

# Package Duplication in Interaction Nets and Weak Head Reduction in the lambda-calculus

Sylvain Lippi<sup>1</sup>

*Institut de Mathématiques de Luminy  
Marseille, France*

---

## Abstract

We present a simple implementation of weak head reduction in the  $\lambda$ -calculus with interaction nets using package duplication.

*Keywords:* Weak head reduction, interaction nets, lambda calculus

---

## 1 Introduction

Interaction nets introduced by Yves Lafont [5] can be considered as a programming language with a real interpreter [9] or as a graphical model of computation. In the second point view, a typical application is the encoding of optimal reduction for the  $\lambda$ -calculus [1]. Those translations are usually complicated giving quite complex nets during the reduction. We propose to focus on weak head reduction. The consequence is that we shift the problem to the duplication of some classes of nets: packages. We present two results on the duplication of packages: a negative one which sets the intrinsic limits of the interaction net paradigm and a positive one by exhibiting a very simple translation of  $\lambda$ -terms.

## 2 The interaction net paradigm

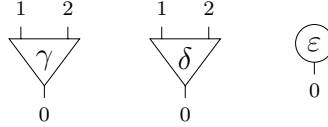
We present the interaction net paradigm in a practical way by introducing a simple example: the interaction combinators. For a more detailed presentation see [5].

---

<sup>1</sup> [lippi@iml.univ-mrs.fr](mailto:lippi@iml.univ-mrs.fr)

### 2.1 Graphical syntax

- The basic ingredient is a *symbol* with its *arity*. Our example system has three symbols:  $\delta$  and  $\gamma$  of arity 2 and  $\varepsilon$  of arity 0.
- Occurrences of symbols are called *cells*. Cells have one *principal port* and their number of *auxiliary ports* is given by the arity of their corresponding symbol. The  $\gamma$ -,  $\delta$ - and  $\varepsilon$ -cells are pictured like this:



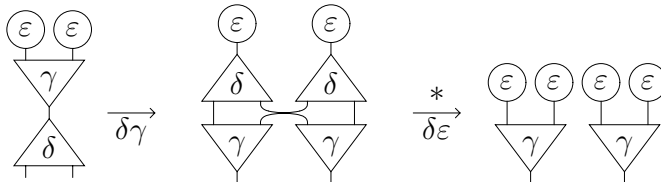
$\gamma$ - and  $\delta$ -cells have three *ports*: one *principal port* (0) and two *auxiliary ports* (1 and 2).  $\varepsilon$ -cells have only one (principal) port. For esthetic reasons, cells with no auxiliary port are pictured with a circle; the important point is to distinguish the principal port of a cell.

**Convention.** Auxiliary ports are not interchangeable. For instance, one can number them from 1 to  $n$ , keeping 0 for the principal port. In practice, the ports will always be implicitly numbered in clockwise order.

- A *net* is a graph built with cells and *free ports*. Ports (principal ports, auxiliary ports or free ports) are connected pairwise by *wires*. An example of a net built with the combinators can be found in figure 1
- A *rule* is a pair of nets (*left member*  $\rightarrow$  *right member*) with the same number of free ports. The important restriction is that the left member must be built with two cells connected by their principal ports; such a net is called a *cut*.<sup>2</sup> There is, at most, one rule for each pair of symbols. Figure 2 gives the rules for the combinators.

### 2.2 Execution

Once a set of symbols and rules has been fixed, we can apply one of those rules to a net obtaining another net and so on until we have reached an irreducible one. The *reduction* relation is denoted by  $\rightarrow$  and its reflexive and transitive closure by  $\rightarrow^*$ . Here is an example of reduction where a net is duplicated by a  $\delta$ -cell:

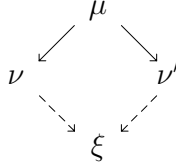


There may be several cuts in a net but the order in which they are eliminated does not matter. The reason is that two instances of a cut are necessarily disjoint,

<sup>2</sup> if the two cells of the left member share the same symbol, there is also a symmetry condition on the right member (see [6]).

so we can apply the corresponding rules independently. All reduction strategies lead to the same irreducible net and have the same length (see [6]).

**Proposition 2.1 (strong confluence)** *If a net  $\mu$  reduces in one step to  $\nu$  and  $\nu'$ , with  $\nu \neq \nu'$ , then  $\nu$  and  $\nu'$  reduce in one step to a common net  $\xi$ .*



**Proof** The two instances of the left members are necessarily distinct so, we can apply the corresponding rules independently.  $\square$

**Corollary 2.2 (reduction)** *If a net  $\mu$  reduces to an irreducible net  $\nu$  in  $n$  steps, then any reduction starting from  $\mu$  eventually reaches  $\nu$  in  $n$  steps.*

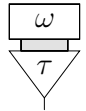
### 3 Package duplication

One of the most important features of the interaction net is that duplication must be explicitly described. To begin with, let us give the definition of three important classes of nets:

- A tree is a net with one distinguished free port, called the *root*. It is either a single wire, in which case the root is fixed arbitrarily, or it is obtained by plugging the auxiliary ports of a cell to the root of smaller trees. In this case, the root is the free port which is connected to the principal port of this cell.
- A wiring is a net built only with free ports and wires *i.e* with no cell.

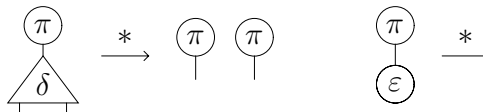
**Notation:**  $\boxed{\phantom{a}}$  represents a bundle of parallel wires ( $|\cdots|$ ).

- A package is obtained by plugging a wiring  $\omega$  with a tree  $\tau$  as follows:



A typical situation is the duplication of packages. An example of implementation is given by the interaction combinators where the following proposition holds:

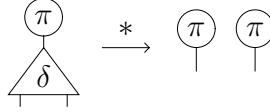
**Proposition 3.1 (Package duplication and erasing)** *For any package  $\pi$  without any  $\delta$ -cell:*



**Proof** The proof relies on the decomposition of a package given above by induction on  $\tau$  and  $\omega$ . The detailed proof can be found in [6].  $\square$

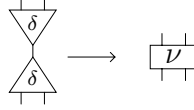
The restriction on the duplicated packages is quite important for duplication. In fact, it is not due to the system of the combinators; it is an intrinsic limitation of the interaction nets.

**Proposition 3.2** *There is no interaction system containing a symbol  $\delta$  such that for any package  $\pi$ :*

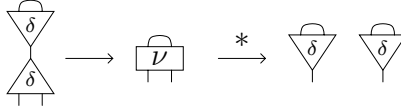


**Proof.**

Let us suppose we have such a system and consider the following rule:



where  $\nu$  is a reduced net invariant by rotation. In particular, we have:



- If  $\boxed{\nu} = \triangle_{\delta} \triangle_{\delta}$ ,

The wire that connect two free ports of  $\nu$  can be identified to each of the four wires of the right member, with two possible orientations in each case. So there are eight cases:

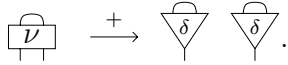
- $\boxed{\nu} = \triangle_{\delta} \triangle_{\delta}$  impossible since  $\nu$  is invariant by rotation.

- $\boxed{\nu} = \triangle_{\delta} \triangle_{\delta}$  idem.

- $\boxed{\nu} = \triangle_{\delta} \triangle_{\delta}$  impossible: we just have to consider the duplication of another package built with  $\delta$ -cells.

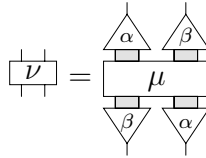
- $\boxed{\nu} = \triangle_{\delta} \triangle_{\delta}$  impossible since  $\nu$  is invariant by rotation.

The four other cases are similar.

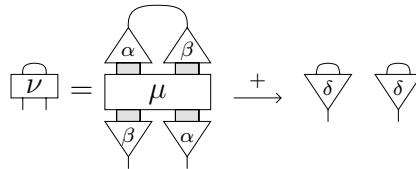
- Otherwise, .

So there is (at least) one cut in the net of the left hand side: the two free ports of “the upper part” of  $\nu$  are connected with principal ports.

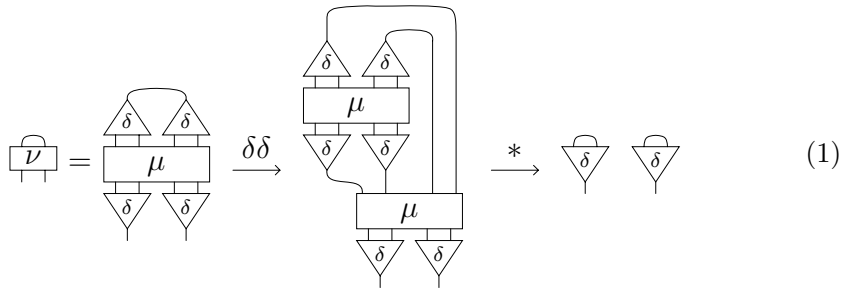
Since  $\nu$  is invariant by rotation the two free ports of “the lower part” of  $\nu$  are connected with principal ports of cells having the same symbols. So we have a reduced net  $\mu$  that is invariant by rotation such that:



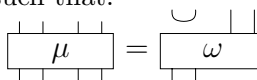
Thus,

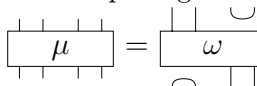


So,  $\alpha = \beta = \delta$  and we have the following reduction:



- If  $\mu$  is a wiring: Let us consider the two nets of the right hand side in reduction 1. Obviously, they are not equal since they do not have the same number of cells. So, the last reduction takes at least one step and its left member contains at least one cut. Thus, there is (at least) two ports of “the upper part” of  $\mu$  that are connected together.  $\mu$  is invariant by rotation, so there is a wiring  $\omega$  such that:

 impossible: we just have to consider the duplication of another package built with  $\delta$ -cells.

 idem

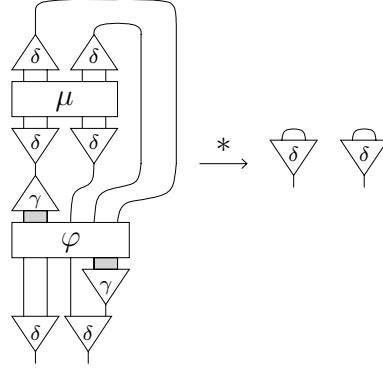
$$\begin{array}{c} \text{---} \text{---} \text{---} \text{---} \\ | \quad | \quad | \quad | \\ \boxed{\mu} \end{array} = \begin{array}{c} \text{---} \text{---} \text{---} \text{---} \\ | \quad | \quad | \quad | \\ \boxed{\omega} \end{array} \text{ impossible because of reduction 1.}$$

The three other cases are similar to the previous one.

- Otherwise,  $\mu$  is a reduced net with at least one cell. So it exists a free port connected to a principal port. Moreover,  $\mu$  is invariant by rotation, so there is a symbol  $\gamma$  and a net  $\varphi$  such that, for example:

$$\begin{array}{c} \text{---} \text{---} \text{---} \text{---} \\ | \quad | \quad | \quad | \\ \boxed{\mu} \end{array} = \begin{array}{c} \text{---} \text{---} \text{---} \text{---} \\ | \quad | \quad | \quad | \\ \boxed{\varphi} \end{array}$$

Because of 1, we have:



this is impossible.

## 4 Weak head reduction

We show that a “naïve” translation of  $\lambda$ -terms into interaction nets is well-adapted for weak head reduction [3]. Whereas other translations such as [7,2,11,8] need auxiliary symbols to tackle with the so-called *bureaucratie*, we just introduce a set of four symbols ( $@$ ,  $\lambda$ ,  $\delta$  and  $\varepsilon$ ). Further more, each  $\lambda$ -term has a unique translation consequently, the correctness proof is much simpler. On the other hand, we just give an implementation of weak head reduction.

**Notation:**  $\lambda$ -terms (denoted by  $u, v, w, t, t_1, t_2, \dots$ ) are written with the Krivine notation:  $x, y, z, x_1, x_2, \dots$  are variables,  $(u)v$  is the application of  $u$  to  $v$  and  $\lambda x u$  is a  $\lambda$ -abstraction.

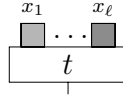
Let us call *process* a pair  $(t, \pi)$  composed of a  $\lambda$ -term  $t$  and a stack  $\pi$  of closed  $\lambda$ -terms. We define *weak head reduction* on processes (noted  $\succ$ ) by:

$$(u)v, \pi \succ u, v \cdot \pi \quad (2)$$

$$\lambda x u, v \cdot \pi \succ u[v/x] \quad (3)$$

#### 4.1 Translations

The translation that is given below is the same as the one into sharing graphs [1] except that special symbols such as *croissants* or brackets are omitted. Intuitively, the translation of a  $\lambda$ -term can be obtained from its syntactic tree by connecting occurrences of variables directly to their corresponding  $\lambda$ . The translation of a  $\lambda$ -term  $t$  is represented by:



where the unique free port at the bottom corresponds to the root of the  $\lambda$ -term and each free port at the top corresponds to an occurrence of a free variable. Those free ports are grouped into bundles corresponding to variable names. Let us define the translation of a  $\lambda$ -term  $t$  by induction on  $t$ :

- A variable  $x_i$  is simply translated by a single wire.
- $(u)v$  is translated by introducing a binary cell: @. The auxiliary ports of @ respectively connected to the translations of  $u$  and  $v$ .
- $\lambda x u$  is translated by connecting the translation of  $u$  to the first auxiliary port of a binary  $\lambda$ -cell. The second auxiliary port of  $\lambda$  is connected to the ports corresponding to free occurrences of  $x$  in  $u$ . At this point, we must introduce a new binary cell  $\delta$  called *duplicator*. Indeed, there may be more than one occurrence of  $x$  in  $u$  and free ports are connected pairwise, so we “gather” the occurrences of  $x$  with a *tree* built with duplicators and connect the root to the second auxiliary port of the  $\lambda$ -cell. In case there is no free occurrence of  $x$ , the second auxiliary port of  $\lambda$  is connected to a cell  $\varepsilon$  of arity zero.

The translation of a  $\lambda$ -term is summed up in figure 3.  $\Delta$  represents a *generalized duplicator* i.e a tree built with  $\delta$ -cells as defined in figure 4.

**Remark 4.1** A tree built with  $\delta$ - and  $\varepsilon$ -cells is not necessarily a generalized duplicator. A generalized duplicator of arity  $n$  is a comb built with  $n$   $\delta$ -cells and one  $\varepsilon$ -cell. We shall see how to use this restriction to duplicate a package containing duplicators.

The translation of a stack  $\pi$  of  $\lambda$ -terms (noted  $\boxed{\pi}$ ) is inductively defined by:

- The translation of the empty stack is

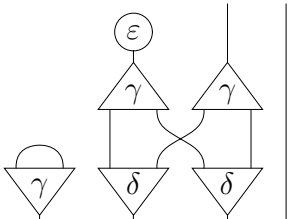


Figure 1. An example of net

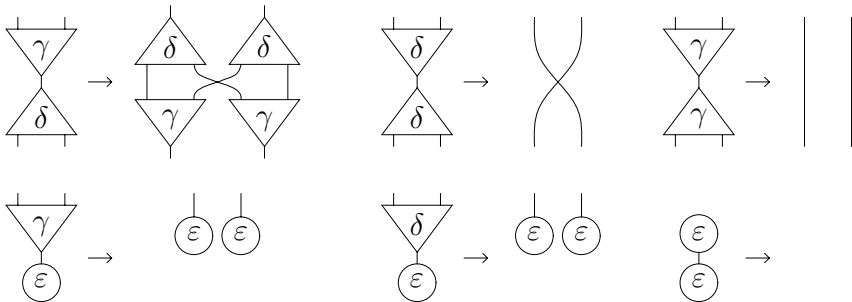


Figure 2. Interaction rules for the combinators

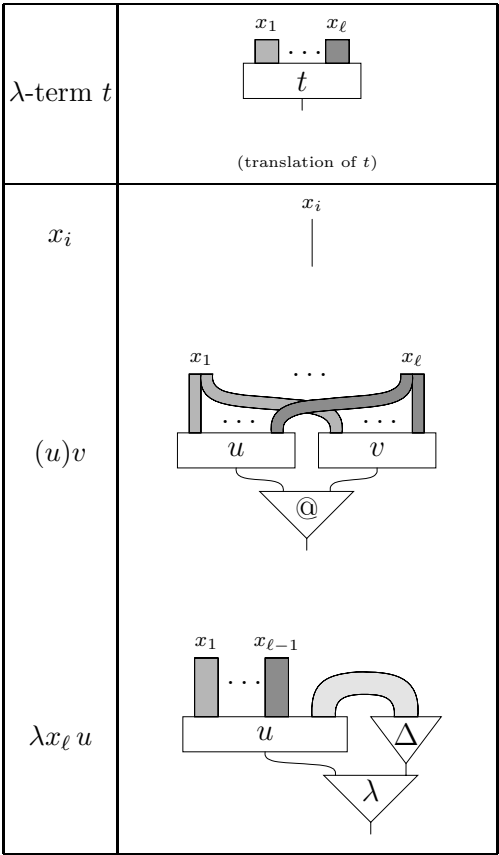
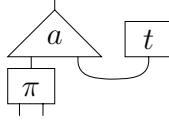


Figure 3. Translation of a  $\lambda$ -term



- The translation of  $t \cdot \pi$  is



Finally a process  $(t, \pi)$  is translated by

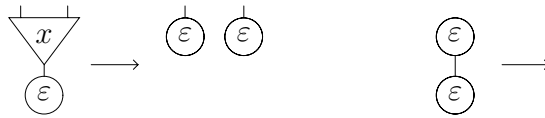


#### 4.2 Duplication and erasing of a $\lambda$ -term

During reduction, some  $\lambda$ -terms are duplicated or erased. The translation of a closed  $\lambda$ -term is a package that contains duplicators. According to proposition 3.2, we can think that our translation is not well-designed. On the other side, we do not have to implement duplication for any packages but only for packages that are translations of a  $\lambda$ -term. So, we can take advantage of the geometry of a translation. In particular, the principal port of a  $\delta$ -cell is connected to the second auxiliary port of a  $\lambda$ - or  $\delta$ -cell and nowhere else.

Consequently, we introduce a binary symbol  $\gamma$  dedicated to the duplication of  $\delta$ -cells. According to the previous remark,  $\gamma$ -cells appear when an abstraction is duplicated ; the  $\lambda$ -cell is duplicated and then the body of the abstraction is duplicated by a  $\delta$ -cell whereas the linker part is duplicated by a  $\gamma$ -cell. The  $\delta$ -cell is duplicated by a  $\gamma$ -cell ; the first auxiliary port of a  $\delta$ -cell is not connected to the principal port of a  $\delta$ -cell whereas the second one is. The rules  $\delta@$  and  $\delta\delta$  are the usual commutations and annihilation rules (see [6]) and correspond respectively to the duplication of an  $@$ -cell and of a link between two auxiliary ports. The set of duplication rules is summed up in figure 5.

Finally, the rest of the rules are devoted to the  $\varepsilon$ -cell and can be summed up by the following scheme of rules:



where  $x \in \{ @, \lambda, \delta, \gamma \}$

**Remark 4.2** The  $\varepsilon\gamma$  rule can be interpreted both as the duplication of  $\varepsilon$ -cells or as the erasing of a  $\gamma$ -cell.

**Remark 4.3** These erasing rules correspond to garbage collection. A reduction can be done even if they are omitted from the system; the consequence is simply a “waste of cells” (see [10] for a precise formulation).

#### 4.3 Weak head reduction

Rule  $a@$  corresponds to equation 2: the argument of an application is pushed on the stack. Rule  $a\lambda$  corresponds to equation 3: the linker part of the abstraction is

connected to the  $\lambda$ -term on the top of the stack and the body of the abstraction to the rest of the stack. Those rules are grouped in figure 6 and an example of reduction can be found in figure 7.

The correctness of the implementation is stated by the following proposition:

**Proposition 4.4** *For any processes  $p$  and  $p'$  if  $p \succ p'$  then,*

$$\boxed{p} \xrightarrow{*} \boxed{p'}$$

**Proof** We have the analog of proposition 3.1 since the translation of a  $\lambda$ -term is a package. The only difference is that  $\delta$ -rules have been slightly modified since a translation may contain  $\delta$ -rules.  $\square$

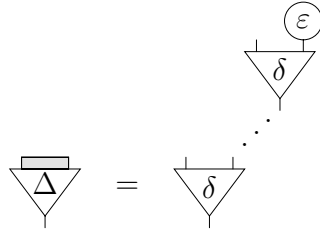


Figure 4. Generalized duplicators

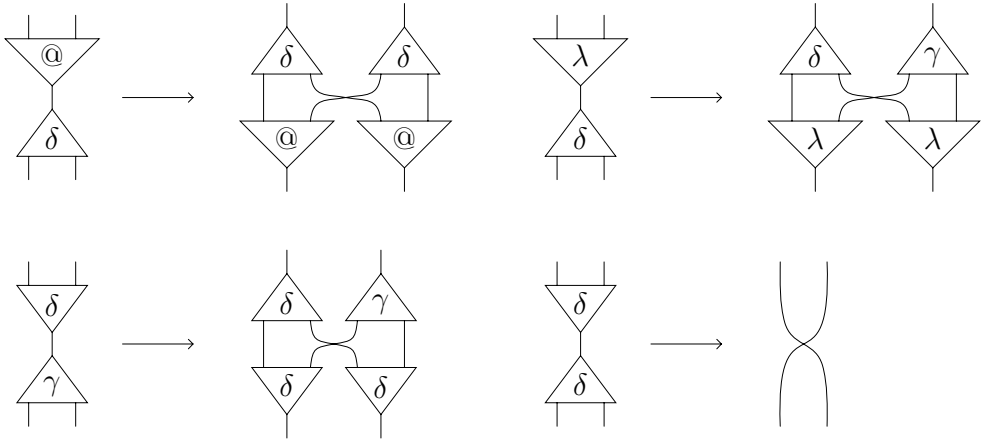


Figure 5. Duplication rules

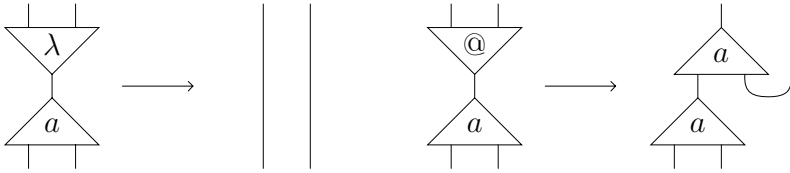


Figure 6. Weak head reduction rules

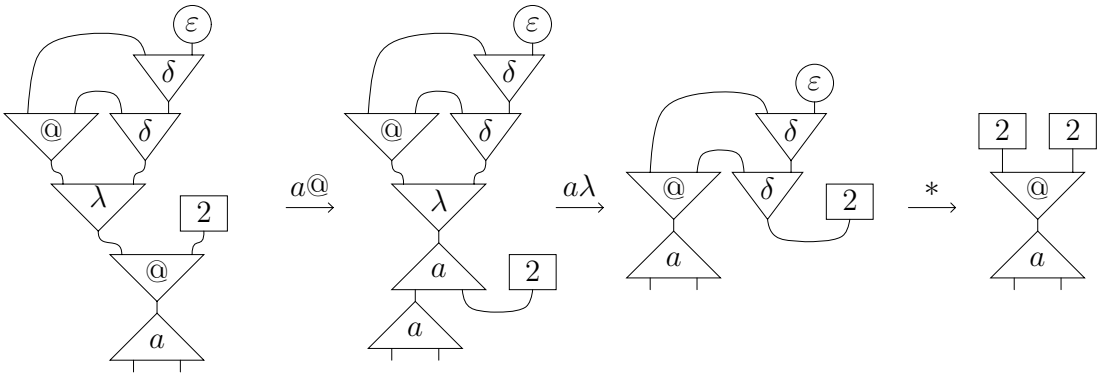


Figure 7. First steps of the reduction for  $(\lambda x (x)x)2$

## References

- [1] 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.
- [2] M. Abadi G. Gonthier and J.-J. Lévy. The geometry of optimal lambda reduction. *In proceedings of the 19th Annual ACM Symposium on Principles of Programming Languages (POPL'92)*, ACM Press., pages 15–26, 1992.
- [3] J.-L. Krivine. *Lambda-calculus, types and models*. Ellis Horwood Series in Computers and their Applications., 1993. Transl. from the French by René Cori. (English).
- [4] Y. Lafont. Interaction nets. *In proceedings of the 17th Annual ACM Symposium on Principles of Programming Languages, Orlando (Fla., USA)*, pages 95–108, 1990.
- [5] Y. Lafont. From proof-nets to interaction nets. In *Advances in Linear Logic*, London Mathematical Society Lecture Note Series 222. Cambridge University Press, 1995.
- [6] Y. Lafont. Interaction combinators. *Information and Computation*, 137(1):69–101, 1997.
- [7] J. Lamping. An algorithm for optimal lambda-calculus reduction. *In proceedings of the 17th Annual ACM Symposium on Principles of Programming Languages, Orlando (Fla., USA)*, 1990.
- [8] Sylvain Lippi. Encoding left reduction in the lambda-calculus with interaction nets. *Mathematical Structures in Computer Science*, 12:1–26, 2002.
- [9] Sylvain Lippi. in<sup>2</sup>: a graphical interpreter for the interaction nets. In *Proceedings of Rewriting Techniques and Applications (RTA '02)*. Springer Verlag, 2002.
- [10] Sylvain Lippi. *Théorie et pratique des réseaux d'interaction*. PhD thesis, Université de la méditerranée, 2002.
- [11] I. Mackie. *The geometry of implementation*. PhD thesis, Department of Computing, Imperial College of Science, Technology and Medicine, 1994.