Is the Optimal Implementation Inefficient? Elementarily Not

Stefano Guerrini, Marco Solieri

Is the Optimal Implementation Inefficient? Elementarily Not*

Stefano Guerrini[1] and Marco Solieri[†2]

1 **LIPN, Université Paris 13 – Sorbonne Paris Cité, Paris, France**
`stefano.guerrini@univ-paris13.fr`
2 **Department of Computer Science, University of Bath, Bath, UK**
`ms@xt3.it`

Abstract

Sharing graphs are a local and asynchronous implementation of lambda-calculus beta-reduction (or linear logic proof-net cut-elimination) that avoids useless duplications. Empirical benchmarks suggest that they are one of the most efficient machineries, when one wants to fully exploit the higher-order features of lambda-calculus. However, we still lack confirming grounds with theoretical solidity to dispel uncertainties about the adoption of sharing graphs. Aiming at analysing in detail the worst-case overhead cost of sharing operators, we restrict to the case of elementary and light linear logic, two subsystems with bounded computational complexity of multiplicative exponential linear logic. In these two cases, the bookkeeping component is unnecessary, and sharing graphs are simplified to the so-called "abstract algorithm". By a modular cost comparison over a syntactical simulation, we prove that the overhead of shared reductions is quadratically bounded to cost of the naive implementation, i.e. proof-net reduction. This result generalises and strengthens a previous complexity result, and implies that the price of sharing is negligible, if compared to the obtainable benefits on reductions requiring a large amount of duplication.

1998 ACM Subject Classification F.1.1 Models of Computation, F.1.3 Complexity Measures and Classes, F.4.1 Mathematical Logic

Keywords and phrases optimality, sharing graphs, λ-calculus, complexity, linear logic

Digital Object Identifier 10.4230/LIPIcs.FSCD.2017.17

1 Introduction

1.1 Intelligence of sharing graphs

Redundancy and non-locality. An ideal implementation of a functional programming language aims at satisfying two properties: sharing, i.e. to avoid the duplication of work, and locality, i.e. to be parallelisable on architectures with multiple computing agents. Although these properties are not orthogonal, a prerequisite for both is a fine-grained implementation of material duplication. Consider the λ-calculus and take for instance the β-redex $T = (\lambda x.M)(\lambda y.N)$ and assume that x occurs $k+1$ times in M. In the λ-term $M\{^{\lambda y.N}/_{x}\}$ obtained by reducing it, we have k new copies of N, and then, k additional copies of any redex in it. Also, such a reduction step cannot be fired in parallel with any reduction in N. To solve this kind of problems, we may consider switching to graph reduction. At the time

* Preliminary results of this work were previously presented [14] with a stronger yet unsolved claim.
† Work mainly carried out during: PhD studentship at Paris 13 and Bologna, ATER at Paris 7.

of writing, the technique is employed for instance in the Glasgow Haskell Compiler, namely by the STG-machine at its core [21], whilst its essence dates back to the early seventies [24]. The key idea can be formalised with proof-nets of linear logic (LL), where T can be expressed as in Figure 1a. With respect to the syntax tree of T we notice: the explicit connection of the λ-link with its variable x; a ?-link connecting the $k+1$ occurrences of x in M; a box (the dotted square) around the argument of the application (the @-link), to mean $\lambda y.N$ is a duplicable term. Such information can be equivalently formalised by associating the boxing depth to each link – e.g. the rightmost λ-link has index 1. Now, a linear-β reduction step rewrites Figure 1a to Figure 1b. The box is now shared, thus reductions in N can be performed without duplicating it. Unfortunately, this is still quite unsatisfactory. Indeed, if an occurrence of x, which is represented in the figure as a premiss of the topmost contraction ?-link, appears in argument position within M, i.e. it is linked to a @-link, then we have a "virtual β-redex". In particular, this @-link and the λ-link will form a redex, once we reduce the exponential redex between the ?-link and the promotion !-link. The problem is that firing the latter entails the duplication of the whole box, and thus the loss of sharing benefits.

Sharing and locality. Sharing graphs [19, 13, 4] include instead, by their very design, the solution to these problems of duplication. Back to our example, instead of performing the duplication of the lowermost box, we can apply the "triggering" rule (t) shown in Figure 1b-c. This introduces a link of a new kind $|\!+\!\rangle$, called *mux* (multiplexer), that has $k+1$ premisses and index 1 as the content of the box. Muxes perform duplications of boxes in a local, link-by-link way, following the approach introduced by interaction nets [18]. For instance, the $|\!+\!\rangle$-link in the example can duplicate the λ-link only, by the $(d^{\pm}\!\circ)$ rule in Figure 1c-d. This allows N being kept shared, whilst copies of the λ-link can now independently interact with @-links in M. Also, from the $(d^{\pm}\!\circ)$ rule originates a sort of co-sharing link, the *negative mux* with kind $|\!-\!\rangle$ and the same index, which implements sharing of contexts (outputs), instead of terms (inputs). In addition to the case of λ-links, there is one duplication for each other link $(@, !, ?)$, but it is applicable only when the mux faces its principal port, i.e. its conclusion, depicted by an outgoing arrow. This makes duplication *lazy* – it is performed only when a mux obstructs the formation of other redexes. The life of a $|\!+\!\rangle$-link may end in two ways: by being merged into it, when it reaches the premiss of a ?-link, or by annihilating with a facing $|\!-\!\rangle$-links (i.e. the facing pair of links reduces to the identity). But positive muxes may also need to swap with negative ones, i.e. they duplicate each other. Therefore, we mark muxes with the index inherited from the exponential redex, e.g. 1 in our running example, so we can distinguish two kinds of redexes with two opposite muxes: annihilation when indices are equal, swap when they are different. In general, we would also need a supplemental mechanism or information tracking, the so-called "oracle", which manages indexes, as a local implementation of digging and dereliction of LL. But this is not the case for proof-nets for terms typed in the elementary and light variants of LL (ELL and LLL) [12, 6]. They are obtained by a restriction on usual exponential boxes that makes indices immutable by definition. Hence, for their sharing implementation we can simply consider the so-called *abstract algorithm* of sharing graphs (ASG). Thanks to muxes, the sharing graph G that is the normal form of T is a (possibly enormously) compressed representation of the proof-net $\overline{T}$ we would have obtained by ordinary cut-elimination. To retrieve $\overline{T}$ from G, we need the *read-back* (RB) – a set of additional rewriting rules for sharing graphs, which unlatch muxes to let them freely duplicate downwardly, i.e. from the root of the term to its inputs.

▷ This paper considers elementary proof-nets (EPN) to represent ELL and LLL typed terms, or proofs (presented in Section 2), and their sharing implementation with ASG, including RB (Section 3).

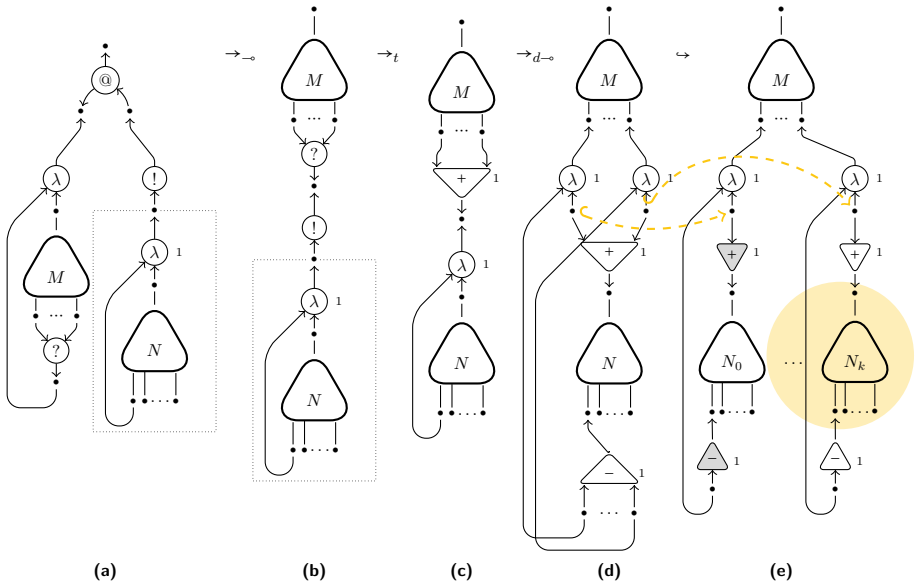


Figure 1 Examples: sharing reductions (a-d) on a proof-net; unshared graph (e) unfolding of d.

1.2 Efficiency of sharing graphs

Question. The possible benefits of sharing were astonishingly evident from the very first sequential implementations of sharing graphs. For example, in the normalisation of λ-terms benchmarks of BOHM [3] recorded polynomial times, against traditional languages (Caml Light and Haskell) [4, Ch. 10] requiring exponential times. But sharing and locality may come with a price. *What is the price of sharing graphs?* To answer this question, very recently broadly surveyed by Asperti [1], we first need the notion of cost. We cannot use the number of β steps, since, even though they are a reasonable [23] measure for the λ calculus [9], in sharing graphs whole families of redexes are reduced simultaneously, (and this is the reason why they realise the Lévy optimal reduction [20]). Nor can we use the number of such parallel β-steps, since it would be an enormously parsimonious measure. Indeed, a polynomial number of family reductions in the size s of a term may hide a concrete cost bigger than a tower of exponentials of s, caused by the oracle rules [5] or by the local duplication rules [2]. Therefore, a study of the complexity of sharing graphs is necessarily limited by the state of the art to take the form of a comparison to some existing reduction system, i.e. by the approach of "ICC in the small" [10], and to use a cost measure based on standard rewriting theory – counting the size of sub-graphs that are erased/introduced at each step. Hence, to employ a reasonable measure we are possibly renouncing to parsimoniousness.

▷ In Section 4.1 we shall define two cost functions for ASG and EPN reductions.

A first partial answer. In the only previous contribution which tackled the complexity study of ASG [7], Baillot, Coppola and Dal Lago exploited the implicit computational complexity of (the affine variants of) ELL and LLL. The cost of the cut-elimination of a proof-net N with maximum boxing index d is related respectively by a Kalmar elementary function in the size of N and rank d, in ELL, or by a polynomial function in the size of N and degree d, in LLL. By means of a quantitative semantical tool [8] inspired by the geometry of interaction [11], the authors proved that the cost of a normalisation with sharing graphs of ELL and LLL proof-nets remains in the two aforementioned complexity classes. But they were not able to

give any explicit bound to the overhead that sharing graphs might introduce in the worst case – for instance, when it becomes less effective, since the reduction does not require a relevant amount of duplications. Moreover, the technical approach hardly seems adaptable to prove a similar result for the more general case featuring the "oracle".

Contributions. We study the complexity of reducing (proof-nets representing) λ-terms typed in the elementary or light linear logic and relate it to the cost of naive reduction. In particular, we give a worst-case bound of the cost of sharing reduction as a quadratic function in the cost of the naive reduction. Three are the axis along which this paper improves the previous literature [7]:

1. *Strength.* We give a clear bound to the overhead of sharing reduction.
2. *Generality.* Our analysis considers strategy-agnostic reductions of arbitrary length, instead of normalisation, and includes the read-back rules (which are sub-optimal). In this way we get a more uniform, and perhaps fairest, complexity comparison.
3. *Scalability.* The technical approach bases on a quantitative extension of an elegant syntactical simulation between sharing graphs and proof-nets; this provides modularity for our complexity analysis, and appears to give room for further investigations also about more general cases.

▷ The complexity bound is obtained in Sections 4.2 and 4.3. Detailed proofs are omitted for lack of space, but can be found in an extended version available on authors' websites.

2 Intuitionistic elementary and light logics

We start by introducing proof-nets of ELL and LLL. We define first the two intuitionistic and weakening-free logics by a levelled sequent calculus inspired by the approach of Guerrini, Martini and Masini [15], which represents a typing system for λI terms. The absence of weakening in λI will remove a considerable technical overhead that have no impact on the complexity question (in the literature [7, for instance] garbage collection is indeed quite commonly postponed at the end) nor on the proof technique. On the other hand, this will greatly ease the exposition of ASG and their complexity analysis, since it removes reduction rules and entails the uniqueness of the root in sharing graphs.

Then, we give the translation of sequent proofs in levelled proof-nets, that are directed hypergraphs where every (occurrence of a) formula is a vertex and every inference rule is a directed hyperedge (called link) that goes from the premisses of the rule to its conclusions. Axioms and cuts correspond instead to a direct plugging of the two sub-proofs. Finally, we will present cut-elimination as reduction of proof-nets.

2.1 Sequent calculus and typed lambda terms

▶ **Definition 1** (Logics and typing)**.** Given L a set of *literal* symbols, the *formulas* of ELL and LLL are built from the following grammar.

$$T ::= L \mid T \multimap T \mid !T \mid \S T. \tag{1}$$

A *levelled formula* is a pair T^n where T is a formula and $n \in \mathbb{N}$. Given a set V of *variables*, the set of *terms* is defined from the standard definition of the Λ calculus:

$$t ::= V \mid \lambda V.t \mid t\,t, \tag{2}$$

$$\frac{}{x : A^n \vdash x : A^n}\ (\mathrm{Ax}) \qquad \frac{\Delta_1^{n+1}, \Gamma_1^n \vdash t : A^n \qquad \Delta_2^{n+1}, \Gamma_2^n, x : A^n \vdash u : B^n}{\Delta_1^{n+1}, \Delta_2^{n+1}, \Gamma_1^n, \Gamma_2^n \vdash u[t/_x] : B^n}\ (\mathrm{Cut})$$

$$\frac{\Delta^{n+1}, \Gamma^n, \{x : A^n\} \vdash t : B^n}{\Delta^{n+1}, \Gamma^n \vdash \lambda x.t : A \multimap B^n}\ (\multimap) \qquad \frac{\Delta_1^{n+1}, \Gamma_1^n \vdash t : A^n \qquad \Delta_2^{n+1}, \Gamma_2^n, x : B^n \vdash u : C^n}{\Delta_1^{n+1}, \Delta_2^{n+1}, \Gamma_1^n, \Gamma_2^n, y : A \multimap B^n \vdash u[yt/_x] : C^n}\ (\bar{\multimap})$$

(a) Common rules: axiom, cut, abstraction, application

$$\frac{\Gamma^{n+1} \vdash t : A^{n+1}}{\Gamma^{n+1} \vdash t : !A^n}\ (!) \qquad \frac{\{x : A^{n+1}\} \vdash t : A^{n+1}}{\{x : A^{n+1}\} \vdash t : !A^n}\ (!) \qquad \frac{\Gamma^{n+1}, \Delta^{n+1} \vdash t : A^{n+1}}{\S\Gamma^n, \Delta^{n+1} \vdash t : \S A^n}\ (\S)$$

(b) Elementary promotion

(c) Light promotion – Γ, Δ may be empty

$$\frac{\Delta^{n+1}, \Gamma^n, (x_i : A^{n+1})_{1 \le i \le m} \vdash t : B^n}{\Delta^{n+1}, \Gamma^n, y : !A^n \vdash t[y/_{x_i}]_{1 \le i \le m} : B^n}\ (?)$$

(d) Contraction

Figure 2 Levelled sequent calculus of ELL and LLL.

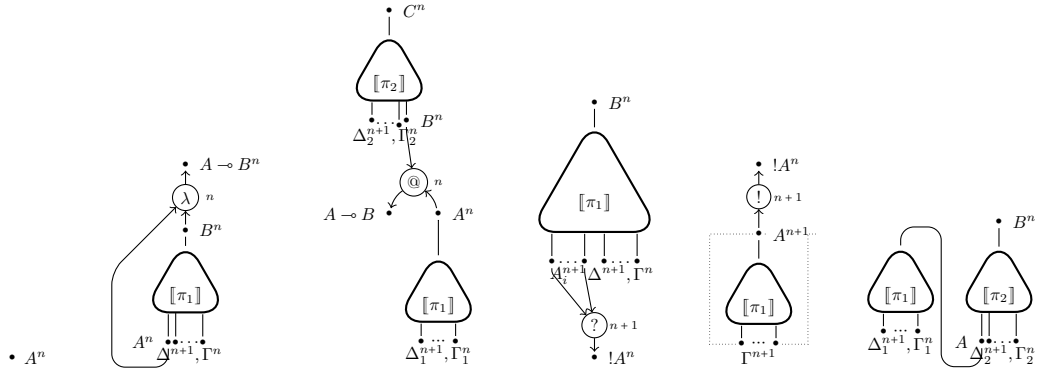
by constraining abstraction so that it bounds at least one occurrence (i.e. x must appear as a free variable of t in order to write $\lambda x.t$). The *sequent calculus* of the *first-order, weakening-free, intuitionistic fragment of ELL and LLL* is defined in Figure 2 and represents a type assignment system for the Λ_I calculus.

▶ Remark 2 (LLL ⊂ ELL). Observe that § cannot appear in ELL formulas and that the light version of the (!) rule is an instance of the elementary one. Moreover, any proof π of LLL can be encoded in ELL by using the logical modality ! instead of §. For this reason and the sake of simplicity, in the rest of the paper we shall refer only to the elementary version.

2.2 Proof-nets

▶ **Definition 3** (Elementary proof-nets)**.** Given a set of *vertices*, a *link* is a directed hyperedge, i.e. a triple $(V\ \kappa^n\ u)$, where: u is a vertex called *conclusion*, V is a non empty sequence of vertices called *premisses*, κ is a label called *kind*, $n \in \mathbb{N}$ is called *level*. Depending on the kind of a link l, a *polarity* is assigned to each of its vertices, i.e. an element of $\{\iota, o\}$ (input and output), and the *arity* may be fixed, i.e. the number of vertices of l. In order to keep a better correspondence with the underlying λ-calculus, the kinds of links will correspond to the syntactical construct associated to the rule and not to the corresponding logical connective. Also, links are depicted so that positioning follows the input/output computational interpretation, o above, ι below; whilst arrows orientation follows the proof-theoretical viewpoint, premisses inward, conclusions outward.

A proof-net of the fragment of ELL of our interest, to which we shall simply refer to as an *elementary proof-net* (EPN), is the hyper-graph N obtained by the *translation* $[\![\pi]\!]$ of some sequent proof π, as inductively defined as follows. Let R be the last rule of π, assume its shape to be as in one of Figures 2a, 2b, and 2d. Also, let π_1, π_2 respectively be the sub-proofs of π (if any) whose conclusions are the leftmost and rightmost premisses of R. Then the translation of π is given by the case analysis in Figure 3. There we assume that $[\![\pi_1]\!], [\![\pi_2]\!]$ are disjoint, and also that, except when otherwise depicted, both are disjoint from the vertices introduced by the inductive steps (i.e. new vertices are "fresh"). Although we shall label vertices with their names, in the picture we used formulas to ease the reading and stress the correspondence with the sequent proof.



(a) Axiom **(b)** Abstraction **(c)** Application **(d)** Contraction **(e)** Promotion **(f)** Cut

Figure 3 Elementary proof-nets, as translated from sequent proofs. See Definition 3 and Figure 2.

The *conclusions* of $[\![\pi]\!]$ are the vertices corresponding to formulas appearing in the sequent proved by π – *input* vertices stand for formulas on the left-hand side of the sequent, *output* ones for those on the right-hand side. The set of the conclusions of $N \in \mathsf{EPN}$ is called its *interface* and denoted by $\mathsf{iface}(N)$; any vertex of N not in $\mathsf{iface}(N)$ is an *internal vertex* of N, and the corresponding set is denoted by $\mathsf{int}(G)$.

▶ Remark 4 (λ-terms). The input-output relation allows to associate a λ-term to a proof-net – if we erase exponential links, we essentially obtain the syntax tree of the term.

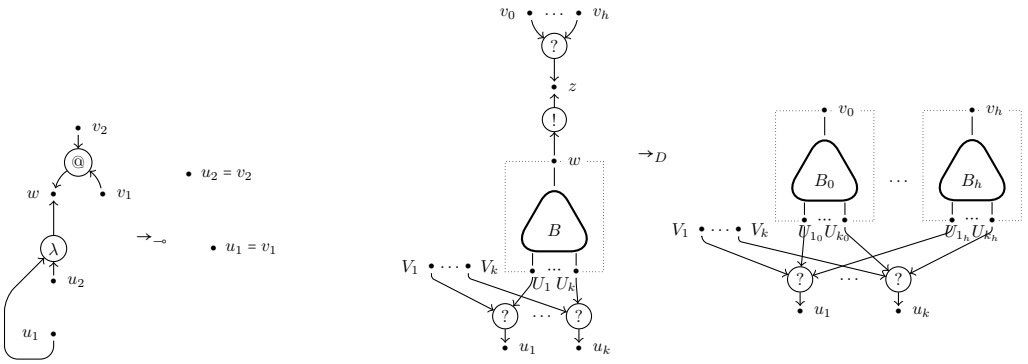
▶ **Definition 5** (Boxes)**.** The level of a vertex v is denoted as $\ell(v)$, and we shall write the same for a link, meaning the level of its premiss(es). The maximum level of links in N is written $\partial(N)$. A *box* of level n in $N \in \mathsf{EPN}$ is a sub-graph B of N whose links and vertices have level not smaller that n and that is maximal with respect to inclusion. The ι conclusion v of B is called its *principal door*, whilst the o conclusions $u_1, \dots, u_k$ *auxiliary doors*. We shall then denote B as $(u_1, \dots, u_k \ \langle\!\langle B \rangle\!\rangle \ v)$. Boxes are depicted with dotted squares.

▶ **Definition 6** (Paths)**.** A *path* from u_0 to u_k in a graph G is a sequence of vertices $(u_0, \dots, u_k)$ for which there is a sequence of links $l_0, \dots, l_{k-1}$ such that $u_i \neq u_{i+1}$ and u_i belongs to both the vertices of l_i and those of l_{i+1}, for any $0 \leq i < k-1$. A *downward path* is a path such that: u_i is not the first premiss of a λ-link and is an o-vertex of l_{i+1}; and u_{i+1} is an ι-vertex of l_{i+1}, for any $0 \leq i < k-1$. If these holds, then $(u_k, \dots, u_0)$ is an *upward path* from u_k to u_0. We shall write $u \sim v$ when there is a path from u to v, $u \rightsquigarrow v$ when there is a downward one, $v \rightsquigarrow u$ when $u \rightsquigarrow v$. A *rooted* path is a downward path starting from the o-conclusion of G.

▶ Remark 7 (Boxes). Auxiliary doors of a box B are always premisses of an exponential link, and that B is always connected: there exists a path between any two of its vertices. Also, boxes properly nest: given two distinct boxes, either they are completely disjoint, or one is included into the other. Our definition of box is slightly different from the standard one, as it is does not include vertices with exponential formulas, but just take the interior of the box.

▶ **Definition 8** (EPN reduction)**.** The rewriting relation $\to_{\mathsf{EPN}}$, which implements the cut-elimination on proof-nets, is obtained by the context closure of the reduction rules $(\overset{+}{\multimap}), (D)$ respectively defined in Figures 4a and 4b.

▶ **Notation 9.** Reductions steps are denoted by Greek letters (e.g. ρ), sequences are marked with overlines ($\overline{\rho}$), reducts are denoted by functional notations ($N \to \rho(N)$).



(a) Multiplicative **(b)** Exponential

Figure 4 EPN reduction.

▶ **Proposition 10** (Stratification). *In EPN, levels are preserved by reduction.*

3 Abstract sharing graphs

We introduce abstract sharing graphs (ASG) and their reduction, and recall their most important qualitative properties as an implementation of EPN.

3.1 Syntax and computation

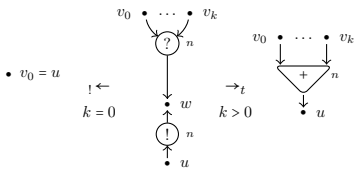
▶ **Definition 11** (Sharing and read-back reductions). The *sharing reduction* is the graph-rewriting relation $\to_{\mathsf{ASG}}$ given by the context closure of the following reduction rules.
Logical $(\multimap), (!), (t)$, defined in Figures 4a and 5a. On unary contractions, $(!) = (D)$.
Duplication $(d\overset{+}{\multimap}), (d\overset{-}{\multimap}), (d!), (d?)$, respectively defined in Figures 5c, 5d, 5e, and 5f.
Muxes $(a), (s)$, defined in Figure 5b.
The *read-back reduction* $\to_{\mathsf{RB}}$ is obtained from the mux interaction rules and the followings.
Read-back duplication $(r\overset{+}{\multimap}), (r?), (m)$, defined in Figures 5g and 5h.
The RB-normal form of a graph G is called its *read-back* and written $\mathcal{R}(G)$. The reduction $\to_{\mathsf{ASGR}}$ is the union of $\to_{\mathsf{ASG}}$ and $\to_{\mathsf{EPN}}$. The set ASG of abstract sharing graphs is obtained by the closure of EPN with respect to $\to_{\mathsf{ASGR}}$.

3.2 Implementation of EPN

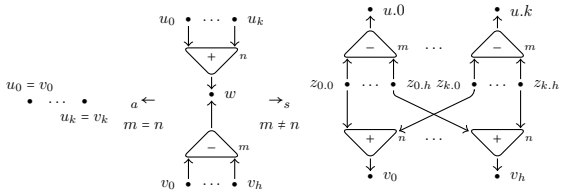
Sharing graphs with ASG and RB reductions represent a well-behaved rewriting system. $\to_{\mathsf{ASG}}$ is locally confluent. $\to_{\mathsf{RB}}$ and $\to_{\mathsf{ASGR}}$ are confluent. All three are strongly normalising, and the last two have normal forms in EPN [16, Thm.s 4, 11.i and 11.ii (for MELL)]. The traditional way of normalising proofs or terms with sharing graphs maximises the amount of sharing by postponing duplication as much as possible, thus performing first an ASG-normalisation and then an RB one. This gives a correct implementation of EPN reduction and a complete implementation of EPN normalisation.

▶ **Theorem 12** (Correctness). *If $N \in EPN, G \in ASG$ and $N \to^*_{ASG} G$, then $N \to^*_{EPN} \mathcal{R}(G)$.*

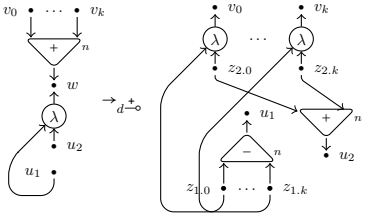
Proof. We refer to the original proof for λ-calculus [13], or the more syntactic one for MELL [16, Thm. 13]. ◀



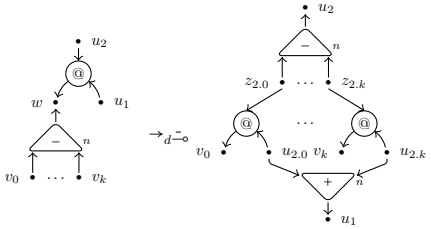
(a) Exponential reductions



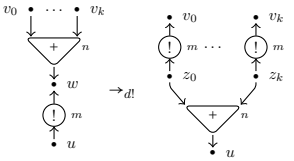
(b) Annihilation and swap



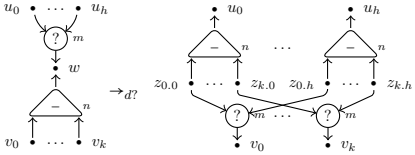
(c) Duplication of abstraction



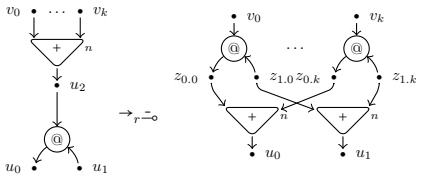
(d) Duplication of application



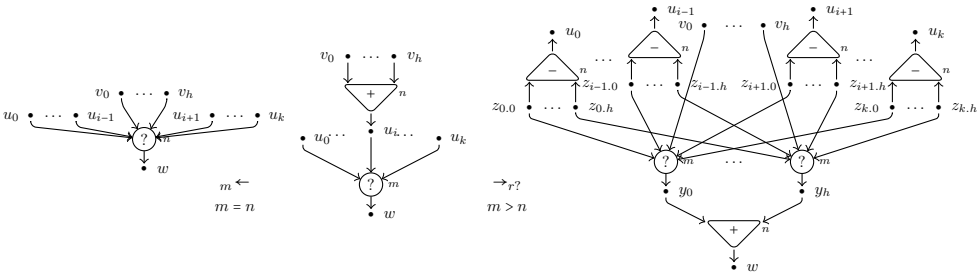
(e) Duplication of promotion



(f) Duplication of contraction



(g) Duplication of application



(h) Merge with, or duplication of, contraction

Figure 5 ASG and RB reduction rules.

▶ **Theorem 13** (Normalisation completeness). *If $N, \bar{N} \in EPN$ where $N \to^*_{EPN} \bar{N}$, with $\bar{N}$ normal, then there is $\bar{G} \in ASG$ being ASG-normal and such that $N \to^*_{ASG} \bar{G} \to^*_{RB} \bar{N}$.*

Proof. See Asperti and Guerrini [4, Thm. 7.9.3.ii]. ◀

If instead we consider $\to_{\mathsf{ASGR}}$, sharing graphs can be shown to be more generally complete with respect to the whole EPN-reduction (not just normalisation). It suffices to prioritise RB redexes, thus enforcing exhaustive duplications of boxes.

▶ **Theorem 14** (Completeness). *For any $N, N' \in EPN$ if $N \to_{EPN} N'$ then $N \to^+_{ASGR} N'$.*

▶ Remark 15 (Optimality). Once equipped with a "call-by-need" strategy [4, §5.6], the number of $(-\circ)$ steps performed by ASG is minimised so the reduction reaches Lévy-optimality [16, Thm. 14 (for MELL)] [4, Thm. 5.6.4 (for λ-calculus)]. For the sake of generality, our focus will not be limited to the optimal strategy, and we shall analyse sharing graphs with the greatest strategy-agnosticism.

4 Computational complexity

4.1 Cost measures

We define two cost functions $\mathcal{C}_{\mathsf{ASG}}$ and $\mathcal{C}_{\mathsf{EPN}}$, respectively for the ASG and EPN reductions. Both measures are essentially equivalent to the size of the rewriting operations required. However, to ease the presentation without affecting fairness, only $\mathcal{C}_{\mathsf{EPN}}$ is formally defined as such, whilst $\mathcal{C}_{\mathsf{ASG}}$ has been accurately hand tuned.

▶ **Definition 16** (Size and variations). The *size* of a graph G, written $\#G$, is the sum of the cardinality of the set of its vertices and the sum of the arities of its links. We remark that for a box $(u_1, \dots, u_k \ \langle\!\langle B \rangle\!\rangle \ v)$, all of its doors, principal and auxiliary ones, belong to the sub-graph B and are accounted by $\#B$. Given M a metric on a graph G, e.g. the size of G or of some set of sub-graphs, and ρ a reduction step, $\Delta(\rho)$ denotes $M(\rho(G)) - M(G)$.

▶ **Definition 17** (EPN-reduction cost). The cost $\mathcal{C}_{\mathsf{EPN}}(\rho)$ of a EPN-reduction step ρ on a levelled proof-net N is defined as the size of the symmetric difference between the vertices and links of N and those of $\rho(N)$. Namely, the cost of a given rule is computed in Table 1a. The cost of a reduction sequence $\bar{\rho}$ is the sum of the costs of each step it is composed of.

▶ **Definition 18** (ASG- and RB-reduction costs). The cost $\mathcal{C}_{\mathsf{ASG}}(\sigma)$ and $\mathcal{C}_{\mathsf{RB}}(\sigma)$ of a ASG- or RB-reduction step σ is given in Table 1b.

4.2 Unshared simulation

Our complexity comparison is directly founded on the most natural correspondence between sharing graphs and proof-nets: a *syntactical simulation*. Given a proof-net P, any ASG reduction sequence σ on P can be simulated by a EPN sequence ρ on the same P. Since the only difference between the two rewriting systems is the style of duplication, local or global, respectively, $\sigma(P)$ and $\rho(P)$ may greatly differ in the number of copies of some subgraphs. To understand precisely the relationship between $\sigma(P)$ and $\rho(P)$, we shall employ an intermediate reduction system – the *unshared graphs* (UG) [16]. Let us taste the main intuitions employing the example that we began to consider in Section 1.1 and Figure 1. The key feature of UG is the fact that the exponential redex in Figure 1b would be rewritten in $k+1$ copies of the box (as one usually does on proof-nets), plus $k+1$ unary muxes (instead

Table 1 Costs assigned to classic and sharing reductions.

Rule	$\mathcal{C}_{\mathsf{EPN}}(\rho)$
$(\multimap)$	9
(D)	$k \times \#B + 2k + 4$

(a) Classic reduction rules. In the case of (D), $k+1$ is the number of premisses of the ?-link, and B is the box enclosed by the !-link.

Rule	$\mathcal{C}_{\mathsf{ASG}}(\sigma)$
$(\multimap)$	9
$(!)$	6
(t)	$j+4$
$(d!)$	$3k$
$(d\overset{+}{\multimap}), (d\overset{-}{\multimap}), (r\overset{-}{\multimap})$	$5k$
$(d?), (r?)$	$(2j+3) \times k$
$(a), (m)$	k
(s)	$k \times l$

(b) Sharing and read-back rules. $j+1$, $k+1$ and $l+1$ respectively are the number of premisses of the ?-link, the first and the second $|*)$-link, where involved.

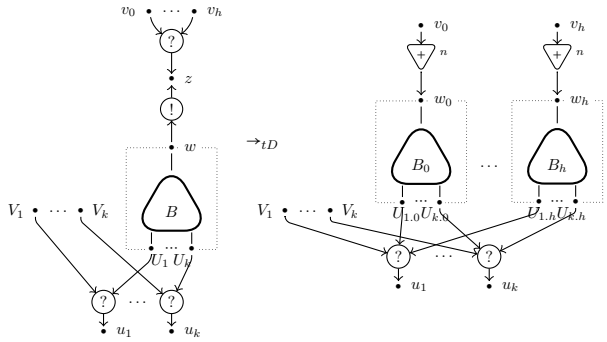


Figure 6 UG reduction rule: duplication and triggering (where $h > 1$).

of one k-ary mux of Figure 1c). Have a peek at Figure 6. The new links are called *lifts*, and play the mere role of markers for the presence of muxes in corresponding sharing graphs. In particular, their propagation along the graph does not affect any other link, and if we erase them we simply obtain a proof-net. For instance, the $(d\lambda)$ step in Figure 1c-d, can be simulated by $k+1$ propagation steps that reach Figure 1e, that is the *unfolding* of Figure 1d. Similarly, any reduction step inside N (which is shared) needs to be performed $k+1$ times in N_i. Yellow arrows illustrate such *unfolding* relation.

4.2.1 Unshared graphs

▶ **Definition 19** (Unshared reduction and graphs)**.** The *unshared reduction* is the rewriting relation $\to_{\mathsf{UG}}$ obtained from $\to_{\mathsf{ASG}}$ by replacing the (t) rule with (tD), defined in Figure 6. The *unshared read-back* $\to_{\mathsf{UR}}$ is the restriction of RB-reduction to UG (i.e. muxes are unary). The union of these two relations is written as UGR. An *unshared graph* UG-graph for short, is either a levelled proof-net, or the reduct of an unshared graph via UG- or UR-reduction.

4.2.2 Unfolding and simulating sharing graphs in unshared graphs

▶ **Definition 20** (Sharing morphism)**.** A *sharing morphism* $\mathcal{M}$ is a surjective homomorphism on hypergraphs [17] from UG to ASG that preserves the kind and level of links. We say that $G \in \mathsf{ASG}$ *unfolds* to $U \in \mathsf{UG}$, written $G \hookrightarrow U$, if there is a sharing morphism $\mathcal{M}$ such that $\mathcal{M}(U) = G$. We shall use the same notation to relate vertices and links: if $v \in V(G)$ and

$W \subseteq V(U)$ we write $v \hookrightarrow W$ to mean $\mathcal{M}(W) = v$, while if $m \in L(G)$ and $N \subseteq L(U)$ we write $m \hookrightarrow N$ when $\mathcal{M}(N) = m$.

▶ **Lemma 21** (Unfolded simulation). *For any $N \in \mathsf{EPN}$, $G \in \mathsf{ASG}$, if $N \xrightarrow{\bar{\sigma}}{}^{*}_{\mathsf{ASG}} G$ then there exists $U \in \mathsf{UG}$ such that $N \xrightarrow{\bar{\mu}}{}^{*}_{\mathsf{UG}} U$ and $G \hookrightarrow U$. Moreover, for any $G' \in \mathsf{ASG}$, if $G \xrightarrow{\bar{\sigma}'}{}^{*}_{\mathsf{RB}} G'$ then there exists $U' \in \mathsf{UG}$ such that $U \xrightarrow{\bar{\mu}'}{}^{*}_{\mathsf{UR}} U'$ and $G' \hookrightarrow U'$. We call $\bar{\mu}$ and $\bar{\mu}'$ an* unfolded simulation *of $\bar{\sigma}$ and $\bar{\sigma}'$, respectively, and write $\bar{\sigma} \hookrightarrow \bar{\mu}$ and $\bar{\sigma}' \hookrightarrow \bar{\mu}'$.*

4.2.3 Simulating sharing graphs into proof-nets

▶ **Definition 22** (Lift erasure). The *lift erasure* is the function $\mapsto$ that maps a $U \in \mathsf{UG}$ to the $N \in \mathsf{EPN}$ obtained by equating any two vertices u, v for which there is $(u \mid\!\ast\!\rangle\ v) \in L(U)$. The function is extended to let it map vertices and links of U to those of N.

▶ **Lemma 23** (Lift erasure simulation). *For any $N \in \mathsf{EPN}, U \in \mathsf{UG}$, if $N \to^{*}_{\mathsf{UG}} U$ then there is a unique $\bar{\sigma} : N \to^{*}_{\mathsf{EPN}} N'$ such that $U \mapsto N'$.*

▶ **Definition 24** (Sharing implementation). Given $N \in \mathsf{EPN}, G \in \mathsf{ASG}$, we say N is *implemented by* G, written $G \twoheadrightarrow N$, if there is $U \in \mathsf{UG}$ such that $G \hookrightarrow U \mapsto N$.

▶ **Theorem 25** (EPN-reduction simulates ASG-reduction). *Let N be a proof-net. If $N \to^{*}_{\mathsf{ASG}} G$ then there exists $N \to^{*}_{\mathsf{EPN}} N'$ such that $G \twoheadrightarrow N'$.*

4.3 Quantitative unshared simulation

Given two simulating reduction sequences on the same proof-net, σ of ASG and ρ of EPN, we can compare $\mathcal{C}_{\mathsf{ASG}}(\sigma)$ to $\mathcal{C}_{\mathsf{EPN}}(\rho)$, finding a quadratic bound. In order to do so, we shall bring the unshared simulation up to a quantitative level, so that these two costs can be expressed as two different cost measures on unshared reduction, thus allowing a direct comparison.

4.3.1 Share

The first step is to define introduce a labelled reduction for UG which decorates lifts: at the creation of a set of lifts we add to each of them a fresh label tracking their number and relative indices. We interpret the 0-th lift as the master one, while all the others as sharing ones. In the previous illustration in Figure 1e, we darkened master lifts. Then, given an unshared graph U being the unfolding of a sharing graph G, we can build the *sharing context* of a vertex v as the sequence of lift labels that are along its access path from the root of U. Its sharing context allows us to understand:

1. if v is shared in G, i.e. if its context contains non master lifts (equivalently: "has v been previously copied in U"?);
2. how much v is shared, i.e. the product of the number of lifts in the label, for any lift in the context of v ("how many other copies of v are in U"?).

The set of such shared objects is called the *share*, and conceptually represents the subtraction of the graph of G from that of U. In Figure 1e, we highlighted the share as a pale yellow circle. Notice that it does not include N_0, since its sharing context include a master lift, we interpret it as a *master* copy of N (the stereotype).

▶ **Definition 26** (Copy identity). We enrich $\to_{\mathsf{UGR}}$ with the *copy identity* labelling (CID). It maps lifts to *labels* of the form $x_{i:k}$ (positive) or $\overline{x}_{i:k}$ (negative), where $x \in \mathcal{V}$ a set of variable symbols, whilst $i, k \in \mathbb{N}$ are the *current* and *maximal index*. Given $U \in \mathsf{UG}$ and μ a (tD)-step

as in Figure 6, we set $\mathsf{CID}((v_j \mid\!*\!\rangle^n u_j)) = x_{j:h}$, for any $0 \le j \le h$, and for some $x \in \mathcal{V}$ not occurring in the labels of U. Labels are negated in negative lifts: if l is a positive lift with $\mathsf{CID}(l) = x_{j:h}$ and l' is a negative residual of l w.r.t. a lift propagation rule $(d\kappa)$ or $(r\kappa)$, then $\mathsf{CID}(l') = \overline{x}_{j:h}$. In any other case, labels are preserved by reduction, in particular under copying.

▶ **Definition 27** (Sharing contexts)**.** The *sharing contexts* $\mathfrak{C}$ are the strings generated by the binary concatenation operator $\cdot$ over labels, and including 1 as the identity element (i.e. the empty string) and 0 as the absorbing element for concatenation. We add two equations to detect whether (or not) labels are well-bracketed in a context by neutralisation, (or nullification). For any labels $a \neq b$,

$$a \cdot \overline{a} = 1 \qquad a \cdot \overline{b} = 0. \tag{3}$$

Also, a is said *positive*, written $a > 0$, when it is empty or contains only positive labels.

A *levelled sharing context*, or simply an l-context, is a map γ from $\mathbb{N}$ to $\mathfrak{C}$, that is uniformly null: if for some $n \in \mathbb{N}$ we have $\gamma(n) = 0$, then $\gamma(m) = 0$ for any $m \in \mathbb{N}$. We write $(\gamma)|_n$ to denote the *restriction* of γ on n: if $m = n$ and $\gamma(m) \neq 0$, then $(\gamma)|_n(m) = 1$; otherwise $(\gamma)|_n(m) = \gamma(m)$. Also, we denote with $!^n a$ the *lifting* of a context a at level n. Namely, if $!^n a = \gamma$ then $\gamma(n) = a$, whilst $\gamma(m) = 1$ for any $m \neq n$. More precisely, $!^n a$ denotes $!(!^{n-1} a)$, where we set that $!1 = 1$, that $!0 = 0$, and also that $!(a \cdot b) = !a \cdot !b$. We say that γ is *positive*, written $\gamma > 0$, if $\gamma(n) > 0$ for any n. Given $\pi : u \rightsquigarrow v$ the *l-context* of π is defined as follows.

$$\mathfrak{c}\,(()) = 1 \tag{4}$$

$$\mathfrak{c}\,(\pi :: (u,v)) = \begin{cases} \mathfrak{c}\,(\pi) \cdot !^n a & \text{if there is } l = (u \mid\!+\!\rangle^n v) \text{ s.t. } \mathsf{CID}(l) = a \\ \mathfrak{c}\,(\pi) \cdot !^n \overline{a} & \text{if there is } l = (v \mid\!-\!\rangle^n u) \text{ s.t. } \mathsf{CID}(l) = a \\ (\mathfrak{c}\,(\pi))|_n & \text{if there is } (u \;?^n\; v) \\ \mathfrak{c}\,(\pi) & \text{if } u, v \text{ belong to a link of kind in } \{\overset{+}{\multimap}, \overset{-}{\multimap}, !\} \end{cases} \tag{5}$$

The *l-context of a vertex* in $U \in \mathsf{UG}$ is the context of any rooted path reaching it.

▶ **Proposition 28** (Positivity)**.** *Let π be a rooted downward path in $U \in \mathsf{UG}$. Then $\mathfrak{c}\,(\pi) > 0$.*

▶ **Proposition 29** (Path irrelevance)**.** *Let π, π' be two rooted paths in some UG ending with the vertex v. Then $\mathfrak{c}\,(\pi) = \mathfrak{c}\,(\pi')$.*

▶ **Definition 30** (Share and master)**.** A lift labelled with $x_{i:k}$ is *master* if $i = 0$, otherwise it is *shared*. A context a is master if a is empty or contains only master labels $x_{0:m}$; otherwise, it is a shared context. Finally, an l-context α is master if $\alpha(n)$ is master for any $n \in \mathbb{N}$, otherwise it is shared. A vertex is shared if its l-context is so, otherwise it is master; a link is shared if it has at least one shared vertex, otherwise it is master. A *share component* is a non-empty, connected, and maximal (w.r.t. inclusion) sub-graph whose vertices and links are shared. The set of the share components of $U \in \mathsf{UG}$ is denoted as $\mathsf{ShC}(U)$, their union is called the *share*, written $\mathsf{Sh}(U)$.

▶ **Definition 31** (Share boundary and interior)**.** Given $U \in \mathsf{UG}$, a shared lift $l = (u \mid\!*\!\rangle\ v)$ is a *boundary lift*, written $l \in \mathsf{bLft}(U)$, when $u \notin \mathsf{Sh}(U) \ni v$, whilst it is an *interior lift*, written $l \in \mathsf{iLft}(U)$, when $u, v \in \mathsf{Sh}(U)$. A given $v \in \mathsf{Sh}(U)$ is *boundary*, written $v \in \mathsf{BSh}(U)$, if it is the conclusion of a boundary lift, or it is linked by a lift to a boundary vertex. If additionally

Table 2 Metrics of UGR reduction: variation in the size of interior share, EPN-cost, variation in the number of boundary share components (Lemma 34); ASG-cost (Definition 35). Notations: μ is the reduction step, U is the net containing the redex, $d\kappa, r\kappa$ stands for a duplication rule; if involved: $h+1$ is the number of premisses of the ?-link, B is the box subnet, l, l' are the lifts. Also, intervals are enclosed in brackets, sets in braces.

Rule(s)	Proviso	$\Delta\mathsf{iSh}(\mu)$	$\mathcal{C}^{\mathsf{EPN}}_{\mathsf{UG}}(\mu)$	$\Delta\mathsf{BShC}(\mu)$	$\mathcal{C}^{\mathsf{ASG}}_{\mathsf{UG}}(\mu)$
$(\multimap)$	$\mu \notin \mathsf{Sh}(U)$	0	9	0	9
	$\mu \in \mathsf{Sh}(U)$	$-9 + \Delta\mathsf{BShC}(\mu)$	$18 - \Delta\mathsf{BShC}(\mu)$	$[0,2]$	0
$(!)$	$\mu \notin \mathsf{Sh}(U)$	0	6	0	6
	$\mu \in \mathsf{Sh}(U)$	-6	12	$[0,1]$	0
(tD)	$\mu \notin \mathsf{Sh}(U)$	$h \times \#\mathcal{E}(B) - h$	$3h+4$	$\{0,h\}$	$h+4$
	$\mu \in \mathsf{Sh}(U)$	$h \times \#\mathcal{E}(B) - 3h - 6$	$5h+10$	0	0
$(d!)$	$l \in \mathsf{bLft}(U)$	$-3 + \Delta\mathsf{BShC}(\mu)$	$3 - \Delta\mathsf{BShC}(\mu)$	$[0,1]$	3
$(d\multimap), (d\multimap), (r\multimap)$	$l \in \mathsf{bLft}(U)$	$-5 + \Delta\mathsf{BShC}(\mu)$	$5 - \Delta\mathsf{BShC}(\mu)$	$[0,2]$	5
$(d?), (r?)$	$l \in \mathsf{bLft}(U)$	$-2h - 3 + \Delta\mathsf{BShC}(\mu)$	$2h+3 - \Delta\mathsf{BShC}(\mu)$	$[0,h+1]$	$2h+3$
$(d\kappa), (r\kappa)$	$l \notin \mathsf{bLft}(U)$	0	0	0	0
(a)	$l, l' \in \mathsf{bLft}(U)$	0	0	-1	1
	otherwise	0	0	0	0
(s)	$l, l' \notin \mathsf{bLft}(U)$, $l, l' \in \mathsf{bLft}(\mu(U))$	$-1 + \Delta\mathsf{BShC}(\mu)$	$1 - \Delta\mathsf{BShC}(\mu)$	$[0,1]$	1
	otherwise	0	0	0	0
(m)	$l \in \mathsf{bLft}(U)$	0	0	-1	1
	otherwise	0	0	0	0

v is linked to a link other than a lift, then it is *boundary-limit*, written $v \in \mathsf{BLSh}(U)$. In such a case, the *boundary lift chain* of v is the longest sequence of lifts that induces a path from v to the conclusion of a boundary lift. If $\mathsf{Sh}(U) \ni v \notin \mathsf{BSh}(U)$ and v is a ι-vertex of a lift, then v is a *pseudo-boundary vertex*, the set of which is denoted by $\mathsf{bSh}(U)$. If $\mathsf{Sh}(U) \ni v \notin \mathsf{BSh}(U) \cup \mathsf{bSh}(U)$, then v is an *interior vertex*, the set of which is $\mathsf{iSh}(U)$. A share component having no interior vertices is a *boundary component*, and $\mathsf{BShC}(U)$ denotes the set of such components.

4.3.2 Reading proof-net cost and sharing cost on unshared reduction

Now we transpose $\mathcal{C}_{\mathsf{EPN}}$ and $\mathcal{C}_{\mathsf{ASG}}$ as two cost notions on UG reduction. *On the proof-net side* we define $\mathcal{C}^{\mathsf{EPN}}_{\mathsf{UG}}$, by subtracting from $\mathcal{C}_{\mathsf{EPN}}$ the variations in the size of the internal share. The first intuition is that the $\#\mathsf{iSh}(\cdot)$ represents a buffer for the amount of duplication work that is performed in big steps by EPN and delayed in small steps by ASG. In such a way, we obtain a definition of $\mathcal{C}^{\mathsf{EPN}}_{\mathsf{UG}}$ that is bounded by constant on any step, including (tD). The second intuition is that a reduction inside $\mathsf{iSh}(\cdot)$ cost up to twice the cost it would have $\mathcal{C}_{\mathsf{EPN}}$, since we need to account not only for the usual graph-rewriting cost, but also for the duplication cost previously performed of the involved links. Finally, we define and compute another notion of cost, $\mathcal{C}^{\mathsf{BShC}}_{\mathsf{UG}}$, which accounts the amount of reductions on boundary lifts in boundary share components. For the moment it only enables simplifications in the computation of $\mathcal{C}^{\mathsf{EPN}}_{\mathsf{UG}}$, but it will play a crucial role later.

▶ **Definition 32** (EPN metrics on UG)**.** Recall the notions of size and variation from Definition 16. Given $U \xrightarrow{\mu}_{\mathsf{UGR}} U'$, the *partial EPN-cost of* μ, is defined as $\mathcal{C}^{\mathsf{EPN}}_{\mathsf{UG}}(\mu) = \mathcal{C}_{\mathsf{EPN}}(\rho) - \Delta\mathsf{iSh}(\mu)$, while the *full* one is defined as $\overline{\mathcal{C}}^{\mathsf{EPN}}_{\mathsf{UG}}(\mu) = \mathcal{C}^{\mathsf{EPN}}_{\mathsf{UG}}(\mu) + \#\mathsf{iSh}(U)$. The *boundary-share-components cost* of μ is $\mathcal{C}^{\mathsf{BShC}}_{\mathsf{UG}}(\mu) = |\Delta\mathsf{BShC}(\mu)|$.

▶ **Fact 33** (Correctness of $\overline{\mathcal{C}}^{\mathsf{EPN}}_{\mathsf{UG}}$)**.** *Let* $N \xrightarrow{\bar{\mu}}{}^{*}_{UG} U$ *and* $N \xrightarrow{\bar{\rho}}{}^{*}_{UG} N'$ *such that* $\bar{\mu} \mapsto \bar{\rho}$*. Then* $\mathcal{C}_{EPN}(\bar{\rho}) = \overline{\mathcal{C}}^{EPN}_{UG}(\bar{\mu})$.

▶ **Lemma 34** (Metrics on UGR-reduction). *Let μ be a UGR step. The possible values of $\Delta\mathsf{iSh}(\mu)$, $\mathcal{C}^{EPN}_{UG}(\mu)$ and $\Delta\mathsf{BShC}(\mu)$ are in Table 2.*

On the sharing graphs side we define $\mathcal{C}^{\mathsf{ASG}}_{\mathsf{UG}}$ as follows: logical redexes are accounted for only if they are a master copy; other redexes are accounted for only if the involved lift is a boundary one. Intuitively, we distribute a $k+1$-ary mux duplication into k propagations of boundary lifts, and a step of another kind into the unique *master copy* of its redex.

▶ **Definition 35** (Cost on unshared graph). Let μ be a UGR-reduction step. The *ASG-cost* of μ, written $\mathcal{C}^{\mathsf{ASG}}_{\mathsf{UG}}(\mu_i)$, is defined in the rightmost column of Table 2.

Let $N \in EPN$, $U \in UG$ and $G \in ASG$ such that $N \xrightarrow{\bar{\sigma}}{}^{*}_{ASG} G$ and $N \xrightarrow{\bar{\mu}}{}^{*}_{UGR} U$, with $\bar{\sigma} \hookrightarrow \bar{\mu}$.

▶ **Lemma 36** (Master copy). *If $v \in V(G)$ and $V' \leftarrow v$, then V' contains a unique master.*

▶ **Lemma 37** (Correctness of $\mathcal{C}^{\mathsf{ASG}}_{\mathsf{UG}}$). $\mathcal{C}_{ASG}(\bar{\sigma}) = \mathcal{C}^{ASG}_{UG}(\bar{\mu})$.

4.3.3 Comparison of the two unshared costs

Finally, we now compare $\mathcal{C}^{\mathsf{EPN}}_{\mathsf{UG}}(\bar{\mu})$ and $\mathcal{C}^{\mathsf{ASG}}_{\mathsf{UG}}(\bar{\mu})$, and find their difference to be bounded by $\mathcal{C}^{\mathsf{BShC}}_{\mathsf{UG}}(\bar{\mu})$. By a mere local observation we find a linear bound in $\mathcal{C}^{\mathsf{EPN}}_{\mathsf{UG}}$ for the portion of $\mathcal{C}^{\mathsf{BShC}}_{\mathsf{UG}}$ induced by logical, duplicating, merging and annihilation rules. For the portion caused by swap rules, instead, we find a quadratic limitation. Therefore, the overhead of ASG with respect to its EPN simulation admits a quadratic bound.

▶ **Lemma 38.** *Let $N \in EPN$, $U \in UG$, s.t. $N \xrightarrow{\bar{\mu}}{}^{*}_{UGR} U$. Then $\mathcal{C}^{ASG}_{UG}(\bar{\mu}) - \mathcal{C}^{\mathsf{BShC}}_{UG}(\bar{\mu}) \leq \mathcal{C}^{EPN}_{UG}(\bar{\mu})$.*

▶ **Lemma 39** (Bound to $\mathcal{C}^{\mathsf{BShC}}_{\mathsf{UG}}$). *For any $N \in EPN$ and any sequence $\bar{\mu}$ of UGR-reduction on N, there exists a quadratic function q such that $\mathcal{C}^{\mathsf{BShC}}_{UG}(\bar{\mu}) \leq q(\mathcal{C}^{EPN}_{UG}(\bar{\mu}))$.*

▶ **Theorem 40** (Complexity comparison). *Let $N, N' \in EPN$, $G \in ASG$ such that $N \xrightarrow{\bar{\sigma}}{}^{*}_{ASGR} G$ and $N \xrightarrow{\bar{\rho}}{}^{*}_{EPN} N'$, where $\bar{\rho} \twoheadrightarrow \bar{\sigma}$. Then $\mathcal{C}_{ASG}(\bar{\sigma}) \leq q\,(\mathcal{C}_{EPN}(\bar{\rho}))$ where q is a quadratic function.*

5 Conclusions

Two reflections and four questions emerge from the study we have presented.

Discussion

1. *A quite positive partial answer to the efficiency of sharing graphs* comes from the quadratic upper-limit to the complexity of their reductions. This is motivated by two arguments.
   - **a.** Hypotheses of our complexity measurement were purposely extremely conservative. Indeed, we not only consider the read-back – an unavoidable portion of the reduction work – by directly including their rewriting rules. But we also allow these rules to be applied freely, also before the end of the β-normalisation, thus allowing duplications of redexes in a sub-optimal fashion [20].
   - **b.** The worst-case overhead of sharing graphs are usually counterbalanced by other benefits. Laziness in the strategy of duplication, for instance, has shown speed-ups up to exponential size [3]. Locality and asynchronicity of the computational model, moreover, allow parallelisable implementation with little effort.

2. *The cost of local duplication is legitimate.* Normalisation with sharing graphs of some ELL-typed λ-terms may cause an elementary explosion in the number of local duplication rules (mux propagations) [2]. This should not surprise, because simply-typed terms in general may require an implementation cost that is more than elementary [22]. To further clarify this point, Lemma 38 shows that duplications performed by sharing graphs have a cost that is linearly bounded by the cost of proof-net reduction.

Open questions

1. Can we improve the quadratic bound? Or there is instead a λ-term typeable in ELL or LLL whose sharing normalisation requires indeed a cost that is quadratic with respect to proof-nets normalisation?
2. Does a similar complexity upper-bound hold for the more general cases of λ-calculus and MELL?
3. Complementarily, is there also a lower bound giving theoretical evidence of performance gains?
4. Can our bound be directly related to the number of β-steps in the leftmost-outermost strategy on the λ-calculus [9]? Are sharing graphs themselves are a reasonable and parsimonious cost model the λ-calculus?

References

1 Andrea Asperti. About the efficient reduction of lambda terms. *arXiv*, January 2017. 1701.04240. URL: `http://arxiv.org/abs/1701.04240`.

2 Andrea Asperti, Paolo Coppola, and Simone Martini. (Optimal) duplication is not elementary recursive. *Inform. and Comput.*, 193(1):21–56, 2004. `doi:10.1016/j.ic.2004.05.001`.

3 Andrea Asperti, Cecilia Giovannetti, and Andrea Naletto. The Bologna optimal higher-order machine. *Journal of Functional Programming*, 6(6):763–810, November 1996. `https://github.com/asperti/BOHM1.1`. `doi:10.1017/S0956796800001994`.

4 Andrea Asperti and Stefano Guerrini. *The Optimal Implementation of Functional Programming Languages*, volume 45 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1998.

5 Andrea Asperti and Harry G. Mairson. Parallel Beta Reduction Is Not Elementary Recursive. *Inf. Comput.*, 170(1):49–80, 2001. `doi:10.1006/inco.2001.2869`.

6 Andrea Asperti and Luca Roversi. Intuitionistic light affine logic. *ACM Trans. Comput. Logic*, 3:137–175, January 2002. `doi:10.1145/504077.504081`.

7 Patrick Baillot, Paolo Coppola, and Ugo Dal Lago. Light logics and optimal reduction: Completeness and complexity. *Information and Computation*, 209(2):118–142, February 2011. `doi:10.1016/j.ic.2010.10.002`.

8 Ugo Dal Lago. Context semantics, linear logic, and computational complexity. *ACM Transactions on Computational Logic (TOCL)*, 10(4):25:1–25:32, August 2009. `doi:10.1145/1555746.1555749`.

9 Ugo Dal Lago and Beniamino Accattoli. (Leftmost-Outermost) Beta Reduction is Invariant, Indeed. *Logical Methods in Computer Science*, 12, 2016. `doi:10.2168/LMCS-12(1:4)2016`.

10 Ugo Dal Lago and Simone Martini. On Constructor Rewrite Systems and the Lambda-Calculus. In *Automata, Languages and Programming*, pages 163–174. Springer, July 2009. `doi:10.1007/978-3-642-02930-1_14`.

11 J.-Y. Girard. Geometry of interaction I. Interpretation of system F. In R. Ferro, C. Bonotto, S. Valentini, and A. Zanardo, editors, *Logic colloquium 1988*, volume 127 of *Studies in*

Logic and The Foundations of Mathematics, pages 221–260. North-Holland, 1989. doi:10.2277/978-0521621120.

12 Jean-Yves Girard. Light linear logic. *Information and Computation*, 143(2):175–204, 1998. doi:10.1006/inco.1998.2700.

13 Georges Gonthier, Martín Abadi, and Jean-Jacques Lévy. The geometry of optimal lambda reduction. In *Conference record of the Nineteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'92)*, pages 15–26, Albequerque, New Mexico, January 1992.

14 Stefano Guerrini, Thomas Leventis, and Marco Solieri. Deep into optimality – complexity and correctness of sharing implementation of bounded logics. Third International Workshop on Developments in Implicit Complexity, 2012.

15 Stefano Guerrini, Simone Martini, and Andrea Masini. Modal Logic, Linear Logic, Optimal Lambda-Reduction. In *Logic and Foundations of Mathematics*, number 280 in Synthese Library, pages 271–282. Springer, 1999. doi:10.1007/978-94-017-2109-7_20.

16 Stefano Guerrini, Simone Martini, and Andrea Masini. Coherence for sharing proof-nets. *Theoretical Computer Science*, 294(3):379–409, February 2003. doi:10.1016/S0304-3975(01)00162-1.

17 Pavol Hell and Jaroslav Nesetril. *Graphs and homomorphisms*. Oxford Univ. Press, 2004.

18 Yves Lafont. Interaction nets. In *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL'90, pages 95–108, New York, NY, USA, 1990. ACM. doi:10.1145/96709.96718.

19 John Lamping. An algorithm for optimal lambda calculus reduction. In *Proc. of Seventeenth Annual ACM Symposyum on Principles of Programming Languages*, pages 16–30, San Francisco, California, January 1990. doi:10.1145/96709.96711.

20 Jean-Jacques Lévy. Optimal reductions in the lambda-calculus. In Jonathan P. Seldin and J. Roger Hindley, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 159–191. Academic Press, 1980.

21 Simon L. Peyton Jones. Implementing lazy functional languages on stock hardware: the Spineless Tagless G-machine. *Journal of Functional Programming*, 2(2):127–202, April 1992. doi:10.1017/S0956796800000319.

22 Richard Statman. The typed λ-calculus is not elementary recursive. *Theoretical Computer Science*, 9(1):73–81, 1979. doi:10.1016/0304-3975(79)90007-0.

23 Peter van Emde Boas. *Machine models and simulations, Handbook of theoretical computer science (vol. A): algorithms and complexity*. MIT Press, Cambridge, MA, 1991.

24 C. P. Wadsworth. *Semantics and pragmatics of the lambda-calculus*. PhD thesis, University of Oxford, 1971.