Sharing Implementations of Graph Rewriting Systems

Stefano Guerrini [1,2]

Università Roma La Sapienza
Dipartimento di Informatica
Via Salaria, 113
00198 Roma, Italy

Abstract

Sharing graphs are a brilliant solution to the implementation of Lévy optimal reductions of λ-calculus. Sharing graphs are interesting on their own and optimal sharing reductions are just a particular reduction strategy of a more general sharing reduction system.

The paper is a gentle introduction to sharing graphs and tries to confute some of the common myths on the difficulty of sharing implementations.

Key words: Sharing graphs, optimal reductions, graph-rewriting, λ-calculus, linear logic.

1 Introduction

Sharing graphs have been introduced by Lamping [Lam89,Lam90] to implement Lévy optimal reduction for λ-calculus [Lév78,Lév80]. Nevertheless, Lamping's sharing graphs are not just a brilliant and ad-hoc solution to an important problem of λ-calculus. In fact, Gonthier, Abadi and Lévy [GAL92a,GAL92b] reformulated Lamping's algorithm inside Geometry of Interaction [Gir89] by means of a generalization of Lafont's Interaction Nets [Laf90], and Asperti and Laneve studied the problem of (optimal) sharing implementations for a general class of higher order rewriting systems that they named Interaction Systems [AL94,AL96].

[1] Partially supported by the MURST PRIN "Fondazioni Logiche di Linguaggi Astratti di Programmazione" and by the projects of the University Roma La Sapienza "Analisi e modelli di sistemi reattivi e multimediali" and "Metodi formali per la progettazione di hardware e software"

[2] Email: `guerrini@di.uniroma1.it`

1.1 Lamping

The main tools introduced by Lamping are a set of graph rewriting operators for sharing and unsharing and for the control of the scope of the sharing operators. Despite the relevance widely acknowledged to Lamping's breakthrough and to the successive studies on sharing graphs, there remain some common myths that bride a widen usage of this technique. The more common criticisms are that sharing graphs are (i) a hard topic to grasp, (ii) difficult to implement, (iii) suitable for λ-calculus optimal reductions only, (iv) inefficient because of Asperti-Mairson's result [AM01] that showed that there is no polynomial bound between the number of shared β-reductions and the whole cost of the reduction.

The paper is a gentle introduction to sharing graphs and tries to confute some of the previous common myths. In particular, the aim is at showing that, on the contrary, sharing graphs are

(i) an easy and natural approach to the implementation of rewriting rules *à la* β-rule of λ-calculus,

(ii) easily implementable,

(iii) applicable to any "orthogonal" graph rewriting system,

(iv) an optimal implementation (according to Lévy) of the underlying system, as efficient as the underlying system.

The real reason that has led to the common myths on the complexity of sharing graphs is the difficulty of the proof of soundness for sharing graphs reductions. But this should not have impact on the application of the result, for instance, in algebra, we commonly use many simple and useful results in spite of the complexity of their proofs.

1.2 The structure of the paper

In the following, we shall follow the general approach to sharing implementations presented in [Gue99]. For an almost complete account of the main research on sharing implementations we refer to [AG98].

In section 2, we shall introduce graph reduction of λ-terms. We shall see the first approach given by Wadsworth [Wad71] and we shall give a first implementation of β-reduction by means of atomic and local graph reduction rules that, however, does not allow any sharing. At the end of the section, we shall see which are the problems of the approach given in section 2 if one wants to allow sharing.

In section 3, we shall discuss a solution to the problem of sharing suggested by the use of Linear Logic, in particular, by the introduction of the boxes introduced by the encoding $A \to B = !A \multimap B$. At the end of the section, we shall also see that a particular reduction strategy of the sharing graph rewriting system presented in the first part of the section is an implementation of Lévy optimal reductions.

In section 4, we shall conclude with an analysis of the results in the paper and with some words on the meaning of Asperti-Mairson's result on the relation between the minimal number of β-rules fired in the shared reduction of a λ-term and the actual complexity of the reduction of that term.

2 λ-calculus Graph Reduction

Looking at the β-rule of λ-calculus from the point of view of its implementation, one can easily see that the execution of $(\lambda x.t)s \longrightarrow_\beta t[s/x]$ involves two main operations:

(i) the assignment of the argument s to the variable x;

(ii) the duplication and displacement of the term s, namely,
   - (a) a new copy of s is created for every occurrence of the variable x,
   - (b) any copy of s is put in the place of the corresponding occurrence of the variable x.

If we represent λ-terms by means of their syntax tree (where the @-node represents application, the λ-node represents λ-abstraction, variable occurrences are represented by $\bullet$-nodes, and the binding relation is represented by the dotted line joining the nodes of a variable to the corresponding binder), the rewriting corresponding to a β-reduction is depicted in Figure 1.

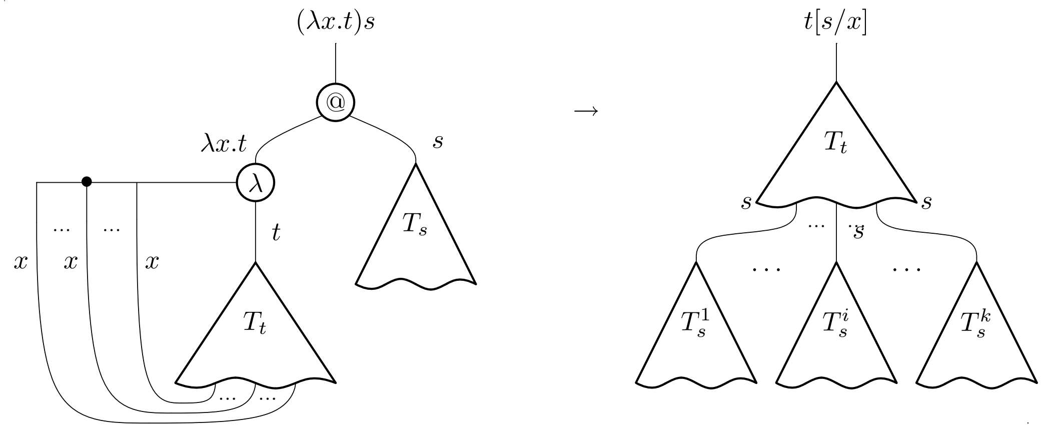

Fig. 1. Term graph implementation of β-rule

The previous considerations also lead to observe that a direct implementation of β-rule (as for instance the term graph rewriting in Figure 1) is global, namely, it is not an atomic step of some abstract reduction machine. Therefore, counting the number of β-rules is not a good measure of reduction complexity, for the execution of a β-reduction might require to copy a huge term. On the other hand, the direct reduction of a β-redex $(\lambda x.t)s$ does not seem the best that one can do from the point of view of efficiency, for the duplication of s would cause the duplication of all the redexes in it.

2.1 Lévy optimal reduction

The aim of Lévy in the definition of his *optimal reductions* was of finding some (parallel) reduction strategy that, at each step, contracts all the redexes "sharable" by a smart reduction machine, avoiding any "useless duplication". By the way, Lévy also had to define the notion of "sharable" and of "useless duplication", or more generally of "useless work", and to motivate them.

The main concept that Lévy introduced to define sharable redexes is that of *redex family*. Given a term t and two redexes r' and r'' in two reducts of t, the redexes r' and r'' are in the same family iff there is a reduction $t \to^* s$ and a redex r of s such that r' and r'' are residuals of r. In other words, two redexes are in the same family iff they have the same origin; as it is also shown by a different characterization of redex families: two redexes are in the same family iff they are residual of the same path in the graph of t (the graph of a term t has been already described: it is the syntax tree of t plus a set of back-connections that link each occurrence of a variable to its binder (w.l.o.g., we can assume that t is closed)). These two characterizations point out the main issue of Lévy families: it is conceptually possible to reduce all the redexes in a family in a single step. For instance, if one takes the reduction $t \to^* s \to u$ where u is obtained by reducing the redex r of s, then, no reduct of s can contain a redex in the same family (a residual) of r. However, it is not possible to find a reduction strategy that reduces the canonical representative of a family at every step. Therefore, we need some astute sharing mechanism that allows to keep always shared the paths of the initial term that during the reduction will become redexes—those paths are sometimes referred to as "virtual redexes".

Family of redexes correspond to sharable redex, while the goal of avoiding any useless work corresponds to the fact that one wants to reduce redexes that must be contracted in any reduction of the given term, and a strategy that allows to avoid useless work is the leftmost-outermost, also known as the normal strategy.

We shall see that sharing graphs allow to reach optimal reductions, as there are reduction strategies that avoid the duplication of paths that can become redexes.

2.2 Graph Reduction

Let us come back to the reduction in Figure 1, some sharing can be reached by using DAGs in the place of trees. In order to avoid the duplication of the term s, we define a new version of the β-rule that replaces the pointers from the occurrences of the variable x to the corresponding binder with pointers to the term s, see Figure 2. Nevertheless, this approach does not completely solve the problem of the need for global copying of terms. In fact, let us assume that, in the example in Figure 2, $s = \lambda y.s'$ and $t = f(xu)(xv)$. The next step can be to reduce either su or sv. Let us reduce su first. It is readily seen that

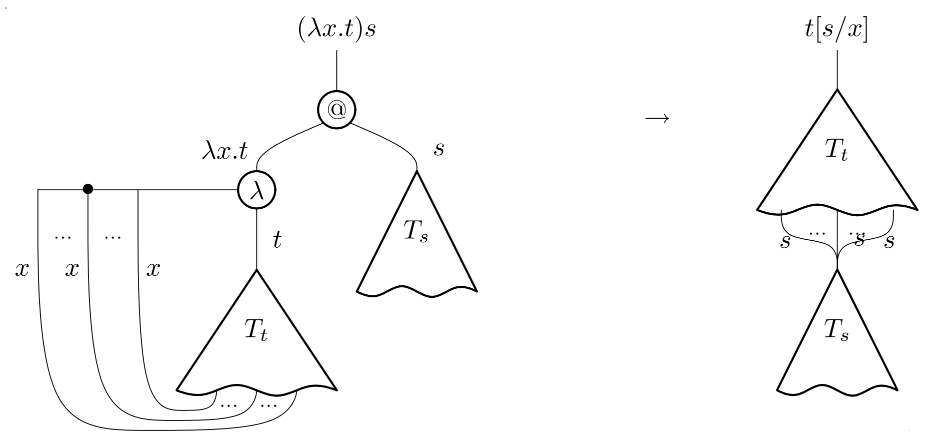

Fig. 2. Wadsworth's graph implementation of β-rule

this cannot be done by replacing u for y in the body of the unique copy of $s = \lambda y.s'$ shared by the redexes, otherwise, this would have the unwanted side effect of replacing $s'[u/y]v$ for sv. The solution proposed by Wadsworth, and followed by the usual graph reduction approaches [PJ86], is to make a new copy of s and to replace u for y in this new copy only. In this way, a term replacing a variable is duplicated only if it starts with a λ-node and when it becomes the functional part of a β-redex.

2.3 A first step-by-step implementation of duplication

We are again in a situation in which, from time to time, we need to execute the duplication of a whole subterm. We want to find a way (i) to completely decompose the duplication process in atomic steps and (ii) to be able to execute it in a completely lazy way. For instance, in the example previously described, we would like to be able to duplicate just a small part of the term s and not the whole of it.

Let us start with the first part of our goal. We can formulate it in the following terms

- define a graph rewriting system that implements the β-rule by means of local and atomic rewritings only

or, equivalently

- decompose the β-rule into a sequence of atomic graph rewritings.

2.3.1 Sharing β-rule

We introduce a node that explicitly represents sharing, namely, in the β-rule in Figure 2, we introduce a sharing node named *multiplexer*, mux for short, or *fan* between the occurrences of the variable x and the root of s. The corresponding sharing β-rule is depicted in Figure 3.

The port of the mux that points to the term s is the principal port of the node, the other ports are the auxiliary ports of the node. The auxiliary

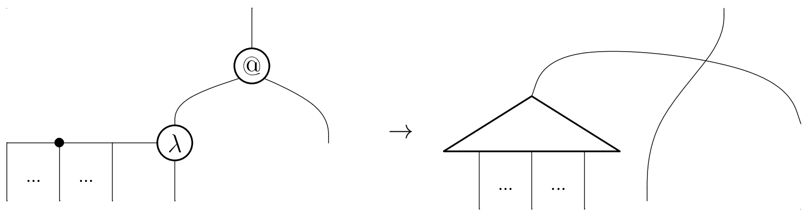

Fig. 3. Sharing β-rule (basic idea)

ports of a mux are distinct, or equivalently, they have different names. In the following, we shall use binary muxes only and in the picture we shall assume that one of the two ports is marked with a $*$ and the other one with a $\circ$.

2.3.2 Mux propagation rules

After the execution of a sharing β-rule, we want a way to completely duplicate the term s (as in the case of the rule in Figure 1) by a step-by-step duplication of the nodes of s obtained by propagating the mux in s. The mux propagation rules that implement such a duplication are depicted in Figure 4 where, to simplify the presentation, we have drawn the rules for the case of binary muxes only, the generalization to the n-ary case is straightforward.

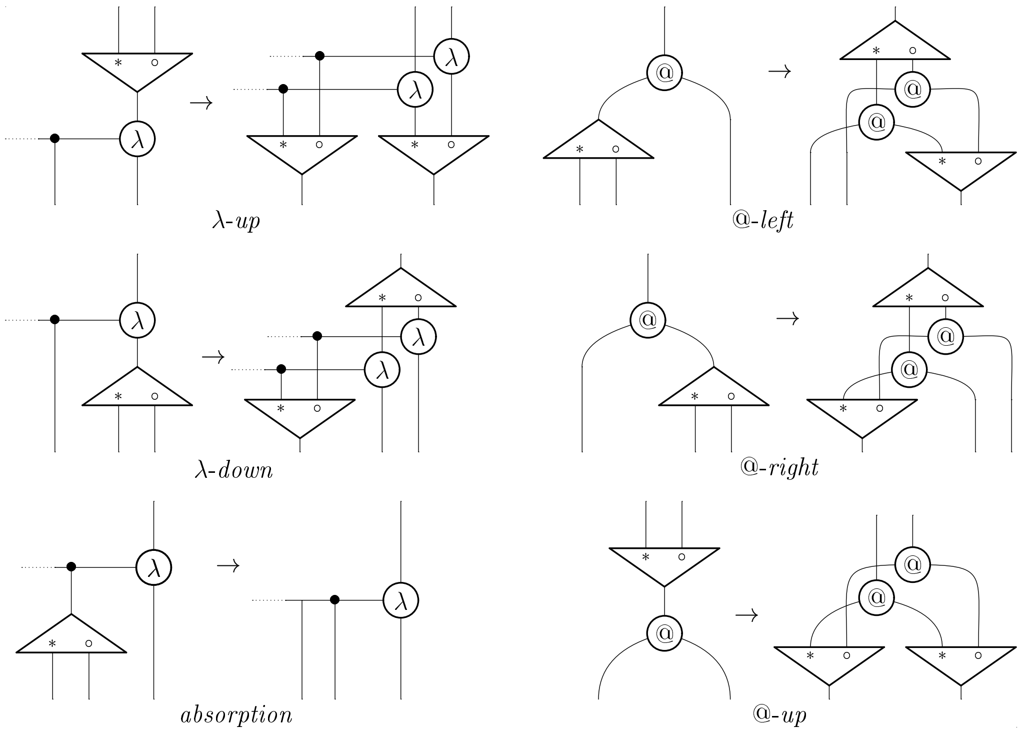

Fig. 4. Mux propagation rules

Let us remark that, when a mux reaches a λ/@-node, the mux duplicates the node and splits in two copies. We can say that the duplication thread of the original mux gives rise to two new duplication threads that can proceed independently and in parallel. Moreover, duplication threads do not propagate downward only, but upward too, according to the orientation of the corresponding mux. E.g., a λ-up rule introduces a mux that propagates upward (from the variables inside the term).

2.3.3 The absorption rule

The absorption rule requires some additional words.

- Absorption is the only rule of the previous ones that erases a mux.
- Absorption corresponds to the end of a duplication thread: it corresponds to a duplication that reaches a free variable of the duplicating term. That is, in the sharing reduction of the redex $(\lambda x.t)s$, it corresponds to a mux that reaches the occurrence of a variable that is free in s.
- Absorption is not always sound: if during the reduction of the redex $(\lambda x.t)s$ a mux reaches the occurrence of a variable bound in s, absorption is not sound. In that case, it would be sound to duplicate the λ-node instead, as in the case of a λ-down rule.

2.3.4 Mux annihilation rule

As already pointed out, we can distinguish two kinds of (complementary) muxes. According to the natural vertical orientation induced by the graph representation of terms, we have that

- a *positive mux* or *fan-in* node is downward oriented and duplicates the tree of a λ-term moving towards the leaves;
- a *negative mux* or *demux* or *fan-out* node is upward oriented and duplicates the tree of a λ-term moving towards the root.

When two complementary muxes face, we have the case of two duplication threads that meet: one duplicating the part of the tree below it, the other one duplicating the part of the tree above it. However, both the muxes have nothing more to duplicate. In fact, the part of the tree below the positive mux has been already duplicated by the negative one, and vice versa. Therefore, the two muxes have completed their tasks and may annihilate, that is, we may replace the two muxes with direct connections between pairs of ports with the same name, as described by the rule in Figure 5.

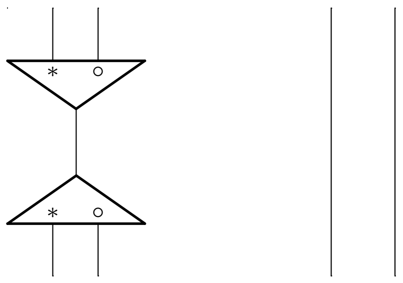

Fig. 5. Mux annihilation rule

2.3.5 Soundness

Let us restrict w.l.o.g. to closed λ-terms. The sharing β-rule in Figure 3 plus the mux propagation rules in Figure 4 plus the mux annihilation rule in Figure 5 give a sound implementation of β-rule provided that one applies the following algorithm:

(i) rewrite a β-redex by means of a sharing β-rule;

(ii) before the firing of another sharing β-rule or of any absorption rule, propagate all the muxes in the graph until they disappear or they reach the binding port of a λ-node;

(iii) remove the muxes left in the graph by means of absorption rules.

Let us remark that, in step (ii), no absorption rule can be fired and that, at the end of this phase, all the muxes left in the graph are at the binding ports of λ-nodes.

More formally, if one denotes by $\to_{\not a}$ the reduction relation induced by the mux propagation rules without absorption plus the mux annihilation rule, by $\to_a$ the reduction relation induced by the absorption rule only, and by $\to_{\beta_s}$ the reduction relation induced by the sharing β-rule, we have the following proposition.

Proposition 2.1 *Let T be the graph representation of a term t. If $t \to t'$, then $T \to_{\beta_s} G_1 \to^*_{\not a} G_2 \to^*_a T'$, where T' is the graph representation of t'.*

$$\begin{tikzcd}
t \arrow[rrr, "\beta"'] \arrow[d, mapsto] & & & t' \arrow[d, mapsto] & \lambda\text{-terms} \\
T \arrow[r, "\beta_s"'] & G_1 \arrow[r, "*", "\pi_{\not a}"'] & G_2 \arrow[r, "*", "\pi_a"'] & T' & \text{graphs}
\end{tikzcd}$$

2.3.6 Troubles and limits of this solution

Let us remark again that, up to now, the soundness of the graph reduction rules rests on a particular reduction strategy, moreover, for that reduction strategy we have that:

(i) after the execution of a β_s-rule we reduce the graph to a new graph without muxes, namely, we unfold all the sharing introduced by muxes;

(ii) as already remarked in 2.3.3, the absorption rule is sound only after the elimination of all the downward oriented muxes in the graph. Absorption is sound when a mux reaches the binding port of a variable free in the duplicating term t. In the case of a mux that reaches the binding port of a λ-node in t, such a mux is waiting to be erased by a complementary mux. In this case, absorption is not sound, but the rule that allows the mux to duplicate the λ-node passing through its binding port is sound.

Therefore, in order to achieve our goal of a rewriting system that allows to share redexes and that implements reduction by means of atomic steps, we need to relax the constraints forced by their reduction strategy. In particular,

(i) we need to be able to fire a β_s-rule independently of the status of the propagation of the muxes introduced by a previously fired β_s-rule;

(ii) we need a way to determine when the absorption rule is sound that does not depend from the reduction strategy.

2.4 Firing two β_s-rules

2.4.1 The mux swap rule

After reducing a β-redex, if we fire another β_s-rule before the completion of the duplication process started by the first sharing rule, it is no longer true that all the muxes in the graph are duplicating the same term and, as a consequence, it is no longer true that two facing muxes annihilate.

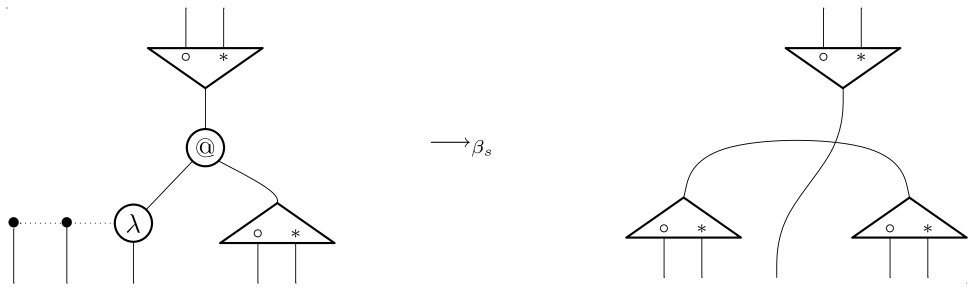

Fig. 6. Muxes originated by distinct β_s-reductions

Figure 6 gives an example in which two facing muxes must not annihilate. After the firing of the β_s-redex in the picture, we get two facing muxes coming from the reduction of two distinct β-redexes. In this case, it is no longer true that the facing muxes may annihilate. Let us remark that (in the pictures of the paper) we are drawing binary muxes only and that in the general case, muxes coming from the execution of distinct β_s-reductions might have different arity. This should make clearer that in the case in Figure 6 annihilation cannot apply, for the facing muxes might also have different arity and we would not know how to connect the opened ports after the annihilation of them. In the case of Figure 7, instead of annihilating, the two facing muxes swap duplicating each-other, as in the case of a mux propagation through a λ or @-node. The corresponding mux swap rule (for the case of binary muxes) is given in Figure 7.

2.4.2 Naming muxes

In order to decide whether to apply swap or annihilation in the case of pairs of facing muxes, we need a way to name muxes s.t.

- two facing muxes with the same name annihilate,

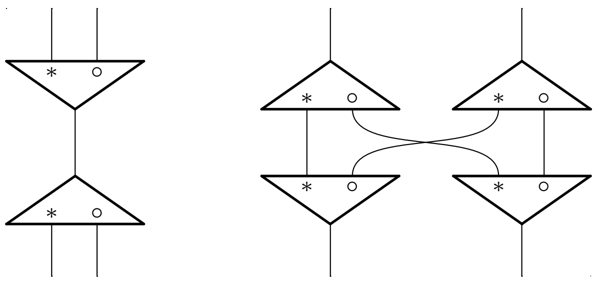

Fig. 7. Mux swap rule

- two facing muxes with different names swap.

We have already seen that two muxes originated by the execution of two distinct β_s-rules must swap. This suggests to assign names to muxes at their creation, in such a way that, muxes that have been originated by the execution of the same β_s-rule have the same name, while muxes that have been originated by distinct β_s-rule have distinct names. Unfortunately, because of the higher-order nature of λ-calculus, there are cases—even in typed λ-calculus—in which two copies of the same mux must swap (an example is given in see [Lam89,Lam90]). However, we remark that static naming of muxes works in the relevant case of the systems with bound complexity coming from the elementary and light linear logic proposed by Girard [Gir98].

3 Sharing reductions

Linear logic gives a way to dynamically assign names to muxes in order to be able to decide when to apply annihilation or swap and, as we shall see, to solve the problem of when absorption is sound.

3.1 Boxes and graphs with levels

In Proof Nets for Linear Logic, a *box* is a subnet with a unique *principal door* and many *auxiliary doors*. In MELL (Multiplicative Exponential Linear Logic), a box corresponds to the introduction of an !-exponential, then, according to the most used MELL translation of λ-calculus

$$A \to B = !A \multimap B$$

the argument of an application is always enclosed into a box. Moreover, a box always corresponds to a subterm: the principal door of the box corresponds to the root of the tree of the subterm, the auxiliary doors correspond to the free variables of the subterm.

3.1.1 Exponential links

Let us introduce two new nodes or links[3] for the principal and auxiliary doors of boxes, the !-link and the ?-link depicted in Figure 8. Every !-link denotes

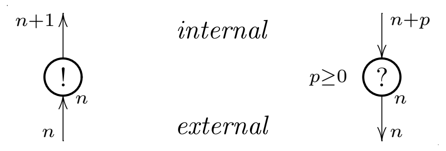

Fig. 8. Exponential links

the principal door of a box: its conclusion, the external port, is always the conclusion of exactly one box. The ?-links correspond instead to the auxiliary doors of boxes: every ?-link can be the auxiliary door of many boxes. The key property of boxes is that they properly nest.

We remark the orientation of the arrows in Figure 8. The orientation of the arrows of the !-link does not correspond to that usually adopted for the exponential links of MELL. In fact, the conclusion of the !-link (the port external to the box associated to the link) is not the port corresponding to the arrow emerging from the link, but the bottom one. The orientation of the arrows has been chosen in this way in order to preserve the natural orientation of the trees associated to λ-terms.

Definition 3.1 [box nesting property] Two boxes B_1 and B_2 of a proof net that have the box nesting property cannot partially overlap, namely, either B_1 and B_2 are disjoint, or B_1 is in B_2, or B_2 is in B_1.

Usually, boxes are represented by a global map that assigns to each !-box the set of the links in its box. In the case that we are considering, boxes are connected, this allows to give a representation of boxes by means of indexes.

3.1.2 Contraction

In the graph representation of λ-terms used in the previous section, we have (back-)connected every occurrence of a bound variable to the binding port of the corresponding λ-node. In proof nets, a λ-link corresponds to a $\parr$-link, that have three ports only (two premises and one conclusion). Binding several occurrences of a variable implies then a contraction.

In the graphs that we shall use in the following, we shall introduce a contraction link (see Figure 9) that takes the conclusions of an arbitrary number of ?-links and contracts them to a unique conclusion. As we shall see by the rules for the construction of the graphs representing λ-terms, see Figure 10, the conclusion of a ?-link corresponds to the occurence of a variable x marked by a ?, that is, to a term $?x$. As a contraction denotes the identification of

[3] The graphs that we shall use in the following to represent λ-terms are a particular case of proof nets. Because of this, according to the usual terminology of proof nets, we shall also use link in the place of node.

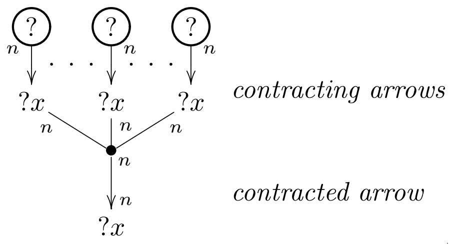

Fig. 9. Contraction

all the occurrences of a given variable, all the premises of a contraction must be marked by a variable named x, and no conclusion of a contraction can be premise of another contraction. The previous provisos have the consequence that the conclusion of a contraction must be connected to the binding port of the λ-link binding x, since this is the only port marked by a $?x$ term (see Figure 10).

3.1.3 Levels

Exploiting the box nesting property, we can assign a level to each link: its *box nesting depth*. The box nesting depth changes at the border of boxes, that is, at the exponential links. In particular, moving from the conclusion of an !-link to its premise, the box nesting depth increases by one, while, moving from the conclusion of a ?-link to its premise, the box nesting depth increases by a positive number p that corresponds to the number of boxes that have that link as auxiliary door (see Figure 8). We can then assign an index, its *level*, to each link and arrow in the graph of a λ-term. The levels assigned to the exponential links were already given in Figure 8. The premises/conclusions of the other links are all at the same level, that is the level of the link also, see Figure 10 and Figure 9.

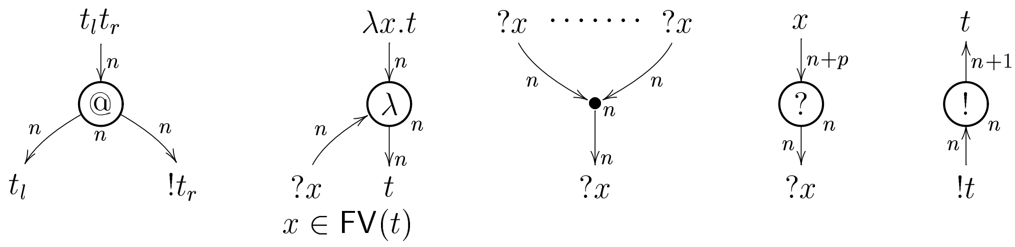

Fig. 10. Links with levels for the representation of λ-terms

The terms at the arrows of the links in Figure 10 and in Figure 9 give the rules for the construction of the syntax tree with levels of a given λ-term. In particular, we see that, at the argument port of an @-link we always have an !-link (this port is marked by an $!t$ term that is the conclusion of a !-link), and that the binding port of any λ-link is the conclusion of a contraction link or of a ?-link.

By means of levels, and by the fact that the boxes that we are considering are connected, we can always recover the box associated to a given !-link.

Definition 3.2 [box] The box of an !-link with level n is the largest connected subgraph of the λ-term with levels that contains the !-link and s.t. all the links, but the ?-links at the auxiliary doors of the box, have a level $> n$.

It is readily seen that, if one opens the connections at the binding ports of the λ-links, the box associated to an !-link corresponds to the subtree of the term rooted at the !-link.

3.1.4 Consequences of the introduction of levels

The introduction of levels implies that muxes too must be indexed and that, after the execution of a β_s-rule, the new copies of the argument must be properly reindexed. Indeed, in the setting with levels, muxes must duplicate and reindex at the same time the links that cross.

The level of a mux, at his creation, is the level (of the !-link) of the box that it has to duplicate. In this way, such a level can be interpreted as a sort of *threshold*: a mux duplicates and reindex every link that it reaches and that has a level lower than its threshold.

The level or threshold of a mux also allows to recognize when a mux must propagate at the binding port of a λ-link or must be absorbed by it. In fact, when the threshold is lower or equal than the level of the top port of the λ-link, the λ-link is outside the box that the mux is duplicating, otherwise, when the threshold is greater than the level of the top port of the λ-link, the λ-link is inside the box. In the first case, the mux can be absorbed, in the second one, the mux can propagate through the λ-link duplicating it.

3.2 The Sharing Rewriting System

3.2.1 The sharing β-rule

First of all, we have to reformulate the sharing β-rule. The new rule is given in Figure 11. The rule includes the rewriting of the contraction and the exponential links connected to the λ and @-link in the β-redex, in this way, we do not have to add new rules for those links (apart for the extension to them of the mux propagation rules).

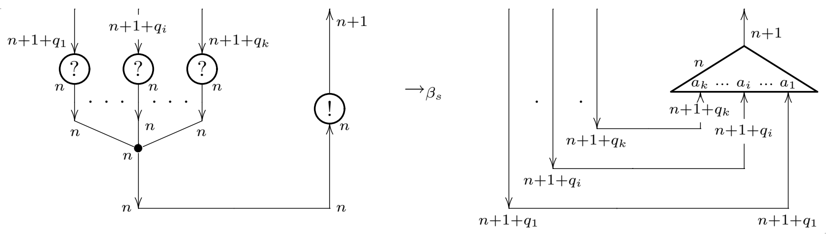

Fig. 11. Sharing β-rule

Let us observe that the difference between the level of the premise of a ?-link and its conclusion is always ≥ 0. In the particular case of a head occurrence of the bound variable (i.e., in a term $\lambda x.xt_1 \dots t_n$), the level of the premise of the ?-link corresponding to that variable is equal to the level of its conclusion. In fact, in $\lambda x.xt_1 \dots t_n$, there is no box enclosing the head occurrence of x.

Let us assume that the difference between the conclusion and the premise of a ?-link in a redex at level n is $q{+}1$. The β_s-reduction of the redex introduces a mux whose principal port is at level $n + 1$ and whose auxiliary port corresponding to the premise of the given ?-link is at level $n{+}q{+}1$, that is, the difference between the principal port and the given auxiliary port is q. Such a value is the *offset* of the port and, by analysis of the rules in Figure 12 and 13, we see that the level of every new copy of a link duplicated by the mux is increased by the offset q of the port that has created the new link. That offset corresponds to the two actions that one has to perform in order to replace a new copy of a give argument/box for an occurrence of a variable enclosed by $q{+}1$ boxes: first of all we have to open the box surrounding the box, and this means to decrease by 1 the levels of every link in the argument, then we have to push inside $q + 1$ boxes the argument, and this means to increase by $q + 1$ the levels of the links in the argument.

We remark that, in the particular case of a head occurrence, the offset is -1, that is, the levels of an argument replacing such a variable are just decreased by 1.

3.2.2 The π-rules

The mux propagation rules are the natural extension of those in Figure 4 and are given in Figure 12. The α-propagation applies whenever a mux reaches any port of a λ-link, of an @-link or of a contraction link. The rule requires that the threshold m of the mux is lower than the level n of the α-link. We shall not prove that but, if one starts with a properly indexed graph, the relation $m < n$ is always verified. The !/?-propagation rule deals with the case of the !-link and of the ?-link. If m is the threshold of the mux and n_2 if the level of the port to which the mux is not connected, the rule apply when $m < n_2$. It is possible to see that the only case in which $m < n_2$ may not hold is when the mux is connected to a ?-link and the level of the link is lower or equal than n (in general, there are more possible cases but, if one starts with a correct graph, that cases cannot arise), that case is covered by the absorption rule (the last rule in Figure 12). The meaning of that rule should be clear: the level of the links in the box that a mux has to duplicate are higher than the threshold of the mux, therefore, when the mux reaches a ?-link whose level is lower than the threshold, it means that it has reached the border of the box on which it has to operate and it can be absorbed.

The mux annihilation and swap rules are given in Figure 13. Two muxes annihilate if they meet with the same threshold, or swap otherwise. Moreover, the mux with the lower threshold operates on the mux with the higher threshold, increasing its threshold in the same way in which it increases the

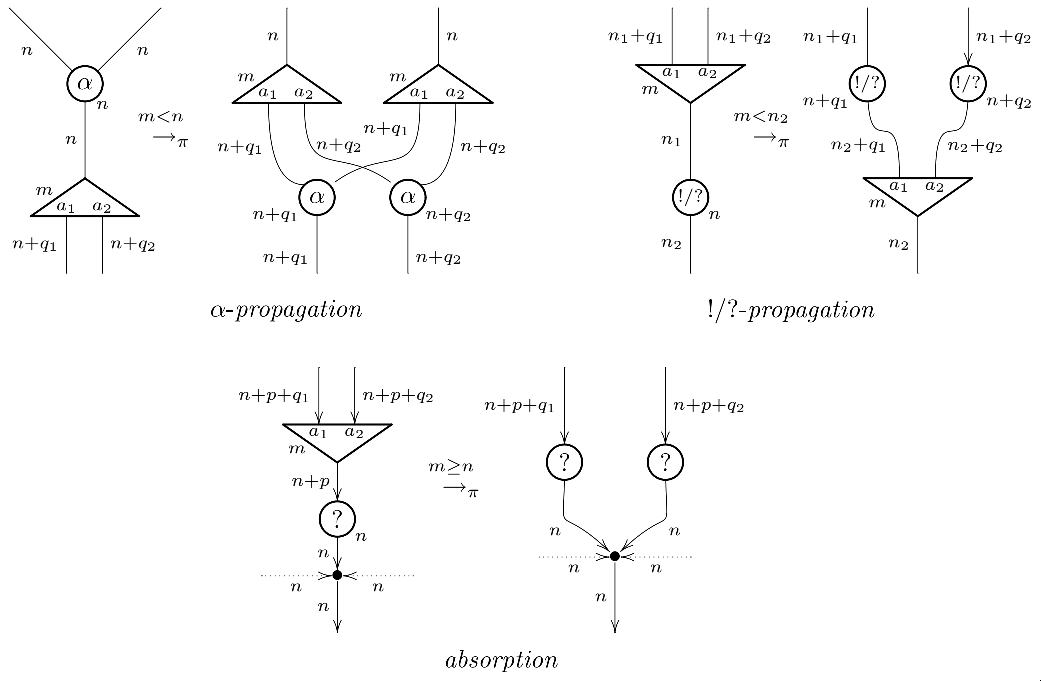

Fig. 12. π-rules: the mux propagation rules

level of the link in an α-rule.

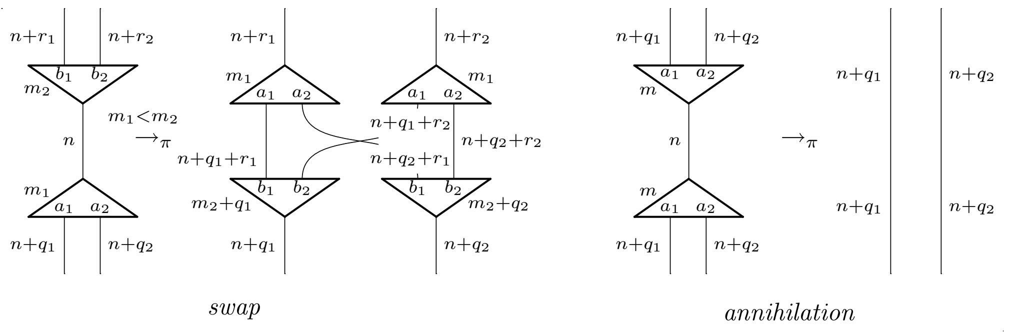

Fig. 13. π-rules: the mux interaction rules

The propagation rules in Figure 12 plus the mux rules (Figure 13) are the π-rules.

3.3 Main Results

The β_s-rules in Figure 11 and the π-rules in Figure 12 and Figure 13 allow to implement β-reduction in a complete shared and asynchronous way.

Theorem 3.3 (soundness) *Let T be the graph representation of a term t s.t. $T \rightarrow^* G$, then the π-normal form $T' = \mathsf{NF}_\pi(G)$ of G is unique and there is $t \rightarrow^* t'$ s.t. T' is the graph representation of t'.*

$$\begin{tikzcd}
T \arrow[r, "*"] & G \arrow[r, "*", "\pi"'] & T' = \mathsf{NF}_\pi(G) & \text{graphs} \\
t \arrow[u, mapsto] \arrow[rr, "*", "\beta"'] & & t' \arrow[u, mapsto] & \lambda\text{-terms}
\end{tikzcd}$$

By Theorem 3.3, we have that every sharing reduction $T \to^* G$ of the graph representation T of a λ-term t corresponds to some reduction $t \to^* t'$. The graph G is a shared representation of the λ-term t' and, since the graph representation T' of t' is the π-normal form of G, we have that the π-rules completely implement sharing unfolding, giving a readback algorithm that allows to find the λ-term represented by a sharing graph.

On the other side, every reduction of a λ-term t has a corresponding reduction of the graph T representing t. This generalization of Proposition 2.1 is a consequence of the fact that, when $t \to t_1$, after the reduction $T \to_{\beta_s} G_1$ that fires the redex of T corresponding to the redex fired in t, we have $G_1 \to_\pi^* T_1$ (by Theorem 3.3). Therefore, by iteration, we get the following diagram

$$\begin{tikzcd}
t \arrow[rr, "\beta"'] \arrow[d, mapsto] & & t_1 \arrow[rr, "\beta"'] \arrow[d, mapsto] & & t_2 \arrow[r, dotted, no head] \arrow[d, mapsto] & t_n \arrow[rr, "\beta"'] \arrow[d, mapsto] & & t' \arrow[d, mapsto] \\
T \arrow[r, "\beta_s"'] & G_1 \arrow[r, "*", "\pi"'] & T_1 \arrow[r, "\beta_s"'] & G_2 \arrow[r, "*", "\pi"'] & T_2 \arrow[r, dotted, no head] & T_n \arrow[r, "\beta_s"'] & G' \arrow[r, "*", "\pi"'] & T'
\end{tikzcd}$$

Theorem 3.4 (completeness) *Let T be the graph representation of a term t. If $t \to^* t'$, there is $T \to^* T'$ s.t. T' is the graph representation of t'.*

3.3.1 Proofs of soundness

As already remarked, the main obstacle in a full understanding of sharing reductions is the proof of soundness. In literature, one can find several versions of that proof. Lamping gave a proof of the soundness of his system in [Lam89]. Gonthier, Abadi and Lévy [GAL92a,GAL92b] gave a sketch of the proof of their reformulation of Lamping's algorithm. Complete proofs of the soundness of the sharing rules restricted to the optimal case (see 3.4) can be found in the work by Asperti and Laneve on Interaction Systems [AL94,AL96] and in Laneve's thesis[Lan93]. A first proof of soundness for a system in the style of that presented here, that is, in which we allow non-optimal duplication also, can be found in [Asp95]. For a detailed analysis of the optimal case, we refer to [AG98]. All the proofs of the statements in this paper can be found in [Gue99]. We also remark that [GMM02,GMM01] presents different proof techniques of the main properties.

3.4 Optimal reductions

Optimal reductions are just a particular reduction strategy of the sharing rewriting system. In fact, as every π-reduction (apart from absorption and mux annihilation) corresponds to the unfolding of some sharing, we may try to restrict π-reductions to those that otherwise might forbid the creation of β-redexes. A mux whose principal port points to the function port of an @-node or to the top port of a λ-node might forbid the creation of a redex. On the contrary, if a mux at the argument or at the top port of an @-node duplicates the node (and analogously for a mux at the binding or at the body port of a λ-node), we might have the duplication of a shared redex.

The set of the *optimal rules* is formed of the sharing β-rule in Figure 11, the mux rules in Figure 12 and the π-rules in Figure 13 with the following constraints:

- the α-propagation applies only when the α-node is
  (i) an @-node and the mux is at its function port,
  (ii) a λ-node and the mux is at its top port,
  (iii) a contraction and the mux is at the conclusion of the node;
- the !/?-propagation applies only when the mux is at the conclusion of the node.

Let $\rightarrow_o$ be the reduction relation induced by the optimal rules. We can prove that

(i) if T is the graph representation of t, for every $t \rightarrow^* t'$, there is $T \rightarrow_o^* G$ and $t' \rightarrow^* t''$ s.t. $G \rightarrow_\pi^* T''$ where T'' is the graph representation of t'' (i.e., by unfolding the sharing we get the representation of t'');

(ii) the term t'' can be obtained by a Lévy family reduction that reduces the families of the redexes reduced in $t \rightarrow^* t'$;

(iii) the number of β_s-rules fired in $T \rightarrow_o^* G$ is equal to the number of families reduced in $t \rightarrow^* t'$, that is, it is optimal in the sense of Lévy.

In particular, if t' is the normal form of t, we have that $G \rightarrow_\pi^* T'$, where T' is the graph representation of t', that is, we have that the sharing system computes the normal form of t in an optimal number of sharing β-rules.

For more details on Lévy optimal reduction and on the optimal sharing graph reductions, we refer the reader to [Lév78,Lév80] and [AG98].

Summing up, we can conclude that optimal reductions are the reductions that minimize the number of β-reductions without blocking the creation and execution of them. For more details see [Gue99] or [AG98].

4 Conclusions

In [AM01], Asperti and Mairson proved that there are terms with a polynomial number of Lévy redex families for which the cost of the reduction cannot be bound by any elementary recursive function. Usually, this is interpreted as a negative result for sharing implementations, instead, it only says that we cannot get the size of its set of redex families to measure the cost of the reduction of a λ-term. In terms of sharing graphs, this implies that there are terms whose sharing reduction requires a number of duplications much bigger than the number of β_s-rules. We remark that this is not a problem of the sharing implementation, this phenomenon is caused by the nature of the reducing term and it is only by the great insight on the reduction mechanism of λ-terms given by sharing graphs that we have been able to understand it.

There are also other approaches to graph implementation of λ-reduction, in particular, we cite the work by Mackie [Mac98,Mac04] based on Interaction Nets [Laf90].

The results in the paper generalize to any graph rewriting system whose rules can be formalized in terms of boxes for which *the box nesting property holds*. In particular they apply to *orthogonal term graph rewriting systems*, see [Gue99].

References

[AG98] Andrea Asperti and Stefano Guerrini. *The Optimal Implementation of Functional Programming Languages*. Number 45 in Cambridges Tracts in Theoretical Computer Science. Cambridge University Press, 1998.

[AL94] Andrea Asperti and Cosimo Laneve. Interaction systems I: The theory of optimal reductions. *Mathematical Structures in Computer Science*, 4:457–504, 1994.

[AL96] Andrea Asperti and Cosimo Laneve. Interaction systems II: The practice of optimal reductions. *Theoretical Computer Science*, 159(2):191–244, 3 June 1996.

[AM01] Andrea Asperti and Harry G. Mairson. Parallel beta reduction is not elementary recursive. *Inf. Comput.*, 170(1):49–80, 2001.

[Asp95] Andrea Asperti. Linear logic, comonads and optimal reductions. *Fundamenta infomaticae*, 22:3–22, 1995.

[GAL92a] Georges Gonthier, Martín Abadi, and Jean-Jacques Lévy. The geometry of optimal lambda reduction. In *Proceedings of Nineteenth Annual ACM Symposyum on Principles of Programming Languages*, pages 15–26, Albequerque, New Mexico, January 1992.

[GAL92b] Georges Gonthier, Martín Abadi, and Jean-Jacques Lévy. Linear logic without boxes. In *Proceedings of 7th Annual Symposium on Logic in Computer Science*, pages 223–234, Santa Cruz, CA, June 1992.

[Gir89] Jean-Yves Girard. Geometry of interaction 1: Interpretation of system F. In R. Ferro, C. Bonotto, S. Valentini, and A. Zanardo, editors, *Logic Colloqium '88*, pages 221–260. Elsevier (North-Holland), 1989.

[Gir98] Jean-Yves Girard. Light linear logic. *Inf. Comput.*, 143(2):175–204, 1998.

[GMM01] Stefano Guerrini, Simone Martini, and Andrea Masini. Proof nets, garbage, and computations. *Theor. Comput. Sci.*, 253(2):185–237, 2001.

[GMM02] Stefano Guerrini, Simone Martini, and Andrea Masini. Coherence for sharing proof nets. *Theoretical Computer Science*, 2002. In printing.

[Gue99] Stefano Guerrini. A general theory of sharing graphs. *Theoretical Computer Science*, 227(1–2):99–151, 1999.

[Laf90] Yves Lafont. Interaction nets. In *Proceedings of Seventeenth Annual ACM Symposyum on Principles of Programming Languages*, pages 95–108, San Francisco, California, January 1990.

[Lam89] John Lamping. An algorithm for optimal lambda calculus evaluation. Technical Report Series SSL-89-27, Xerox PARC, Palo Alto, May 1989.

[Lam90] John Lamping. An algorithm for optimal lambda calculus reduction. In *Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages*, pages 16–30, San Francisco, California, January 1990.

[Lan93] Cosimo Laneve. *Optimality and Concurrency in Interaction Systems.* PhD thesis, Dipartmento di Informatica, Universit‘a degli Studi di Pisa, 1993.

[Lév78] Jean-Jacques Lévy. *Réductions Correctes et Optimales dans le lambda-calcul.* PhD Thesis, Université Paris VII, Paris, 1978.

[Lév80] 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.

[Mac98] Ian Mackie. Linear logic with boxes. In IEEE, editor, *13th Annual IEEE Symposium on Logic in Computer Science (LICS '98)*, pages 309–320, Indianapolis, Indiana, 1998.

[Mac04] Ian Mackie. Efficient lambda-evaluation with interaction nets. In *RTA*, pages 155–169, 2004.

[PJ86] Simon L. Peyton Jones. *The Implementation of Functional Programming Languages.* Series in Computer Science. Prentice-Hall International, Englewood Cliffs, NJ, 1986.

[Wad71] C.P. Wadsworth. *Semantics and pragmatics of the lambda-calculus.* Phd thesis, University of Oxford, 1971.