Complexity Analysis in Presence of Control Operators and Higher-Order Functions

Ugo Dal Lago, Giulio Pellitta

Complexity Analysis in Presence of Control Operators and Higher-Order Functions*

Ugo Dal Lago and Giulio Pellitta

Università di Bologna & INRIA Sophia Antipolis
{dallago,pellitta}@cs.unibo.it

Abstract. A polarized version of Girard, Scedrov and Scott's Bounded Linear Logic is introduced and its normalization properties studied. Following Laurent [25], the logic naturally gives rise to a type system for the $\lambda\mu$-calculus, whose derivations reveal bounds on the time complexity of the underlying term. This is the first example of a type system for the $\lambda\mu$-calculus guaranteeing time complexity bounds for typable programs.

1 Introduction

Among non-functional properties of programs, bounds on the amount of resources (like computation time and space) programs need when executed are particularly significant. The problem of deriving such bounds is indeed crucial in safety-critical systems, but is undecidable whenever non-trivial programming languages are considered. If the units of measurement become concrete and close to the physical ones, the problem becomes even more complicated and architecture-dependent. A typical example is the one of WCET techniques adopted in real-time systems [29], which not only need to deal with how many machine instructions a program corresponds to, but also with how much time each instruction costs when executed by possibly complex architectures (including caches, pipelining, etc.), a task which is becoming even harder with the current trend towards multicore architectures.

A different approach consists in analysing the *abstract* complexity of programs. As an example, one can take the number of instructions executed by the program as a measure of its execution time. This is of course a less informative metric, which however becomes more accurate if the actual time taken *by each instruction* is kept low. One advantage of this analysis is the independence from the specific hardware platform executing the program at hand: the latter only needs to be analysed once. A variety of *complexity analysis* techniques have been employed in this context, from abstract interpretation [21] to type systems [22] to program logics [10] to interactive theorem proving. Properties of programs written in higher-order functional languages are for various reasons well-suited to be verified by way of type systems. This includes not only safety properties

* An extended version of this paper including more details is available [9].

(e.g. well-typed programs do not go wrong), but more complex ones, including resource bounds [22,5,15,7].

In this paper, we delineate a methodology for complexity analysis of higher-order programs *with control operators*. The latter are constructs which are available in most concrete functional programming languages (including Scheme and OCaml), and allow control to flow in non-standard ways. The technique we introduce takes the form of a type system for de Groote's $\lambda\mu$-calculus [12] derived from Girard, Scedrov and Scott's Bounded Linear Logic [19] (BLL in the following). We prove it to be sound: typable programs can indeed be reduced in a number of steps lesser or equal to a (polynomial) bound which can be read from the underlying type derivation. A similar result can be given when the cost model is the one induced by an abstract machine. To the authors' knowledge, this is the first example of a complexity analysis methodology coping well not only with higher-order functions, but also with control operators.

In the rest of this section, we explain the crucial role Linear Logic has in this work, in the meantime delineating its main features.

1.1 Linear Logic and Complexity Analysis

Linear Logic [16] is one of the most successful tools for characterizing complexity classes in a higher-order setting, through the Curry-Howard correspondence. Subsystems of it can indeed be shown to correspond to the polynomial time computable functions [19,18,23] or the logarithmic space computable functions [28]. Many of the introduced fragments can then be turned into type systems for the λ-calculus [5,15], some of them being relatively complete in an intensional sense [7].

The reason for this success lies in the way Linear Logic decomposes intuitionistic implication into linear implication, which has low complexity, and an *exponential modality*, which marks those formulas to which structural rules can be applied. This gives a proper status to proof duplication, without which cut-elimination can be performed in a linear number of steps. By tuning the rules governing the exponential modality, then, one can define logical systems for which cut-elimination can be performed within appropriate resource bounds. Usually, this is coupled with an encoding of all functions in a complexity class $\mathcal{C}$ into the system at hand, which makes the system a *characterization* of $\mathcal{C}$.

Rules governing the exponential modality can be constrained in (at least) two different ways:

- On the one hand, one or more of the rules governing the modality (e.g., dereliction or digging) can be *dropped* or *restricted*. This is what happens, for example, in Light Linear Logic [18] or Soft Linear Logic [23].
- On the other, the logic can be further refined and *enriched* so as to control the number of times structural rules are applied. In other words, rules for the modality are still all there, but in a refined form. This is what happens in Bounded Linear Logic [19]. Similarly, one could control so-called modal impredicativity by a system of levels [4].

The first approach corresponds to cutting the space of proofs with an axe: many proofs, and among them many corresponding to efficient algorithms, will not be part of the system because they require one of the forbidden logical principles. The second approach is milder in terms of the class of good programs that are "left behind": there is strong evidence that with this approach one can obtain a quite expressive logical system [8,7].

Not much is known about whether this approach scales to languages in which not only functions but also first-class continuations and control operators are present. Understanding the impact of these features to the complexity of programs is an interesting research topic, which however has received little attention in the past.

1.2 Linear Logic and Control Operators

On the other hand, more than twenty years have passed since Classical Logic has been shown to be amenable to the Curry-Howard paradigm [20]. And, interestingly enough, classical axioms (e.g. Pierce's law or the law of the Excluded Middle) can be seen as the type of control operators like Scheme's `callcc`. In the meantime, the various facets of this new form of proofs-as-programs correspondence have been investigated in detail, and many extensions of the λ-calculus for which Classical Logic naturally provides a type discipline have been introduced (e.g. [27,6]).

Moreover, the decomposition provided by Linear Logic is known to scale up to Classical Logic [17]. Actually, Linear Logic was known to admit an involutive notion of negation from its very inception [16]. A satisfying embedding of Classical Logic into Linear Logic, however, requires restricting the latter by way of polarities [24]: this way one is left with a logical system with most of the desirable dynamical properties.

In this paper, we define BLLP, a polarized version of Bounded Linear Logic. The kind of enrichment resource polynomials provide in BLL is shown to cope well with polarization. Following the close relationship between Polarized Linear Logic and the $\lambda\mu$-calculus [25], BLLP gives rise to a type system for the $\lambda\mu$-calculus. Proofs and typable $\lambda\mu$-terms are both shown to be reducible to their cut-free or normal forms in a number of steps bounded by a polynomial weight. Such a result for the former translates to a similar result for the latter, since any reduction step in $\lambda\mu$-terms corresponds to one or more reduction steps in proofs. The analysis is then extended to the reduction of $\lambda\mu$-terms by a Krivine-style abstract machine [13].

2 Bounded Polarized Linear Logic as A Sequent Calculus

In this section, we define BLLP as a sequent calculus. Although this section is self-contained, some familiarity with both Bounded [19] and Polarized [25] Linear Logic would certainly help. Some more details can be found in an extended version of the present paper [9].

2.1 Polynomials and Formulas

A *resource monomial* is any (finite) product of binomial coefficients in the form $\prod_{i=1}^{m} \binom{x_i}{n_i}$, where the x_i are distinct variables and the n_i are non-negative integers. A *resource polynomial* is any finite sum of resource monomials. Given resource polynomials p, q we write $p \sqsubseteq q$ to denote that $q - p$ is a resource polynomial. If $p \sqsubseteq r$ and $q \sqsubseteq s$ then also $q \circ p \sqsubseteq s \circ r$. Resource polynomials are closed by addition, multiplication, bounded sums and composition [19].

A *polarized formula* is a formula (either positive or negative) generated by the following grammar

$$P ::= V \;\Big|\; P \otimes P \;\Big|\; 1 \;\Big|\; !_{x<p} N;$$
$$N ::= V^{\perp} \;\Big|\; N \parr N \;\Big|\; \bot \;\Big|\; ?_{x<p} P;$$

where V ranges over a countable sets of atoms. Throughout this paper, formulas (but also terms, contexts, etc.) are considered modulo α-equivalence. Formulas (either positive or negative) are ranged over by metavariables like A, B. Formulas like $V^{\perp}$ are sometimes denoted as X, Y.

In a polarized setting, contraction can be performed on any negative formula. As a consequence, we need the notion of a *labelled formula* $[A]^p_x$, namely the *labelling* of the formula A with respect to x and p. The labelled formula $[N]^p_x$ (resp. $[P]^p_x$) can be thought of roughly as $?_{x<p} N^{\perp}$ (resp. $!_{x<p} P^{\perp}$), i.e., in a sense we can think of labelled formulas as formulas hiding an implicit exponential modality. All occurrences of x in A are bound in $[A]^p_x$. Metavariables for labellings of positive (respectively, negative) formulas are $\mathbf{P}, \mathbf{Q}, \mathbf{R}$ (respectively, $\mathbf{N}, \mathbf{M}, \mathbf{L}$). Labelled formulas are sometimes denoted with metavariables $\mathbf{A}, \mathbf{B}$ when their polarity is not essential. Negation, as usual in classical linear systems, can be applied to any (possibly labelled) formula, *à la* De Morgan. When the resource variable x does not appear in A, then we do not need to mention it when writing $[A]^p_x$, which becomes $[A]^p$. Similarly for $!_{x<p} N$ and $?_{x<p} P$.

Both the space of formulas and the space of labelled formulas can be seen as partial orders by stipulating that two (labelled) formulas can be compared iff they have *exactly* the same skeleton and the polynomials occurring in them can be compared. As an example,

$$!_{x<p} N \sqsubseteq !_{x<q} M \text{ iff } q \sqsubseteq p \wedge N \sqsubseteq M;$$
$$?_{x<p} P \sqsubseteq ?_{x<q} Q \text{ iff } p \sqsubseteq q \wedge P \sqsubseteq Q.$$

In a sense, then, polynomials occurring next to atoms or to the *whynot* operator are in positive position, while those occurring next to the *bang* operator are in negative position. In all the other cases, $\sqsubseteq$ is defined component-wise, in the natural way, e.g. $P \otimes Q \sqsubseteq R \otimes S$ iff both $P \sqsubseteq R$ and $Q \sqsubseteq S$. Finally $[N]^p_x \sqsubseteq [M]^q_x$ iff $N \sqsubseteq M \wedge p \sqsupseteq q$. And dually, $[P]^p_x \sqsubseteq [Q]^q_x$ iff $N \sqsubseteq M \wedge p \sqsubseteq q$.

Certain operators on resource polynomials can be lifted to formulas. As an example, we want to be able to *sum* labelled formulas provided they have a

proper form:

$$[N]_x^p \uplus [N\{x/y+p\}]_y^q = [N]_x^{p+q}.$$

We are assuming, of course, that x, y are not free in either p or q. This construction can be generalized to *bounded* sums: suppose that a labelled formula is in the form

$$[M]_y^r = [N\{x/y + \sum_{u<z} r\{z/u\}\}]_y^r,$$

where y and u are not free in N nor in r and z is not free in N. Then the labelled formula $\sum_{z<q}[M]_y^r$ is defined as $[N]_x^{\sum_{z<q} r}$. See [19, §3.3] for more details about the above constructions.

2.2 Sequents and Rules

The easiest way to present BLLP is to give a sequent calculus for it. Actually, proofs will be structurally identical to proofs of Laurent's LLP. Of course, only *some* of LLP proofs are legal BLLP proofs — those giving rise to an exponential blow-up cannot be decorated according to the principles of Bounded Linear Logic.

A *sequent* is an expression in the form $\vdash \Gamma$, where $\Gamma = \mathbf{A}_1, \ldots \mathbf{A}_n$ is a multiset of labelled formulas such that at most one among $\mathbf{A}_1, \ldots, \mathbf{A}_n$ is positive. If Γ only contains (labellings of) negative formulas, we indicate it with metavariables like $\mathcal{N}, \mathcal{M}$. The operator $\uplus$ can be extended to one on multisets of formulas component-wise, so we can write expressions like $\mathcal{N} \uplus \mathcal{M}$: this amounts to sum the polynomials occurring in $\mathcal{N}$ and those occurring in $\mathcal{M}$. Similarly for bounded sums.

The rules of the sequent calculus for BLLP are in Figure 1. Please observe

$$\frac{\mathbf{N} \sqsubseteq \mathbf{M} \qquad \mathbf{M}^\perp \sqsupseteq \mathbf{P}}{\vdash \mathbf{N}, \mathbf{P}}\ \mathsf{Ax} \qquad \frac{\vdash \Gamma, \mathbf{N} \qquad \vdash \mathcal{N}, \mathbf{N}^\perp}{\vdash \Gamma, \mathcal{N}}\ \mathsf{Cut}$$

$$\frac{\vdash \Gamma, [N]_x^p, [M]_x^q \qquad p \sqsubseteq r \quad q \sqsubseteq r}{\vdash \Gamma, [N \parr M]_x^r}\ \parr \qquad \frac{\vdash \mathcal{N}, [P]_x^p \qquad \vdash \mathcal{M}, [Q]_x^q \quad r \sqsubseteq p \quad r \sqsubseteq q}{\vdash \mathcal{N}, \mathcal{M}, [P \otimes Q]_x^r}\ \otimes$$

$$\frac{\vdash \mathcal{N}, [N]_x^p \qquad \mathcal{M} \sqsubseteq \sum_{y<q} \mathcal{N}}{\vdash \mathcal{M}, [!_{x<p} N]_y^q}\ ! \qquad \frac{\vdash \mathcal{N}, [P\{y/0\}]_x^{p\{y/0\}} \qquad \mathbf{N} \sqsubseteq [?_{x<p} P]_y^1}{\vdash \mathcal{N}, \mathbf{N}}\ ?d$$

$$\frac{\vdash \Gamma}{\vdash \Gamma, \mathbf{N}}\ ?w \qquad \frac{\vdash \Gamma, \mathbf{N}, \mathbf{M} \qquad \mathbf{L} \sqsubseteq \mathbf{N} \uplus \mathbf{M}}{\vdash \Gamma, \mathbf{L}}\ ?c \qquad \frac{\vdash \Gamma}{\vdash \Gamma, [\bot]_x^p}\ \bot \qquad \frac{}{\vdash [1]_x^p}\ 1$$

Fig. 1. BLLP, Sequent Calculus Rules

that:

- The relation $\sqsubseteq$ is implicitly applied to both formulas and polynomials whenever possible in such a way that "smaller" formulas can always be derived (see Section 2.3).

- As in LLP, structural rules can act on any negative formula, and not only on exponential ones. Since all formulas occurring in sequents are labelled, however, we can still keep track of how many times formulas are "used", in the spirit of BLL.
- A byproduct of taking sequents as multisets of *labeled* formulas is that multiplicative rules themselves need to deal with labels. As an example, consider rule $\otimes$: the resource polynomial labelling the conclusion $P \otimes Q$ is anything smaller or equal to the polynomials labeling the two premises.

The sequent calculus we have just introduced could be extended with second-order quantifiers and additive logical connectives. For the sake of simplicity, however, we have kept the language of formulas very simple here. The interested reader can check [24] for a treatment of these connectives in a polarized setting or [9] for more details.

As already mentioned, BLLP proofs can be seen as obtained by decorating proofs from Laurent's LLP [25] with resource polynomials. Given a proof π, $\langle \pi \rangle$ is the LLP proof obtained by erasing all resource polynomials occurring in π. If π and ρ are two BLLP proofs, we write $\pi \sim \rho$ iff $\langle \pi \rangle = \langle \rho \rangle$, i.e., iff π and ρ are two decorations of the same LLP proof.

Even if structural rules can be applied to all negative formulas, only certain proofs will be copied or erased along the cut-elimination process, as we will soon realize. A *box* is any proof which ends with an occurrence of the ! rule. In non-polarized systems, only boxes can be copied or erased, while here the process can be applied to $\otimes$*-trees*, which are proofs inductively defined as follows:

- Either the last rule in the proof is Ax or ! or 1;
- or the proof is obtained from two $\otimes$-trees by applying the rule $\otimes$.

A $\otimes$-tree is said to be *closed* if it does not contain any axiom nor any box having auxiliary doors (i.e., no formula in the context of the ! rules).

2.3 Malleability

The main reason for the strong (intensional) expressive power of BLL [8] is its *malleability*: the conclusion of any proof π can be modified in many different ways without altering its structure. Malleability is not only crucial to make the system expressive, but also to prove that BLLP enjoys cut-elimination. In this section, we give four different ways of modifying a sequent in such a way as to preserve its derivability. Two of them are anyway expected and also hold in BLL, while the other two only make sense in a polarized setting.

First of all, taking smaller formulas (i.e., more general — cf. [19, §3.3, p. 21]) preserves derivability:

Lemma 1 (Subtyping). *If* $\pi \triangleright \vdash \Gamma, \mathbf{A}$ *and* $\mathbf{A} \sqsupseteq \mathbf{B}$*, then there is* $\rho \triangleright \vdash \Gamma, \mathbf{B}$ *such that* $\pi \sim \rho$*.*

Substituting resource variables for polynomials itself preserves typability:

Lemma 2 (Substitution). *Let* $\pi \triangleright \vdash \Gamma$*. Then there is a proof* $\pi\{x/p\}$ *of* $\vdash \Gamma\{x/p\}$*. Moreover,* $\pi\{x/p\} \sim \pi$*.*

Both Lemma 1 and Lemma 2 can be proved by easy inductions on the structure of π.

As we have already mentioned, one of the key differences between Linear Logic and its polarized version is that in the latter, arbitrary proofs can potentially be duplicated (and erased) along the cut-elimination process, while in the former only special ones, namely boxes, can. This is, again, a consequence of the fundamentally different nature of structural rules in the two systems. Since BLLP is a refinement of LLP, this means that the same phenomenon is expected. But beware: in a bounded setting, contraction is not symmetric, i.e., the two copies of the proof π we are duplicating are not identical to π.

What we need to prove, then, is that proofs can indeed be *split* in BLLP:

Lemma 3 (Splitting). *If $\pi \triangleright \vdash \mathcal{N}, [P]^p_x$ is a $\otimes$-tree and $p \sqsupseteq r + s$ then there exist $\mathcal{M}, \mathcal{O}$ such that $\rho \triangleright \vdash \mathcal{M}, [P]^r_x$, $\sigma \triangleright \vdash \mathcal{O}, [P\{x/y + r\}]^s_y$. Moreover, $\mathcal{N} \sqsubseteq \mathcal{M} \uplus \mathcal{O}$ and $\rho \sim \pi \sim \sigma$.*

Observe that not every proof can be split, but only $\otimes$-trees can. The proof of Lemma 3 is not trivial and requires some auxiliary results (see [9] for more details). A parametric version of splitting is also necessary here:

Lemma 4 (Parametric Splitting). *If $\pi \triangleright \vdash \mathcal{N}, [P]^p_x$, where π is a $\otimes$-tree and $p \sqsupseteq \sum_{x<r} s$, then there exists $\rho \triangleright \vdash \mathcal{M}, [P]^s_x$ where $\sum_{x<r} \mathcal{M} \sqsupseteq \mathcal{N}$ and $\rho \sim \pi$.*

While splitting allows to cope with duplication, parametric splitting implies that an arbitrary $\otimes$-tree can be modified so as to be lifted into a box through one of its auxiliary doors. Please observe that p^π continues to be such an upper bound even if any natural number is substituted for any of its free variables, an easy consequence of Lemma 2. The following is useful when dealing with cuts involving the rule $?d$:

Lemma 5. *If $q \sqsupseteq 1$, then $\sum_{z<q} [M]^r_y \sqsubseteq [M]^r_y\{z/0\}$.*

3 Cut Elimination

In this section, we give some ideas about how cuts can be eliminated from BLLP proofs.

Logical cuts (i.e., those in which the two immediate subproofs end with a rule introducing the formula involved in the cut) can be reduced as in LLP [24], but exploiting malleability whenever polynomials need to be modified. This defines the reduction relation $\longmapsto$ (see [9] for more details). All instances of the Cut rule which are not logical are said to be *commutative*, and induce an equivalent relation $\cong$ on proofs. In general, not all cuts in a proof are logical, but any cut can be turned into a logical one:

Lemma 6. *Let π be any proof containing an occurrence of the rule Cut. Then, there are two proofs ρ and σ such that $\pi \cong \rho \longmapsto \sigma$, where ρ can be effectively obtained from π.*

The proof of Lemma 6 goes as follows: given any instance of the Cut rule

$$\frac{\pi \triangleright \vdash \Gamma, [N]^p_x \qquad \rho \triangleright \vdash \mathcal{N}, [P]^p_x}{\vdash \Gamma, \mathcal{N}}\ \mathsf{Cut}$$

consider the path (i.e., the sequence of formula occurrences) starting from $[N]^p_x$ and going upward inside π, and the path starting from $[P]^p_x$ and going upward inside ρ. Both paths end either at an Ax rule or at an instance of a rule introducing the main connective in N or P. The game to play is then to show that these two paths can always be *shortened* by way of commutations, thus exposing the underlying logical cut.

Lemma 6 is implicitly defining a cut-elimination procedure: given any instance of the Cut rule, turn it into a logical cut by the procedure from Lemma 6, then fire it. This way we are implicitly defining another reduction relation $\longrightarrow$. The next question is the following: is this procedure going to terminate for every proof π (i.e., is $\longrightarrow$ strongly, or weakly, normalizing)? How many steps does it take to turn π to its cut-free form?

Actually, $\longrightarrow$ produces reduction sequences of very long length, but is anyway strongly normalizing. A relatively easy way to prove it goes as follows: any BLLP proof π corresponds to a LLP sequent calculus proof $\langle\pi\rangle$, and the latter itself corresponds to a polarized proof net $\langle\!\langle\pi\rangle\!\rangle$ [25]. Moreover, $\pi \longrightarrow \rho$ implies that $\langle\!\langle\pi\rangle\!\rangle \mapsto \langle\!\langle\rho\rangle\!\rangle$, where $\mapsto$ is the canonical cut-elimination relation on polarized proof-nets. Finally, $\langle\!\langle\pi\rangle\!\rangle$ is identical to $\langle\!\langle\rho\rangle\!\rangle$ whenever $\pi \cong \rho$. Since $\mapsto$ is known to be strongly normalizing, $\longrightarrow$ does not admit infinite reduction sequences:

Proposition 1 (Cut-Elimination). *The relation $\longrightarrow$ is strongly normalizing.*

This does not mean that cut-elimination can be performed in (reasonably) bounded time. Already in BLL this can take exponential time: the whole of Elementary Linear Logic [18] can be embedded into it.

3.1 Soundness

To get a soundness result, then, we somehow need to restrict the underlying reduction relation $\longrightarrow$. Following [19], one could indeed define a subset of $\longrightarrow$ just by imposing that in dereliction, contraction, or box cut-elimination steps, the involved $\otimes$-trees are closed. Moreover, we could stipulate that reduction is external, i.e., it cannot take place inside boxes. Closed and external reduction, however, is not enough to simulate head-reduction in the $\lambda\mu$-calculus, and not being able to reduce under the scope of μ-abstractions does not make much sense anyway. We are forced, then, to consider an extension of closed reduction. The fact that this new notion of reduction still guarantees polynomial bounds is technically a remarkable strengthening with respect to BLL's Soundness Theorem [19].

There is a quite natural notion of *downward* path in proofs: from any occurrence of a negative formula $\mathbf{N}$, just proceed downward until you either find (the main premise of) a Cut rule, or a conclusion. In the first case, the occurrence of $\mathbf{N}$ is said to be *active*, in the second it is said to be *passive*. Proofs can then be endowed with a new notion of reduction: all dereliction, contraction or box

digging cuts can be fired only if the negative formula occurrences in its rightmost argument are all passive. In the literature, this is sometimes called a *special cut* (e.g. [3]). Moreover, reduction needs to be external, as usual. This notion of reduction, as we will see, is enough to mimic head reduction, and is denoted with $\Longrightarrow$.

The next step consists in associating a weight, in the form of a resource polynomial, to every proof, similarly to what happens in BLL. The *pre-weight* $\pi^{\diamond}$ of a proof π with conclusion $\vdash \mathbf{A}_1, \ldots, \mathbf{A}_n$ consists in:

- a resource polynomial p^{π}.
- n disjoints sets of resource variables $S_1^{\pi}, \ldots, S_n^{\pi}$, each corresponding to a formula in $\mathbf{A}_1, \ldots, \mathbf{A}_n$; if this does not cause ambiguity, the set of resource variables corresponding to a formula $\mathbf{A}$ will be denoted by $S^{\pi}(\mathbf{A})$. Similarly for $S^{\pi}(\Gamma)$, where Γ is a multiset of formulas.

If π has pre-weight $p^{\pi}, S_1^{\pi}, \ldots, S_n^{\pi}$, then the *weight* q^{π} of π is simply p^{π} where, however, all the variables in $S_1^{\pi}, \ldots, S_n^{\pi}$ are substituted with 0: $p^{\pi}\{\cup_{i=1}^{n} S_i^{\pi}/0\}$. The pre-weight of a proof π is defined by induction on the structure of π (see [9] for more details). The idea is that every occurrence of negative formulas is attributed a fresh variable, which later is instantiated with either 0 (if the formula is passive) or 1 (if it is active). This allows to discriminate between the case in which rules can "produce" time complexity along the cut-elimination, and the case in which they do not. Ultimately, this leads to:

Lemma 7. *If $\pi \cong \rho$, then $q^{\pi} = q^{\rho}$. If $\pi \Longrightarrow \rho$, then $q^{\pi} \sqsupset q^{\rho}$.*

The main idea behind Lemma 7 is that even if the logical cut we perform when going from π to ρ is "dangerous" (e.g. a contraction) *and* the involved $\otimes$-tree is not closed, the residual negative rules have null weight, because they are passive.

We can conclude that:

Theorem 1 (Polystep Soundness). *For every proof π, if $\pi \Longrightarrow^{n} \rho$, then $n \leq q^{\pi}$.*

In a sense, then, the weight of any proof π is a resource polynomial which can be easily computed from π (rules in [9] are anyway inductively defined), but which is also an upper bound on the number of logical cut-elimination steps separating π from its normal form. Please observe that q^{π} continues to be such an upper bound even if any natural number is substituted for any of its free variables, an easy consequence of Lemma 2.

Why then, are we talking about *polynomial* bounds? In BLL, and as a consequence also in BLLP, one can write programs in such a way that the size of the input is reflected by a resource variable occurring in its type. As an example, the type of (Church encodings of) binary strings of length at most x could be the following in BLLP:

$$(X \multimap^{1} X) \multimap^{x} (X \multimap^{1} X) \multimap^{x} (X \multimap^{1} X)$$

(where $N \multimap^{p} M$ stands for $?_p N^{\perp} \parr M$). The weight, then, turns out to be a tool to study the behavior of terms seen as functions taking arguments of varying length. A more in-depth discussion about these issues is outside the scope of this paper. Please refer to [19].

4 A Type System for the $\lambda\mu$-Calculus

We describe here a version of the $\lambda\mu$-calculus as introduced by de Groote [11]. Terms are as follows

$$t, u \ ::= \ x \ \Big| \ \lambda x.t \ \Big| \ \mu\alpha.t \ \Big| \ [\alpha]t \ \Big| \ (t)t,$$

where x and α range over two infinite disjoint sets of variables (called λ-variables and μ-variables, respectively). In contrast with the $\lambda\mu$-calculus as originally formulated by Parigot [27], μ-abstraction is not restricted to terms of the form $[\alpha]t$ here.

4.1 Notions of Reduction

The reduction rules we consider are the following ones:

$$(\lambda x.t)u \to_\beta t[{}^u/_x]; \qquad (\mu\alpha.t)u \to_\mu \mu\alpha.t[{}^{[\alpha](v)u}/_{[\alpha]v}]; \qquad \mu\alpha.[\alpha]t \to_\theta t;$$

where, as usual, $\to_\theta$ can be fired only if $\alpha \notin FV(t)$. In the following, $\to$ is just $\to_{\beta\mu\theta}$. In so-called *weak reduction*, denoted $\to_{\mathsf{w}}$, reduction simply cannot take place in the scope of binders, while *head reduction*, denoted $\to_{\mathsf{h}}$, is a generalization of the same concept from pure λ-calculus [13]. Details are in Figure 2. Please

$$\frac{t \to u}{t \to_{\mathsf{w}} u} \qquad \frac{t \to_{\mathsf{w}} u}{tv \to_{\mathsf{w}} uv} \qquad \frac{t \to_{\mathsf{w}} u}{[\alpha]t \to_{\mathsf{w}} [\alpha]u}$$

$$\frac{t \to_{\mathsf{w}} u}{t \to_{\mathsf{h}} u} \qquad \frac{t \to_{\mathsf{h}} u}{\lambda x.t \to_{\mathsf{h}} \lambda x.u} \qquad \frac{t \to_{\mathsf{h}} u}{\mu\alpha.t \to_{\mathsf{h}} \mu\alpha.u}$$

Fig. 2. Weak and Head Notions of Reduction

notice how in head reduction, redexes can indeed be fired even if they lie in the scope of λ-or-μ-abstractions, which, however, cannot themselves be involved in a redex. This harmless restriction, which corresponds to taking the *outermost* reduction order, is needed for technical reasons that will become apparent soon.

4.2 The Type System

Following Laurent [25], types are just negative formulas. Not all of them can be used as types, however: in particular, $N \parr M$ is a legal type only if N is in the form $?_{x<p}O^\perp$, and we use the following abbreviation in this case: $N \multimap^p_x M = (?_{x<p}N^\perp) \parr M$. In particular, if M is $\perp$ then $N \multimap^p_x M$ can be abbreviated as $\neg^p_x N$. *Typing formulas* are negative formulas which are either $\perp$, or X, or in the form $N \multimap^p_x M$ (where N and M are typing formulas themselves). A *modal formula* is one in the form $?_{x<p}N^\perp$ (where N is a typing formula). Please

observe that all the constructions from Section 2.1 (including labellings, sums, etc.) easily apply to typing formulas. Finally, we use the following abbreviation for labeled modal formulas: ${}^{q}_{y}[N]^{p}_{x} = [?_{y<q} N^{\perp}]^{p}_{x}$.

A *typing judgment* is a statement in the form $\Gamma \vdash t : \mathbf{N} \mid \Delta$, where:

- Γ is a context assigning labelled modal formulas to λ-variables;
- t is a $\lambda\mu$-term;
- $\mathbf{N}$ is a typing formula;
- Δ is a context assigning labelled typing formulas to μ-variables.

The way typing judgments are defined allows to see them as BLLP sequents. This way, again, various concepts from Section 2.2 can be lifted up from sequents to judgments, and this remarkably includes the subtyping relation $\sqsubseteq$.

Typing rules are in Figure 3. The typing rule for applications, in particular,

$$\frac{1 \sqsubseteq p, r\{y/0\} \sqsubseteq q, M \sqsubseteq N\{y/0\}}{\Gamma, x : {}^{r}_{z}[N]^{p}_{y} \vdash x : [M]^{q}_{z} \mid \Delta}\ \mathsf{var} \qquad \frac{\Gamma, x : {}^{s}_{z}[N]^{p}_{y} \vdash t : [M]^{q}_{y} \mid \Delta \qquad r \sqsupseteq q, r \sqsupseteq p}{\Gamma \vdash \lambda x.t : [N \multimap^{s}_{z} M]^{r}_{y} \mid \Delta}\ \mathsf{abs}$$

$$\frac{\Theta \vdash t : [N \multimap^{p}_{x} M]^{q}_{y} \mid \Psi \qquad \Xi \vdash u : [N]^{p}_{x} \mid \Phi \qquad \begin{array}{c} h \sqsupseteq q \qquad k \sqsupseteq q \\ \Gamma \sqsubseteq \Theta \uplus \Upsilon \qquad \Upsilon \sqsubseteq \sum_{b<h} \Xi \\ \Delta \sqsubseteq \Psi \uplus \Pi \qquad \Pi \sqsubseteq \sum_{b<h} \Phi \end{array}}{\Gamma \vdash (t)u : [M]^{k}_{y} \mid \Delta}\ \mathsf{app}$$

$$\frac{\Gamma \vdash t : \mathbf{N} \mid \alpha : \mathbf{M}, \Delta \qquad \mathbf{L} \sqsubseteq \mathbf{N} \uplus \mathbf{M}}{\Gamma \vdash [\alpha]t : [\perp]^{q}_{z} \mid \alpha : \mathbf{L}, \Delta}\ \mu\text{-}\mathsf{name} \qquad \frac{\Gamma \vdash t : [\perp]^{q}_{z} \mid \beta : \mathbf{N}, \Delta}{\Gamma \vdash \mu\beta t : \mathbf{N} \mid \Delta}\ \mu\text{-}\mathsf{abs}$$

Fig. 3. Type Assignment Rules

can be seen as overly complicated. In fact, all premises except the first two are there to allow the necessary degree of malleability for contexts, without which even subject reduction would be in danger. Alternatively, one could consider an explicit subtyping rule, the price being the loss of syntax directedness. Indeed, all malleability results from Section 2.3 can be transferred to the just defined type assignment system.

4.3 Subject Reduction and Polystep Soundness

The aim of this section is to show that *head* reduction preserves types, and as a corollary, that the number of reduction steps to normal form is bounded by a polynomial, along the same lines as in Theorem 1. Actually, the latter will easily follow from the former, because so-called Subject Reduction will be formulated (and in a sense proved) with a precise correspondence between type derivations and proofs in mind.

In order to facilitate this task, Subject Reduction is proved on a modified type-assignment system, called $\mathsf{BLLP}^{\mathsf{mult}}_{\lambda\mu}$ which can be proved equivalent to $\mathsf{BLLP}_{\lambda\mu}$. The only fundamental difference between the two systems lies in

how structural rules, i.e., contraction and weakening, are reflected into the type system. As we have already noticed, $\mathsf{BLLP}_{\lambda\mu}$ has an *additive* flavour, since structural rules are implicitly applied in binary and 0-ary typing rules. This, in particular, makes the system syntax directed and type derivations more compact. The only problem with this approach is that the correspondence between type derivations and proofs is too weak to be directly lifted to a dynamic level (e.g., one step in $\to_{\mathsf{h}}$ could correspond to possibly many steps in $\Longrightarrow$). In $\mathsf{BLLP}^{\mathsf{mult}}_{\lambda\mu}$, on the contrary, structural rules are explicit, and turns it into a useful technical tool to prove properties of $\mathsf{BLLP}_{\lambda\mu}$. The rules of $\mathsf{BLLP}^{\mathsf{mult}}_{\lambda\mu}$ are in [9].

Whenever derivability in one of the systems needs to be distinguished from derivability on the other, we will put the system's name in subscript position (e.g. $\Gamma \vdash_{\mathsf{BLLP}^{\mathsf{mult}}_{\lambda\mu}} t : \mathbf{N} \mid \Delta$). Not so surprisingly, $\mathsf{BLLP}_{\lambda\mu}$ and $\mathsf{BLLP}^{\mathsf{mult}}_{\lambda\mu}$ type exactly the same class of terms:

Lemma 8. $\Gamma \vdash_{\mathsf{BLLP}^{\mathsf{mult}}_{\lambda\mu}} t : \mathbf{N} \mid \Delta$ *iff* $\Gamma \vdash_{\mathsf{BLLP}_{\lambda\mu}} t : \mathbf{N} \mid \Delta$

Proof. The left-to-right implication follows from weakening and contraction lemmas for $\mathsf{BLLP}_{\lambda\mu}$, which are easy to prove. The right-to-left implication is more direct, since additive var and app are multiplicatively derivable. □

Given a $\mathsf{BLLP}^{\mathsf{mult}}_{\lambda\mu}$ type derivation π, one can define a BLLP proof $\pi^{\diamond}$ by induction on the structure of π, closely following Laurent's translation [25]. This way one not only gets some guiding principles for subject-reduction, but can also prove that the underlying transformation process is nothing more than cut-elimination:

Theorem 2 (Subject Reduction). *Let* $\pi \rhd \Gamma \vdash t : \mathbf{N} \mid \Delta$ *and suppose* $t \to_{\mathsf{h}} u$. *Then there is* $\rho \rhd \Gamma \vdash u : \mathbf{N} \mid \Delta$. *Moreover* $\pi^{\diamond} \Longrightarrow^{+} \rho^{\diamond}$.

Observe how performing head reduction corresponds to $\Longrightarrow$, instead of the more permissive $\longrightarrow$. The following, then, is an easy corollary of Theorem 2 and Theorem 1:

Theorem 3 (Polystep Soundness for Terms). *Let* $\pi \rhd \Gamma \vdash t : \mathbf{N} \mid \Delta$ *and let* $t \to^{n}_{\mathsf{h}} u$. *Then* $n \leq q^{\pi^{\diamond}}$.

5 Control Operators

In this section, we show that $\mathsf{BLLP}_{\lambda\mu}$ is powerful enough to type (the natural encoding of) two popular control operators, namely Scheme's `callcc` and Felleisen's $\mathcal{C}$ [2,25].

Control operators change the evaluation context of an expression. This is simulated by the operators μ and $[\cdot]$ which can, respectively, save and restore a stack of arguments to be passed to subterms. This idea, by the way, is the starting point of an extension of Krivine's machine for de Groote's $\lambda\mu$ [13] (see Section 6).

5.1 callcc

An encoding of `callcc` into the $\lambda\mu$-calculus could be, e.g.,

$$\kappa = \lambda x.\mu\alpha.[\alpha](x)\lambda y.\mu\beta.[\alpha]y.$$

Does κ have the operational behavior we would expect from `callcc`? First of all, it should satisfy the following property (see [14]): if $k \notin FV(e)$, then $(\kappa)\lambda k.e \to^* e$. Indeed:

$$(\lambda x.\mu\alpha.[\alpha](x)\lambda y.\mu\beta.[\alpha]y)\lambda k.e \to_{\mathsf{h}} \mu\alpha.[\alpha](\lambda k.e)\lambda y.\mu\beta.[\alpha]y \to_{\mathsf{h}} \mu\alpha.[\alpha]e \to_{\mathsf{h}} e,$$

where the second β-reduction step replaces $e\{k/\lambda y.\mu\beta.[\alpha]y\}$ with e since $k \notin FV(e)$ by hypothesis. It is important to observe that the second step replaces a variable for a term with a free μ-variable, hence weak reduction gets stuck. Actually, our notion of weak reduction is even more restrictive than the one proposed by de Groote in [13]. Head reduction, on the contrary, is somehow more liberal. Moreover, it is also straightforward to check that the reduction of `callcc` in [27, §3.4] can be simulated by head reduction on κ.

But is κ typable in $\mathsf{BLLP}_{\lambda\mu}$? The answer is positive: a derivation typing it with (an instance of) Pierce's law is in Figure 4, where π is the obvious derivation of $x : {}^{r}_{v}[(X \multimap^{s} Y) \multimap^{1} X]^{1} \vdash x : [(X \multimap^{s} Y) \multimap^{1} X]^{r}_{v} \mid \alpha : [X]^{0}$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{y : {}^{s}[X]^{1} \vdash y : [X]^{s} \mid \alpha : [X]^{0}, \beta : [Y]^{0}}\,\mathsf{var}}{y : {}^{s}[X]^{1} \vdash [\alpha]y : [\bot]^{0} \mid \alpha : [X]^{s}, \beta : [Y]^{0}}\,\mu\text{-name}}{y : {}^{s}[X]^{1} \vdash \mu\beta.[\alpha]y : [Y]^{0} \mid \alpha : [X]^{s}}\,\mu\text{-abs}}{\vdash \lambda y.\mu\beta.[\alpha]y : [X \multimap^{s} Y]^{1} \mid \alpha : [X]^{s}}\,\mathsf{abs}}{x : {}^{r}_{v}[(X \multimap^{s} Y) \multimap^{1} X]^{1} \vdash (x)\lambda y.\mu\beta.[\alpha]y : [X]^{r}_{v} \mid \alpha : [X]^{\sum_{v<r} s}_{v}}\,\mathsf{app}}{x : {}^{r}_{v}[(X \multimap^{s} Y) \multimap^{1} X]^{1} \vdash [\alpha](x)\lambda y.\mu\beta.[\alpha]y : [\bot]^{1} \mid \alpha : [X]^{k}}\,\mu\text{-name}$$

π ; $k \sqsupseteq r + \sum_{v<r} s$, $k \sqsupseteq 1$

$$\dfrac{\dfrac{x : {}^{r}_{v}[(X \multimap^{s} Y) \multimap^{1} X]^{1} \vdash \mu\alpha.[\alpha](x)\lambda y.\mu\beta.[\alpha]y : [X]^{k} \mid}{\vdash \lambda x.\mu\alpha.[\alpha](x)\lambda y.\mu\beta.[\alpha]y : [((X \multimap^{s} Y) \multimap^{1} X) \multimap^{r}_{v} X]^{k} \mid}\,\mathsf{abs}}{x : {}^{r}_{v}[(X \multimap^{s} Y) \multimap^{1} X]^{1} \vdash [\alpha](x)\lambda y.\mu\beta.[\alpha]y : [\bot]^{1} \mid \alpha : [X]^{k}}\,\mu\text{-abs}$$

Fig. 4. A Type Derivation for κ

5.2 Felleisen's $\mathcal{C}$

The canonical way to encode Felleisen's $\mathcal{C}$ as a $\lambda\mu$-term is as the term $\aleph = \lambda f.\mu\alpha.(f)\lambda x.[\alpha]x$. Its behavior should be something like

$$(\aleph)wt_1 \dots t_k \to (w)\lambda x.(x)t_1 \dots t_k,$$

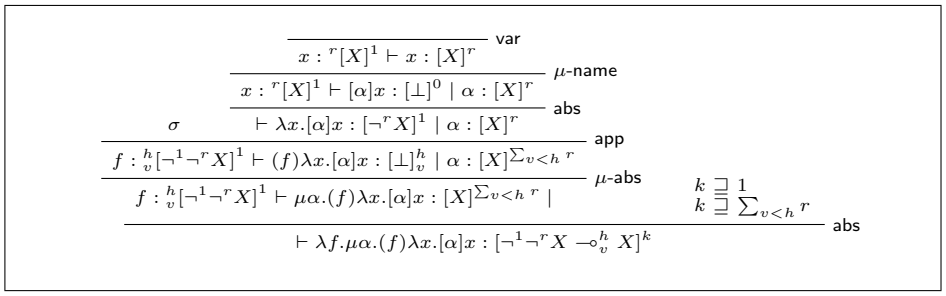

Fig. 5. A Type Derivation for ℵ

where $x \notin FV(t_1, \ldots, t_k)$, i.e., x is a fresh variable. Indeed:

$$(\aleph)wt_1 \ldots t_k \to_{\mathsf{h}} (\mu\alpha.(w)\lambda x.[\alpha](x))t_1 \ldots t_k \to_{\mathsf{h}}^{k} \mu\alpha.(w)\lambda x.[\alpha](x)t_1 \ldots t_k.$$

A type derivation for ℵ is in Figure 5, where σ is a derivation for

$$f : {}^{h}_{v}[\neg^1\neg^r X]^1 \vdash f : [\neg^1\neg^r X]^h_v \mid \alpha : [X]^0.$$

It is worth noting that weak reduction is strong enough to properly simulating the operational behavior of $\mathcal{C}$. It is not possible to type $\mathcal{C}$ in Parigot's $\lambda\mu$, unless an open term is used. Alternatively, a free continuation constant must be used (obtaining yet another calculus [2]). This is one of the reasons why we picked the version of $\lambda\mu$-calculus proposed by de Groote over other calculi. See [12] for a discussion about $\lambda\mu$-and-λ-calculi and Felleisen's $\mathcal{C}$.

6 Abstract Machines

Theorem 3, the main result of this paper so far, tells us that the number of *head-reduction steps* performed by terms typable in $\mathsf{BLLP}_{\lambda\mu}$ is bounded by the weight of the underlying type derivation. One may wonder, however, whether taking the number of reduction steps as a measure of term complexity is sensible or not — substitutions involve arguments which can possibly be much bigger than the original term. Recent work by Accattoli and the first author [1], however, shows that in the case of λ-calculus endowed with head reduction, the unitary cost model is polynomially invariant with respect to Turing machines. We conjecture that those invariance results can be extended to the $\lambda\mu$-calculus.

It can be shown that $\mathsf{BLLP}_{\lambda\mu}$ is polystep sound for another cost model, namely the one induced by de Groote's K, an abstract machine for the $\lambda\mu$-calculus. This is done following a similar proof for PCF typed with linear dependent types [7] and Krivine's Abstract Machine (of which K is a natural extension). The main idea consists in extending BLLP to a type system for K's configurations, this way defining a *weight* for each of them in the form of a resource polynomial. The weight, as expected, can then be shown to decrease at each K's computation step. It is worth noting that the weight defined this way

is fundamentally different than the one from Section 3.1. See [9] for some more details.

7 Conclusions

In this paper we have presented some evidence that the enrichment to Intuitionistic Linear Logic provided by Bounded Linear Logic is robust enough to be lifted to Polarized Linear Logic and the $\lambda\mu$-calculus. This paves the way towards a complexity-sensitive type system, which on the one hand guarantees that typable terms can be reduced to their normal forms in a number of reduction steps which can be read from their type derivation, and on the other allows to naturally type useful control operators.

Many questions have been purposely left open here: in particular, the language of programs is the pure, constant-free, $\lambda\mu$-calculus, whereas the structure of types is minimal, not allowing any form of polymorphism. We expect that endowing BLLP with second order quantification or $\mathsf{BLLP}_{\lambda\mu}$ with constants and recursion should not be particularly problematic, although laborious: the same extensions have already been considered in similar settings in the absence of control [19,7]. Actually, a particularly interesting direction would be to turn $\mathsf{BLLP}_{\lambda\mu}$ into a type system for Ong and Stewart's $\mu\mathsf{PCF}$ [26], this way extending the linear dependent paradigm to a language with control. This is of course outside the scope of this paper, whose purpose was only to delineate the basic ingredients of the logic and the underlying type system.

As we stressed in the introduction, we are convinced this work is the first one giving a time complexity analysis methodology for a programming language with higher-order functions *and control*. One could of course object that complexity analysis of $\lambda\mu$-terms could be performed by translating them into equivalent λ-terms, e.g. by way of a suitable CPS-transform [11]. This, however, would force the programmer (or whomever doing complexity analysis) to deal with programs which are structurally different from the original one. And of course, translations could introduce inefficiencies, which are maybe harmless from a purely qualitative viewpoint, but which could make a difference for complexity analysis.

References

1. B. Accattoli and U. Dal Lago. On the invariance of the unitary cost model for head reduction. In *RTA*, volume 15 of *LIPIcs*, pages 22–37, 2012.
2. Z. M. Ariola and H. Herbelin. Minimal classical logic and control operators. In *ICALP*, volume 2719 of *LNCS*, pages 871–885. Springer, 2003.
3. P. Baillot, P. Coppola, and U. Dal Lago. Light logics and optimal reduction: Completeness and complexity. *Information and Computation*, 209(2):118–142, 2011.
4. P. Baillot and D. Mazza. Linear logic by levels and bounded time complexity. *Theoretical Computer Science*, 411(2):470–503, 2010.
5. P. Baillot and K. Terui. Light types for polynomial time computation in lambda-calculus. *Information and Computation*, 207(1):41–62, 2009.

6. P.-L. Curien and H. Herbelin. The duality of computation. In *ICFP*, pages 233–243. ACM, 2000.
7. U. Dal Lago and M. Gaboardi. Linear dependent types and relative completeness. *Logical Methods in Computer Science*, 8(4), 2012.
8. U. Dal Lago and M. Hofmann. Bounded linear logic, revisited. In *TLCA*, volume 5608 of *LNCS*, pages 80–94. Springer, 2009.
9. U. Dal Lago and G. Pellitta. Complexity analysis in presence of control operators and higher-order functions (long version). `http://arxiv.org/abs/1310.1763`.
10. J. W. de Bakker, A. de Bruin, and J. Zucker. *Mathematical theory of program correctness*. Prentice-Hall International Series in Computer Science. Prentice Hall, 1980.
11. P. de Groote. A CPS-translation of the $\lambda\mu$-calculus. In *CAAP*, volume 787 of *LNCS*, pages 85–99. Springer, 1994.
12. P. de Groote. On the relation between the $\lambda\mu$-calculus and the syntactic theory of sequential control. In *Logic Programming and Automated Reasoning*, pages 31–43. Springer, 1994.
13. P. de Groote. An environment machine for the $\lambda\mu$-calculus. *Mathematical Structures in Computer Science*, 8(6):637–669, 1998.
14. M. Felleisen. On the expressive power of programming languages. In *ESOP*, volume 432 of *LNCS*, pages 134–151. Springer, 1990.
15. M. Gaboardi and S. Ronchi Della Rocca. A soft type assignment system for *lambda*-calculus. In *CSL*, volume 4646 of *LNCS*, pages 253–267. Springer, 2007.
16. J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987.
17. J.-Y. Girard. A new constructive logic: Classical logic. *Mathematical Structures in Computer Science*, 1(3):255–296, 1991.
18. J.-Y. Girard. Light linear logic. *Information and Computation*, 143(2):175–204, 1998.
19. J.-Y. Girard, A. Scedrov, and P. Scott. Bounded linear logic: a modular approach to polynomial-time computability. *Theoretical Computer Science*, 97(1):1–66, 1992.
20. T. Griffin. A formulae-as-types notion of control. In *POPL*, pages 47–58. ACM Press, 1990.
21. S. Gulwani. Speed: Symbolic complexity bound analysis. In *CAV*, volume 5643 of *LNCS*, pages 51–62. Springer, 2009.
22. S. Jost, K. Hammond, H.-W. Loidl, and M. Hofmann. Static determination of quantitative resource usage for higher-order programs. In *POPL*, Madrid, Spain, 2010. ACM Press.
23. Y. Lafont. Soft linear logic and polynomial time. *Theoretical Computer Science*, 318(1):163–180, 2004.
24. O. Laurent. *Étude de la polarisation en logique*. Thèse de doctorat, Université Aix-Marseille II, Mar. 2002.
25. O. Laurent. Polarized proof-nets and $\lambda\mu$-calculus. *Theoretical Computer Science*, 290(1):161–188, 2003.
26. C.-H. L. Ong and C. A. Stewart. A Curry-Howard foundation for functional computation with control. In *POPL*, pages 215–227. ACM Press, 1997.
27. M. Parigot. $\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction. In *LPAR*, volume 624 of *LNCS*, pages 190–201. Springer, 1992.
28. U. Schöpp. Stratified bounded affine logic for logarithmic space. In *LICS*, pages 411–420, 2007.
29. R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström. The worst case execution time problem - overview of methods and survey of tools. *ACM Transactions on Embedded Computing Systems*, 2008.