lambda\mathtt{z}.\mathtt{z}\mathtt{y}_1\mathtt{y}_2)(\lambda\mathtt{x}_1.\lambda\mathtt{x}_2.\mathtt{M}) & : \ (\mathtt{A}_1 \otimes \mathtt{A}_2) \multimap \tau \\ \lambda^!(\mathtt{x}_1 \otimes \mathtt{x}_2).\mathtt{M} & \overset{\text{def}}{=} & \lambda\mathtt{x}.(\mathtt{x}\lambda^!\mathtt{y}_1.\lambda^!\mathtt{y}_2.\lambda\mathtt{z}.\mathtt{z}!\mathtt{y}_1!\mathtt{y}_2)(\lambda^!\mathtt{x}_1.\lambda^!\mathtt{x}_2.\mathtt{M}) & : \ (!\sigma_1 \otimes !\sigma_2) \multimap \tau \\ & & & \\ \pi_i & \overset{\text{def}}{=} & \lambda(\mathtt{x}_1 \otimes \mathtt{x}_2).\mathtt{x}_i & : \ (\mathtt{A}_1 \otimes \mathtt{A}_2) \multimap \mathtt{A}_i \quad i \in \{1,2\} \\ \pi_i^! & \overset{\text{def}}{=} & \lambda^!(\mathtt{x}_1 \otimes \mathtt{x}_2).!\mathtt{x}_i & : \ (!\sigma_1 \otimes !\sigma_2) \multimap !\sigma_i \quad i \in \{1,2\}\end{array}$$

Then the two following reduction rules are derivable:

$$(\lambda(\mathtt{x}_1 \otimes \mathtt{x}_2).\mathtt{N})(\mathtt{M}_1 \otimes \mathtt{M}_2) \to \mathtt{N}\{\mathtt{M}_1/\mathtt{x}_1, \mathtt{M}_2/\mathtt{x}_2\}$$
$$(\lambda^!(\mathtt{x}_1 \otimes \mathtt{x}_2).\mathtt{N})(!\mathtt{M}_1 \otimes !\mathtt{M}_2) \to \mathtt{N}\{\mathtt{M}_1/\mathtt{x}_1, \mathtt{M}_2/\mathtt{x}_2\}.$$

For more detail one can see [25].

Observe that, contrary to what one could expect, we chose not to define $\lambda^!(\mathtt{x}_1 \otimes \mathtt{x}_2).\mathtt{M}$ simply as $\lambda\mathtt{x}.\mathtt{x}(\lambda^!\mathtt{x}_1.\lambda^!\mathtt{x}_2.\mathtt{M})$. The reason behind this choice is that the application of the latter could require the substitution of a non-linear type to a type

variable, an operation which is forbidden by our typing system: indeed such behavior would undermine subject reduction, a crucial property for the characterization we aim to prove.

In the following it will be useful to handle beside pairs also tuples of n elements, for $n \geq 2$. For that the previous constructions can be generalized to the n-ary case, with:

$$\sigma_1 \otimes \sigma_2 \otimes \cdots \otimes \sigma_n \stackrel{\text{def}}{=} \forall \mathtt{a}.(\sigma_1 \multimap \sigma_2 \multimap \ldots \multimap \sigma_n \multimap \mathtt{a}) \multimap \mathtt{a}.$$

Now, let us observe that Scott words allow for an easy definition of the basic operations on binary words and of the $\mathtt{case}$ function:

$$\begin{array}{lll}
\mathtt{cons}^0 & \stackrel{\text{def}}{=} & \lambda \mathtt{w}.\lambda \mathtt{s}_0.\lambda \mathtt{s}_1.\lambda \mathtt{x}.\mathtt{s}_0\mathtt{w} : \mathsf{W}_\mathsf{S} \multimap \mathsf{W}_\mathsf{S} \\
\mathtt{cons}^1 & \stackrel{\text{def}}{=} & \lambda \mathtt{w}.\lambda \mathtt{s}_0.\lambda \mathtt{s}_1.\lambda \mathtt{x}.\mathtt{s}_1\mathtt{w} : \mathsf{W}_\mathsf{S} \multimap \mathsf{W}_\mathsf{S} \\
\mathtt{tail} & \stackrel{\text{def}}{=} & \lambda \mathtt{w}.\mathtt{w}\ I\ I\ \mathtt{nil} : \mathsf{W}_\mathsf{S} \multimap \mathsf{W}_\mathsf{S} \\
\mathtt{case} & \stackrel{\text{def}}{=} & \lambda \mathtt{s}_0.\lambda \mathtt{s}_1.\lambda \mathtt{x}.\lambda \mathtt{w}.\mathtt{w}\mathtt{s}_0\mathtt{s}_1\mathtt{x} : \forall \mathtt{a}.(\mathsf{W}_\mathsf{S} \multimap \mathtt{a}) \multimap (\mathsf{W}_\mathsf{S} \multimap \mathtt{a}) \multimap \mathtt{a} \multimap (\mathsf{W}_\mathsf{S} \multimap \mathtt{a})
\end{array}$$

Moreover we define
$\mathtt{length} \stackrel{\text{def}}{=} \lambda \mathtt{w}.\lambda^! \mathtt{s}.(\lambda^! \mathtt{x}.!(\lambda \mathtt{z}.\mathtt{xz}))(\mathtt{wss}) : \mathsf{W} \multimap \mathsf{N}$,
returning the length of a Church binary word as a tally integer, and
$\mathtt{conv} \stackrel{\text{def}}{=} \lambda \mathtt{w}.(\lambda^! \mathtt{x}.!(\mathtt{x}\ \mathtt{nil}))(\mathtt{w}\ !\mathtt{cons}^0\ !\mathtt{cons}^1) : \mathsf{W} \multimap !\mathsf{W}_\mathsf{S}$,
turning a Church binary word into the corresponding Scott binary word.

By combining some basic operations on Church binary words, we can define a program raising the depth of a Church word, namely $\mathtt{coer} : \mathsf{W} \multimap !\mathsf{W}$, whose behavior is $\mathtt{coer}\ \underline{w} = !\underline{w}$. By applying Proposition 28 to such term, it is possible to build the program $\mathtt{coer}^k \stackrel{\text{def}}{=} \lambda^! \mathtt{w}.\mathtt{coer}_k(\mathtt{coer}_{k-1}(\ldots(\mathtt{coer}_1\ !\mathtt{w})\ldots))$ of type $!\mathsf{W} \multimap !^{k+1}\mathsf{W}$ such that $\mathtt{coer}^k !\underline{w} = !^{k+1}\underline{w}$, for every $k \geq 1$ and for any Church word $\underline{w}$.

Finally, we define $\vdash \mathtt{iter} : !\mathtt{A} \multimap !(\mathtt{A} \multimap \mathtt{A}) \multimap \mathsf{N} \multimap !\mathtt{A}$ as the term that, when applied to a base, a step function and a value n, iterates the step function n times starting from the base; it is easy to check that, given terms $\mathtt{base}$ and $\mathtt{step}$, the terms $(\lambda^! \mathtt{b}.\lambda^! \mathtt{s}.\lambda \mathtt{y}.(\lambda^! \mathtt{x}.!(\mathtt{x}\ \mathtt{b}))(\mathtt{y}\ !\mathtt{s}))\ !\mathtt{base}\ !\mathtt{step}\ \underline{n}$ and $!(\mathtt{step}^n\ \mathtt{base})$ reduce to the same normal form.

Properties of typed $\lambda^!$-terms In order to give a characterization of functions through programs of $\lambda^!$-calculus, we need to be able to extract the shape of the term from its type, when both the term and the type satisfy some conditions of normalization and closure.

We start by examining under which conditions a $\lambda^!$-term with a ! type is either a ! term or an application:

Lemma 29. *Let $\Pi \triangleright \Gamma \mid \Delta \mid \emptyset \vdash \mathtt{M} : !\sigma$, where $\mathtt{M} \in \mathbf{nf}_0$. Then:*

i. $\Gamma = \emptyset$ implies $\mathtt{M} = !\mathtt{N}$ for some $\mathtt{N}$.
ii. if there is $\mathtt{x} \in FV(\mathtt{M})$ such that $\Gamma(\mathtt{x})$ is defined, then $\mathtt{M} = \mathtt{NQ}$ for some $\mathtt{N}, \mathtt{Q}$.

Proof. Both points follow by induction on Π. Observe that the only rules that can assign a non-linear type are (Ax^P), (!) and $(\multimap E)$.

Secondly, we identify some criteria according to which a term is *not* an application:

Lemma 30. *Let $\Pi \triangleright \Gamma \mid \emptyset \mid \emptyset \vdash \mathtt{M} : \sigma$ such that $\mathtt{M} \in \mathbf{nf}_0$. Then we have:*

i. if for every $\mathtt{x}$ in $FV(\mathtt{M})$ the type $\Gamma(\mathtt{x})$ is a type variable, then $\mathtt{M}$ is not an application;
ii. if σ is not a type variable and for every $\mathtt{x} \in FV(\mathtt{M})$ either $\Gamma(\mathtt{x})$ is a type variable or it is of the shape $\Gamma(\mathtt{x}) = \mathtt{A} \multimap \mathtt{a}$, where $\mathtt{a}$ is a type variable, then $\mathtt{M}$ is not an application.

Proof. Both points follow by induction on Π.

With these ingredients, we are finally able to prove the *reading property* of B and W_S, namely the fact that every derivation assigning such datatype is inhabited by a corresponding value, where both the type and the term are at depth $k \geq 0$:

Lemma 31 *(Reading property of B and W_S).*

i. If $\vdash \mathtt{M} : !^k\mathsf{B}$ and $\mathtt{M} \in \mathbf{nf}_k$, then $\mathtt{M} = !^k\mathtt{true}$ or $!^k\mathtt{false}$, for $k \geq 0$.
ii. If $\Gamma \mid \emptyset \mid \emptyset \vdash \mathtt{M} : !^k\mathsf{W}_\mathsf{S}$ and $\mathtt{M} \in \mathbf{nf}_k$, such that for each $\mathtt{x} \in dom(\Gamma)$ either $\Gamma(\mathtt{x}) = \mathtt{a}$ or $\Gamma(\mathtt{x}) = \mathtt{A} \multimap \mathtt{a}$, then $\mathtt{M} = !^k\widehat{w}$ for some Scott binary word $\widehat{w}$, for $k \geq 0$.

Please cite this article in press as: P. Baillot et al., Characterizing polynomial and exponential complexity classes in elementary lambda-calculus, Inf. Comput. (2018), https://doi.org/10.1016/j.ic.2018.05.005

Proof. Both points follow by induction on k.

5.1. Soundness

We are interested in giving a precise account of the hierarchy of classes characterized by the typed $\lambda^!$-calculus. In particular, we want to show that it is possible to represent exactly the class k-EXP by considering a subclass of typed $\lambda^!$-terms, whose type depends on the parameter k. Before presenting the main complexity result, we briefly introduce the notations used to identify the hierarchy of complexity classes.

Let $\mathtt{FDTIME}(f(n))$ and $\mathtt{DTIME}(f(n))$ be 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 that we wish to capture are the following ones:

$$\begin{array}{rcll} \mathtt{k\text{-}EXP} & = & \cup_{i\in\mathbb{N}}\mathtt{DTIME}(2_k^{n^i}) & \text{for } k\geq 0 \\ \mathtt{k\text{-}FEXP} & = & \cup_{i\in\mathbb{N}}\mathtt{FDTIME}(2_k^{n^i}) & \text{for } k\geq 0 \end{array}$$

In particular, observe that the classes of polynomial time predicates and the ones of polynomial time functions are defined respectively as $\mathtt{PTIME}=\cup_{i\in\mathbb{N}}\mathtt{DTIME}(n^i)=\mathtt{0\text{-}EXP}$ and $\mathtt{FPTIME}=\cup_{i\in\mathbb{N}}\mathtt{FDTIME}(n^i)=\mathtt{0\text{-}FEXP}$.

Now we want to study the relation between the type of a program representing a function and the complexity class to which the function belongs. In particular, we show that a closed term of type $!\mathrm{W}\multimap!^{k+2}\mathrm{B}$ and a closed term of type $!\mathrm{W}\multimap!^{k+2}\mathrm{W_S}$ represent respectively a predicate and a function which, when applied to a word of length n, can be evaluated in time $O(2_k^{p(n)})$ for some polynomial p.

Let $F(\sigma)$ denote the class of functions represented by programs of type σ. First, we prove that $F(!\mathrm{W}\multimap!^{k+2}\mathrm{B})\subseteq\mathtt{k\text{-}EXP}$, namely, that every program having type $!\mathrm{W}\multimap!^{k+2}\mathrm{B}$ represents (in the sense of Section 3:Representation of functions) a predicate of k-EXP:

Theorem 32. *Let* $\vdash \mathtt{P}:!\mathrm{W}\multimap!^{k+2}\mathrm{B}$ *and* $\vdash \underline{w}:\mathrm{W}$, *where* P *is a program and* $\mathbf{length}(w)=n$; *then the reduction* $\mathtt{P}!\underline{w}\xrightarrow{*}!^{k+2}\mathcal{D}$ *can be computed in time* $2_k^{p(n)}$, *where* $\mathcal{D}$ *is either* true *or* false *and* p *is a polynomial.*

Proof. Let $\mathtt{M}'$ be the normal form of $\mathtt{P}!\underline{w}$. By Proposition 22, we know that $\overline{\mathtt{R}!\underline{w}}^{k+2}$ 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 $\overline{\mathtt{M}'}^{k+2}=\mathtt{N}$ by combining Lemma 21.i and Lemma 8.ii.

Since $\mathtt{P}!\underline{w}$ has type $!^{k+2}\mathrm{B}$, by Theorem 25 $\mathtt{M}'$ is a closed term of type $!^{k+2}\mathrm{B}$: then by Lemma 31.i $\mathtt{M}'$ is either $!^{k+2}\mathtt{true}$ or $!^{k+2}\mathtt{false}$.

Since $\mathtt{N}=\overline{\mathtt{M}'}^{k+2}=\mathtt{M}'$, $\mathtt{P}!\underline{w}$ can be computed in time $O(2_k^{p(n)})$.

This result can be extended to show that $F(!\mathrm{W}\multimap!^{k+2}\mathrm{W_S})\subseteq\mathtt{k\text{-}FEXP}$, that is, every program of type $!\mathrm{W}\multimap!^{k+2}\mathrm{W_S}$ represents a function of k-FEXP. This can be easily proved by retracing the same steps of the previous proof and by applying the reading property of Scott binary words in order to extract the output:

Theorem 33. *Let* $\vdash \mathtt{P}:!\mathrm{W}\multimap!^{k+2}\mathrm{W_S}$ *and* $\vdash \underline{w}:\mathrm{W}$, *where* P *is a program and* $\mathbf{length}(w)=n$; *then the reduction* $\mathtt{P}!\underline{w}\xrightarrow{*}!^{k+2}\widehat{w'}$ *can be computed in time* $2_k^{p(n)}$, *where* $\widehat{w'}$ *is a Scott binary word and* p *is a polynomial.*

Proof. We proceed exactly as before, up to the application of the reading property: by Lemma 31.ii, the output of the computation is $!^{k+2}\widehat{w'}$, where $\widehat{w'}$ is a Scott binary word, and the evaluation of $\mathtt{P}!\underline{w}$ is done in time $O(2_k^{p(n)})$.

One might wonder why we should use Scott binary words as the output type for the representation of functions, instead of Church binary words, as the latter would seem like a more obvious direction to take.

Considering the case of $k=0$, the reason behind such refusal is easily explained by the fact that Church binary words are data of depth 1: from the point of view of expressivity, there are some FPTIME functions which are not captured by terms of type $!\mathrm{W}\multimap!\mathrm{W}$, while programs of type $!\mathrm{W}\multimap!^2\mathrm{W}$ allow also the typing of functions outside the FPTIME class, like exponential functions.

5.2. Completeness

We now want to show that for every predicate of k-EXP (resp. function of k-FEXP) there is *at least* one program of type $!\mathrm{W}\multimap!^{k+2}\mathrm{B}$ (resp. $!\mathrm{W}\multimap!^{k+2}\mathrm{W_S}$) representing it. In order to achieve such results, we simulate time-bounded Turing machines through terms of $\lambda^!$-calculus; we then prove that the computation of a desired function can be performed through a term of $\lambda^!$-calculus respecting the given time complexity bound.

First we show how to encode polynomials on tally integers, which are needed to bound the iterations of the machine.

Please cite this article in press as: P. Baillot et al., Characterizing polynomial and exponential complexity classes in elementary lambda-calculus, Inf. Comput. (2018), https://doi.org/10.1016/j.ic.2018.05.005

The multiplication and the addition of two tally integers are given respectively by $\vdash \mathtt{mult} \stackrel{\text{def}}{=} \lambda z_1.\lambda z_2.\lambda^! s.z_1(z_2 !s) : N \multimap N \multimap N$ and $\vdash \mathtt{add} \stackrel{\text{def}}{=} \lambda z_1.\lambda z_2.\lambda^! s.(\lambda^! x.\lambda^! y.!(\lambda z.x(yz)))(z_1 !s)(z_2 !s) : N \multimap N \multimap N$. As to exponentiation, mapping n to 2^n, it is given by $\vdash \mathtt{exp} \stackrel{\text{def}}{=} \lambda z.(\lambda^! f.!(f\ \underline{1}))(z\ !\underline{2}) : N \multimap !N$.

We will now see that by composing addition and multiplication with the term representing the exponential function, it is possible to represent all the functions of the form $2_k^{q(n)}$ on tally integers, for some polynomial $q(n)$ and $k \geq 0$:

Lemma 34. *If p is a polynomial over one variable with coefficients in $\mathbb{N}$, then*

i) there is $\mathtt{M}$ representing $p(n)$ such that $\vdash \mathtt{M} : !N \multimap !N$;
ii) there is $\mathtt{M}$ representing $2_k^{p(n)}$ such that $\vdash \mathtt{M} : !N \multimap !^{k+1} N$, for any $k \geq 1$.

Proof.

i) Let us write $p(n)$ as $\sum_{i=0}^{k} a_i n^i$. Any monomial $a_i n^i$ can be represented by a term $\mathtt{N}_i$ obtained by composing $\mathtt{mult}$ $(i+1)$ times using the variable $\mathtt{x}$ and the Church integer $\underline{a_i}$. This term is then typable as $\emptyset \mid \emptyset \mid \mathtt{x} : N \vdash \mathtt{N}_i : N$. Then a term $\mathtt{N}$ for $p(n)$ can be obtained from the $\mathtt{N}_i$s by composing $\mathtt{add}$ k times, and it can be typed as $\emptyset \mid \emptyset \mid \mathtt{x} : N \vdash \mathtt{N} : N$. Finally the term $\mathtt{M}$ is obtained as $\lambda^! x.!\mathtt{N}$ and is typable as $\vdash \lambda^! x.!\mathtt{N} : !N \multimap !N$.
ii) Let $\Pi_{\mathtt{exp}^k} \rhd \vdash \mathtt{exp}^k : N \multimap !^k N$, where $\mathtt{exp}^k n = \mathtt{exp}_{k-1}(\ldots(\mathtt{exp}_1(\mathtt{exp}\ n))\ldots)$ is the program representing the function 2_k^n, for every $k \geq 0$.
The derivation obtained by applying Proposition 28 to $\Pi_{\mathtt{exp}^k}$ is composed with $\vdash \mathtt{M} : !N \multimap !N$, where $\mathtt{M}$ represents the polynomial $q(n)$; then the term of type $!N \multimap !^{k+1} N$ representing $2_k^{q(n)}$ is $\lambda^! x.(\lambda^! y.!(\mathtt{exp}^k y))(\mathtt{M} x)$.

In the previous section we showed how to represent polynomials and exponentials on tally integers in $\lambda^!$-calculus: now, such results are employed to simulate $\mathtt{k}$-$\mathtt{EXP}$-time bounded Turing machines.

First, we can generalize the boolean datatype to the n-ary case by defining $B^n \stackrel{\text{def}}{=} \forall a.a \multimap \ldots \multimap a \multimap a$ with $(n+1)$ occurrences of $\mathtt{a}$, typing normal forms of the shape $\lambda x_1.\ldots.\lambda x_n.x_i$ for some $i \in \{1, \ldots, n\}$ and $n \geq 1$. Note that $B^2 = B$.

Let $\mathfrak{M}$ be a Turing machine with alphabet $\Sigma = \{0, 1\}$ and a set of n states $Q_n = \{q_1, \ldots, q_n\}$. We represent the configurations of a one-tape Turing machine $\mathfrak{M}$ over a binary alphabet with n states through a term $\mathtt{N}_L \otimes \mathtt{P} \otimes \mathtt{N}_R \otimes \mathtt{P}$, having type $C \stackrel{\text{def}}{=} W_S \otimes B \otimes W_S \otimes B^n$, where:

1. the first component of type W_S represents the portion of the tape on the left-hand side of the scanned symbol, in reverse order;
2. the second component of type B represents the currently scanned symbol;
3. the third component of type W_S represents the portion of the tape on the right-hand side of the scanned symbol;
4. the fourth and final component of type B^n represents the current state of the machine.

We define $\tilde{1}$ to be the first value of type B^n, where n is the number of states of $\mathfrak{M}$, which is conventionally chosen to represent the initial state of the machine.

As a last tool in order to prove the completeness result, we need to define three more terms, which allow us to simulate the operations of a Turing machine through $\lambda^!$-calculus:

Lemma 35 *(Transitions of a Turing machine). Let $\mathfrak{M}$ be a one-tape deterministic Turing machine over a binary alphabet; then the following programs can be typed:*

i) $\vdash \mathtt{init} : W_S \multimap C$, mapping a Scott binary word to the corresponding initial configuration of $\mathfrak{M}$;
ii) $\vdash \mathtt{step} : C \multimap C$, computing the next configuration of $\mathfrak{M}$ based on the current configuration;
iii) $\vdash \mathtt{accept} : C \multimap B$, returning $\mathtt{true}$ (respectively $\mathtt{false}$) if the state of the current configuration is accepting (respectively rejecting);
iv) $\vdash \mathtt{extract} : C \multimap W_S$, returning the binary word written on the tape.

Proof. We show how to define each of the described terms; the respective typings are trivial and thus their proof is omitted.

i. We define $\mathtt{init}$ as $\mathtt{init} \stackrel{\text{def}}{=} \mathtt{case}\ M_1 M_2 M_3$, where $\mathtt{M}_1 = \lambda w.\mathtt{nil} \otimes \mathtt{true} \otimes \mathtt{w} \otimes \tilde{1}$, $\mathtt{M}_2 = \lambda w.\mathtt{nil} \otimes \mathtt{false} \otimes \mathtt{w} \otimes \tilde{1}$ and $\mathtt{M}_3 = \mathtt{nil} \otimes \mathtt{false} \otimes \mathtt{nil} \otimes \tilde{1}$.
ii. The term $\mathtt{step}$ can be defined by retracing the transition function of $\mathfrak{M}$ and by doing a $\mathtt{case}$ distinction.
iii. We define $\mathtt{accept} \stackrel{\text{def}}{=} \lambda(y_L \otimes x \otimes y_R \otimes s).s Q_1 \ldots Q_n$ where, for $1 \leq i \leq n$, $Q_i = \mathtt{true}$ (respectively $Q_i = \mathtt{false}$) if the state encoded by the i-th element of type B^n is accepting (respectively rejecting) for $\mathfrak{M}$.
iv. The last term, returning the right-hand tape portion of the tape, is trivially defined by the projection π_3.

Please cite this article in press as: P. Baillot et al., Characterizing polynomial and exponential complexity classes in elementary lambda-calculus, Inf. Comput. (2018), https://doi.org/10.1016/j.ic.2018.05.005

Finally we have all the necessary ingredients to prove the main completeness result:

Theorem 36 *(Extensional completeness).*

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

Proof.

i. Let $\mathfrak{M}$ be a deterministic Turing machine of time $2_k^{q(n)}$ computing a binary function f on binary words.
By Lemma 34 there is a derivation $\vdash \mathtt{Q} : !\mathsf{N} \multimap !^{k+1}\mathsf{N}$ representing $2_k^{q(n)}$. Moreover, the following derivations can be easily built:

$$\begin{array}{lcl} \Pi & \triangleright & \emptyset \mid \mathtt{w} : !\mathsf{W} \mid \emptyset \vdash \mathtt{init}_{k+2}(\mathtt{conv}_{k+1}(\mathtt{coer}^k\ !\mathtt{w})) : !^{k+2}\mathsf{C} \\ \Sigma & \triangleright & \emptyset \mid \mathtt{w} : !\mathsf{W} \mid \emptyset \vdash !^{k+2}\mathtt{step} : !^{k+2}(\mathsf{C} \multimap \mathsf{C}) \\ \Phi & \triangleright & \emptyset \mid \mathtt{w} : !\mathsf{W} \mid \emptyset \vdash \mathtt{Q}(\mathtt{length}_1\ !\mathtt{w}) : !^{k+1}\mathsf{N} \end{array}$$

Recall that the notation $\mathtt{M}_I$, used for instance for the term $\mathtt{init}_{k+2}$ has been defined in Proposition 28. Observe that Π types the initial configuration at depth $k+2$, which is obtained by applying the initializing function to the input word w, while Σ types the step function at depth $k+2$ and Φ types the tally integer representation of $2_k^{q(n)}$ at depth $k+1$.
Let $\mathtt{P} = \mathtt{iter}_{k+1}(\mathtt{init}_{k+2}(\mathtt{conv}_{k+1}(\mathtt{coer}^k\ !\mathtt{w})))(!^{k+2}\mathtt{step})(\mathtt{Q}(\mathtt{length}_1\ !\mathtt{w}))$ be the term obtained by iterating the step function $2_k^{q(n)}$ times starting from the initial configuration; the latter is typed by applying
$\emptyset \mid \mathtt{w} : !\mathsf{W} \mid \emptyset \vdash \mathtt{iter}_{k+1} : !^{k+2}\mathsf{C} \multimap !^{k+2}(\mathsf{C} \multimap \mathsf{C}) \multimap !^{k+1}\mathsf{N} \multimap !^{k+2}\mathsf{C}$ to Π, Σ and Φ in turns; let $\Psi \triangleright \emptyset \mid w : !\mathsf{W} \mid \emptyset \vdash \mathtt{P} : !^{k+2}\mathsf{C}$ be the derivation obtained by this construction.
Then we can apply rule $(\multimap E)$ to $\emptyset \mid \mathtt{w} : !\mathsf{W} \mid \emptyset \vdash \mathtt{accept}_{k+2} : !^{k+2}\mathsf{C} \multimap !^{k+2}\mathsf{B}$ and to Ψ, followed by one application of rule $(\multimap I^!)$ to abstract over $\mathtt{w}$, in order to obtain the desired derivation.
ii. The proof retraces the same steps of the previous point, up to the construction of Ψ; then the desired derivation is obtained by applying rule $(\multimap E)$ to $\emptyset \mid \mathtt{w} : !\mathsf{W} \mid \emptyset \vdash \mathtt{extract}_{k+2} : !^{k+2}\mathsf{C} \multimap !^{k+2}\mathsf{W_S}$ and to Ψ, followed by one application of rule $(\multimap I^!)$ to abstract over $\mathtt{w}$.

Observe that, as usual, such completeness results hold in the sense of *function* representation, whereas few polynomial time *algorithms* can actually be implemented by $\lambda^!$-term of the desired type. We do not claim our typed language to be algorithmically expressive in this sense; instead, the interest of the present approach lies in the simple setting it provides in order to obtain a characterization of a family of complexity classes in a generic way.

By looking at the type of programs representing functions of FPTIME, one obvious drawback of such characterization is the fact that it does not account for compositionality; indeed polytime functions are closed by composition, while the mismatch of input and output type of our programs, paired with the nonexistence of a coercion from Scott to Church binary words, makes it so that they cannot be composed: for such reason, in the next section we offer an alternative characterization which is capable of solving the issue.

6. Composite types for an alternative characterization

In order to be able to compose programs representing FPTIME functions, we want to define a datatype which is better suited to serve as both input and output type of a function. Recall that in Theorem 36 (ii) we were using Church words (type W) as input and Scott words (type $\mathsf{W_S}$) as output. To identify input and output domain we now think of representing a word by an hybrid representation, a pair of a Church integer and a Scott word. More precisely the idea is to represent the word $w' \in \{0,1\}^\star$ by a pair $\langle n, w\rangle$, where $n \in \mathbb{N}$, $w \in \{0,1\}^\star$, such that the following invariant holds:

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

The $\otimes$ operator on types, defined in Section 5, comes in handy for the representation of a pair of datatypes. Observe that the iter term is typed in such a way that, if the natural number it expects is at depth i, then its other arguments and its result are at depth $i+1$. We thus introduce a new combined datatype defined as $!^{k+1}\mathsf{N} \otimes !^{k+2}\mathsf{W}_S$, for $k \geq 0$, containing a pair of a Church integer $\underline{n}$ and a Scott binary word $\widehat{w}$, where $\underline{n}$ is meant to represent the length of a list, whose content is described by $\widehat{w}$; so, in particular, we choose the datatype $!\mathsf{N} \otimes !^{!2}\mathsf{W}_S$ for $k = 0$.

It will be convenient to define a shorthand notation for case distinction on Scott words:

Notation 1 *(Match). We use the following notation:*

$$\begin{array}{lll} \mathtt{match}\ w\ \mathtt{with} & \mathtt{cons}_0 u & \Rightarrow \mathtt{M}_0[u,\overline{x}] \\ & \mathtt{cons}_1 u & \Rightarrow \mathtt{M}_1[u,\overline{x}] \\ & \mathtt{nil} & \Rightarrow \mathtt{N}[\overline{x}] \end{array}$$

as syntactic sugar for $((w\ \lambda u.\lambda\overline{x}.\mathtt{M}_0\ \lambda u.\lambda\overline{x}.\mathtt{M}_1\ \lambda\overline{x}.\mathtt{N})\ \overline{x})$*, where* $\overline{x}$ *stands for a sequence of variables* $\mathtt{x}_1 \ldots \mathtt{x}_n$*, while* $\lambda\overline{x}.\mathtt{M}$ *stands for the abstraction* $\lambda\mathtt{x}_1.\ \ldots\lambda\mathtt{x}_n.\mathtt{M}$*, for* $n \geq 1$*.*

This construction is intended to be used with w a term of type W_S. When computing on elements $!^{k+1}\underline{n}\otimes!^{k+2}\widehat{w}$ of type $!^{k+1}\mathrm{N}\otimes!^{k+2}\mathrm{W}_S$, we want to maintain the invariant that $\mathbf{length}(w) \leq n$. This can be enforced by the following lemma:

Lemma 37. *For any* $k \geq 0$*, there exists a term* $\vdash \mathtt{M} : !\mathrm{N} \multimap !^2\mathrm{W}_S \multimap !^{k+2}\mathrm{W}_S$ *for* $k \geq 0$ *such that, for any* $\underline{n}$ *and* $\widehat{w}$*,*

$$\mathtt{M}!\underline{n}!^2\widehat{w} \xrightarrow{*} !^{k+2}\widehat{w'}$$

where $w' = w$ *if* $\mathbf{length}(w) \leq n$*, otherwise* w' *is the prefix of* w *of length* n*.*

Proof. The idea for constructing the term $\mathtt{M}$ is a bit similar to that for defining coercions on datatype W (see Sect. 5 for these coercions). Here the first argument $!n$ of type !N will be used to drive an iteration allowing to read the Scott word given in second argument (of type W_S) and to "reconstruct" it at depth $k+2$, up to its nth bit. Concretely this iteration will be done by $\mathtt{M} = \lambda^! n.\lambda^! w.!((\lambda^! \mathtt{y}.\lambda^! \mathtt{z}.!((\pi_1(\mathtt{y}(I\otimes \mathtt{z})))!^k\widehat{\epsilon}))(n!\mathrm{step}_k)w)$, for a suitable term step_k depending on k. Let us now describe the construction in more detail, and in particular make step_k explicit.

Let $\mathtt{A}^i = (!^i\mathrm{W}_S \multimap !^i\mathrm{W}_S) \otimes \mathrm{W}_S = \forall \mathtt{a}.((!^i\mathrm{W}_S \multimap !^i\mathrm{W}_S) \multimap \mathrm{W}_S \multimap \mathtt{a}) \multimap \mathtt{a}$ and consider the following terms:

$$\begin{array}{lcll} \vdash \mathrm{step}_0 & = & \lambda(f\otimes u). & \mathtt{match}\ u\ \mathtt{with} \quad \mathtt{cons}_0 v \Rightarrow \lambda\mathtt{y}.(f\ (\mathtt{cons}^0\mathtt{y}))\otimes v \\ & & & \qquad\qquad\qquad\quad \mathtt{cons}_1 v \Rightarrow \lambda\mathtt{y}.(f\ (\mathtt{cons}^1\mathtt{y}))\otimes v \\ & & & \qquad\qquad\qquad\quad \mathtt{nil} \Rightarrow f\otimes \mathtt{nil} \\ & : & \mathtt{A}^0 \multimap \mathtt{A}^0 & \\ \vdash \mathrm{step}_{i+1} & = & \lambda(f\otimes u). & \mathtt{match}\ u\ \mathtt{with} \quad \mathtt{cons}_0 v \Rightarrow \lambda^!\mathtt{y}.(f\ !(\mathtt{cons}_i^0\mathtt{y}))\otimes v \\ & & & \qquad\qquad\qquad\quad \mathtt{cons}_1 v \Rightarrow \lambda^!\mathtt{y}.(f\ !(\mathtt{cons}_i^1\mathtt{y}))\otimes v \\ & & & \qquad\qquad\qquad\quad \mathtt{nil} \Rightarrow f\otimes \mathtt{nil} \\ & : & \mathtt{A}^{i+1} \multimap \mathtt{A}^{i+1} & \end{array}$$

where $\vdash \mathtt{cons}_i^0 : !^i\mathrm{W}_S \multimap !^i\mathrm{W}_S$ and $\vdash \mathtt{cons}_i^1 : !^i\mathrm{W}_S \multimap !^i\mathrm{W}_S$ are obtained as usual by applying Proposition 28 to $\vdash \mathtt{cons}^0 : \mathrm{W}_S \multimap \mathrm{W}_S$ and $\vdash \mathtt{cons}^1 : \mathrm{W}_S \multimap \mathrm{W}_S$ respectively.

Then the desired program $\mathtt{M}$ is built as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Sigma \quad \vdots \quad \Pi \quad \emptyset \mid \emptyset \mid \Theta \vdash n!\mathrm{step}_k : !(\mathtt{A}^k \multimap \mathtt{A}^k)}{\emptyset \mid \emptyset \mid \Theta \vdash (\lambda^!\mathtt{y}.\lambda^!\mathtt{z}.!((\pi_1(\mathtt{y}(I\otimes\mathtt{z})))!^k\widehat{\epsilon}))(n!\mathrm{step}_k) : !\mathrm{W}_S \multimap !^{k+1}\mathrm{W}_S}(\multimap E) \quad \dfrac{}{\emptyset \mid \emptyset \mid \Theta \vdash w : !\mathrm{W}_S}(Ax^P)}{\emptyset \mid \emptyset \mid \Theta \vdash (\lambda^!\mathtt{y}.\lambda^!\mathtt{z}.!((\pi_1(\mathtt{y}(I\otimes\mathtt{z})))!^k\widehat{\epsilon}))(n!\mathrm{step}_k)w : !^{k+1}\mathrm{W}_S}(\multimap E)}{\emptyset \mid n : !\mathrm{N}, w : !^2\mathrm{W}_S \mid \emptyset \vdash !((\lambda^!\mathtt{y}.\lambda^!\mathtt{z}.!((\pi_1(\mathtt{y}(I\otimes\mathtt{z})))!^k\widehat{\epsilon}))(n!\mathrm{step}_k)w) : !^{k+2}\mathrm{W}_S}(!)}{\vdash \lambda^! n.\lambda^! w.!((\lambda^!\mathtt{y}.\lambda^!\mathtt{z}.!((\pi_1(\mathtt{y}(I\otimes\mathtt{z})))!^k\widehat{\epsilon}))(n!\mathrm{step}_k)w) : !\mathrm{N} \multimap !^2\mathrm{W}_S \multimap !^{k+2}\mathrm{W}_S}(\multimap I^I_\otimes)$$

where Π is:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots \quad \emptyset \mid \emptyset \mid \Theta' \vdash \pi_1 : \mathtt{A}^k \multimap (!^k\mathrm{W}_S \multimap !^k\mathrm{W}_S) \quad \dfrac{\dfrac{}{\emptyset \mid \emptyset \mid \Theta' \vdash \mathtt{y} : \mathtt{A}^k \multimap \mathtt{A}^k}(Ax^P) \quad \Sigma}{\emptyset \mid \emptyset \mid \Theta' \vdash \mathtt{y}(I\otimes\mathtt{z}) : \mathtt{A}^k}}{\emptyset \mid \emptyset \mid \Theta' \vdash \pi_1(\mathtt{y}(I\otimes\mathtt{z})) : !^k\mathrm{W}_S \multimap !^k\mathrm{W}_S} \quad \vdots \quad \emptyset \mid \emptyset \mid \Theta' \vdash !^k\widehat{\epsilon} : !^k\mathrm{W}_S}(\multimap E)}{\emptyset \mid \emptyset \mid \Theta' \vdash (\pi_1(\mathtt{y}(I\otimes\mathtt{z})))!^k\widehat{\epsilon} : !^k\mathrm{W}_S}}{\emptyset \mid !\Theta' \mid \Theta \vdash !((\pi_1(\mathtt{y}(I\otimes\mathtt{z})))!^k\widehat{\epsilon}) : !^{k+1}\mathrm{W}_S}(!)}{\emptyset \mid \emptyset \mid \Theta \vdash \lambda^!\mathtt{y}.\lambda^!\mathtt{z}.!((\pi_1(\mathtt{y}(I\otimes\mathtt{z})))!^k\widehat{\epsilon}) : !(\mathtt{A}^k \multimap \mathtt{A}^k) \multimap !\mathrm{W}_S \multimap !^{k+1}\mathrm{W}_S}(\multimap I^L)$$

and Σ is

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \emptyset \mid \emptyset \mid \Theta' \vdash I : \forall \mathtt{a}.\mathtt{a} \multimap \mathtt{a}\end{array}}{\emptyset \mid \emptyset \mid \Theta' \vdash I : !^k \mathtt{W}_S \multimap !^k \mathtt{W}_S}\ (\forall E) \qquad \dfrac{}{\emptyset \mid \emptyset \mid \Theta' \vdash \mathtt{z} : \mathtt{W}_S}\ (Ax^P)}{\emptyset \mid \emptyset \mid \Theta' \vdash I \otimes \mathtt{z} : \mathtt{A}^k}\ (\otimes I)$$

with $\Theta = n : \mathrm{N}, w : !\mathrm{W}_S$ and $\Theta' = \mathtt{y} : \mathtt{A}^k \multimap \mathtt{A}^k, \mathtt{z} : \mathrm{W}_S$.

It is easy to check that $\mathtt{M}!\underline{n}!^2\widehat{w}$ and $!^2((\pi_1(\mathrm{step}_k^n(I \otimes \widehat{w})))!^k\widehat{\epsilon})$ reduce to the same normal form, which is the word $!^{k+2}\widehat{w'}$ such that w' is a prefix of w and $\mathbf{length}(w') \leq n$.

Retracing the reading properties of datatypes B and W_S, we need to examine under which conditions we can read out a value of the composite datatype. In order to do so, we need a few intermediary results about the shape of terms which are typed either with a ! type or with a type variable:

Lemma 38.

i. Let $\Pi \triangleright \Gamma \mid \emptyset \mid \Theta \vdash \mathtt{M} : !\tau$ *and* $\mathtt{M} \in \mathbf{nf}_0$*, such that* $\Gamma, \Theta \subseteq \mathtt{x} : \sigma_1 \multimap \ldots \multimap \sigma_n \multimap \mathtt{a}, \mathtt{y}_1 : \mathtt{a}_1, \ldots, \mathtt{y}_k : \mathtt{a}_k$ *($n \geq 1, k \geq 0$): then* $\mathtt{M} = !\mathtt{N}$ *for some* N.

ii. Let $\Pi \triangleright \Gamma \mid \emptyset \mid \Theta \vdash \mathtt{M} : \mathtt{a}$*, such that* $\mathtt{M} \in \mathbf{nf}_0$ *and* $\Gamma, \Theta = \mathtt{x} : \sigma_1 \multimap \ldots \multimap \sigma_n \multimap \mathtt{a}, \mathtt{y}_1 : \mathtt{a}_1, \ldots, \mathtt{y}_k : \mathtt{a}_k$ *($n \geq 1, k \geq 0$): then either* $\mathtt{M} = \mathtt{y}_i$*, if* $\mathtt{a} = \mathtt{a}_i$*, or* $\mathtt{M} = \mathtt{x}\mathtt{P}_1 \ldots \mathtt{P}_n$ *for some* $\mathtt{P}_1, \ldots, \mathtt{P}_n$.

Proof. Both points follow by induction on Π.

Now we are able to prove a reading result, similar to the ones of Lemma 31, for both pairs and Church integers:

Lemma 39 *(Reading property).*

i. If $\vdash \mathtt{M} : (\sigma_1 \otimes \sigma_2)$ *and* $\mathtt{M} \in \mathbf{nf}_0$*, then there are* $\vdash \mathtt{M}_1 : \sigma_1$ *and* $\vdash \mathtt{M}_2 : \sigma_2$ *such that* $\mathtt{M} = \mathtt{M}_1 \otimes \mathtt{M}_2$.

ii. If $\vdash \mathtt{M} : !^k\mathrm{N}$ *for* $k \geq 0$ *and* $\mathtt{M} \in \mathbf{nf}_{k+1}$*, then there exists* $n \in \mathbb{N}$ *such that* $\mathtt{M} = !^k\underline{n}$.

Proof.

i. Let Π be a clean derivation proving $\vdash \mathtt{M} : \forall \mathtt{a}.((\sigma_1 \multimap \sigma_2 \multimap \mathtt{a}) \multimap \mathtt{a})$; then it is sufficient to consider the shape of Π.

ii. Recall that $\underline{n} = \lambda^! f.!(\lambda \mathtt{x}.f^n\mathtt{x})$ for any $n \in \mathbb{N}$. Let Π be a clean derivation for $\vdash \mathtt{M} : !^k\mathrm{N}$: then the proof follows by induction on k. Recall that a clean derivation is a derivation without detours, as defined in Sect. 1 and that existence of a clean derivation is ensured by the subject-reduction Theorem 25.

Such results can be combined in order to prove a reading result for pairs of datatypes:

Lemma 40 *(Reading property of pairs). 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 are* $m \in \mathbb{N}$ *and* $w \in \{0,1\}^\star$ *such that* $\mathtt{M} = !^k\underline{m} \otimes !^{k+1}\widehat{w}$.

Proof. By Lemma 39.i, $\vdash \mathtt{M} : !^k\mathrm{N} \otimes !^{k+1}\mathrm{W}_S$ and $\mathtt{M} \in \mathbf{nf}_0$ imply there are derivations $\vdash \mathtt{M}_1 : !^k\mathrm{N}$ and $\vdash \mathtt{M}_2 : !^{k+1}\mathrm{W}_S$ such that $\mathtt{M} = \mathtt{M}_1 \otimes \mathtt{M}_2$: then the result follows easily by Properties 39.ii and 31.ii.

It is possible to prove a result similar to the one of Proposition 17, where a stratified bound on the size of the reduction is given for programs taking combined data as input:

Lemma 41. *Given a program* P*, for any* $k \geq 2$*, there exists a polynomial* q *such that, for any* $m \in \mathbb{N}$*,* $w \in \{0,1\}^\star$*,* $\mathtt{P}(!\underline{m} \otimes !^2\widehat{w}) \overset{*}{\rightsquigarrow} \mathtt{M}_k^1 \in \mathbf{nf}_{k-1}$ *in at most* $2_{k-2}^{q(n)}$ *steps, and* $|\mathtt{M}_k^1| \leq 2_{k-2}^{q(n)}$*, where* $n = m + \mathbf{length}(w)$*. In particular, in the case where* $k = 2$ *we have a polynomial bound* $q(n)$.

Proof. The statement can be proved in a way similar to Proposition 17. For $k = 2$, it is easy to check that the number of steps at depths 0 and 1 are bounded by a constant, since the size of the term $\mathtt{P}(!\underline{m} \otimes !^2\widehat{w})$ at depth lower than or equal to 1 does not depend on m nor w.

The complexity soundness result can be proved, in a similar way to that of Theorem 33, by combining the bound for the reduction in the untyped $\lambda^!$-calculus with the reading property of pairs:

Please cite this article in press as: P. Baillot et al., Characterizing polynomial and exponential complexity classes in elementary lambda-calculus, Inf. Comput. (2018), https://doi.org/10.1016/j.ic.2018.05.005

Theorem 42 *(Soundness). 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* $n = m + \mathbf{length}(w)$*.*

Proof. Easy, by combining Lemma 41, Theorem 25 and Lemma 39.i.

Finally we examine the matter of expressivity with respect to the combined datatype:

Theorem 43 *(Completeness). 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* $\vdash \mathtt{M} : (!\mathrm{N} \otimes !^2\mathrm{W}_S) \multimap (!^{k+1}\mathrm{N} \otimes !^{k+2}\mathrm{W}_S)$*.*

Proof. We show how to simulate a Turing machine for a function in `k-FEXP` through a term of type $(!\mathrm{N} \otimes !^2\mathrm{W}_S) \multimap (!^{k+1}\mathrm{N} \otimes !^{k+2}\mathrm{W}_S)$, in a way similar to the proof of Theorem 36.ii.

Consider a Turing machine $\mathfrak{M}$ of time $2_k^{q(n)}$ computing f, thus the size of the output is also bounded by $2_k^{q(n)}$. Let C, as defined in Sect. 5.2, denote the type of its configurations. By Lemma 34 there is $\vdash \mathtt{Q} : !\mathrm{N} \multimap !^{k+1}\mathrm{N}$ such that $\mathtt{Q}$ is a term representing $2_k^{q(n)}$. Let $!\underline{n} \otimes !^2\widehat{w}$ be the input of $\mathtt{M}$, and let w' be the binary word represented by $\langle n, w\rangle$; by Lemma 37 there is $\vdash \mathtt{P} : !\mathrm{N} \multimap !^2\mathrm{W}_S \multimap !^{k+2}\mathrm{W}_S$ such that $\mathtt{P}!\underline{n}!^2\widehat{w} \xrightarrow{*} !^{k+2}\widehat{w'}$.

Let $\Delta = m : !\mathrm{N}, u : !^2\mathrm{W}_S$; it is easy to build the following derivations:

$$\begin{array}{lll} \Pi & \triangleright & \emptyset \mid \Delta \mid \emptyset \vdash \mathtt{init}_{k+2}(\mathtt{P}!m!u) : !^{k+2}\mathrm{C} \\ \Sigma & \triangleright & \emptyset \mid \Delta \mid \emptyset \vdash !^{k+2}\mathtt{step} : !^{k+2}(\mathrm{C} \multimap \mathrm{C}) \\ \Phi & \triangleright & \emptyset \mid \Delta \mid \emptyset \vdash \mathtt{Q}\ !m : !^{k+1}\mathrm{N} \end{array}$$

Let $\mathtt{N} = \mathtt{iter}_{k+1}(\mathtt{init}_{k+2}(\mathtt{P}!m!u))(!^{k+2}\mathtt{step})(\mathtt{Q}\ !m)$ be obtained by applying $\mathtt{iter}_{k+1}$ to the typed terms above; by suitable applications of rule $(\multimap E)$ to $\emptyset \mid \Delta \mid \emptyset \vdash \mathtt{iter}_{k+1} : !^{k+2}\mathrm{C} \multimap !^{k+2}(\mathrm{C} \multimap \mathrm{C}) \multimap !^{k+1}\mathrm{N} \multimap !^{k+2}\mathrm{C}$ and derivations Π, Σ, Φ, we can build the derivation $\Psi \triangleright \emptyset \mid \Delta \mid \emptyset \vdash \mathrm{N} : !^{k+2}\mathrm{C}$. Then the desired derivation is $\vdash \lambda^!(m \otimes u).(\mathtt{Q}!m) \otimes (\mathtt{extract}_{k+2}\mathrm{N}) : (!\mathrm{N} \otimes !^2\mathrm{W}_S) \multimap (!^{k+1}\mathrm{N} \otimes !^{k+2}\mathrm{W}_S)$. Observe here that the size of the output word w'' is bounded by the running time $2_k^{q(n)}$, namely the clock of the Turing machine. Then the composite term $!^{k+1}n' \otimes !^{k+2}\widehat{w''}$ representing the output of the computation, where $n' = 2_k^{q(n)}$, respects the invariant stated at the beginning of Section 6 and thus is a correct representation of w''.

Note that, in particular, terms of type $(!\mathrm{N} \otimes !^2\mathrm{W}_S) \multimap (!^{k+1}\mathrm{N} \otimes !^{k+2}\mathrm{W}_S)$ correspond to the class `k-FEXP`. Some aspects of this alternative characterization are worth mentioning: on the one hand, we can now compose two terms of type $(!\mathrm{N} \otimes !^2\mathrm{W}_S) \multimap (!\mathrm{N} \otimes !^2\mathrm{W}_S)$, mirroring the fact that `FPTIME` is closed under composition; on the other hand, if $f \in$ `FPTIME` and $g \in$ `k-FEXP`, then by Theorem 43 there are $\Pi \triangleright \vdash \mathtt{M} : (!\mathrm{N} \otimes !^2\mathrm{W}_S) \multimap (!\mathrm{N} \otimes !^2\mathrm{W}_S)$ representing f and $\Sigma \triangleright \vdash \mathrm{N} : (!\mathrm{N} \otimes !^2\mathrm{W}_S) \multimap (!^{k+1}\mathrm{N} \otimes !^{k+2}\mathrm{W}_S)$ representing g: then we can compose these programs as $\vdash \lambda x.\mathrm{N}(\mathtt{M}x) : (!\mathrm{N} \otimes !^2\mathrm{W}_S) \multimap (!^{k+1}\mathrm{N} \otimes !^{k+2}\mathrm{W}_S)$, showing that $g \circ f \in$ `k-FEXP`. While the previous characterization of `k-FEXP`, given in Section 5, offers the advantage of simplicity by employing classical datatypes (Church and Scott binary words), this alternative composite characterization offers a better account of the closure properties of the considered complexity classes, at the price of a slightly more involved representation of words.

Moreover, as shown in the next subsection, the flexibility of the latter choice allows also to tackle other less explored characterizations.

6.1. Other characterizations through composite datatypes

We give two examples of characterizations, beside that of the `k-FEXP` hierarchy, which can be achieved by carefully tuning the composite datatype introduced at the beginning of Section 6.

Characterization of poly`FEXP` We denote by *poly*`FEXP` the class of functions computable in exponential time on a deterministic Turing machine, whose output size is polynomially bounded with respect to the input. In order to obtain this effect, we choose the type $!\mathrm{N} \otimes !^3\mathrm{W}_S$ as the output type of the program: the intuition is that, when looking at the components of the pair, we can actually see only a part of the binary word, since the tally integer represented by the first component is smaller than the second component. This general reasoning can be exploited in order to characterize classes of functions whose output's size is bounded by a function of the input:

Theorem 44 *(Characterization of poly*`FEXP`*).*

i. Let $\vdash \mathtt{P} : !\mathrm{N} \otimes !^2\mathrm{W}_S \multimap !\mathrm{N} \otimes !^3\mathrm{W}_S$*; then there exists a polynomial* q *such that, for any* $n \in \mathbb{N}$ *and* $w \in \{0,1\}^\star$ *such that* $\mathbf{length}(w) \leq n$*,* $\mathtt{P}(!\underline{n} \otimes !^2\widehat{w}) \overset{*}{\rightsquigarrow} \mathtt{Q} \in \mathbf{nf}_3$ *can be computed in time* $O(2^{q(n)})$ *on a Turing machine and* $\mathtt{Q} = !\underline{n'} \otimes !^3\widehat{w'}$ *represents a word* w'' *with* $\mathbf{length}(w'') \leq q(n)$*.*

ii. Let f *be a function on binary words in poly*`FEXP`*: then there is a term* $\mathtt{M}$ *representing* f *such that* $\vdash \mathtt{M} : (!\mathrm{N} \otimes !^2\mathrm{W}_S) \multimap (!\mathrm{N} \otimes !^3\mathrm{W}_S)$*.*

Please cite this article in press as: P. Baillot et al., Characterizing polynomial and exponential complexity classes in elementary lambda-calculus, Inf. Comput. (2018), https://doi.org/10.1016/j.ic.2018.05.005

Proof.

i. By examining the reduction of $\mathtt{P}(\lambda \mathtt{y}.\mathtt{y}\ !\underline{n}\ !^2\widehat{w})$, of type $!\mathrm{N}\otimes !^3\mathrm{W}_S$, using the level-by-level reduction strategy we obtain that there exist polynomials p and q such that: the number of steps for the reduction at depth 3 is bounded by $2^{p(n)}$, and if the result is denoted as $!\underline{n'}\otimes !^3\widehat{w'}$, then as $\underline{n'}$ is at depth 1 we have that $n' \leq q(n)$. It thus follows (by definition) that the word w'' represented by $!\underline{n'}\otimes !^3\widehat{w'}$ has length bounded by $q(n)$.
ii. Without loss of generality we can consider a Turing machine $\mathfrak{M}$ computing f in time $2^{q(n)}$ and with an output of size bounded by $q(n)$. Proceeding in a similar way as in the proof of Theorem 43 we can construct a term M_1 of type $(!\mathrm{N}\otimes !^2\mathrm{W}_S) \multimap !^3\mathrm{W}_S$ which, when it is fed with $(!\underline{n}\otimes !^2\widehat{w})$ representing w, computes $!^3\widehat{w'}$ such that $w' = f(w)$. We also know that there exists a term M_2 of type $!\mathrm{N} \multimap !\mathrm{N}$ computing the polynomial q. By combining M_1 and M_2 we can then define M of type $!\mathrm{N}\otimes !^2\mathrm{W}_S \multimap !\mathrm{N}\otimes !^3\mathrm{W}_S$ such that $M\ (!\underline{n}\otimes !^2\widehat{w})$ reduces to $!\underline{n'}\otimes !^3\widehat{w'}$ where $n' = q(n)$.

Characterization of NP problems The class NP is known as the class of languages that have polynomial time verifiers [26], that is, those languages $\mathcal{L}$ for which given an input u and an additional polysize word w' one can check in polynomial time whether w' is a witness of the fact u belongs to $\mathcal{L}$. More formally, given a language $\mathcal{L}$, we say that $\mathcal{L}$ is in NP if and only if there are two polynomials p, q and a deterministic Turing machine $\mathfrak{M}$ such that:

- for all u, w' the machine $\mathfrak{M}$ runs in time $p(|u| + |w'|))$ on input (u, w');
- for all $u \in \mathcal{L}$, there is w' of length $q(|u|)$ such that $\mathfrak{M}$ accepts on input (u, w');
- for all $u \notin \mathcal{L}$, for all w' of length $q(|u|)$, $\mathfrak{M}$ rejects on input (u, w').

We want to show that it is possible, using the deterministic language of $\lambda^!$-calculus, to also characterize the class NP.

We say that a predicate is *w-representable* if it is representable through a witness:

Definition 45. Let $g : \{0,1\}^* \to \{0,1\}$ be a predicate: we say that g is w-represented by $\vdash \mathtt{M} : !\mathrm{N}\otimes !^2\mathrm{W}_S \otimes !^2\mathrm{W}_S \multimap !^2\mathrm{B}$ if, for any word u represented by $!\underline{n}\otimes !^2\widehat{w}$,

$$g(u) = \begin{cases} 1 & \text{if } \exists w' \in \{0,1\}^*.\mathtt{M}\ (!\underline{n}\otimes !^2\widehat{w}\otimes !^2\widehat{w'}) \overset{*}{\rightsquigarrow} !^2\mathtt{true}; \\ 0 & \text{if } \forall w' \in \{0,1\}^*.\mathtt{M}\ (!\underline{n}\otimes !^2\widehat{w}\otimes !^2\widehat{w'}) \overset{*}{\rightsquigarrow} !^2\mathtt{false}. \end{cases}$$

Note that the main difference between this definition and that of NP languages is that here there is no explicit bound on the length of the candidate witnesses w'. We will however show that the predicates w-representable with such type are exactly those of NP:

Theorem 46 *(Characterization of NP).*

i. If g is w-represented by $\vdash \mathtt{P} : !\mathrm{N}\otimes !^2\mathrm{W}_S \otimes !^2\mathrm{W}_S \multimap !^2\mathrm{B}$, then $g \in$ NP.
ii. If $g \in$ NP, then there exists a program P such that the function g is w-represented by $\vdash \mathtt{P} : !\mathrm{N}\otimes !^2\mathrm{W}_S \otimes !^2\mathrm{W}_S \multimap !^2\mathrm{B}$.

Proof.

i. Let $\mathtt{P} = \lambda^!(\mathtt{x}_1 \otimes \mathtt{x}_2 \otimes \mathtt{x}_3).\mathrm{N}$: again, it is sufficient to study the reduction of terms $\vdash (\lambda^!(\mathtt{x}_1 \otimes \mathtt{x}_2 \otimes \mathtt{x}_3).\mathrm{N})(!\underline{n}\otimes !^2\widehat{w}\otimes !^2\widehat{w'}) : !^2\mathrm{B}$ using the level-by-level reduction strategy, where $!\underline{n}\otimes !^2\widehat{w}$ represents the word w'' such that $\mathbf{length}(w'') \leq n$. To conclude we need to show that there exists a polynomial q such that if $g(w) = 1$ then there is a witness w' of size $q(n)$, and if $g(w) = 1$ then there is no witness of size $q(n)$.
We know that:
   - there exists a polynomial p such that the reduction of all such terms up to depth 2 can be done in a number of steps bounded by $p(n)$;
   - if $g(w) = 1$ there exists one word w'_0 such that this term reduces to $\mathtt{true}$.

   We choose as polynomial q for the candidate witnesses the polynomial p. If $g(w) = 0$ then the claim holds because there is no witness of any size. If $g(w) = 1$ consider the word w'_1 which is the prefix of w'_0 of length $q(n)$. As the number of steps of the computation is bounded by $q(n)$ and as reading an extra bit on a Scott word *costs* one step, during the whole computation only the first $q(n)$ bits of w'_0 can be read. Thus the computations on inputs $(!\underline{n}\otimes !^2\widehat{w}\otimes !^2\widehat{w'_0})$ and $(!\underline{n}\otimes !^2\widehat{w}\otimes !^2\widehat{w'_1})$ yield the same result, which is $!^2\mathtt{true}$. Therefore w'_1 is a suitable witness of length $q(n)$, and we have proved that the language belongs to NP.
ii. Consider a Turing machine $\mathfrak{M}$ of time $p(n)$, such that $\mathfrak{M}$ is a verifier for g; then there exists a polynomial q such that:
   - if $g(u) = 0$ then for all w' of size $q(n)$ the machine rejects on input (u, w');
   - if $g(u) = 1$ then there is a w' of size $q(n)$ such that the machine accepts on input (u, w').

   Then one can define a Turing machine $\mathfrak{M}'$ of polynomial time $r(n)$ such that on input (u, w'):
   - $\mathfrak{M}'$ first examines if its second input is of size inferior to $q(n)$, and if it is not it rejects right away;

Please cite this article in press as: P. Baillot et al., Characterizing polynomial and exponential complexity classes in elementary lambda-calculus, Inf. Comput. (2018), https://doi.org/10.1016/j.ic.2018.05.005

• otherwise it behaves as $\mathfrak{M}$ on input (u, w').

Then we have that: if $g(u) = 0$ then on any input (u, w') the machine $\mathfrak{M}'$ rejects; if $g(u) = 1$ then there exists a w' such that on input (u, w') $\mathfrak{M}'$ accepts.

Then by proceeding as in the proof of Theorem 43 we can show that there exists a term P that simulates the machine $\mathfrak{M}'$, so we conclude that g is w-represented by P.

7. 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 [2] [7] is that the complexity bound can be deduced by looking only at the interface of the program (its type) without referring 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] N.D. Jones, Computability and Complexity – from a Programming Perspective, Foundations of Computing Series, MIT Press, 1997.
[2] N.D. Jones, The expressive power of higher-order types or, life without CONS, J. Funct. Program. 11 (1) (2001) 5–94.
[3] S. Bellantoni, S.A. Cook, A new recursion-theoretic characterization of the polytime functions (extended abstract), in: S.R. Kosaraju, M. Fellows, A. Wigderson, J.A. Ellis (Eds.), Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4–6, 1992, Victoria, British Columbia, Canada, ACM, 1992, pp. 283–293.
[4] D. Leivant, Predicative recurrence and computational complexity I: word recurrence and poly-time, in: Feasible Mathematics II, Birkhauser, 1994, pp. 320–343.
[5] D. Leivant, J. Marion, Lambda calculus characterizations of poly-time, Fundam. Inform. 19 (1/2) (1993) 167–184.
[6] J. Girard, Light linear logic, Inf. Comput. 143 (2) (1998) 175–204, https://doi.org/10.1006/inco.1998.2700.
[7] D. Leivant, Calibrating computational feasibility by abstraction rank, in: 17th IEEE Symposium on Logic in Computer Science (LICS 2002), 22–25 July 2002, Copenhagen, Denmark, Proceedings, IEEE Computer Society, 2002, p. 345.
[8] P. Baillot, Elementary linear logic revisited for polynomial time and an exponential time hierarchy, in: H. Yang (Ed.), Programming Languages and Systems – 9th Asian Symposium, APLAS 2011, Kenting, Taiwan, December 5–7, 2011, Proceedings, in: Lecture Notes in Computer Science, vol. 7078, Springer, 2011, pp. 337–352, http://dx.doi.org/10.1007/978-3-642-25318-8.
[9] V. Danos, J. Joinet, Linear logic and elementary time, Inf. Comput. 183 (1) (2003) 123–137, https://doi.org/10.1016/S0890-5401(03)00010-5.
[10] D. Mazza, Linear logic and polynomial time, Math. Struct. Comput. Sci. 16 (6) (2006) 947–988, https://doi.org/10.1017/S0960129506005688.
[11] U. Dal Lago, Context semantics, linear logic, and computational complexity, ACM Trans. Comput. Log. 10 (4) (2009), https://doi.org/10.1145/1555746.1555749.
[12] K. Terui, Light affine lambda calculus and polynomial time strong normalization, Arch. Math. Log. 46 (3–4) (2007) 253–280, https://doi.org/10.1007/s00153-007-0042-6.
[13] A. Madet, R.M. Amadio, An elementary affine λ-calculus with multithreading and side effects, in: C.L. Ong (Ed.), Typed Lambda Calculi and Applications – 10th International Conference, TLCA 2011, Novi Sad, Serbia, June 1–3, 2011, Proceedings, in: Lecture Notes in Computer Science, vol. 6690, Springer, 2011, pp. 138–152, http://dx.doi.org/10.1007/978-3-642-21691-6.
[14] A. Madet, Complexité Implicite de Lambda-Calculus Concurrents, Theses, Université Paris-Diderot, Paris VII, Dec. 2012, https://tel.archives-ouvertes.fr/tel-00794977.
[15] P. Coppola, U. Dal Lago, S. Ronchi Della Rocca, Light logics and the call-by-value lambda calculus, Log. Methods Comput. Sci. 4 (4) (2008), https://doi.org/10.2168/LMCS-4(4:5)2008.
[16] S. Ronchi Della Rocca, L. Roversi, Lambda calculus and intuitionistic linear logic, Stud. Log. 59 (3) (1997) 417–448, https://doi.org/10.1023/A:1005092630115.
[17] U. Dal Lago, A. Masini, M. Zorzi, Quantum implicit computational complexity, Theor. Comput. Sci. 411 (2) (2010) 377–409, https://doi.org/10.1016/j.tcs.2009.07.045.
[18] P. Baillot, E. De Benedetti, S. Ronchi Della Rocca, Characterizing polynomial and exponential complexity classes in elementary lambda-calculus, in: J. Diaz, I. Lanese, D. Sangiorgi (Eds.), Theoretical Computer Science – 8th IFIP TC 1/WG 2.2 International Conference, TCS 2014, Rome, Italy, September 1–3, 2014, Proceedings, in: Lecture Notes in Computer Science, vol. 8705, Springer, 2014, pp. 151–163, http://dx.doi.org/10.1007/978-3-662-44602-7.
[19] I. Bethke, J.W. Klop, R.C. de Vrijer, Descendants and origins in term rewriting, Inf. Comput. 159 (1–2) (2000) 59–124, https://doi.org/10.1006/inco.2000.2876.
[20] S. Ronchi Della Rocca, L. Paolini, The Parametric λ-Calculus: A Metamodel for Computation, Texts in Theoretical Computer Science: an EATCS Series, Springer-Verlag, Berlin, 2004.
[21] P. Baillot, Stratified coherence spaces: a denotational semantics for light linear logic, Theor. Comput. Sci. 318 (1–2) (2004) 29–55, https://doi.org/10.1016/j.tcs.2003.10.015.
[22] U. Dal Lago, P. Baillot, On light logics, uniform encodings and polynomial time, Math. Struct. Comput. Sci. 16 (4) (2006) 713–733, https://doi.org/10.1017/S0960129506005421.
[23] L. Roversi, L. Vercelli, Safe recursion on notation into a light logic by levels, in: Baillot [27], pp. 63–77, https://doi.org/10.4204/EPTCS.23.5.
[24] A. Brunel, K. Terui, Church => scott = ptime: an application of resource sensitive realizability, in: Baillot [27], pp. 31–46, https://doi.org/10.4204/EPTCS.23.3.
[25] E. De Benedetti, Linear logic, type assignment systems and implicit computational complexity, Theses, Università degli Studi di Torino – ENS de Lyon, Feb. 2014, http://www.di.unito.it/~debenede/docs/phdthesis.pdf.
[26] M. Sipser, Introduction to the Theory of Computation, PWS Publishing Company, 1997.
[27] P. Baillot (Ed.), Proceedings International Workshop on Developments in Implicit Computational complExity, DICE 2010, Paphos, Cyprus, 27–28th, March 2010, EPTCS, vol. 23, 2010, https://doi.org/10.4204/EPTCS.23.

Please cite this article in press as: P. Baillot et al., Characterizing polynomial and exponential complexity classes in elementary lambda-calculus, Inf. Comput. (2018), https://doi.org/10.1016/j.ic.2018.05.005