IMELL Cut Elimination with Linear Overhead

Beniamino Accattoli
Inria & LIX, École Poytechnique, UMR 7161, Palaiseau, France

Claudio Sacerdoti Coen
Alma Mater Studiorum – Università di Bologna, Italy

Abstract

Recently, Accattoli introduced the Exponential Substitution Calculus (ESC) given by untyped proof terms for Intuitionistic Multiplicative Exponential Linear Logic (IMELL), endowed with rewriting rules at-a-distance for cut elimination. He also introduced a new cut elimination strategy, dubbed *the good strategy*, and showed that its number of steps is a time cost model with polynomial overhead for ESC/IMELL, and the first such one.

Here, we refine Accattoli's result by introducing an abstract machine for ESC and proving that it implements the good strategy and computes cut-free terms/proofs within a *linear* overhead.

2012 ACM Subject Classification Theory of computation → Operational semantics

Keywords and phrases Lambda calculus, linear logic, abstract machines

Digital Object Identifier 10.4230/LIPIcs.FSCD.2024.24

Related Version *Extended Version*: `https://arxiv.org/abs/2405.03669` [11]

Supplementary Material *Software (Source Code)*: `https://github.com/sacerdot/sesame/` [41] archived at `swh:1:dir:93776e14435d0dd5cde6deee3d9cf8f515f18fac`

Funding *Claudio Sacerdoti Coen*: Supported by the INdAM-GNCS project "Modelli Composizionali per l'Analisi di Sistemi Reversibili Distribuiti" and by the Cost Action CA2011 EuroProofNet.

1 Introduction

One of the advances of the last decade in the theory of λ-calculus is the study of reasonable cost models carried out by Accattoli, Dal Lago, and co-authors [14, 15, 10, 17, 19], completing the research program started by Dal Lago and Martini almost 20 years ago [27, 34, 29, 28].

The General Problem. Focusing on time, the underlying problem is whether the number of β-steps (of a fixed evaluation strategy) can be taken as a measure of time that is polynomially equivalent to that of random access machines (RAMs); or, equivalently, to that of Turing machines. This is a priori unclear because there are families of terms suffering of *size explosion*: the size of terms grows exponentially with the number of β-steps, independently of the adopted evaluation strategy. The difficulty is finding a strategy, together with a simulation on RAMs of the strategy, working within a overhead polynomial in:

- *Length*: the number of steps of the strategy, and
- *Input*: the size of the initial term.

When a strategy admits such a simulation we say that it is a *a polynomial cost model*. Note that the number of steps of the strategy can be whatever (it need not be polynomial), it is the *overhead* of the simulation that has to be polynomial in the above parameters.

The overhead is required to be polynomial because reasonable frameworks are not necessarily linearly related; this is the reason why the complexity class P is so important. For instance, Turing machines simulate RAMs only within a quadratic overhead. When possible, however, it is interesting to know what is the exact degree of the overhead with

respect to RAMs, which are the commonly accepted abstraction of modern computers. When the overhead is linear – which is the optimal situation – then the polynomial cost model is trustable in practice: it does not hide an acceptable but possibly costly polynomial overhead.

Today, it is well-known that strategies such as weak head, head, and leftmost β-reduction provide polynomial time cost models, and that the same is true for their call-by-value and call-by-need variants [24, 42, 34, 14, 15, 10, 17, 22, 20, 23].

LSC and the Sub-Term Property. The key tool for the polynomially bounded simulations underlying the recent advances is Accattoli and Kesner's *linear substitution calculus* (shortened to LSC) [2, 9], which is an intermediary setting between the λ-calculus and RAMs.

The LSC is a simple λ-calculus with explicit substitutions refining a previous calculus by Milner [40, 32] and where evaluation is *micro-step*, that is, substitution acts on a variable occurrence at a time, rather than on all at once (which is *small-step*). A distinguished feature of the LSC is its rewriting rules *at a distance*, that is, rules where explicit substitutions do not percolate through the term structure but rather act through some contexts. The relevance of the LSC for reasonable time is linked to the *sub-term property* of its standard strategies [15]: at any point, only sub-terms of the initial term are duplicated, thus allowing one to bound the cost of each duplication with the size of the input.

No strategy of the λ-calculus has the sub-term property, which, as discussed by Accattoli [4], is the essence of the size explosion problem. The LSC crystallizes exactly what is needed for refining the λ-calculus to retrieve the sub-term property and circumvent size explosion.

LSC and Abstract Machines. It is also well-known that strategies of the LSC can usually be implemented with environment-based abstract machines whose overhead is *bi-linear*, that is, linear in both the input and length parameters, as shown by Accattoli et al. [6]. Abstract machines handle three tasks, namely the *decomposition of the substitution process*, the *search for redexes*, and *α-renaming*. The difference between the LSC and abstract machines is that the LSC only handles the substitution process, the critical one for avoiding size explosion. Search and α-renaming, indeed, tend to take only a linear overhead. That said, handling search and α-renaming provides an in-depth understanding of evaluation and it is mandatory for obtaining precise bounds from the cost model.

Lifting to Linear Logic. In 2022, Accattoli started a generalization of the mentioned recent results for the λ-calculus to the wider framework of linear logic [4]. Linear logic can be seen as a micro-step system with a tight control over duplications, somewhat similar to the LSC. Accattoli's starting point is the observation that, despite such similarity, no reasonable cut elimination strategy for linear logic was available. In particular, no cut elimination strategy with the sub-term property was known.

In [4], he introduces a generalization of the LSC to intuitionistic multiplicative exponential linear logic (IMELL). His *exponential substitution calculus* (ESC) is an untyped calculus of proof terms for the sequent calculus proofs of IMELL, endowed with cut elimination at a distance and having IMELL as typing system. He then designs a new cut elimination strategy, dubbed *good strategy*, which has the sub-term property and which is a polynomial cost model for ESC, the first such cost model for an expressive fragment of linear logic.

This Paper. Here, we design an environment-based abstract machine implementing the good strategy of ESC, show that it has bi-linear overhead, and provide an OCaml implementation. The result is not surprising, and yet we believe that it is interesting, for various reasons.

Firstly, it is the first linear overhead result for an expressive fragment of linear logic. It allows one to extract precise time bounds from the length of cut elimination, and can thus be useful, for instance, in linear-logic-based implicit computational complexity.

Secondly, our study differs in many details from other environment-based machines. There are in fact at least two literatures related to our work (surveyed in the next paragraph). A recent one about cost analyses of machines for λ-calculi and an older one about machines for linear logic, here referred to as *linear machines*. The main differences are the following:

- *Natural deduction vs sequent calculus*: machines usually implement calculi based on natural deduction, while ESC is based on sequent calculus, and some aspects are different. For instance, our machine has no argument stack (as it is expected, after Herbelin [31]).
- *Micro-step source and no structural equivalence*: the implemented calculus is usually small-step, while here it is micro-step. In the literature, when the calculus is micro-step it is usually implemented by the machine up to a notion of structural equivalence. Here, there is no need of structural equivalence.
- *Strong vs weak*: our machine computes cut-free proofs, while other linear machines in the literature do not, as they usually only perform weak (or surface) evaluation (that is, they do not evaluate under abstraction and inside promotions). To our knowledge, ours is the first strong linear machine.
- *Strong with no backtracking*: machines for strong evaluation usually have transitions for sequentially backtracking once the evaluation of a sub-term is over. Here, we adopt a recent new technique for strong machines by Accattoli and Barenbaum [5], that avoids sequential backtracking. The idea is to assign distinct jobs to each sub-term and simply jumping to the next job once the current one is over. This approach structures the machine in a very different way and provides drastically simpler strong machines.
- *Complexity analysis*: the complexity of linear machines is never studied in the literature.
- *OCaml implementation*: linear machines are usually studied theoretically and never implemented. We provide an OCaml implementation verifying our complexity analysis. In particular, it is the first implementation of the technique for strong machines of [5].

Summing up, our machine is simple, given the complex setting that it implements, thanks to the absence of stacks and backtracking. In fact, it uses just one (non-trivial) data structure.

Methodologically, the main difference between our work and the literature on linear machines is that those machines were developed to provide insights about λ-calculi and functional programming (such as no garbage collection, in-place updates, and single pointer property) while here we proceed the other way around, using the recent theory of the λ-calculus to provide insights about linear logic.

Architecture of the Result. The good strategy that we implement is a non-deterministic but *diamond* strategy, where diamond means that the choices do not affect termination nor evaluation lengths (otherwise the number of steps would not be a well-defined cost model). Our implementation is in two phases. The first and main one, performed by a deterministic machine dubbed *SESAME* (Strong Exponential Substitution Abstract Machine without Erasure), never erases. The second one is a simple garbage collection pass over the output of SESAME, that produces a cut-free term. For the correctness of SESAME, we thus need to relate a non-deterministic strategy and a deterministic machine, which is slightly unusual. For that, we follow the abstract recipe of Accattoli et al. [12] which is here simplified because our case is not up to structural equivalence and there is no implosiveness.

Related Work. Accattoli and co-authors studied at length the overhead of machines for λ-calculi both micro-step [6, 7, 10] and small-step [3, 8, 18, 13, 12, 19, 5]. Biernacka et al. similarly studied machines for strong call-by-value [22] and strong call-by-need [23].

Lafont considered the first linear logic (LL) machine, based on categorical combinators for intuitionistic LL (ILL) [33]. Abramsky considered an environment-based machine for ILL and a chemical-style machine for LL, both doing only weak evaluation [1]. The latter machine was then studied by Mikami and Akama [39] and Sato and co-authors [43, 38]. Turner and Wadler study the use of IMELL for memory management and give two machines, one with the single pointer property and one without it but with a memoization mechanism [45]. Alberti and Ritter also deal with the single pointer property [21]. Bonelli gives a machine based on an unusual sequent calculus for ILL [25].

The interaction abstract machine (IAM) is an unusual token-based machine for LL proofs first studied by Danos and Regnier [30] and Mackie [35]. Recent work by Accattoli et al. showed that the IAM overhead is unreasonable for both time and space [16, 17, 46]. Mackie also studied interaction nets-based implementations of LL [36, 37].

2 The Exponential Substitution Calculus

In this section, we briefly recall Accattoli's exponential substitution calculus (ESC) [4] which is an untyped calculus having IMELL as system of simple types.

For more explanations, we refer the reader to [4]. The main differences with respect to [4] is that, for lack of space, we omit tensor and we only deal with micro-step rewriting rules (omitting the small-step exponential rule, which is derivable). In IMELL, linear implication is "more important" than tensor, as tensors are not needed to simulate the λ-calculus. In Appendix B of the technical report [11], we explain how to straightforwardly extend our study to tensor.

Values and Terms. The grammars of ESC are in the upper part of Fig. 1. Variables are of two disjoint kinds, multiplicative and exponential, and we refer to variables of unspecified kind using x, y, z. Values are the proof terms associated to axioms or to the right rules and, beyond variables, are *abstractions* $\lambda x.t$ and *promotions* $!t$. The proof terms decorating left rules are *subtractions* $[m \otimes v, x]t$, *derelictions* $[e?x]t$, and *cuts* $[v \to x]t$, which is in red because of its special role. Note that cuts and subtractions are *split*, that is, have values (rather than terms) as left sub-terms. The constructors $\lambda x.t$, $[m \otimes v, x]t$, $[e?x]t$, and $[v \to x]t$ bind x in t. We identify terms up to α-renaming. Free variables (resp. multiplicative/exponential variables) are defined as expected, and noted $\mathtt{fv}(t)$ (resp. $\mathtt{mfv}(t)$ and $\mathtt{efv}(t)$). We use $|t|$ for the number of constructors in t, and $|t|_x$ for the number of free occurrences of x in t. There is a notion of *proper term* ensuring the linearity of multiplicative variables and the exponential boundary of promotions, detailed in Appendix A of the tech report [11]. The only relevant case is: $!t$ *is proper if* t *is proper and* $\mathtt{mfv}(t) = \emptyset$. In the following, terms are assumed to be proper.

Contexts and Plugging. The broadest notion of context that we consider is *general contexts* C, which simply allow the hole $\langle\cdot\rangle$ to replace any sub-term in a term. Because of split cuts and subtractions (that is, the fact that their left sub-term is a value rather than a term), the definition relies on the auxiliary notion of *value context* V. The definition also uses *left contexts* L, which are contexts under left constructors (or, for binary left constructors, under the right sub-term) that play a key role in the system – their use in defining C is just to keep the grammar compact. We also need *cut contexts* E, which are noted with E because they shall play the role of machine environments in Sect. 5.

Mult. values	$v_{\mathtt{m}}$	$::=$	$m \mid \lambda x.t$	Values	v, v'	$::=$	$v_{\mathtt{m}} \mid v_{\mathtt{e}}$
Exp. values	$v_{\mathtt{e}}$	$::=$	$e \mid !t$	Terms	t	$::=$	$v \mid [v{\to}x]t \mid [m{\otimes}v, x]t \mid [e?x]t$

Cut ctxs	$E ::= \langle\cdot\rangle \mid [v{\to}x]E$	Left ctxs	$L ::= \langle\cdot\rangle \mid [v{\to}x]L \mid [m{\otimes}v, x]L \mid [e?x]L$
Value ctxs	$V ::= \langle\cdot\rangle \mid \lambda x.C \mid !C$	Ctxs	$C ::= V \mid [V{\to}x]t \mid [m{\otimes}V, x]t \mid L\langle C\rangle$

Micro-step root multiplicative rules

$$
\begin{array}{rcll}
[v_{\mathtt{m}}{\to}m]C\langle\!\langle m\rangle\!\rangle & \mapsto_{\mathtt{axm}_1} & C\langle\!\langle v_{\mathtt{m}}\rangle\!\rangle & \\
[n{\to}m]C\langle[m{\otimes}v, x]t\rangle & \mapsto_{\mathtt{axm}_2} & C\langle[n{\otimes}v, x]t\rangle & \\
[\lambda y.s{\to}m]C\langle[m{\otimes}v, x]t\rangle & \mapsto_{\multimap} & C\langle[v{\to}y]L\langle[v'{\to}x]t\rangle\rangle & \text{with } s = L\langle v'\rangle
\end{array}
$$

Micro-step root exponential rules

$$
\begin{array}{rcll}
[v_{\mathtt{e}}{\to}e]C\langle\!\langle e\rangle\!\rangle & \mapsto_{\mathtt{axe}_1} & [v_{\mathtt{e}}{\to}e]C\langle\!\langle v_{\mathtt{e}}\rangle\!\rangle & \\
[f{\to}e]C\langle[e?x]t\rangle & \mapsto_{\mathtt{axe}_2} & [f{\to}e]C\langle[f?x]t\rangle & \\
[!s{\to}e]C\langle[e?x]t\rangle & \mapsto_{!} & [!s{\to}e]C\langle L\langle[v{\to}x]t\rangle\rangle & \text{with } s = L\langle v\rangle \\
[v_{\mathtt{e}}{\to}e]t & \mapsto_{\mathtt{w}} & t & \text{if } e \notin \mathtt{fv}(t)
\end{array}
$$

Contextual closure $\dfrac{t \mapsto_a s}{C\langle t\rangle \to_a C\langle s\rangle}$ for $a \in \{\mathtt{axm}_1, \mathtt{axm}_2, \multimap, \mathtt{axe}_1, \mathtt{axe}_2, !, \mathtt{w}\}$

Notations

$$
\begin{array}{rcl}
\text{Multiplicative } \to_{\mathtt{m}} & := & \to_{\mathtt{axm}_1} \cup \to_{\mathtt{axm}_2} \cup \to_{\multimap} \\
\text{Exponential } \to_{\mathtt{e}} & := & \to_{\mathtt{axe}_1} \cup \to_{\mathtt{axe}_2} \cup \to_{!} \cup \to_{\mathtt{w}} \\
\text{ESC } \to_{\mathtt{ESC}} & := & \to_{\mathtt{m}} \cup \to_{\mathtt{e}} \\
\text{Non-erasing ESC } \to_{\mathtt{ESC}_{-\mathtt{w}}} & := & \to_{\mathtt{m}} \cup \to_{\mathtt{axe}_1} \cup \to_{\mathtt{axe}_2} \cup \to_{!}
\end{array}
$$

Figure 1 The Exponential Substitution Calculus (ESC).

A fact used pervasively is that every term t writes, or *splits* uniquely as $t = L\langle v\rangle$. For instance $[e?m][v{\to}n]\lambda g.[m{\otimes}n, f]f$ splits as $L = [e?m][v{\to}n]\langle\cdot\rangle$ and $v = \lambda g.[m{\otimes}n, f]f$.

Because of split cuts and subtractions, the definition of plugging $C\langle t\rangle$ (or $C\langle D\rangle$) of a term t (or a context D) in a context C is slightly tricky, as it has to preserve the split shape. We refer the reader to [4] for such details, the definition is mostly as expected (the only two subtle cases are for $[\langle\cdot\rangle{\to}x]t$ and $[m{\otimes}\langle\cdot\rangle, x]$). Plugging can capture variables and we use $C\langle\!\langle t\rangle\!\rangle$ when we want to prevent it.

Types. The formulas of IMELL, and the deductive rules of the sequent calculus annotated with ESC terms, are in Fig. 2. They are taken directly from [4]. The typing system for ESC is exactly the standard sequent calculus for IMELL. Both formulas and rules are standard, but for the decoration with proof terms and the side conditions about variable names of the form $\Gamma\#\Delta$, which is a shortcut for $\mathtt{dom}\Gamma \cap \mathtt{dom}\Delta = \emptyset$. Linear implication $\multimap$ is also referred to as *lolli*. The only atomic formula that we consider, $X_{\mathtt{m}}$, is multiplicative. There is no multiplicative unit because, in presence of the exponentials, 1 can be simulated by $!(X_{\mathtt{m}} \multimap X_{\mathtt{m}})$. We distinguish between multiplicative and exponential axioms, in order to decorate them with the corresponding kind of variable.

Note that the weakening and contraction rules do not add constructors to terms. This is crucial in order to keep the calculus manageable. Note also that the decorations of the cut and $\multimap_l$ rules are *split*, as explained in the previous section. Clearly, typed terms are proper.

In this paper, types are only referred to in Lemma 3 below.

Multiplicative Cut Elimination Rules. The rewriting rules are in Fig. 1. The ESC has three multiplicative rules, in particular two for axioms, depending on whether they are acted upon ($\to_{\mathtt{axm}_1}$) or used to rename another multiplicative (thus linear) variable ($\to_{\mathtt{axm}_2}$). Rule

$$\textsc{Formulas} \quad A,B,C \quad ::= \quad X_{\mathtt{m}} \mid A \otimes B \mid A \multimap B \mid !A$$

$$\textsc{Multiplicative rules}$$

$$\frac{A \neq !B}{m:A \vdash m:A}\ \mathtt{ax_m} \qquad \frac{x:A,\Gamma \vdash t:B}{\Gamma \vdash \lambda x.t : A \multimap B}\ \multimap_r$$

$$\frac{\Gamma \vdash L\langle v\rangle : A \qquad \Delta, x:B \vdash t:C \qquad \Gamma\#(\Delta, x:B),\ m \text{ fresh}}{\Gamma,\Delta, m:A \multimap B \vdash L\langle [m \otimes v, x]t\rangle : C}\ \multimap_l$$

$$\textsc{Exponential rules}$$

$$\frac{}{e:!A \vdash e:!A}\ \mathtt{ax_e}$$

$$\frac{\Gamma, x:A \vdash t:B \qquad e \text{ fresh}}{\Gamma, e:!A \vdash [e?x]t : B}\ !_l \qquad \frac{!\Gamma \vdash t:A}{!\Gamma \vdash !t : !A}\ !_r$$

$$\frac{\Gamma \vdash t:A \qquad e \text{ fresh}}{\Gamma, e:!B \vdash t:A}\ \mathtt{w} \qquad \frac{\Gamma, e:!B, f:!B \vdash t:A}{\Gamma, e:!B \vdash \{e{\to}f\}t : A}\ \mathtt{c}$$

$$\textsc{Cut}$$

$$\frac{\Gamma \vdash L\langle v\rangle : A \qquad \Delta, x:A \vdash t:B \qquad \Gamma\#(\Delta, x:A)}{\Gamma,\Delta \vdash L\langle [v{\to}x]t\rangle : B}\ \mathtt{cut}$$

Figure 2 IMELL has a type system for ESC.

$\to_{\mathtt{axm}_1}$ is expressed generically for multiplicative values $v_{\mathtt{m}}$ (that is, multiplicative variables m and abstractions $\lambda x.t$). In $\to_{\mathtt{axm}_1}$, $\to_{\mathtt{axm}_2}$, and $\to_{\multimap}$, it is silently assumed that C does not capture m in $\to_{\mathtt{axm}_2}$ and $\to_{\multimap}$ (what is noted $C\langle\!\langle m\rangle\!\rangle$ in $\to_{\mathtt{axm}_1}$, while $\langle\!\langle\cdot\rangle\!\rangle$ is not used in $\to_{\mathtt{axm}_2}$, and $\to_{\multimap}$ because C might capture other variables in t and v). Note that, since C cannot capture m in these rules and terms are assumed to be proper, the hole of C cannot be contained in a !; this kind of context is called a *multiplicative context* in [4].

In $\to_{\multimap}$, the rule has to respect split cuts, which is why, for writing the reduct, the sub-term s is split on-the-fly. An example of $\to_{\multimap}$ step follows:

$$[\lambda e.[e?m]m{\to}n][n \otimes !f, o]o \quad \to_{\multimap} \quad [!f{\to}e][e?m][m{\to}o]o.$$

Exponential Rules. There are also four exponential rules, with again two rules for axioms. Replacement of variables ($\to_{\mathtt{axe}_1}$) and erasure ($\to_{\mathtt{w}}$) are expressed generically for exponential values $v_{\mathtt{e}}$ (that is, exponential variables f and promotions $!t$), interaction with derelictions (in $\to_{\mathtt{axe}_2}$ and $\to_!$) instead requires inspecting $v_{\mathtt{e}}$. Rule $\to_!$ removes the dereliction, copies the promotion body, and puts it in a cut – in proof nets jargon, *it opens the box*. To preserve the split shape, the body of the promotion is split and only the value is cut. An example:

$$[![f{\to}g]g{\to}e]\lambda m.[e?e'][m \otimes e', n]n \quad \to_! \quad [![f{\to}g]g{\to}e]\lambda m.[f{\to}g][g{\to}e'][m \otimes e', n]n.$$

Note that $\to_!$ entangles *interaction with a dereliction* and *duplication*, which is not what proof nets usually do (but this is what the LSC does). It is silently assumed that C does not capture e in $\to_{\mathtt{axe}_2}$ and $\to_!$ but it might capture other variables in t.

Comments about the Rewriting Rules. Unsurprisingly, proper terms are stable by reduction [4]. We also use a notion of position in terms.

▶ **Definition 1** (Positions). *A* position *in a term t is a decomposition $t = C\langle s\rangle$ such that s is a sub-term of t.*

For later defining the good strategy, we identify a redex with its position, which is a context. Every step $t \to_{\mathtt{ESC}_{\neg \mathtt{w}}} s$ reduces a redex of shape $t = C\langle [v{\to}x]D\langle t_x\rangle\rangle$ where t_x is an *occurrence* of x, *i.e.* a sub-term of t of shape x, $[x{\otimes}v, y]s$, or $[x?y]s$. The *redex position of $\to_{\mathtt{ESC}_{\neg \mathtt{w}}}$ steps* is the context $C\langle [v_{\mathtt{e}}{\to}x]D\rangle$. The *redex position of $\to_{\mathtt{w}}$ steps* is the context closing the root step. We write $C : t \to_{\mathtt{ESC}} s$ for a redex of position C in t.

Clashes. The presence of many constructors in an untyped setting gives rise to *clashes*, that is, irreducible cuts.

▶ **Definition 2** (Clashing, smooth, clash-free terms). *A* clash *is a term of the form $[v_{\mathtt{m}}{\to}e]t$ or $[v_{\mathtt{e}}{\to}m]t$. A term t is* clashing *if it has a clash as sub-term. Moreover, t is* clash-free *if, co-inductively, t is not clashing and s is clash-free for any step $t \to_{\mathtt{ESC}} s$.*

Clash-freeness is guaranteed by all forms of types (simple, polymorphic, recursive, and multi types, for instance), in particular by the IMELL types in Fig. 2, as ensured by the next lemma. Here, we adopt the untyped calculus but consider only clash-free terms, as in [4].

▶ **Lemma 3** ([4]). *Let t be a proper term.*
1. *If t has no clashes and $t \not\to_{\mathtt{ESC}}$ then t is cut-free.*
2. *If t is typable (by the IMELL type system in Fig. 2) then t is clash-free.*

Postponement of Garbage Collection. The erasing rule $\to_{\mathtt{w}}$, that models garbage collection, can be postponed. This point shall be crucial for our study.

▶ **Proposition 4** (Postponement of garbage collection, [4]). *If $t \to^*_{\mathtt{ESC}} s$ then $t \to^*_{\mathtt{ESC}_{\neg \mathtt{w}}} \to^*_{\mathtt{w}} s$.*

Basic Evaluation. As an intermediary step towards our results, we consider also a restricted form of ESC evaluation, forbidding reduction under all constructors but the right sub-terms of cuts, and dub it *basic evaluation*. It is one possible ESC analogous of weak evaluation in the λ-calculus (which forbids evaluation under abstraction). In λ-calculi with explicit substitutions, weak normal forms of closed terms are *answers*, i.e. abstractions possibly surrounded by explicit substitutions. We obtain a similar property for basic evaluation.

▶ **Definition 5** (Basic evaluation, answer). *A step $C : t \to_{\mathtt{ESC}} s$ (that is, a redex of position C in t) is* basic *if C is a cut context, which is also noted with $t \to_{\mathtt{b}} s$, or $t \to_{\mathtt{b}a} s$ with $a \in \{\mathtt{axm}_1, \mathtt{axm}_2, \multimap, \mathtt{axe}_1, \mathtt{axe}_2, !, \mathtt{w}\}$ if we want to specify the kind of step. Moreover, $\to_{\mathtt{b}\neg\mathtt{w}}$ denotes a non-erasing (that is, not $\mathtt{w}$) basic step. A term t is an* answer *if $t = E\langle v\rangle$ with E a cut context and v not a variable.*

▶ **Lemma 6.** *Let t be a closed clash-free term. Either t has a $\to_{\mathtt{b}\neg\mathtt{w}}$-step or it is an answer.*

Basic evaluation is enough to simulate in ESC the call-by-name/value weak evaluation of closed λ-terms, via Girard's translations to linear logic. Moreover, basic non-erasing evaluation is deterministic (and diamond, a property defined in Sect. 3, when erasing steps are considered). We omit the proofs of these facts, because neither basic evaluation nor λ-terms are the focus of the paper.

Dominating free variables of contexts

$$\begin{array}{rcl|rcl}
\mathtt{dv}(\langle\cdot\rangle) & := & \emptyset & \mathtt{dv}([m\otimes V, x]t) & := & \{m\} \cup \mathtt{dv}(V) \\
\mathtt{dv}(\lambda x.C) & := & \mathtt{dv}(C)\backslash\{x\} & \mathtt{dv}([m\otimes v, x]C) & := & \begin{cases} \{m\} \cup (\mathtt{dv}(C)\backslash\{x\}) & \text{if } x\in\mathtt{dv}(C) \\ \mathtt{dv}(C) & \text{otherwise.} \end{cases} \\
\mathtt{dv}(!C) & := & \mathtt{dv}(C) & & & \\
\mathtt{dv}([v\to x]C) & := & \mathtt{dv}(C)\backslash\{x\} & \mathtt{dv}([e?x]C) & := & \begin{cases} \{e\} \cup (\mathtt{dv}(C)\backslash\{x\}) & \text{if } x\in\mathtt{dv}(C) \\ \mathtt{dv}(C) & \text{otherwise.} \end{cases} \\
\mathtt{dv}([V\to x]t) & := & \mathtt{dv}(V) & & &
\end{array}$$

Good value contexts

$$V_G ::= \langle\cdot\rangle \mid \lambda x.G \mid !G$$

Good contexts

$$G ::= V_G \mid [m\otimes v, x]G \mid [m\otimes V_G, x]t \mid [e?x]G \mid [v\to x]G \text{ if } x \notin \mathtt{dv}(G)$$

Figure 3 Definitions for the good strategy: dominating free variables and good contexts.

Out Cuts and Garbage Collection. We shall mainly deal with cuts that are not contained in any other cut, the *out cuts*, which induce a notion of *out variable*.

▶ **Definition 7** (Out cuts, out variables). *The* out cuts *of a term t are those cuts in t that are not contained in any other cut, that is, if $t = C\langle[v\to x]s\rangle$ then $[v\to x]s$ is an out cut of t if C cannot be written as $C = D\langle[V\to y]u\rangle$ for some D, V, y, and u. The set of* out variables $\mathtt{ov}(t)$ *of t contains the variables having at least one occurrence out of all cuts of t.*

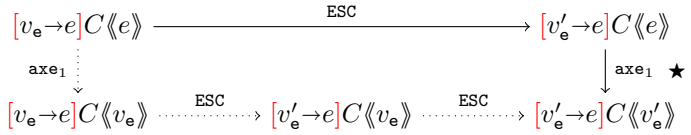
Out variables allow us to define a lax notion of cut-free terms where there are some cuts, but they can only be garbage collection cuts, or concern variable occurrences contained in garbage, so that the term becomes cut-free after garbage collection.

▶ **Definition 8** (Cut-freeness up to garbage). *A term t is cut-free up to garbage if $x \notin \mathtt{ov}(s)$ for all the out cuts $[v\to x]s$ of t.*

3 The Good Strategy

Here, we define the *good cut elimination strategy* $\to_{\mathtt{G}}$ for ESC, recalling the discussion from [4] that explains the design of the strategy. The strategy is conceived as to have the *sub-term property*, which is crucial for time analyses and which is defined as follows: every duplicated (or erased) sub-term during a cut elimination sequence is a sub-term of the initial term (up to variable renamings). See [4] for extensive discussions about the sub-term property.

Breaking the Sub-Term Property. When does the sub-term property not hold? One has to duplicate an exponential value v_{e} *touched* by previous steps. In our setting, *touched* can mean two things. Either a redex *fully* contained in v_{e} is reduced, obtaining v'_{e}, and then v'_{e} is duplicated (or erased), as in the step marked with ★ in the following diagram (the other, dashed path of which has the sub-term property):



Preventing these situations from happening, thus forcing evaluation to follow the other (dashed) side of the diagram, is easy. It is enough to forbid the position of the reduced redex

to be inside the left sub-term of a cut – we say inside a *cut value* for short. It is however not enough, because cuts are also *created*. Consider:

$$[\lambda e.e{\to}m][m{\otimes}v_{\mathsf{e}},x]t \quad \to_{\mathtt{ESC}} \quad [\lambda e.e{\to}m][m{\otimes}v'_{\mathsf{e}},x]t \quad \to_{\multimap} \quad [v'_{\mathsf{e}}{\to}e][e{\to}x]t \quad \overset{\star}{\to}_{\mathtt{axe}_1} \quad [v'_{\mathsf{e}}{\to}e][v'_{\mathsf{e}}{\to}x]t$$

Reducing inside the subtraction value v_{e} leads to a *later* breaking of the sub-term property by the $\mathtt{axe}_1$ step, because the $\to_{\multimap}$ step creates a cut with v'_{e} inside. Preventing these cases is tricky, because forbidding reducing subtraction values leads to cut elimination stopping too soon, without producing a cut-free term. In the λ-calculus, it corresponds to forbidding reducing inside arguments, which leads to *head* reduction, that does not compute normal λ-terms. We shall then forbid reducing only subtraction values which are *at risk* of becoming cuts. In λ-calculus, leftmost reduction does reduce arguments but only when the left sub-term of the application is normal and not an abstraction, so that the argument is not involved in a β-redex. We shall do something similar here, but the sequent calculus formalizes this constraint differently, by checking that some variables are not captured.

Dominating Variables. The key notion is the one of *dominating (free) variables* $\mathtt{dv}(C)$ of a context (where C is meant to be the position of a redex), defined in Fig. 3, the base case of which is for $[m{\otimes}V,x]t$. If C is a position and $x \in \mathtt{dv}(C)$ then $[v{\to}x]C$ turns C into a dangerous position, that is, a redex of position $[v{\to}x]C : t \to_{\mathtt{ESC}} s$ might lead to a breaking of the sub-term property later on during cut elimination. In the example, e belongs to $\mathtt{dv}(C)$ for every context $C := [e?m][m{\otimes}V,x]s$ of $[e?m][m{\otimes}v,x]s$, for every V.

The Good Strategy. These considerations lead to the notion of *good contexts* in Fig. 3. A good context forbids the two ways of breaking the sub-term property: its hole cannot be in a cut value (note the absence of the production $[V_G{\to}x]t$) nor in a subtraction value such that one of its dominating variables is cut (because of the production $[v{\to}x]G$ if $x \notin \mathtt{dv}(G)$).

▶ **Definition 9** (Good steps, good strategy). *A step $C : t \to_{\mathtt{ESC}} s$ is* good *if its position C is good. In such a case, we write $t \to_{\mathtt{G}} s$. The* good cut elimination strategy *is simply $\to_{\mathtt{G}}$. We also use $\to_{\mathtt{G}a}$ to stress that the good step is of kind $a \in \{\mathtt{axm}_1, \mathtt{axm}_2, \multimap, \mathtt{axe}_1, \mathtt{axe}_2, !, \mathtt{w}\}$. The non-erasing good strategy $\to_{\mathtt{G}_{\neg\mathtt{w}}}$ is the variant of $\to_{\mathtt{G}}$ excluding $\to_{\mathtt{Gw}}$ steps.*

The sub-term property in the following theorem captures the quantitative aspect for cost analyses, i.e. the bound on the size of duplicated values by the size of the initial term.

▶ **Theorem 10** (Properties of the good strategy, [4]).

1. Quantitative sub-term property*: if $e : t \to_{\mathtt{G}}^{*} s$ and v be a value erased or duplicated along e, then $|v| \leq |t|$.*
2. Diamond*: if $s_1 \; {}_{\mathtt{G}}{\leftarrow} t \to_{\mathtt{G}} s_2$ and $s_1 \neq s_2$ then $s_1 \to_{\mathtt{G}} u \; {}_{\mathtt{G}}{\leftarrow} s_2$ for some u.*
3. Fullness*: if t is clash-free and not $\to_{\mathtt{ESC}}$-normal then $t \to_{\mathtt{G}} s$ for some s.*
4. Good polynomial cost model*: if $\Gamma \vdash t : A$ is a typed term then there exist k and a cut-free term s such that $t \to_{\mathtt{G}}^{k} s$, and such a reduction sequence is implementable on RAMs in time polynomial in k and $|t|$.*

The algorithmic aspect of Theorem 10.4 is proved in [4] via arguments that do not establish the degree of the polynomial bound. The aim of this paper is exactly to show that a carefully designed abstract machine provides a bound that is linear in both k and $|t|$.

The Diamond Property. Let us provide some background about the diamond property. Following Dal Lago and Martini [34], we say that a relation $\to_{\mathtt{r}}$ is *diamond* if $s_1 \ {}_{\mathtt{r}}\!\leftarrow t \to_{\mathtt{r}} s_2$ and $s_1 \neq s_2$ imply $s_1 \to_{\mathtt{r}} u \ {}_{\mathtt{r}}\!\leftarrow s_2$ for some u. The terminology in the literature is inconsistent: Terese [44, Exercise 1.3.18] dubs this property CR[1], and defines the diamond more restrictively, without requiring $s_1 \neq s_2$ in the hypothesis: s_1 and s_2 have to join even if $s_1 = s_2$.

Standard corollaries of Dal Lago and Martini's notion are that, if $\to_{\mathtt{r}}$ is diamond, then:

1. *Confluence*: $\to_{\mathtt{r}}$ is confluent, that is, $s_1 \ {}^{*}_{\mathtt{r}}\!\leftarrow t \to^{*}_{\mathtt{r}} s_2$ implies $s_1 \to^{*}_{\mathtt{r}} u \ {}^{*}_{\mathtt{r}}\!\leftarrow s_2$ for some u;
2. *Length invariance*: all $\mathtt{r}$-evaluations with the same start and $\mathtt{r}$-normal end terms have the same length (*i.e.* if $e\colon t \to^{k}_{\mathtt{r}} s$ and $e'\colon t \to^{h}_{\mathtt{r}} s$ with s $\mathtt{r}$-normal, then $h = k$);
3. *Uniform normalization*: t is weakly $\mathtt{r}$-normalizing if and only if it is strongly $\mathtt{r}$-normalizing.

Basically, the diamond property captures a more liberal form of determinism. In particular, length invariance is essential in order to take the number of steps of a strategy as a cost model. Without it, indeed, the number of steps of a non-deterministic strategy would be an ambiguously defined notion of cost.

4 Preliminaries on Abstract Machines

Abstract Machines Glossary. Abstract machines manipulate *pre-terms*, that is, terms without implicit α-renaming. In this paper, an *abstract machine* is a quadruple $\mathtt{M} = (\mathtt{States}, \rightsquigarrow, \cdot \triangleleft \cdot, \underline{\cdot})$ the components of which are as follows.

- *States.* A state $Q \in \mathtt{States}$ is composed by the *active term* t, plus one data structure which depends on the actual machine. Terms in states are actually pre-terms.
- *Transitions.* The pair $(\mathtt{States}, \rightsquigarrow)$ is a transition system with transitions $\rightsquigarrow$ partitioned into *principal transitions*, whose union is noted $\rightsquigarrow_{\mathtt{pr}}$ and that are meant to correspond to cut-elimination steps on the calculus, and *search transitions*, whose union is noted $\rightsquigarrow_{sea}$, that take care of searching for (principal) redexes.
- *Initialization.* The component $\triangleleft \subseteq \Lambda \times \mathtt{States}$ is the *initialization relation* associating terms to initial states. It is a *relation* and not a function because $t \triangleleft Q$ maps a λ-term t (considered modulo α) to a state Q having a *pre-term representant* of t (which is not modulo α) as active term. Intuitively, any two states Q and Q' such that $t \triangleleft Q$ and $t \triangleleft Q'$ are α-equivalent. The initializing terms (i.e. those t such that $t \triangleleft Q$ for some Q) are always *proper* and *clash-free*. A state Q is *reachable* if it can be reached starting from an initial state, that is, if $Q' \rightsquigarrow^{*} Q$ where $t \triangleleft Q'$ for some t and Q', shortened as $t \triangleleft Q' \rightsquigarrow^{*} Q$.
- *Read-back.* The read-back function $\underline{\cdot} : \mathtt{States} \to \Lambda$ turns reachable states into terms and satisfies the *initialization constraint*: if $t \triangleleft Q$ then $\underline{Q} =_{\alpha} t$.

Further Terminology and Notations. A state is *final* if no transitions apply. A *run* $r : Q \rightsquigarrow^{*} Q'$ is a possibly empty finite sequence of transitions, the length of which is noted $|r|$; note that the first and the last states of a run are not necessarily initial and final. If a and b are transitions labels (that is, $\rightsquigarrow_a \subseteq \rightsquigarrow$ and $\rightsquigarrow_b \subseteq \rightsquigarrow$) then $\rightsquigarrow_{a,b} := \rightsquigarrow_a \cup \rightsquigarrow_b$ and $|r|_a$ is the number of a transitions in r, and $|r|_{\neg a}$ is the number of transitions in r that are not $\rightsquigarrow_a$.

Well-Boundness and Renamings. For the machines at work in this paper, the pre-terms in initial states shall be *well-bound*, that is, they have pairwise distinct bound names; for instance $[m \otimes \lambda n.n, e]\lambda f.f$ is well-bound while $[m \otimes \lambda n.n, e]\lambda n.n$ is not. We shall also write t^{α} in a state Q for a *fresh well-bound renaming* of t, *i.e.* t^{α} is α-equivalent to t, well-bound, and its bound variables are fresh with respect to those in t and in the other components of Q.

Implementation Theorem, Abstractly. We now formally define the notion of a machine implementing a strategy. Since the good strategy is non-deterministically diamond but the machine that shall implement it is deterministic, we need a slightly unusual form of implementation theorem. As it is standard, machine transitions shall be mapped to equalities or steps on the calculus. For the other direction, however, we obtain only a *big-step* simulation, that is, if the strategy on the calculus terminates / diverges then the same does the machine, and with a related number of steps. But there is no step-by-step simulation of the calculus by the machine, because the non-deterministic strategy might do a step which is not the one done by the machine. Everything works fine because of the properties of diamond strategies. The general scheme is inspired by Accattoli et al.'s scheme for strong call-by-value [12], but it is here simpler because of the absence of structural equivalence and implosive sharing.

It would also be possible to have a more symmetric setting by either designing a more complex diamond machine, as done by Accattoli and Barenbaum [5], or by designing a deterministic variant of the good strategy. For the sake of simplicity, we prefer the asymmetric setting as to keep a simple machine and reuse the notion of good strategy from the literature.

▶ **Definition 11** (Big-step implementations). *A machine* $\mathtt{M} = (\mathtt{States}, \rightsquigarrow, \cdot \triangleleft \cdot, \underline{\cdot})$ *is a* big-step implementation of a strategy $\to_{\mathtt{str}}$ *on terms when, given a (proper and clash-free) term* t*:*

1. Runs to evaluations*: for any* $\mathtt{M}$*-run* $r : t \triangleleft Q' \rightsquigarrow^* Q$ *there is a* $\to_{\mathtt{str}}$*-evaluation* $e : t \to^*_{\mathtt{str}} \underline{Q}$ *with* $|r|_{\mathtt{pr}} = |e|$.
2. Normalizing evaluations to runs*: if* $e \colon t \to^*_{\mathtt{str}} s$ *with* s $\to_{\mathtt{str}}$*-normal then there is a* $\mathtt{M}$*-run* $r \colon t \triangleleft Q' \rightsquigarrow^* Q$ *such that* $\underline{Q} = s$ *with* $|r|_{\mathtt{pr}} = |e|$.
3. Diverging evaluations to runs*: if* $t \triangleleft Q$ *and* $\to_{\mathtt{str}}$ *diverges on* t *then* $\mathtt{M}$ *diverges on* Q *doing infinitely many principal transitions.*

Next, we isolate sufficient conditions for big-step implementations.

▶ **Definition 12** (Lax implementation system). *A lax implementation system is given by a machine* $\mathtt{M} = (\mathtt{States}, \rightsquigarrow, \cdot \triangleleft \cdot, \underline{\cdot})$ *and a strategy* $\to_{\mathtt{str}}$ *such that for every reachable state* Q*:*

1. Principal projection*: if* $Q \rightsquigarrow_{\mathtt{pr}} Q'$ *then* $\underline{Q} \to_{\mathtt{str}} \underline{Q'}$*;*
2. Search transparency*: if* $Q \rightsquigarrow_{sea} Q'$ *then* $\underline{Q} = \underline{Q'}$*;*
3. Search terminates*:* $\rightsquigarrow_{sea}$ *terminates;*
4. Halt*: if* Q *is final then* $\underline{Q}$ *is* $\to_{\mathtt{str}}$*-normal;*
5. Diamond*:* $\to_{\mathtt{str}}$ *is diamond.*

▶ **Theorem 13** (Abstract big-step implementation). *Let* $\mathtt{M}$ *and* $\to_{\mathtt{str}}$ *form a lax implementation system. Then,* $\mathtt{M}$ *is a big-step implementation of* $\to_{\mathtt{str}}$*.*

Clash-Free and Proper States. Note that we have *not* taken into account properness and clashes on states. One can say that a state Q is proper (resp. clash-free) if its decoding $\underline{Q}$ is. In this way, these notions are trivially seen to be preserved by runs in a lax implementation system. Note, indeed, that transitions are mapped via decoding to either equalities or rewriting steps (and the proofs of these facts shall not need properness nor clash-freeness). Since states are initialized with proper and clash-free terms, and that these notions on terms are preserved by reduction (clash-freeness by definition, for properness see [4]), proper and clash-free states are preserved by transition. Therefore, we shall omit all considerations about properness and clashes for machines.

$$\begin{array}{rrcl} \textsc{States} & Q, Q' & ::= & (E \mid t) \\ \textsc{Read back} & \underline{(E \mid t)} & := & E\langle t\rangle \end{array} \qquad \textsc{Initialization} \quad t \triangleleft Q \quad \text{if} \quad Q = (\langle\cdot\rangle \mid t^{\alpha})$$

Cut Ctx	Active Tm	Tran.	Cut Ctx	Active Tm	
E	$[v{\to}x]t$	$\rightsquigarrow_{sea}$	$E\langle[v{\to}x]\langle\cdot\rangle\rangle$	t	
$E\langle[n{\to}m]E'\rangle$	$[m{\otimes}v, x]t$	$\rightsquigarrow_{\mathtt{axm}_2}$	$E\langle E'\rangle$	$[n{\otimes}v, x]t$	
$E\langle[\lambda y.L\langle v'\rangle{\to}m]E'\rangle$	$[m{\otimes}v, x]t$	$\rightsquigarrow_{\multimap}$	$E\langle E'[v{\to}y]\rangle$	$L\langle v'{\to}x]t\rangle$	
$E\langle[f{\to}e]E'\rangle$	$[e?x]t$	$\rightsquigarrow_{\mathtt{axe}_2}$	$E\langle[f{\to}e]E'\rangle$	$[f?x]t$	
$E\langle[!L\langle v\rangle{\to}e]E'\rangle$	$[e?x]t$	$\rightsquigarrow_{!}$	$E\langle[!L\langle v\rangle{\to}e]E'\rangle$	$L'\langle[v'{\to}x]t\rangle$	#
$E\langle[v_{\mathtt{m}}{\to}m]E'\rangle$	m	$\rightsquigarrow_{\mathtt{axm}_1}$	$E\langle E'\rangle$	$v_{\mathtt{m}}$	
$E\langle[v_{\mathtt{e}}{\to}e]E'\rangle$	e	$\rightsquigarrow_{\mathtt{axe}_1}$	$E\langle[v_{\mathtt{e}}{\to}e]E'\rangle$	$v_{\mathtt{e}}^{\alpha}$	

with $L'\langle v'\rangle = L\langle v\rangle^{\alpha}$

Figure 4 The Basic Abstract Machine (BAM).

5 A Machine for the Closed Basic Case

Here, we study a machine implementing basic evaluation on closed ESC terms. The aim is to give a gentle introduction to some machine concepts.

BAM. The *basic abstract machine* (BAM) is defined in Fig. 4. States $Q = (E \mid t)$ are simply given by the active (pre-)term t and a cut context E which is a list containing the cuts encountered so far by the search mechanism, playing the role of the (global) environment in machines such as Accattoli et al.'s Milner Abstract Machine [6].

The initialization relation $t \triangleleft (\langle\cdot\rangle \mid t^{\alpha})$ pairs (proper and clash-free) terms t with states composed by an empty cut context $\langle\cdot\rangle$ and a well-bound renaming t^{α} of t. The BAM has six principal transitions, mimicking the ESC rewriting rules but for the weakening one (BAM never erases), plus one search transition $\rightsquigarrow_{sea}$, moving cuts from the term to the cut context.

The BAM looks at the topmost constructor of the active term and proceeds applying a transition belonging to one of the following three groups. If the constructor is:

- *Search*: a cut $[v{\to}x]$, the BAM adds it to the cut context, by applying $\rightsquigarrow_{sea}$;
- *Computation*: a subtraction $[m{\otimes}v, x]$ or a dereliction $[e?x]$, then the BAM looks for the associated cut in the cut context, it applies the corresponding cut elimination rule, and goes on to execute the modified active term, by applying $\rightsquigarrow_{\mathtt{axm}_2}$, $\rightsquigarrow_{\multimap}$, $\rightsquigarrow_{\mathtt{axe}_2}$, or $\rightsquigarrow_{!}$;
- *Terminal replacements*: a variable m or e, the BAM looks for the associated cut in the cut context and it applies the matching replacement via $\rightsquigarrow_{\mathtt{axm}_1}$ or $\rightsquigarrow_{\mathtt{axe}_1}$. Note that terminal replacements can only be followed by further terminal replacements, hence the name.

Note that in transitions $\rightsquigarrow_{!}$ and $\rightsquigarrow_{\mathtt{axe}_1}$ some renaming takes place, using names that are fresh with respect to the whole state. The domain $\mathtt{dom}(E)$ of a cut context is defined as $\mathtt{dom}(\langle\cdot\rangle) := \emptyset$ and $\mathtt{dom}([v{\to}x]E) := \{x\} \cup \mathtt{dom}(E)$.

▶ **Lemma 14** (BAM qualitative invariants). *Let $Q = (E \mid t)$ be a BAM reachable state.*

1. Closure*: $\mathtt{fv}(t) \subseteq \mathtt{dom}(E)$ and if $E = E'\langle[v{\to}x]E''\rangle$ then $\mathtt{fv}(v) \subseteq \mathtt{dom}(E')$.*
2. Well-bound*: if $\lambda x.s$, $[e?x]s$, $[m{\otimes}v, x]s$, or $[v{\to}x]s$ occur in Q and x has any other occurrence in Q then it is a free variable of s, and if $E = E'\langle[v{\to}x]E''\rangle$ and x has any other occurrence in Q then it is a free variable in E'' or t.*

▶ **Proposition 15.** *Let Q be a BAM reachable state and $a \in \{\mathtt{axm}_1, \mathtt{axm}_2, \mathtt{axe}_1, \mathtt{axe}_2, \multimap, !\}$.*

1. Search transparency*: if $Q \rightsquigarrow_{sea} Q'$ then $\underline{Q} = \underline{Q'}$.*
2. Principal projection*: if $Q \rightsquigarrow_{a} Q'$ then $\underline{Q} \to_{\mathtt{ba}} \underline{Q'}$.*
3. Search termination*: transition $\rightsquigarrow_{sea}$ terminates.*

$$\begin{array}{rrcl|rrcl} \textsc{Pools} & P,P' & ::= & \epsilon \mid \langle t\rangle_a{:}P & \textsc{Init.} & t \triangleleft Q & \text{if} & Q = (\langle\cdot\rangle_a \mid t^\alpha) \\ \textsc{States} & Q,Q' & ::= & (\mathbb{C} \mid P) & \textsc{Read back} & \underline{(\mathbb{C} \mid P)} & := & \mathbb{C}\langle t_1\rangle_{a_1}\dots\langle t_k\rangle_{a_k} \\ & & & & & & & \text{with } P = \langle t_1\rangle_{a_1}{:}\dots{:}\langle t_k\rangle_{a_k}{:}\epsilon \end{array}$$

MCtx	Pool	Tran.	MCtx	Pool	
$\mathbb{C}$	$\langle [v{\to}x]t\rangle_a{:}P$	$\rightsquigarrow_{sea_1}$	$\mathbb{C}\langle [v{\to}x]\langle\cdot\rangle_a\rangle_a$	$\langle t\rangle_a{:}P$	
$\mathbb{C}$ with $m \notin \mathtt{dom}(\mathbb{C})$	$\langle [m{\otimes}v, x]t\rangle_a{:}P$	$\rightsquigarrow_{sea_2}$	$\mathbb{C}\langle [m{\otimes}\langle\cdot\rangle_b, x]\langle\cdot\rangle_a\rangle_a$	$\langle v\rangle_b{:}\langle t\rangle_a{:}P$	*
$\mathbb{C}$ with $e \notin \mathtt{dom}(\mathbb{C})$	$\langle [e?x]t\rangle_a{:}P$	$\rightsquigarrow_{sea_3}$	$\mathbb{C}\langle [e?x]\langle\cdot\rangle_a\rangle_a$	$\langle t\rangle_a{:}P$	
$\mathbb{C}$	$\langle \lambda x.t\rangle_a{:}P$	$\rightsquigarrow_{sea_4}$	$\mathbb{C}\langle \lambda x.\langle\cdot\rangle_a\rangle_a$	$\langle t\rangle_a{:}P$	
$\mathbb{C}$	$\langle !t\rangle_a{:}P$	$\rightsquigarrow_{sea_5}$	$\mathbb{C}\langle !\langle\cdot\rangle_a\rangle_a$	$\langle t\rangle_a{:}P$	
$\mathbb{C}$ with $x \notin \mathtt{dom}(\mathbb{C})$	$\langle x\rangle_a{:}P$	$\rightsquigarrow_{sea_6}$	$\mathbb{C}\langle x\rangle_a$	P	
$\mathbb{C}\langle [n{\to}m]\mathbb{C}'\rangle_\circ$	$\langle [m{\otimes}v, x]t\rangle_a{:}P$	$\rightsquigarrow_{\mathtt{axm}_2}$	$\mathbb{C}\langle \mathbb{C}'\rangle_\circ$	$\langle [n{\otimes}v, x]t\rangle_a{:}P$	
$\mathbb{C}\langle [\lambda y.L\langle v'\rangle{\to}m]\mathbb{C}'\rangle_\circ$	$\langle [m{\otimes}v, x]t\rangle_a{:}P$	$\rightsquigarrow_{-\circ}$	$\mathbb{C}\langle \mathbb{C}'\rangle_\circ$	$\langle [v{\to}y]L\langle [v'{\to}x]t\rangle\rangle_a{:}P$	
$\mathbb{C}\langle [f{\to}e]\mathbb{C}'\rangle_\circ$	$\langle [e?x]t\rangle_a{:}P$	$\rightsquigarrow_{\mathtt{axe}_2}$	$\mathbb{C}\langle [f{\to}e]\mathbb{C}'\rangle_\circ$	$\langle [f?x]t\rangle_a{:}P$	
$\mathbb{C}\langle [!L\langle v\rangle{\to}e]\mathbb{C}'\rangle_\circ$	$\langle [e?x]t\rangle_a{:}P$	$\rightsquigarrow_!$	$\mathbb{C}\langle [!L\langle v\rangle{\to}e]\mathbb{C}'\rangle_\circ$	$\langle L'\langle [v'{\to}x]t\rangle\rangle_a{:}P$	#
$\mathbb{C}\langle [v_{\mathtt{m}}{\to}m]\mathbb{C}'\rangle_\circ$	$\langle m\rangle_a{:}P$	$\rightsquigarrow_{\mathtt{axm}_1}$	$\mathbb{C}\langle \mathbb{C}'\rangle_\circ$	$\langle v_{\mathtt{m}}\rangle_a{:}P$	
$\mathbb{C}\langle [v_{\mathtt{e}}{\to}e]\mathbb{C}'\rangle_\circ$	$\langle e\rangle_a{:}P$	$\rightsquigarrow_{\mathtt{axe}_1}$	$\mathbb{C}\langle [v_{\mathtt{e}}{\to}e]\mathbb{C}'\rangle_\circ$	$\langle v_{\mathtt{e}}^\alpha\rangle_a{:}P$	

* b is fresh. # with $L'\langle v'\rangle = L\langle v\rangle^\alpha$

Figure 5 The Strong Exponential Substitution Abstract Machine without Erasure (SESAME).

4. Halt*: if Q is final then $Q = (E \mid v)$ with v not a variable, and $\underline{Q}$ is normal for non-erasing basic evaluation.*

▶ **Theorem 16.** *The BAM implements ESC non-erasing basic evaluation on closed terms.*

6 SESAME

In this section, we extend the BAM as to perform the analogous of strong evaluation, that is, as to perform cut-elimination also inside values, and implement the good strategy. The obtained machine shall ignore garbage collection and – when it terminates – it returns a term that is cut-free *up to garbage* (Def. 8). Garbage collection shall be addressed in Sect. 8.

Pools and Jobs. The *Strong Exponential Substitution Abstract Machine without Erasure* (SESAME), defined in Fig. 5, relies on a technique for strong evaluation recently introduced by Accattoli and Barenbaum [5], which we are now going to explain.

When evaluation goes under binders, the closure invariant of the basic case (Lemma 14.1), for which free variables of the active term are associated to a cut in the cut context, is lost. Thus, when the active term is a subtraction $[m{\otimes}v, x]t$ with no associated cut for m, the subtraction is kept, and the machine has to evaluate v and t. Now, the evaluations of v and t cannot affect each other, they are independent, but one of the two sub-terms has to be evaluated first, say v. Usually, strong machines (such as Crégut's [26]) would run through v, and if such process does terminate, producing a value v', then they *backtrack* to the subtraction generating the fork by *moving sequentially* through v', and then start evaluate t.

Accattoli and Barenbaum's technique simply circumvents the sequential backtracking process by directly jumping back to the forking point. For that, the machine is equipped with a *pool* P of *jobs*, each one paired to a unique name a. In our example, there would be a job $\langle v\rangle_a$ for v and a job $\langle t\rangle_b$ for t. The idea is that when one job ends then the machine jumps to the next one, *without* moving through the structure of the finished job.

In [5], the pool is any data structure satisfying a certain interface. The idea is that different data structures implementing the interface realize different job scheduling policies, compactly accounting for different strong strategies within the same framework. We here omit this abstract aspect and fix the pool to be the simplest such structure, namely a LIFO list of named jobs. In the λ-calculus, LIFO list pools implement leftmost evaluation [5].

Another simplification with respect to [5] is that here jobs are simply named terms, while in [5] they are named pairs of a term and an applicative stack. The change induces a simpler notion of read back. This difference is not a design choice, it is simply induced by having a machine for sequent calculus terms (here) rather than natural deduction terms (in [5]).

A crucial point of SESAME is that job forking happens on subtractions only, that is, it does not happen on cuts $[v{\to}x]t$: the machine ignores v and goes straight to evaluate t. This happens in particular to prevent the breaking of the sub-term property.

We write $\mathtt{names}(P)$ for the set of names associated to the jobs in P, that is, if $P = \langle t_1\rangle_{a_1}{:}\ldots{:}\langle t_k\rangle_{a_k}$ then $\mathtt{names}(P) := \{a_1, \ldots, a_k\}$.

Multi-Contexts and Approximants. Another aspect of the technique in [5] is that the parts of the term that have been evaluated and that shall not be touched again by the machine – sometimes referred to as *stable parts* – are accumulated in the *approximant* $\mathbb{A}$ (of the normal form), which is a multi-context (that is, a context with possibly many holes, possibly none). The idea is that the name a of a job $\langle t\rangle_a$ in the pool is associated to a (unique) named hole $\langle\cdot\rangle_a$ in $\mathbb{A}$, and that, whenever a stable piece of term is produced by the job $\langle t\rangle_a$, that piece is moved to $\langle\cdot\rangle_a$ in $\mathbb{A}$, incrementally building the normal form.

We follow this pattern from [5], but our setting induces a few differences. Firstly, in [5] the machine has also a global environment, akin to the cut context of the BAM. Here, we include the environment/cut context *into* the approximant, having only one data structure. Thus, the approximant shall have cuts, but these cuts bind variables that have non-garbage occurrences only in the active term, i.e., only out of the approximant itself, where instead they have only garbage occurrences. At the end of a complete run, the approximant shall be normal for the non-erasing good strategy, i.e. it shall be cut-free up to garbage (Def. 8).

Secondly, in [5] the machine is defined using approximants, while here we define it using the weaker notion of multi-contexts, and then prove invariants guaranteeing that the multi-contexts of reachable states are approximants.

▶ **Definition 17** (Multi-contexts and approximants). *A* (named) multi-context $\mathbb{C}$ *is an ESC term in which there might be occurrences of holes* $\langle\cdot\rangle_a$ *indexed with names, as follows:*

$$\begin{array}{l|l} \textsc{Names} \quad a, b, c, a_1, b_2, \ldots & \textsc{Value multi-ctxs} \quad \mathbb{V} ::= \langle\cdot\rangle_a \mid x \mid \lambda x.\mathbb{C} \mid\, !\mathbb{C} \\ & \textsc{Multi-ctxs} \quad \mathbb{C} ::= \mathbb{V} \mid [\mathbb{V}{\to}x]\mathbb{C} \mid [m \otimes \mathbb{V}, x]\mathbb{C} \mid [e?x]\mathbb{C} \end{array}$$

The plugging $\mathbb{C}\langle\mathbb{C}'\rangle_a$ *of* $\mathbb{C}'$ *on* a *in* $\mathbb{C}$*, is the capture-allowing substitution of* $\langle\cdot\rangle_a$ *by* $\mathbb{C}'$ *in* $\mathbb{C}$*. We write* $\mathbb{C}\langle\mathbb{C}'\rangle_\circ$ *for when* $\circ$ *is an irrelevant name occurring exactly once in* $\mathbb{C}$*; this notation is meant to be used for decomposing a multi-context in two, as in* $\mathbb{C} = \mathbb{C}'\langle\mathbb{C}''\rangle_\circ$*. We write* $\mathtt{names}(\mathbb{C})$ *for the set of names that occur in* $\mathbb{C}$*. Out variables and out cuts extend to multi-contexts as expected.*

The domain $\mathtt{dom}(\mathbb{C}) = \{x_1, \ldots, x_n\}$ *of a multi-context* $\mathbb{C}$ *contains the variables on which there is a cut in* $\mathbb{C}$ *(the formal definition is in Appendix E of the tech report [11]).*

An approximant $\mathbb{A}$ *is a multi-context such that:*

1. Unique names*: every name* $a \in \mathtt{names}(\mathbb{A})$ *has exactly one occurrence in* $\mathbb{A}$*;*
2. Out cuts are hereditary garbage and hole-free*: for every out cut* $[\mathbb{V}{\to}x]\mathbb{C}'$ *in* $\mathbb{A}$*, if* $x \in \mathtt{fv}(\mathbb{C}')$ *then* x *occurs only inside cut values, i.e.* $x \notin \mathtt{ov}(\mathbb{C}')$*, and* $\mathbb{V}$ *is a term.*

Note that a multi-context $\mathbb{C}$ without holes is simply a term, thus the defined notion of plugging subsumes the plugging $\mathbb{C}\langle t\rangle_a$ of terms in multi-contexts.

Transitions. The principal transition of SESAME are as for the BAM up to the generalization of the cut context E to a multi-context $\mathbb{C}$. Note the mute name $\circ$ for decomposing the multi-context in the transitions. Clearly, the name $\circ$ on the LHS and the RHS of each transition is the same. SESAME has a search transition for each constructor of the calculus. Note that transitions $\rightsquigarrow_{\mathtt{axm}_1}$ and $\rightsquigarrow_{\mathtt{axe}_1}$ are no longer *terminal*, because now they might be followed by a search transition, after which the run might jump to a different job.

Read Back. The read back $\underline{Q}$ of a state $Q = (\mathbb{C} \mid P)$ is defined in Fig. 5 and simply plugs each job $\langle t \rangle_a$ of the pool P in the hole of name a of the multi context $\mathbb{C}$. We also write $\mathbb{C}\langle P \rangle$ for $\underline{Q}$. Since the holes of $\mathbb{C}$ are all independent and jobs contains terms (with no holes), we have that $\mathbb{C}\langle P \rangle = \mathbb{C}\langle P' \rangle$ for any pool P' obtained by permuting the elements of P.

Example of SESAME run. As an example, we show a SESAME run on the following term:

$$t \quad := \quad [!\lambda m_1.m_1 \to e_1][e_1 ? m_2][e_1 ? m_3][m_2 \ominus m_3, m_4]m_4$$

that just applies an identity function to itself, by making two copies of $!\lambda m_1.m_1$ and applying one to the other via the subtraction. To fit it into the margins, we use the abbreviation $s := [m_2 \ominus m_3, m_4]m_4$.

Tr.	MCtx	Pool
	$\langle \cdot \rangle_a$	$\langle t \rangle_a : \epsilon$
$\rightsquigarrow_{sea_1}$	$[!\lambda m_1.m_1 \to e_1]\langle \cdot \rangle_a$	$\langle [e_1 ? m_2][e_1 ? m_3]s \rangle_a : \epsilon$
$\rightsquigarrow_{!}$	$[!\lambda m_1.m_1 \to e_1]\langle \cdot \rangle_a$	$\langle [\lambda m_5.m_5 \to m_2][e_1 ? m_3]s \rangle_a : \epsilon$
$\rightsquigarrow_{sea_1}$	$[!\lambda m_1.m_1 \to e_1][\lambda m_5.m_5 \to m_2]\langle \cdot \rangle_a$	$\langle [e_1 ? m_3]s \rangle_a : \epsilon$
$\rightsquigarrow_{!}$	$[!\lambda m_1.m_1 \to e_1][\lambda m_5.m_5 \to m_2]\langle \cdot \rangle_a$	$\langle [\lambda m_6.m_6 \to m_3]s \rangle_a : \epsilon$
$\rightsquigarrow_{sea_1}$	$[!\lambda m_1.m_1 \to e_1][\lambda m_5.m_5 \to m_2][\lambda m_6.m_6 \to m_3]\langle \cdot \rangle_a$	$\langle s \rangle_a : \epsilon$
$\rightsquigarrow_{\multimap}$	$[!\lambda m_1.m_1 \to e_1][\lambda m_6.m_6 \to m_3]\langle \cdot \rangle_a$	$\langle [m_3 \to m_5][m_5 \to m_4]m_4 \rangle_a : \epsilon$
$\rightsquigarrow_{sea_1}$	$[!\lambda m_1.m_1 \to e_1][\lambda m_6.m_6 \to m_3][m_3 \to m_5]\langle \cdot \rangle_a$	$\langle [m_5 \to m_4]m_4 \rangle_a : \epsilon$
$\rightsquigarrow_{sea_1}$	$[!\lambda m_1.m_1 \to e_1][\lambda m_6.m_6 \to m_3][m_3 \to m_5][m_5 \to m_4]\langle \cdot \rangle_a$	$\langle m_4 \rangle_a : \epsilon$
$\rightsquigarrow_{\mathtt{axm}_1}$	$[!\lambda m_1.m_1 \to e_1][\lambda m_6.m_6 \to m_3][m_3 \to m_5]\langle \cdot \rangle_a$	$\langle m_5 \rangle_a : \epsilon$
$\rightsquigarrow_{\mathtt{axm}_1}$	$[!\lambda m_1.m_1 \to e_1][\lambda m_6.m_6 \to m_3]\langle \cdot \rangle_a$	$\langle m_3 \rangle_a : \epsilon$
$\rightsquigarrow_{\mathtt{axm}_1}$	$[!\lambda m_1.m_1 \to e_1]\langle \cdot \rangle_a$	$\langle \lambda m_6.m_6 \rangle_a : \epsilon$
$\rightsquigarrow_{sea_4}$	$[!\lambda m_1.m_1 \to e_1]\lambda m_6.\langle \cdot \rangle_a$	$\langle m_6 \rangle_a : \epsilon$
$\rightsquigarrow_{sea_6}$	$[!\lambda m_1.m_1 \to e_1]\lambda m_6.m_6$	ϵ

Note that in the last state the cut $[!\lambda m_1.m_1 \to e_1]$ is garbage but it is not removed. This shall be taken care of by an extra garbage collection phase described in Sect. 8.

7 Analysis of SESAME

Invariants. As for the BAM, we prove some invariants. The first one states that the multi context of a reachable state is an approximant. A second invariant states that names in the multi context are exactly those in the pool, where they have exactly one occurrence each. Then, we need a well-bound invariant about binders, relying on a technical definition in Appendix E of the tech report [11], along the lines of the invariant of the BAM. Lastly, the contextual decoding invariant, the most important and sophisticated invariant, states that a reachable state less the first job reads back to a good context. For the invariant to hold, the statement has to be generalized in a technical way, analogously to similar invariants in [5].

▶ **Lemma 18** (SESAME qualitative invariants). *Let $Q = (\mathbb{C} \mid P)$ be a SESAME state reachable from a well-bound initial term t_0.*

1. Approximant*: $\mathbb{C}$ is an approximant.*
2. Names*: jobs in the pool P have pairwise distinct names, and $\mathtt{names}(\mathbb{C}) = \mathtt{names}(P)$.*
3. Well-bound*: Q is a well-bound state.*
4. Contextual decoding*: if $P = \langle t_1\rangle_{a_1} : \ldots : \langle t_k\rangle_{a_k}$ has length $k \geq 1$ then:*

$$\mathbb{C}^Q_{s_1,\ldots,s_{i-1}|a_i|s_{i+1},\ldots,s_k} := \mathbb{C}\langle s_1\rangle_{a_1} \ldots \langle s_{i-1}\rangle_{a_{i-1}} \langle\langle\cdot\rangle\rangle_{a_i} \langle s_{i+1}\rangle_{a_{i+1}} \ldots \langle s_k\rangle_{a_k}$$

is a good context for $i \in \{1,\ldots,k\}$ and any terms $s_1,\ldots,s_{i-1},s_{i+1},\ldots,s_k$ such that $\mathbb{C}^Q_{s_1,\ldots,s_{i-1}|a_i|s_{i+1},\ldots,s_k}$ is a term.

Implementation Theorem. The well-bound and contextual decoding invariants are used to prove principal projection. The approximant and names invariants are used to prove the halt property. Search transparency and termination are straightforward. Then we apply the abstract implementation theorem (Theorem 13), and obtain the implementation theorem.

▶ **Proposition 19.** *Let $a \in \{\mathtt{axm}_1, \mathtt{axm}_2, \mathtt{axe}_1, \mathtt{axe}_2, \multimap, !\}$.*

1. Search transparency*: if $Q \rightsquigarrow_{sea} Q'$ then $\underline{Q} = \underline{Q'}$.*
2. Principal projection*: if $Q \rightsquigarrow_a Q'$ then $\underline{Q} \to_{\mathtt{G}a} \underline{Q'}$.*
3. Search termination*: transition $\rightsquigarrow_{sea}$ terminates.*
4. Halt*: if Q is final then $Q = (t \mid \epsilon)$ and $\underline{Q} = t$ is cut-free up to garbage.*

▶ **Theorem 20** (SESAME is good). *SESAME is a big-step implementation of the non-erasing good strategy $\to_{\mathtt{G}_{\neg\mathtt{w}}}$ of ESC.*

Complexity Analysis

The complexity analysis of SESAME amounts to bound the cost of implementing a run $r : t \triangleleft Q \rightsquigarrow^*_{\mathtt{SESAME}} Q'$ on random access machines (RAMs) as a function of two parameters: the number $|r|_{\mathtt{pr}}$ of principal transitions in r (which are in bijection with the number of $\to_{\mathtt{G}_{\neg\mathtt{w}}}$ steps) and the size $|t|$ of the initial term/state t/Q.

Sub-Term Property. The analysis relies on the *sub-term property*, ensuring that the duplicating principal transitions $\rightsquigarrow_!$ and $\rightsquigarrow_{\mathtt{axe}_1}$ manipulate only sub-terms of the initial term. Therefore, the cost of duplications is connected to size of the initial term. The property for SESAME can be inferred by the one for the good strategy, but we prefer to give a direct simple proof. The statement of the related invariant is about cut values, not duplications, but note that such values are the only terms duplicated by SESAME.

▶ **Lemma 21** (Sub-term). *Let $r\colon t \triangleleft Q \rightsquigarrow^*_{\mathtt{SESAME}} (\mathbb{C} \mid P)$ be a SESAME run.*

1. Invariant*: if v is a cut value in $\mathbb{C}$ or a value in P then $|v| \leq |t|$.*
2. Property*: if v is a value duplicated along r (by $\rightsquigarrow_!$ or $\rightsquigarrow_{\mathtt{axe}_1}$) then $|v| \leq |t|$.*

Since search transitions decrease the number of term constructors in the pool, which is only increased by principal transitions and of a quantity bounded by $|t|$ (by the sub-term invariant), we obtain the following bound.

▶ **Lemma 22** (Search transitions are bi-linear in number). *Let $r\colon t \triangleleft Q \rightsquigarrow^*_{\mathtt{SESAME}} Q'$ be a SESAME run. Then $|r|_{\mathtt{sea}} \leq |t| \cdot (|r|_{\mathtt{pr}} + 1)$.*

Cost of Single Transitions. For bounding the total cost, we need to make some hypotheses on how SESAME is going to be itself implemented on RAMs:

1. *Variable occurrences, binders, and cuts*: a variable is a memory location, a variable occurrence is a reference to it, and a cut $[v{\to}x]$ is the fact that the location associated with x contains v;
2. *Random access to variables*: the cuts in $\mathbb{C}$ can be accessed in $\mathcal{O}(1)$ by just following the reference given by the variable occurrence, with no need to search through $\mathbb{C}$;
3. *Named holes and jobs*: a named hole $\langle\cdot\rangle_a$ is again a memory location a, and a job $\langle t\rangle_a$ is a pointer to a and expresses the fact that location a contains t;
4. *Sequences of left rules*: cuts have a back pointer to the constructors before them (for instance in $[e?m][v{\to}x][m{\otimes}v', y]t$ the cut $[v{\to}x]$ has a pointer to $[e?m]$) so that the removal of a cut from the multi context $\mathbb{C}$ of a state – needed for the multiplicative transitions $\rightsquigarrow_{\mathtt{axm}_1}$, $\rightsquigarrow_{\mathtt{axm}_2}$, and $\rightsquigarrow_{\multimap}$ – can be performed in $\mathcal{O}(1)$.

As it is standard for time analyses, we also assume that pointers can be managed in $\mathcal{O}(1)$. These hypotheses mimic similar ones behind machines for λ-calculi, which are shown to be implementable in OCaml by Accattoli and Barras [8] and Accattoli et al. [12], and are followed by the OCaml implementation outlines in Sect. 9. They allow us to consider search transitions as having constant cost, and principal transitions as having cost bound by the initial term, by the sub-term property.

▶ **Lemma 23** (Cost of single transitions). *Let $r\colon t \triangleleft Q \rightsquigarrow^*_{\mathtt{SESAME}} Q'$ be a SESAME run. Search (resp. principal) transitions of r are implementable in $\mathcal{O}(1)$ (resp. $\mathcal{O}(|t|)$).*

Summing Up. By putting together the bounds on the number of search transitions with the cost of single transitions we obtain the complexity of SESAME.

▶ **Theorem 24** (SESAME bi-linear overhead bound). *Let $r\colon t \triangleleft Q \rightsquigarrow^*_{\mathtt{SESAME}} Q'$ be a SESAME run. Then r is implementable on RAMs in $\mathcal{O}\left(|t| \cdot (|r|_{\mathtt{pr}} + 1)\right)$.*

8 Final Garbage Collection and Full Cut Elimination

By the halt property (Proposition 19.4), SESAME stops on final state $Q = (\mathbb{A} \mid \epsilon)$ where $\mathbb{A}$ is a (pre-)term t that is cut-free *up to garbage*. The final garbage collection process turning t into a cut-free proof term $\mathtt{GC}(t)$ is formalized as the following function:

Final garbage Collection

$$\begin{array}{lll} \mathtt{GC}(x) := x & \mathtt{GC}(\lambda x.t) := \lambda x.\mathtt{GC}(t) & \mathtt{GC}(!t) := !\mathtt{GC}(t) \\ \mathtt{GC}([v{\to}x]t) := \mathtt{GC}(t) & \mathtt{GC}([m{\otimes}v, x]t) := [m{\otimes}\mathtt{GC}(v), x]\mathtt{GC}(t) & \mathtt{GC}([e?x]t) := [e?x]\mathtt{GC}(t) \end{array}$$

Clearly, $\mathtt{GC}(t)$ is cut-free. The following proposition states the qualitative and quantitative properties of garbage collection. Its first point uses the lemma before it, while the second point rests on the SESAME bi-linear bound.

▶ **Lemma 25.** *Let t be a term that is cut-free up to garbage but not cut-free. Then in t there is at least an out cut that is a $\to_{\mathtt{Gw}}$-redex.*

▶ **Proposition 26.** *Let $r : t \triangleleft Q \rightsquigarrow^*_{\mathtt{SESAME}} (s \mid \epsilon)$ a SESAME run ending on a final state.*

1. Good weakening steps simulate GC*: $s \to^k_{\mathtt{Gw}} \mathtt{GC}(s)$ where k is the number of out cuts in s;*
2. GC is bi-linear*: the final garbage collection function $s \mapsto \mathtt{GC}(s)$ can be implemented on RAMs in $\mathcal{O}(|t| \cdot (|r|_{\mathtt{pr}} + 1))$.*

By simulating good cut-elimination via the SESAME followed by final garbage collection, we obtain our main result, which – beyond the bi-linear bound – is a further slight (but sort of obvious) refinement of Accattoli's result, as it allows one to count only *non-erasing* good steps, rather than general good steps.

▶ **Theorem 27** (Good cut-elimination is big-step implementable with bi-linear overhead)**.** *Let $e : t \to_{\mathtt{G}}^{*} s$ be a good evaluation sequence with s cut-free. Then s is computable from t on RAMs in $\mathcal{O}(|t| \cdot (|e|_{\neg \mathtt{w}} + 1))$.*

Proof. If $e : t \to_{\mathtt{G}}^{*} s$ with s cut-free then $\to_{\mathtt{G}_{\neg \mathtt{w}}}$ terminates, by the *uniform normalization* corollary of the diamond property for $\to_{\mathtt{G}}$ (see the end of Sect. 3). Then there is an evaluation $e' : t \to_{\mathtt{G}_{\neg \mathtt{w}}}^{*} s'$ with s' cut-free up to garbage. Since SESAME is a big-step implementation of $\to_{\mathtt{G}_{\neg \mathtt{w}}}$ (Theorem 20), we obtain a SESAME run $r : t \triangleleft Q \rightsquigarrow_{\mathtt{SESAME}}^{*} Q'$ with $\underline{Q'} = s'$, of cost $\mathcal{O}\left(|t| \cdot (|r|_{\mathtt{pr}} + 1)\right) = \mathcal{O}(|t| \cdot (|e'| + 1))$. By the halt property of SESAME and cut-freeness up to garbage of SESAME, we obtain that Q' is final, that is, $Q' = (u \mid \epsilon)$ with $u =_{\alpha} s'$.

Since good weakening steps simulate GC (Proposition 26.1), we obtain $e'' : u \to_{\mathtt{Gw}}^{*} \mathtt{GC}(s')$ with $\mathtt{GC}(s')$ cut-free. Concatenating e' and e'' we obtain a good and normalizing evaluation sequence $e''' : t \to_{\mathtt{G}_{\neg \mathtt{w}}}^{*} \to_{\mathtt{Gw}}^{*} \mathtt{GC}(s')$. The diamond property of the good strategy (Theorem 10.2) implies *confluence* and *length invariance* of $\to_{\mathtt{G}}$ (see the end of Sect. 3), that is, we obtain $s = \mathtt{GC}(s')$ and $|e| = |e'''|$. Since computing GC is bi-linear (Proposition 26.1), and precisely costs again $\mathcal{O}\left(|t| \cdot (|r|_{\mathtt{pr}} + 1)\right) = \mathcal{O}(|t| \cdot (|e'| + 1))$, we obtain that the total cost of SESAME followed by GC is $2 \cdot \mathcal{O}(|t| \cdot (|e'| + 1)) = \mathcal{O}(|t| \cdot (|e'| + 1))$.

Lastly, the diamond property of the good strategy (Theorem 10.2) as stated in [4] does not give information about the kinds of steps, but by looking at its proof it is easily seen that diamond diagrams preserve the kind of steps. Therefore, not only we have $|e| = |e'''|$, but also $|e|_{\neg \mathtt{w}} = |e'''|_{\neg \mathtt{w}}$. Finally, note that $|e'''|_{\neg \mathtt{w}} = |e'|$, so the cost actually is $\mathcal{O}(|t| \cdot (|e|_{\neg \mathtt{w}} + 1))$. ◀

9 Implementation in OCaml

An implementation in OCaml can be found on GitHub at `https://github.com/sacerdot/sesame/`. All the datatypes and functions of the implementation are documented at `https://sacerdot.github.io/sesame`.

Aim and Design. The implementation is provided to support evidence for the cost of transitions claimed by Lemma 23. Moreover, it allows one to easily study the computational behaviour of family of terms by observing their evaluation.

The implementation has *not* been heavily optimized, in order to keep the code readable and close to the pen-and-paper presentation. Nevertheless, the employed data structures are reasonably close to those of an optimized implementation. Advanced recent OCaml features, like Generalized Algebraic Data Types, could have been used to statically enforce more invariants. We sticked instead to a simpler subset of OCaml, to make the code readable to non experts of the language. As a consequence, in several places there are assertions to abort the program in case the invariants are violated (possible only in case of bugs).

Overview. The implementation consists of a Read-Eval-Print-Loop (REPL) that asks the user to enter an IMELL term to be reduced, in ESC syntax. The accepted BNF is printed by the executable before starting the REPL. The term is first checked to be proper, then its strong normal form up to garbage is computed by the SESAME machine and finally garbage is removed. All intermediate machine steps are shown.

Before starting the REPL, a few test terms are reduced and printed (among which the example at the end of Sect. 6), to check the machine functionalities and as examples of the input syntax. In particular, the last tests are terms from the exploding family discussed in the next section.

Data Structures. Terms are encoded at runtime as term graphs, which are Direct Acyclic Graphs (DAG) augmented with a few back edges. Nodes in the graph are given by constructors of Algebraic Data Types whose arguments are mutable, to imperatively change the graph during reduction. All type declarations are given in the `termGraphs.ml` file. The data structures used in the implementation are described in Appendix H of the tech report [11].

Machine Runs: Auxiliary Functions and Their Complexity. The code that implements the runs of SESAME to normal form and the final garbage collection process can be found in `reduction.ml`, which is independent from all the remaining files but `termGraphs.ml`. Both files together amount to 412 lines of commented OCaml code.

A quick inspection of the code in `reduction.ml` shows only a few non-trivial functions, all the others being constant time:

1. `copy_{term,value,var,bvar}` that copies the DAG in input while visiting it. A $\mathcal{O}(1)$ fresh name generator is used to assign a new `name` value to copies of variables.
2. `alpha` and `enter_bo`, used respectively in the ! and $\multimap$ transitions, that take the body of a promotion/abstraction, traverse it to split it into a left context and a value, and build a new term by glueing together the obtained left context, a new cut that uses the value, and a remaining term. The two functions differ only in the fact that `alpha` also copies (α-renames) the term in input, calling the `copy_*` functions on the sub-terms, while `enter_bo` consumes the given term.
3. `steps`, the SESAME main loop, that runs until the pool is empty and a normal form is therefore reached, or it diverges otherwise.
4. `gc_{value,term}` that traverse in linear time the input to remove all garbage cuts from a normal form.

Linearity for all the previous functions but `steps` is easily established observing that the functions are based on visits of DAGs that never visit a node twice. In particular, the functions of Point 2 are linear in the size of the given term, which, by the sub-term invariant, is a (copy of a) sub-term of the initial term.

The only major source of technicalities – which however do not affect the complexity – is the fact that multiplicative transitions remove the acting cut from the multi context. For example, consider a multiplicative step $[e?f][v{\to}m][v'{\to}y]t \to [e?f][v'{\to}y]t'$ involving the cut on m. To implement it, beyond manipulating v, t, and t', one also has to connect $[e?f]$ and $[v'{\to}y]$. For that, cuts have a back-pointer to the preceding constructor (as mentioned at Point 4, labeled *sequences of left rule*, before Lemma 23), which induces inelegant imperative manipulations of the term graph, in particular in the copy function. The back-pointer on $[v{\to}m]$ is used to retrieve the dereliction $[e?f]$ from $[v{\to}m]$, while making the new connection of $[e?f]$ and $[v'{\to}y]$ requires changing two pointers: the one from the dereliction to its body, which has to target $[v'{\to}y]$, plus the back-pointer from $[v'{\to}y]$, which now has to target $[e?f]$.

The general scheme for addressing this issue is to augment the input t of functions that take a term with the pointer to the parent node of t, in order to be able to reassign the back-pointer when t is a cut.

Parsing, Pretty Printing, and Check for Properness. We have not attempted to achieve the best asymptotic costs for these functionalities since they are not relevant for the paper and since in practice we are manipulating small terms in input. To keep the code reproducible in the long term, we have not used any external library or tool for parsing and pretty-printing. The printed version of the example run at the end of Sect. 6 is shown in Appendix I of the tech report [11].

10 What's Next: an Interesting Family of Terms

We here hint at what we consider the most interesting future work, already mentioned by Accattoli in the conclusions of [4]. The difference is that here we provide a family of terms showing that such a future work is challenging.

The Question. *One* β-step of the λ-calculus is simulated in IMELL by *one* multiplicative step (actually a $\to_{\_\circ}$ step) followed by possibly *many* exponential steps. Since for many strategies the number of β-steps is a polynomial cost model, it means that one can count only the number of multiplicative (or even $\to_{\_\circ}$) steps, that is, *one can count zero for exponential steps*. Such a surprising fact is established easily in the case of weak evaluation with closed terms (roughly, it can be proved via standard abstract machines), while for strong evaluation it requires a sophisticated additional technique called *useful sharing* [15, 12, 20]. A question naturally arises: *is the number of ESC multiplicative/*$\to_{\_\circ}$ *good steps a polynomial time cost model as well?* This question – left to future work – is far from obvious. In the λ-calculus, there is a strong, hardcoded correlation between multiplicatives and exponentials, not present in IMELL. In the standard call-by-name/value encodings of λ-calculus in IMELL, indeed, multiplicatives and exponentials connectives *rigidly alternate*, while IMELL also has consecutive exponentials, as in $!!A$, enabling wilder exponential behaviour. A hint that that question might have a positive answer is given by the strong normalization of untyped exponentials in ESC/IMELL shown in [4] (while they are *not* SN in MELL, see [4]).

The Question is Challenging. We here show that the question mentioned above is challenging, and – surprisingly – it is already challenging in the apparently simple setting of basic evaluation with closed terms, that is the ESC analogous of weak evaluation for the λ-calculus, for which instead the question has an easy answer. The challenging aspect is here shown by building a family of ESC terms σ_n having size linear in n, whose evaluation is basic and made out of exponential steps only, and of a number of steps that is exponential in n. This is impossible in the λ-calculus, and it is crucially related to iterated exponential terms such as $!!t$. Since the evaluation of σ_n uses 0 multiplicative steps, it suggests that the number of multiplicative steps is *not* a polynomial cost model. More precisely, it shows that a smart additional mechanism – akin to useful sharing – is needed in order to circumvent the exponential number of exponential steps. Useful sharing as it appears in the literature, however, cannot be the answer, since useful sharing addresses exponential inefficiencies that are induced by strong evaluation, which is an orthogonal issue.

The Exponential Exploding Family. The family is built in three steps.

1. For $k \geq 1$, define $\pi_k := \overbrace{[f?\_]\dots[f?\_][f?e]}^{k} e$, where $\_$ is an exponential variable with no occurrences, and whose name is irrelevant. These terms can be composed as to reduce to (the cut-free term) $\pi_{k\cdot h}$ via exponential basic evaluation, that is, $[!\pi_k \to f]\pi_h \to_{\mathsf{e}}^* \pi_{k\cdot h}$.

2. Define $\delta_n := \overbrace{[!\pi_2 \to f] \dots [!\pi_2 \to f]}^{n-1} \pi_2$, and note that, by Point 1, one has $\delta_n \to_{\mathsf{e}}^* \pi_{2^n}$.
3. Finally, define the exponential exploding family as $\sigma_n := [!!\lambda m.m \to f]\delta_n$. By Point 2, we have $\sigma_n \to_{\mathsf{e}}^* [!!\lambda m.m \to f]\pi_{2^n}$. Then note that $[!!\lambda m.m \to f]\pi_{2^n} \to_{\mathsf{e}}^{\Omega(2^n)} !\lambda m.m$.

Summing up, σ_n is an example of term of size $\mathcal{O}(n)$ that reduces in $\Omega(2^n)$ exponential steps and 0 multiplicative steps to a cut-free proof using only basic evaluation. As mentioned in the previous section, the OCaml implementation starts by running some tests, including examples of the three points above, respectively for $k = 3, h = 4$, for $n = 3$, and for $n = 3$.

References

1 Samson Abramsky. Computational interpretations of linear logic. *Theor. Comput. Sci.*, 111(1&2):3–57, 1993. doi:10.1016/0304-3975(93)90181-R.

2 Beniamino Accattoli. An abstract factorization theorem for explicit substitutions. In Ashish Tiwari, editor, *23rd International Conference on Rewriting Techniques and Applications (RTA'12) , RTA 2012, May 28 - June 2, 2012, Nagoya, Japan*, volume 15 of *LIPIcs*, pages 6–21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPICS.RTA.2012.6.

3 Beniamino Accattoli. The useful MAM, a reasonable implementation of the strong λ-calculus. In *Logic, Language, Information, and Computation - 23rd International Workshop, WoLLIC 2016, Puebla, Mexico, August 16-19th, 2016. Proceedings*, pages 1–21, 2016. doi:10.1007/978-3-662-52921-8_1.

4 Beniamino Accattoli. Exponentials as substitutions and the cost of cut elimination in linear logic. *Log. Methods Comput. Sci.*, 19(4), 2023. doi:10.46298/LMCS-19(4:23)2023.

5 Beniamino Accattoli and Pablo Barenbaum. A diamond machine for strong evaluation. In Chung-Kil Hur, editor, *Programming Languages and Systems - 21st Asian Symposium, APLAS 2023, Taipei, Taiwan, November 26-29, 2023, Proceedings*, volume 14405 of *Lecture Notes in Computer Science*, pages 69–90. Springer, 2023. doi:10.1007/978-981-99-8311-7_4.

6 Beniamino Accattoli, Pablo Barenbaum, and Damiano Mazza. Distilling abstract machines. In Johan Jeuring and Manuel M. T. Chakravarty, editors, *Proceedings of the 19th ACM SIGPLAN international conference on Functional programming, Gothenburg, Sweden, September 1-3, 2014*, pages 363–376. ACM, 2014. doi:10.1145/2628136.2628154.

7 Beniamino Accattoli, Pablo Barenbaum, and Damiano Mazza. A strong distillery. *CoRR*, abs/1509.00996, 2015. arXiv:1509.00996.

8 Beniamino Accattoli and Bruno Barras. Environments and the complexity of abstract machines. In Wim Vanhoof and Brigitte Pientka, editors, *Proceedings of the 19th International Symposium on Principles and Practice of Declarative Programming, Namur, Belgium, October 09 - 11, 2017*, pages 4–16. ACM, 2017. doi:10.1145/3131851.3131855.

9 Beniamino Accattoli, Eduardo Bonelli, Delia Kesner, and Carlos Lombardi. A nonstandard standardization theorem. In Suresh Jagannathan and Peter Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 659–670. ACM, 2014. doi:10.1145/2535838.2535886.

10 Beniamino Accattoli and Claudio Sacerdoti Coen. On the relative usefulness of fireballs. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 141–155. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.23.

11 Beniamino Accattoli and Claudio Sacerdoti Coen. IMELL Cut Elimination with Linear Overhead, 2024. arXiv:2405.03669.

12 Beniamino Accattoli, Andrea Condoluci, and Claudio Sacerdoti Coen. Strong call-by-value is reasonable, implosively. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–14. IEEE, 2021. doi:10.1109/LICS52264.2021.9470630.

13 Beniamino Accattoli, Andrea Condoluci, Giulio Guerrieri, and Claudio Sacerdoti Coen. Crumbling abstract machines. In *Proceedings of the 21st International Symposium on Principles and Practice of Programming Languages, PPDP 2019, Porto, Portugal, October 7-9, 2019*, pages 4:1–4:15. ACM, 2019. doi:10.1145/3354166.3354169.

14 Beniamino Accattoli and Ugo Dal Lago. On the Invariance of the Unitary Cost Model for Head Reduction. In *23rd International Conference on Rewriting Techniques and Applications (RTA'12) , RTA 2012, May 28 - June 2, 2012, Nagoya, Japan*, pages 22–37. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.RTA.2012.22.

15 Beniamino Accattoli and Ugo Dal Lago. (Leftmost-outermost) Beta reduction is invariant, indeed. *Logical Methods in Computer Science*, 12(1), 2016. doi:10.2168/LMCS-12(1:4)2016.

16 Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. The (in)efficiency of interaction. *Proc. ACM Program. Lang.*, 5(POPL):1–33, 2021. doi:10.1145/3434332.

17 Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. The space of interaction. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470726.

18 Beniamino Accattoli and Giulio Guerrieri. Abstract machines for open call-by-value. *Sci. Comput. Program.*, 184, 2019. doi:10.1016/j.scico.2019.03.002.

19 Beniamino Accattoli, Ugo Dal Lago, and Gabriele Vanoni. Reasonable space for the λ-calculus, logarithmically. In Christel Baier and Dana Fisman, editors, *LICS '22: 37th Annual ACM/IEEE Symposium on Logic in Computer Science, Haifa, Israel, August 2 - 5, 2022*, pages 47:1–47:13. ACM, 2022. doi:10.1145/3531130.3533362.

20 Beniamino Accattoli and Maico Leberle. Useful open call-by-need. In Florin Manea and Alex Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14-19, 2022, Göttingen, Germany (Virtual Conference)*, volume 216 of *LIPIcs*, pages 4:1–4:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.CSL.2022.4.

21 Francisco Alberti and Eike Ritter. An efficient linear abstract machine with single-pointer property, 1998. ESSLLI.

22 Malgorzata Biernacka, Witold Charatonik, and Tomasz Drab. A derived reasonable abstract machine for strong call by value. In Niccolò Veltri, Nick Benton, and Silvia Ghilezan, editors, *PPDP 2021: 23rd International Symposium on Principles and Practice of Declarative Programming, Tallinn, Estonia, September 6-8, 2021*, pages 6:1–6:14. ACM, 2021. doi:10.1145/3479394.3479401.

23 Malgorzata Biernacka, Witold Charatonik, and Tomasz Drab. A simple and efficient implementation of strong call by need by an abstract machine. *Proc. ACM Program. Lang.*, 6(ICFP):109–136, 2022. doi:10.1145/3549822.

24 Guy E. Blelloch and John Greiner. Parallelism in sequential functional languages. In *Proceedings of the seventh international conference on Functional programming languages and computer architecture, FPCA 1995, La Jolla, California, USA, June 25-28, 1995*, pages 226–237, 1995. doi:10.1145/224164.224210.

25 Eduardo Bonelli. The linear logical abstract machine. In Stephen D. Brookes and Michael W. Mislove, editors, *Proceedings of the 22nd Annual Conference on Mathematical Foundations of Programming Semantics, MFPS 2006, Genova, Italy, May 23-27, 2006*, volume 158 of *Electronic Notes in Theoretical Computer Science*, pages 99–121. Elsevier, 2006. doi:10.1016/J.ENTCS.2006.04.007.

26 Pierre Crégut. Strongly reducing variants of the Krivine abstract machine. *High. Order Symb. Comput.*, 20(3):209–230, 2007. doi:10.1007/s10990-007-9015-z.

27 Ugo Dal Lago and Simone Martini. An invariant cost model for the lambda calculus. In Arnold Beckmann, Ulrich Berger, Benedikt Löwe, and John V. Tucker, editors, *Logical Approaches to Computational Barriers, Second Conference on Computability in Europe, CiE 2006, Swansea, UK, June 30-July 5, 2006, Proceedings*, volume 3988 of *Lecture Notes in Computer Science*, pages 105–114. Springer, 2006. doi:10.1007/11780342_11.

28 Ugo Dal Lago and Simone Martini. Derivational complexity is an invariant cost model. In Marko C. J. D. van Eekelen and Olha Shkaravska, editors, *Foundational and Practical Aspects of Resource Analysis - First International Workshop, FOPARA 2009, Eindhoven, The Netherlands, November 6, 2009, Revised Selected Papers*, volume 6324 of *Lecture Notes in Computer Science*, pages 100–113. Springer, 2009. doi:10.1007/978-3-642-15331-0_7.

29 Ugo Dal Lago and Simone Martini. On constructor rewrite systems and the lambda-calculus. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *Automata, Languages and Programming, 36th Internatilonal Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part II*, volume 5556 of *Lecture Notes in Computer Science*, pages 163–174. Springer, 2009. doi:10.1007/978-3-642-02930-1_14.

30 Vincent Danos and Laurent Regnier. Reversible, irreversible and optimal lambda-machines. In Jean-Yves Girard, Mitsuhiro Okada, and Andre Scedrov, editors, *Linear Logic Tokyo Meeting 1996, Keio University, Mita Campus, Tokyo, Japan, March 29 - April 2, 1996*, volume 3 of *Electronic Notes in Theoretical Computer Science*, pages 40–60. Elsevier, 1996. doi:10.1016/S1571-0661(05)80402-5.

31 Hugo Herbelin. A lambda-calculus structure isomorphic to gentzen-style sequent calculus structure. In Leszek Pacholski and Jerzy Tiuryn, editors, *Computer Science Logic, 8th International Workshop, CSL '94, Kazimierz, Poland, September 25-30, 1994, Selected Papers*, volume 933 of *Lecture Notes in Computer Science*, pages 61–75. Springer, 1994. doi:10.1007/BFB0022247.

32 Delia Kesner and Shane Ó Conchúir. Milner's lambda-calculus with partial substitutions. *CoRR*, abs/2312.13270, 2023. arXiv:2312.13270, doi:10.48550/arXiv.2312.13270.

33 Yves Lafont. The linear abstract machine. *Theor. Comput. Sci.*, 59:157–180, 1988. doi:10.1016/0304-3975(88)90100-4.

34 Ugo Dal Lago and Simone Martini. The weak lambda calculus as a reasonable machine. *Theor. Comput. Sci.*, 398(1-3):32–50, 2008. doi:10.1016/J.TCS.2008.01.044.

35 Ian Mackie. The geometry of interaction machine. In Ron K. Cytron and Peter Lee, editors, *Conference Record of POPL'95: 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Francisco, California, USA, January 23-25, 1995*, pages 198–208. ACM Press, 1995. doi:10.1145/199448.199483.

36 Ian Mackie. Interaction nets for linear logic. *Theor. Comput. Sci.*, 247(1-2):83–140, 2000. doi:10.1016/S0304-3975(00)00198-5.

37 Ian Mackie and Jorge Sousa Pinto. Encoding linear logic with interaction combinators. *Inf. Comput.*, 176(2):153–186, 2002. doi:10.1006/INCO.2002.3163.

38 Ian Mackie and Shinya Sato. A calculus for interaction nets based on the linear chemical abstract machine. In Vincent Danos and Mariangiola Dezani, editors, *Proceedings of the Third International Workshop on Developments in Computational Models, DCM@ICALP 2007, Wroclaw, Poland, July 15, 2007*, volume 192 of *Electronic Notes in Theoretical Computer Science*, pages 59–70. Elsevier, 2007. doi:10.1016/J.ENTCS.2008.10.027.

39 Seikoh Mikami and Yohji Akama. A study of abramsky's linear chemical abstract machine. In Jean-Yves Girard, editor, *Typed Lambda Calculi and Applications, 4th International Conference, TLCA'99, L'Aquila, Italy, April 7-9, 1999, Proceedings*, volume 1581 of *Lecture Notes in Computer Science*, pages 243–257. Springer, 1999. doi:10.1007/3-540-48959-2_18.

40 Robin Milner. Local bigraphs and confluence: Two conjectures (extended abstract). In Roberto M. Amadio and Iain Phillips, editors, *Proceedings of the 13th International Workshop on Expressiveness in Concurrency, EXPRESS 2006, Bonn, Germany, August 26, 2006*, volume 175 of *Electronic Notes in Theoretical Computer Science*, pages 65–73. Elsevier, 2006. doi:10.1016/J.ENTCS.2006.07.035.

41 Claudio Sacerdoti Coen. sesame. Software, swhId: swh:1:dir:93776e14435d0dd5cde6deee3d9cf8f515f18fac (visited on 2024-06-19). URL: https://github.com/sacerdot/sesame/.

42 David Sands, Jörgen Gustavsson, and Andrew Moran. Lambda calculi and linear speedups. In Torben Æ. Mogensen, David A. Schmidt, and Ivan Hal Sudborough, editors, *The Essence of Computation, Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones [on occasion of his 60th birthday]*, volume 2566 of *Lecture Notes in Computer Science*, pages 60–84. Springer, 2002. doi:10.1007/3-540-36377-7_4.

43 Shinya Sato, Toru Sugimoto, and Shinichi Yamada. An implementation model of the typed lambda-calculus based on linear chemical abstract machine. In Michael Hanus, editor, *International Workshop on Functional and (Constraint) Logic Programming, WFLP 2001, Kiel, Germany, September 13-15, 2001, Selected Papers*, volume 64 of *Electronic Notes in Theoretical Computer Science*, pages 292–307. Elsevier, 2001. doi:10.1016/S1571-0661(04)80356-6.

44 Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

45 David N. Turner and Philip Wadler. Operational interpretations of linear logic. *Theor. Comput. Sci.*, 227(1-2):231–248, 1999. doi:10.1016/S0304-3975(99)00054-7.

46 Gabriele Vanoni. *On Reasonable Space and Time Cost Models for the λ-Calculus. (Sur les modèles de coût raisonnable en espace et en temps pour le λ-calcul)*. PhD thesis, University of Bologna, Italy, 2022. URL: https://tel.archives-ouvertes.fr/tel-03923206.