Characterizing polynomial and exponential complexity classes in elementary lambda-calculus (*abstract*)

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

[1] CNRS, ENS de Lyon, Inria, UCBL, Université de Lyon. Laboratoire LIP.
[2] Dipartimento di Informatica, Università degli Studi di Torino

1 Introduction

This paper fits in the research line of *implicit computational complexity* (ICC), whose goal is to characterize complexity classes without relying on explicit bounds on resources but instead by considering restrictions on programming languages and calculi. One can distinguish in this body of work between some *monovalent* characterisations, which provide a language for *one* given complexity class, e.g. PTIME in [4], and *polyvalent* characterisations in which the language allows to capture several complexity classes [5, 10].

Denote by FDTIME($F(n)$) and by DTIME($F(n)$) respectively the class of functions and the class of predicates on binary words computable on a deterministic Turing machine in time $O(F(n))$; in the following we will refer to complexity classes k-EXP $= \cup_{i\in\mathbb{N}}$DTIME($2_k^{n^i}$) and k-FEXP $= \cup_{i\in\mathbb{N}}$FDTIME($2_k^{n^i}$), for $k \geq 0$.

A particularly fruitful technical tool in this setting is linear logic together with its light versions, that exploit the isomorphism between proofs and programs. This logic indeed provides a powerful system to analyse the duplication and sharing of arguments in typed λ-calculus, and hence in functional languages, thanks to the introduction of a specific logical connective for the duplication of arguments, the ! modality. As in λ-calculus the reuse of an argument can cause a complexity explosion, the idea here is to use weak versions of ! to characterize complexity classes. This intuition is illustrated by elementary linear logic (ELL) [4, 3], a simple variant of linear logic which provides a monovalent characterisation of elementary complexity, that is to say computation in time bounded by a tower of exponentials of fixed height. Other variants of linear logic provide characterizations of PTIME, but they either use a more complicated language, like light linear logic (LLL) [4], or a more specific programming discipline, like soft linear logic (SLL) [6].

The ! modality in ELL has the effect of dividing the proofs-programs into strata, and the flow of information between these strata proceeds only in one-way. Let us call this phenomenon !-stratification. It was shown recently in [1] that a simple extension of ELL with type fixpoints and weakening actually provides a polyvalent characterization of the hierarchy of classes k-EXP for $k \geq 0$, by means of a hierarchy of types. However this result was obtained by using *proof-nets*, which are a powerful tool but require some background in linear logic.

The goal of the present work is to recast the methodology of !-stratification in the more standard setting of λ-calculus and to show that the resulting simple language is sufficient to obtain a polyvalent characterization of the hierarchy k-EXP for $k \geq 0$. As a consequence this language also enables, beside the characterization of classes of predicates, the characterization of classes of functions on binary words. Here we will use a λ-calculus close to that of [12], without a `let` construction but with two kinds of λ-binders. We consider a type system, inspired by [2], bringing together the advantage of simplicity and the characterization of sub-elementary classes. We think that this λ-calculus-based approach pinpoints the key mechanisms at work in ELL and thus could serve as a basis for future characterizations of other complexity classes.

Related works. The first results on ELL [4, 3] had been carried out in the setting of proof-nets, and later works further analyzed computation in this setting [13, 7]. Terui [15] introduced a term calculus for LLL in order to study its complexity properties. Madet and Amadio [11, 12] took inspiration from it to design an elementary λ-calculus, typed in ELL. In [2] a type system derived from ELL is designed for standard λ-calculus. [1] gave an extension of ELL to characterize subclasses inside elementary time; see also [9] for a related investigation in an untyped setting. Here we will use a λ-calculus with a syntax similar to e.g. [14, 8] and a well-formedness discipline following [12] in the style of elementary linear logic. We consider a type system, which is a slight variant of that of [2], bringing together the advantage of simplicity and the characterization of sub-elementary classes.

2 The $\lambda^!$-calculus

We will use a calculus, $\lambda^!$-calculus, adding to ordinary λ-calculus a !-modality:

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

where $\mathtt{x}$ ranges over a countable set of term variables $\mathtt{Var}$; here the usual notions of free variables and substitution are extended respectively with $\mathrm{FV}(!\mathtt{M}) = \mathrm{FV}(\mathtt{M})$ and $(!\mathtt{M})[\mathtt{N}/\mathtt{x}] = !(\mathtt{M}[\mathtt{N}/\mathtt{x}])$.

Depth. We introduce a notion coming from proof-nets and crucial for the study of computation in $\lambda^!$-calculus: the *depth of an occurrence* $\mathtt{N}$ in a term $\mathtt{M}$, denoted $\delta(\mathtt{N}, \mathtt{M})$, is the number of ! enclosing the occurrence $\mathtt{N}$ in $\mathtt{M}$. Moreover, the *depth* $\delta(\mathtt{M})$ of a term $\mathtt{M}$ is the maximal nesting of ! in $\mathtt{M}$.

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

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

The notation $\overset{*}{\rightarrow}$ denotes the reflexive and transitive closure of $\rightarrow$.

Observe that a term of the form $(\lambda^!\mathtt{x}.\mathtt{M})\mathtt{P}$ is a redex only if $\mathtt{P}$ is of the form $\mathtt{P} = !\mathtt{N}$. These two kinds of redexes suggest the following intuition: the abstraction λ expects an input at depth 0 while $\lambda^!$ expects an input at depth 1.

A *redex at depth* i in $\mathtt{M}$ is an occurrence $\mathtt{N}$ in $\mathtt{M}$ which is a β or ! redex, and such that $\delta(\mathtt{N}, \mathtt{M}) = i$. We say that a term is in i*-normal form* if it does not have any redex at depth inferior or equal to i. Thus if $d = \delta(\mathtt{M})$, then $\mathtt{M}$ is in normal form iff it is in d-normal form. We denote as $\mathbf{nf}_i$ the set of terms in i-normal form. Moreover we denote by $\rightarrow_i$ a reduction where the reduced redex is at depth i. It is important to notice that the reduction of a redex at depth j cannot introduce a redex at a strictly inferior depth:

Proposition 1. *Let* $\mathtt{M} \in \mathbf{nf}_i$ *and* $\mathtt{M} \rightarrow_j \mathtt{M}'$*, where* $j \geq i + 1$ *then* $\mathtt{M}' \in \mathbf{nf}_i$*.*

We will consider a subclass of $\lambda^!$-terms, inspired by $\mathtt{ELL}$ [4, 11]:

Definition 1 (Well-formed term). *A term* $\mathtt{M}$ *is well-formed (w.f.) if and only if*

1. *in a subterm* $\lambda\mathtt{x}.\mathtt{N}$*,* $\mathtt{x}$ *occurs at most once and at depth* 0 *in* $\mathtt{N}$*,*
2. *in a subterm* $\lambda^!\mathtt{x}.\mathtt{N}$*,* $\mathtt{x}$ *occurs any number of times and at depth* 1 *in* $\mathtt{N}$*.*

From now on, we will consider only well-formed terms. Observe that the depth of subterms during reduction does not change, and moreover the class of w.f. terms is preserved by reduction.

As a consequence of the definition of w.f. terms we also get that their depth does not increase during reduction:

Proposition 2. $\mathtt{M} \rightarrow \mathtt{M}'$ *implies* $\delta(\mathtt{M}') \leq \delta(\mathtt{M})$*.*

We define the *size of* $\mathtt{M}$ *at depth* i, denoted by $|\mathtt{M}|_i$, by:

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

We examine what happens to the size of a term during reduction:

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

The fact that by Prop. 1 reducing a redex does not create any redex at strictly lower depth suggests considering a *level-by-level reduction strategy*: given a term $\mathtt{M}$, if $\mathtt{M} \in \mathbf{nf}_i$ and $\mathtt{M} \notin \mathbf{nf}_{i+1}$, reduce (non-deterministically) a redex at depth $i+1$.

Proposition 3. *Any reduction of a term* $\mathtt{M}$ *by the level-by-level strategy terminates.*

Representation of functions. In order to represent functions by terms, we first need to encode data. For booleans we can use the familiar encoding by $\mathtt{true} = \lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{x}$ and $\mathtt{false} = \lambda \mathtt{x}.\lambda \mathtt{y}.\mathtt{y}$, while for integers we use the following encoding, respectively for Church integers and binary words:

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

Observe that the terms encoding booleans are of depth 0, while those representing Church integers and binary words are of depth 1. We denote the length of a word $w \in \{0,1\}^\star$ by $\mathbf{length}(w)$.

We will represent computation on a binary word by considering applications of the form $\mathtt{P}!\underline{w}$; since we want to allow the possibility to compute at arbitrary depth, the output must be of the form $!^k\mathtt{D}$, where $k \in \mathbb{N}$ and $\mathtt{D}$ is one of the data representations above

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

Complexity of reduction. We study the complexity of the reduction of terms $\mathtt{P}!\underline{w}$ by the level-by-level strategy; it is useful to analyze the complexity of the reduction of such terms to their k-normal form, for $k \geq 2$, as follows:[3]

Proposition 4. *Given a program* $\mathtt{P}$*, for any* $k \geq 2$*, there exists a polynomial* q *such that for any* $w \in \{0,1\}^\star$*, the term* $\mathtt{P}!\underline{w}$ *reduces by the level-by-level strategy in at most* $2^{q(n)}_{k-2}$ *steps to a* $(k-1)$*-normal form* $\mathtt{M}^1_k$*, with* $|\mathtt{M}^1_k| \leq 2^{q(n)}_{k-2}$*, where* $n = \mathbf{length}(w)$*. In particular, when* $k = 2$ *we have a polynomial bound* $q(n)$*.*

3 Type system

Since we are interested in characterizing predicates and functions, we want the result of the application of a program to a binary word to have the shape of either a boolean or a word. To this aim we now introduce a type assignment system for $\lambda^!$-calculus, based on ELL, such that any well-typed term in the system is also well-formed and all previous results are preserved in the typed setting.

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

$$\begin{array}{lr} \mathtt{A}, \mathtt{B} ::= \mathtt{a} \mid \sigma \multimap \sigma \mid \forall \mathtt{a}.\mathtt{A} \mid \mu \mathtt{a}.\mathtt{A} & \text{(linear types)} \\ \sigma, \tau ::= \mathtt{A} \mid !\sigma & \text{(types)} \end{array}$$

where $\mathtt{a}$ ranges over a countable set of type variables.

A *basis* is a partial function from variables to types, with finite domain; following the work of [2], we consider three different bases $\Gamma \mid \Delta \mid \Theta$, being respectively the linear, modal and parking basis. While the meaning of the linear and modal bases is intuitive, the parking basis has the purpose of holding variables, (possibly) occurring more than once in the term, which eventually will be moved to the modal basis; indeed, observe that there is no abstraction rule for variables in the parking basis.

The typing system proves statements of the shape $\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \sigma$, and derivations are denoted by Π, Σ. The rules are given in Table 1; observe that rule (!) moves variables from the parking to the modal basis, and

[3] We define the notation 2^n_i in the following way: $2^x_0 = x$ and $2^x_{i+1} = 2^{2^x_i}$.

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

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

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

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

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

Table 1. Derivation rules.

moreover it allows to add any suitable choice of bases by weakening. Here $\Gamma_1 \# \Gamma_2$ iff $\mathrm{dom}(\Gamma_1) \cap \mathrm{dom}(\Gamma_2) = \emptyset$, and $\Delta_1 \subseteq \Delta_2$ iff $\mathtt{x} : \sigma \in \Delta_1$ implies $\mathtt{x} : \sigma \in \Delta_2$.

We say that a term M is *well-typed* (w.t.) iff there is a derivation $\Pi \triangleright \Gamma \mid \Delta \mid \emptyset \vdash \mathtt{M} : \sigma$ for some Γ, Δ, σ, since parking variables will eventually become modal variables. When all three basis are empty we denote the derivation by $\Pi \triangleright \vdash \mathtt{M} : \sigma$.

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

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

Observe that, given $\Gamma \mid \Delta \mid \Theta \vdash \mathtt{M} : \mathtt{A}$, we have:

– if $\mathtt{x} \in \mathrm{dom}(\Gamma) \cup \mathrm{dom}(\Theta)$, then each occurrence of x in M is at depth 0 in M,
– if $\mathtt{x} \in \mathrm{dom}(\Delta)$, then each occurrence of x in M is at depth 1 in M.

This property is reminiscent of the definition of w.f. terms, and indeed it is the key to prove:

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

Therefore, Prop. 2 holds both in the untyped and in the typed settings and it ensures that, during reduction of a typed term, the depth does not increase.

We define the following types (adapted from system F):

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

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

representing booleans (B), Church tally integers (N) and Church binary words (W).

Complexity soundness. We are interested in giving a precise account of the hierarchy of classes characterized by this typed $\lambda^!$-calculus; in particular, we prove that a closed term of type $!\mathrm{W} \multimap !^{k+2}\mathrm{B}$ (resp. $!\mathrm{W} \multimap !^{k+2}\mathrm{W}_S$) represents a predicate (resp. a function) which, when applied to a word of length n, can be evaluated with a level-by-level strategy in a number of steps in $O(2_k^{p(n)})$ for some polynomial p:

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

The result can be easily extended to functions, while changing the output data types. Scott binary words are defined inductively as:

$$\widehat{\epsilon} \overset{\text{def}}{=} \lambda \mathtt{x}.\lambda \mathtt{y}.\lambda \mathtt{z}.\mathtt{z} \qquad \widehat{0w} \overset{\text{def}}{=} \lambda \mathtt{x}.\lambda \mathtt{y}.\lambda \mathtt{z}.\mathtt{x}\widehat{w} \qquad \widehat{1w} \overset{\text{def}}{=} \lambda \mathtt{x}.\lambda \mathtt{y}.\lambda \mathtt{z}.\mathtt{y}\widehat{w}$$

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

Theorem 3. *Let $\vdash \mathtt{P} : !\mathrm{W} \multimap !^{k+2}\mathrm{W}_S$ where $\mathtt{P}$ is a program, and let $\vdash \underline{w} : \mathrm{W}$ where $\underline{w}$ is a Church binary word such that* **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.*

Completeness. We proved that all programs of type $!\mathrm{W} \multimap !^{k+2}\mathrm{B}$ (resp. $!\mathrm{W} \multimap !^{k+2}\mathrm{W}_S$) fall into the k-EXP (resp. k-FEXP) class; we want to strengthen this result by proving the converse inclusion. By simulating time-bounded Turing machines, we obtain the following result:

Theorem 4 (Extensional completeness). *Let f be a binary predicate in k-EXP, for $k \geq 0$; then there is a term $\mathtt{M}$ representing f such that $\vdash \mathtt{M} : !\mathrm{W} \multimap !^{k+2}\mathrm{B}$.*

Once again, the same result can be easily extended to functions.

Theorem 5 (Extensional 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 $\vdash \mathtt{M} : !\mathrm{W} \multimap !^{k+2}\mathrm{W}_S$.*

References

1. P. Baillot. Elementary linear logic revisited for polynomial time and an exponential time hierarchy. In *Proceedings of APLAS 2011*, LNCS 7078. Springer, 2011.
2. P. Coppola, U. Dal Lago, and S. Ronchi Della Rocca. Light logics and the call-by-value lambda calculus. *Logical Methods in Computer Science*, 4(4), 2008.
3. V. Danos and J.-B. Joinet. Linear logic & elementary time. *Information and Computation*, 183:123–137, 2003.
4. J.-Y. Girard. Light linear logic. *Information and Computation*, 143:175–204, 1998.
5. N. D. Jones. The expressive power of higher-order types or, life without cons. *J. Funct. Program.*, 11(1):5–94, 2001.
6. Y. Lafont. Soft linear logic and polynomial time. *Theoret. Comput. Sci.*, 318(1–2):163–180, 2004.
7. Ugo Dal Lago. Context semantics, linear logic, and computational complexity. *ACM Transactions in Computational Logic*, 10(4), 2009.
8. Ugo Dal Lago, Andrea Masini, and Margherita Zorzi. Quantum implicit computational complexity. *Theor. Comput. Sci.*, 411(2):377–409, 2010.
9. O. Laurent. Polynomial time in untyped elementary linear logic. Unpublished note, 2013.
10. D. Leivant. Calibrating computational feasibility by abstraction rank. In *Proceedings LICS'02*, pages 345–353. IEEE Computer Society, 2002.
11. A. Madet and R. M. Amadio. An elementary affine lambda-calculus with multithreading and side effects. In *Proceedings of TLCA 2011*, LNCS. Springer, 2011.
12. Antoine Madet. *Implicit Complexity in Concurrent Lambda-Calculi.* PhD thesis, Université Paris 7, December 2012. http://tel.archives-ouvertes.fr/tel-00794977.
13. Damiano Mazza. Linear logic and polynomial time. *Mathematical Structures in Computer Science*, 16(6):947–988, 2006.
14. Simona Ronchi Della Rocca and Luca Roversi. Lambda calculus and intuitionistic linear logic. *Studia Logica*, 59(3), 1997.
15. Kazushige Terui. Light affine lambda calculus and polynomial time strong normalization. *Arch. Math. Log.*, 46(3-4):253–280, 2007.