

# Hard combinators

Denis Béchet <sup>1,2</sup>

*LINA  
Université de Nantes  
Nantes, France*

Sylvain Lippi<sup>3</sup>

*I3S  
Université de Nice  
Sophia Antipolis, France*

---

## Abstract

Hard Interaction systems can be presented as graph relabeling with a handshake mechanism that provide local synchronization. We present a particular one with only four symbols and seven rules that can be used to simulate all the other hard interaction systems.

*Keywords:* Interaction nets, asynchronous circuits, universal machines, graph rewriting

---

## 1 Introduction

Interaction nets introduced by Yves Lafont [3] can be considered as a generalization of linear logic multiplicative proof nets. Syntactically they are presented as graph rewriting systems where rules are applied on pairs of nodes (called *cells*) connected by an “active edge” called *cut* by logicians. Lafont presented in [4] a system of three symbols and six rules called *interaction combinators* that is *universal*: any interaction system can be translated (in a sense that we shall detail below) into the system of the combinators. Interaction nets have been successfully used to implement various reduction strategies for the  $\lambda$ -calculus ([8] and [5]) and several interpreters (in particular a parallel one by [10] and a graphical one by [6]) for interaction nets have been proposed. More recently, non-deterministic extensions have been studied [9].

---

<sup>1</sup> Thanks to the referees and to Yves Lafont for their useful advises.

<sup>2</sup> Email: [denis.bechet@univ-nantes.fr](mailto:denis.bechet@univ-nantes.fr)

<sup>3</sup> Email: [lippi@i3s.unice.fr](mailto:lippi@i3s.unice.fr)

In this paper, we shall focus on a restriction called *hard interaction nets* where the geometry of the net is invariant during reduction and propose a universal system (called *hard combinators*) for such systems. The translation of an arbitrary hard interaction system into hard combinators has a quite different character from the corresponding translation for interaction nets where the key technical point is implementing the duplication of some nets.<sup>4</sup> Here we shall represent nodes as binary words and calculate the transformations with boolean functions. The name *hard interaction nets* is well-chosen, since they are a form of abstract hardware. In this perspective, it is interesting to sum up the important rules and give the basic components that can be used to construct asynchronous circuits. Indeed, it is possible to implement multiplexors, registers, memories and ALUs with hard combinators [7] so it should be possible to build an asynchronous computer simply by following classical Von Neumann computer architecture and using hard combinators [1].

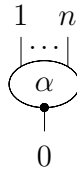
**Notation.** The domain of some variables is implicitly given by their names with the following conventions:  $x, y, z, x_0, x_1, x_2, \dots$  are binary digits,  $p, q, r, s, t$  are binary words and  $\sigma, \rho$  and  $\tau$  are signatures (+ or  $-$ ). Concatenation of  $p$  and  $q$  is noted  $pq$  so  $xy$  is a word with two digits and the scalar product of  $x$  and  $y$  is explicitly noted  $x \times y$ .  $x^n$  denotes the word  $x\dots x$  with  $n$  letters.  $|p|$  is the length of  $p$ . The set of boolean values  $\{0, 1\}$  is noted  $\mathbb{B}$  and the set of natural numbers  $\mathbb{N}$ .

## 2 Hard interaction nets

We present hard interaction nets informally from scratch without any reference to linear logic or even to interaction nets. A hard interaction system (or hard system for short) is composed with a set of *symbols* and their corresponding arity and with a set of *interaction rules*.

### 2.1 Cells, Ports, Nets and Cuts

Occurrence of symbols are called *cells* and have  $n + 1$  *ports* where  $n$  is the corresponding arity. Each cell has exactly one principal port (pictured with a blob) and  $n$  auxiliary ones:

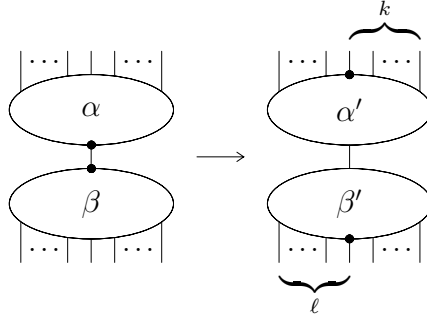


*Nets* are build with a set of cells and *free ports* where ports (principal, auxiliary and free ones) are connected pairwise. *Cuts* are particular nets composed of two cells connected by their principal ports.

<sup>4</sup> more precisely principal nets for the connoisseur.

## 2.2 Interaction rules

The difference between the principal port and the auxiliary ones is essential since rewriting (or *interaction*) can be applied only on cuts. In other words, the left member of an interaction rule is composed of two cells connected by their principal ports. Interaction consists in relabeling cells and changing the orientation of the principal ports; we shall say that the cell is turning. To sum up, an interaction rule is pictured as follows,

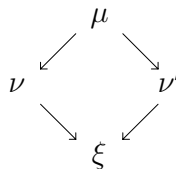


and we say that if an  $\alpha$ -cell interacts with a  $\beta$ -cell it becomes  $\alpha'$  and turns  $k$  times counter clockwise. Similarly,  $\beta$ -cell becomes  $\beta'$  and turns  $\ell$  times. Note we are interested only in *deterministic* hard interaction systems so there is at most one interaction rule for each pair of symbols.

## 2.3 Reduction

Starting from an initial net containing cuts, we can apply an interaction rule obtaining another net and so on until an irreducible net if the reduction finishes. Hard interaction systems are very simple since the computation is local (only two cells are involved in a reduction) and the geometry of the net is invariant. However one can show that it is complete from a computational point of view *i.e* for each Turing machine one can define a hard interaction system that simulates this machine. The reader can find a simple translation in Lafont's paper on combinators [4] (the proof is straightforward) where the proposed system happens to be a hard interaction system where all the cells are unary. Of course, an arbitrarily extendable tape is represented with an infinite number of cells but the alphabet and the set of rules is finite. Let us finish with an essential property due to the local synchronization.

**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  $\xi$ .*



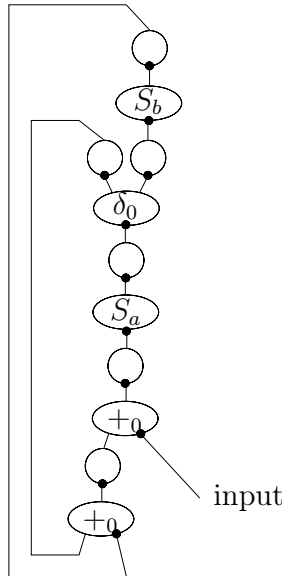
**Proof** The left member of an interaction rule is a cut and  $\nu \neq \nu'$ . Consequently the above reductions are applied on two different instances of cuts. Two instances of cuts are necessarily disjoint (a cell is in one cut at most) so the corresponding interaction rules can be applied independently.  $\square$

Consequently reduction is deterministic in a strong way: any reduction strategy gives the same result with the same number of steps.

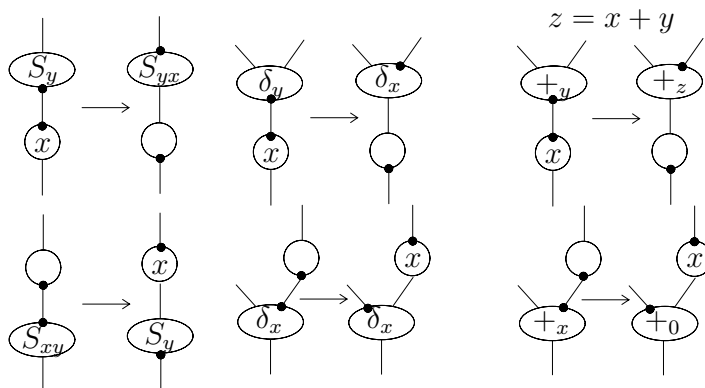
**Corollary 2.2 (reduction)** *If a net  $\mu$  reduces to an irreducible net  $\nu$  in  $n$  steps, then any reduction starting from  $\mu$  eventually reaches  $\nu$  in  $n$  steps.*

#### 2.4 Example

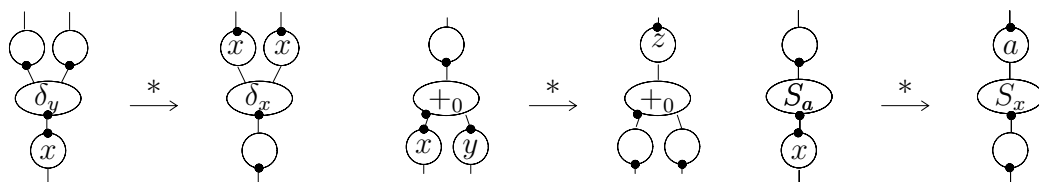
Let us present a simple hard interaction system that implements a “real function”. Given an input stream of bits  $x_1, x_2, x_3, \dots, x_n$  the following net computes an error detection value  $ab$  with two bits that gives a criterium that is comparable to simple parity computation. This kind of calculus is used, for instance, to compute a checksum of Ethernet frames. In this case, the checksum has 32 bits.



Cells represented by circles can have three different values: 0, 1 or *void*. The other cells correspond to the duplication operator  $\delta_x$  (two symbols), the exclusive or operator  $+$  (two symbols) and finally the shift register  $S_x$  and  $S_{xy}$  (six symbols). The values  $a_i$  and  $b_i$  are recursively defined by  $a_{i+1} = a_i + b_i + x_i$  and  $b_{i+1} = a_i$ . Let us give the corresponding rules of this system:



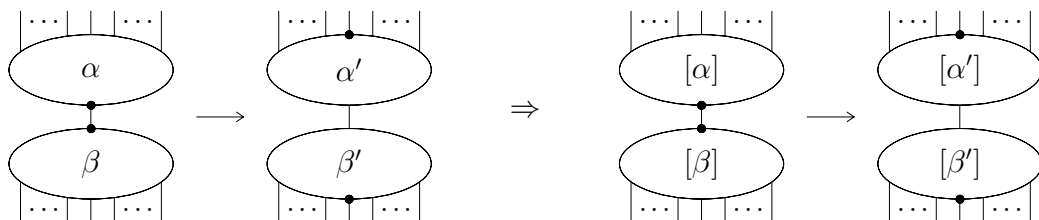
The correctness of the implementation comes from the following three reductions that detail the execution of the three operators:



**Remark 2.3** This system can be encoded in a system with only two symbols and three rules that happens to be the binary hard combinators presented in the next section.

### 3 A universal system: hard combinators

We present a particular hard system called *hard combinators* with four symbols and seven rules that is sufficient to simulate all other hard systems. More precisely, we can translate each cell  $\alpha$  by a net  $[\alpha]$  built with hard combinators such that,



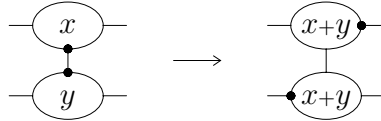
#### 3.1 Cells

Our system is composed of four different symbols: two binary ones, 0 and 1, and two unary ones,  $+$  and  $-$ .

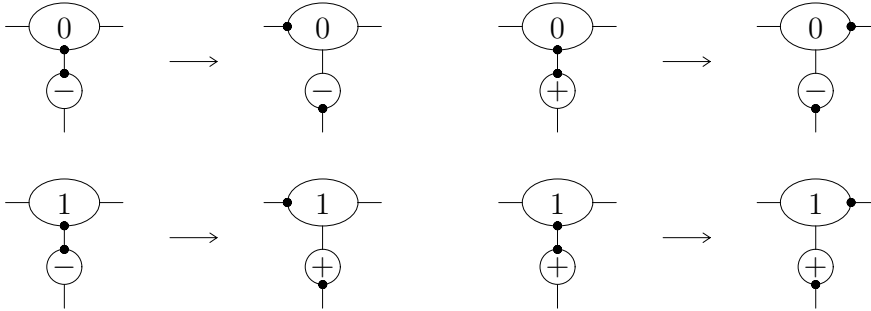


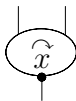
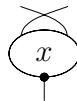
### 3.2 Rules

There are seven rules that can be split into two groups: three rules between binary cells and four rules between unary and binary cells. There is no rules between unary cells. Binary rules are also called *uniform rules* because the principal port “turns in the same direction” (counter clockwise) for each interaction. The three uniform rules can be summed up by the following schema where  $+$  denotes sum modulo 2.

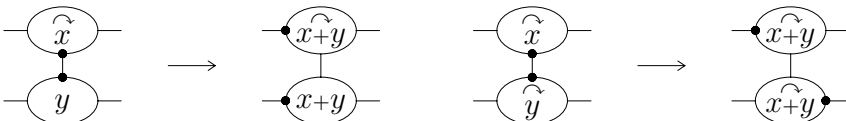


Consequently, the four other rules are called *non-uniform rules* because the orientation of a binary cell depends on the unary cell interacting with it. Intuitively,  $(+)$ -cells let binary cells turn counter clockwise and  $(-)$ -cells force them to turn clockwise.



**Definition 3.1** [clocks] for any bit  $x$ ,  = 

Clocks are introduced for graphical convenience to avoid complicated crossing of wires. They are noted  $\hat{x}$  because they interact as binary cells except their principal port turns clockwise. For example, we have the following reductions.



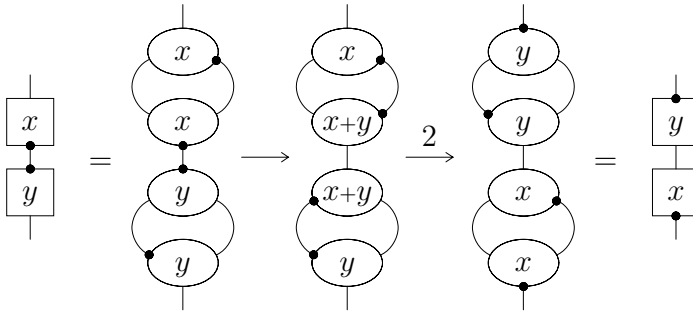
## 4 Uniform components

In this section, we consider the subsystem composed only with the two binary cells and the corresponding three uniform rules. Surprisingly, non trivial functions can be built in this restriction and, indeed it is a decisive step in the construction of a universal translation.

**Definition 4.1** [binary pipes] for any bit  $x$ ,  $\boxed{x} \bullet = \text{---} \bullet \begin{array}{c} \circlearrowleft \\ x \end{array} \begin{array}{c} \circlearrowright \\ x \end{array} \bullet \text{---}$

**Lemma 4.2** for any bits  $x$  and  $y$ ,  $\boxed{x} \bullet \bullet \boxed{y} \text{---} \xrightarrow{*} \text{---} \bullet \boxed{y} \bullet \bullet \boxed{x} \bullet \text{---}$

**Proof** We apply uniform rules and the equality  $x + x + y = y \pmod 2$ .



□

**Definition 4.3** [pipes] for any word  $p = x_1 \dots x_n$ ,  $\boxed{p} \bullet = \text{---} \bullet \boxed{x_n} \bullet \dots \bullet \boxed{x_1} \bullet \text{---}$

**Notation.** We also picture an *unknown pipe*  $\boxed{\phantom{x}} \bullet$  for pipes corresponding to any word of size  $n$  or simply  $\boxed{\phantom{x}} \bullet$  if there is no ambiguity. Those blank representations come from the idea that if one does not know what is stored in a pipe then, the place is free !

**Lemma 4.4** for any words  $p$  and  $q$ ,  $\boxed{p} \bullet \bullet \boxed{q} \text{---} \xrightarrow{*} \text{---} \bullet \boxed{q} \bullet \bullet \boxed{p} \bullet \text{---}$

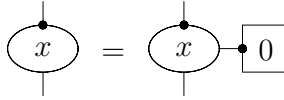
**Proof** by induction on  $p$  and  $q$ .

□

**Definition 4.5** [zero]  $\bullet \boxed{0} = \text{---} \bullet \begin{array}{c} \circlearrowleft \\ 0 \end{array} \begin{array}{c} \circlearrowright \\ 0 \end{array} \bullet \begin{array}{c} \circlearrowleft \\ 0 \end{array} \begin{array}{c} \circlearrowright \\ 0 \end{array} \bullet \begin{array}{c} \circlearrowleft \\ 0 \end{array} \begin{array}{c} \circlearrowright \\ 0 \end{array} \bullet \begin{array}{c} \circlearrowleft \\ 0 \end{array} \begin{array}{c} \circlearrowright \\ 0 \end{array} \bullet \text{---}$

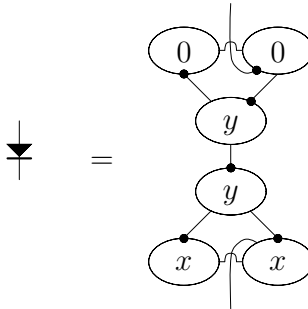
**Lemma 4.6** for any bit  $x$ ,  $\boxed{0} \bullet \bullet \begin{array}{c} \circlearrowleft \\ x \end{array} \begin{array}{c} \circlearrowright \\ x \end{array} \bullet \text{---} \xrightarrow{*} \text{---} \bullet \boxed{0} \bullet \bullet \begin{array}{c} \circlearrowleft \\ x \end{array} \begin{array}{c} \circlearrowright \\ x \end{array} \bullet \text{---}$

**Proof** The above reduction can be easily checked with the binary rules.  $\square$

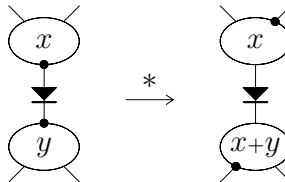
**Definition 4.7** [seesaws] for any bit  $x$ , 

As clocks, seesaws are introduced to simplify the definitions of the other nets and do not have any functional property. Seesaws interact as binary cells: they change their principal port and their symbol is summed with the interacting cell.

**Remark 4.8** Do not confuse between pipes (binary words in a square box), seesaws (bits in a round box) and unary cells (signatures in a round box).

**Definition 4.9** [diodes] 

**Remark 4.10** Unlike pipes or zero, diodes correspond to a set of nets not to a unique one. Indeed, bits  $x$  and  $y$  in the above definition can have any binary values so there are four different representation of a diode. We shall use this kind of definition for other components.

**Lemma 4.11** For any bits  $x$  and  $y$ , 

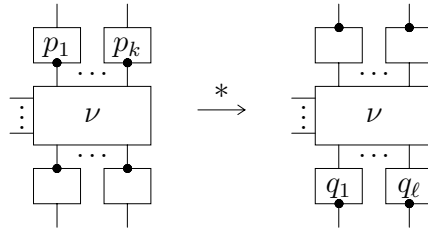
**Proof** The above reduction can be easily checked with the uniform rules.  $\square$

**Remark 4.12** According to remark 4.10, the above lemma should be read “starting from any representation of the diode in the left member, we obtain another (possibly different) representation of the diode in the right member.”

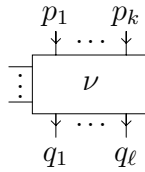
## 5 Invariant nets

**Definition 5.1** [Invariant nets] Let us consider a net  $\nu$  where free ports are partitioned into three sets: *inputs*, pictured with an in-going arrow, *outputs*, pictured with an out-going arrow, and *unused*, pictured with no arrow. We say that  $\nu$  is invariant on inputs  $p_1, \dots, p_k$  and produces outputs  $q_1, \dots, q_\ell$  when we have the following reduction,





where the length of the “input” pipes are respectively  $|p_1|, \dots, |p_k|$  and the length of the “output” ones  $|q_1|, \dots, |q_\ell|$ . We shall use the following notation for invariant nets,



**Remark 5.2** We do not mention where are the principal ports of  $\nu$ . Indeed, the important point is to identify the inputs and the outputs and to know how they interact with pipes.

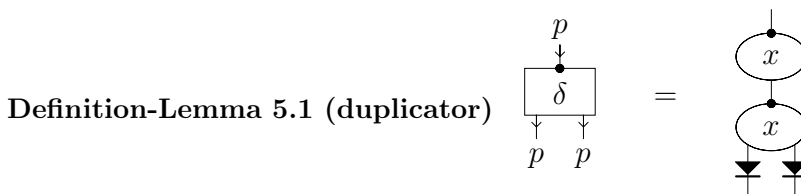
As explained in remark 4.10, the net  $\nu$  corresponds to a class of nets and the reduction above means that the right member is in the same class of nets as the left member. For example, in definition 5.6,  $x_0$  and  $y_0$  range over  $\{0, 1\}$  and  $\sigma$  ranges over  $\{+, -\}$  so there are eight different representations.

**Remark 5.3** According to the previous definition, an invariant net is a pair composed of a net and a partition of its free ports and there may be several invariant nets corresponding to a unique net. However, we also say that a net is invariant when such a partition exists.

**Remark 5.4** In the previous section we introduced unknown pipes and zero which are invariant. More precisely,  $p \rightarrow \boxed{\phantom{0}} \rightarrow p$  and  $\boxed{0} \rightarrow 0$ .

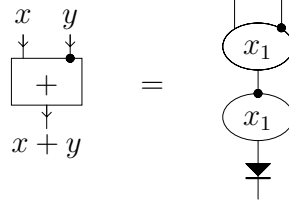
### 5.1 Duplicator and arithmetic operations

To avoid cumbersome repetitions, we give the definition and the corresponding invariance property of the following nets in one shot. For example, the net  $\delta$  is defined by the right member of the equality and we show that it is invariant on input  $p$  and produces output  $p$  twice.



**Proof** We apply lemma 4.11 for the diode and the uniform rules. □

**Definition-Lemma 5.2 (plus)**

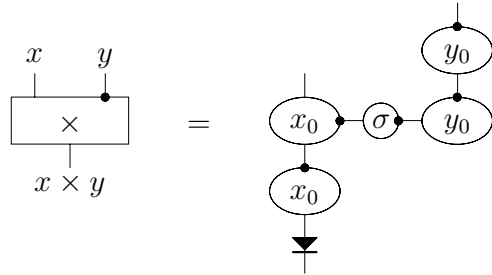


**Remark 5.5**  $+$  denotes the sum modulo 2.

**Proof** We apply lemma 4.11 for the diode and the uniform rules.  $\square$

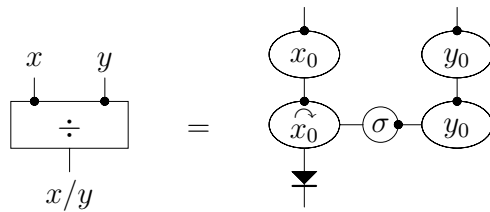
In the uniform subsystem, we have defined constants, pipes, duplication and plus. So one may wonder if it is possible to define product as well in this subsystem. The answer is probably negative. Indeed, the plus operation (binary `xor`) is weaker than binary addition that is computing the sum and but also the carry. Moreover, one can prove that is impossible to build a uniform system that is universal.

**Definition 5.6** [sequential product]



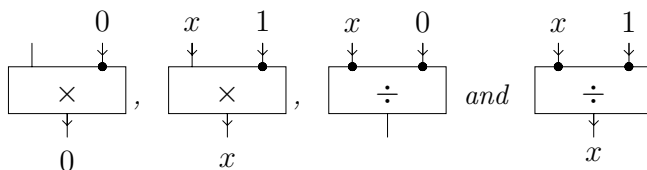
The *sequential product* use input  $y$  first. If  $y$  is zero the result is directly returned and input  $x$  is not used.

**Definition 5.7** [partial quotient]



The *partial quotient* can be considered as the dual of the sequential product. Both inputs are used but it returns no result when input  $y$  is zero.

**Lemma 5.8** We have the following invariants for sequential product and partial quotient,



**Proof** Trivial with uniform but also non-uniform rules.  $\square$

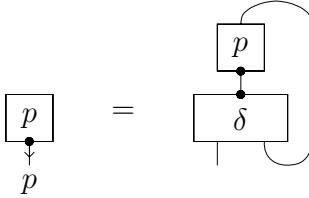
## 5.2 Composition

The first steps, building invariant nets from scratch can be compared to bootstrap in the sense that the difficult part is only to build the very first components (constant zero, duplicator, product). It is now easy to compose invariant nets with pipes and build other more complicated nets.

However, for synchronizations reasons, it is not always possible to compose two invariant nets by plugging directly outputs of the first one with inputs of the second one. To avoid this problem, outputs of invariant nets are connected to unknown pipes. It is not difficult to verify that such “buffered” invariant nets can be freely composed. In some cases, we can suppress those “output pipes” but the proof of the invariance property is tedious. Consequently, from now on, all outputs of invariant nets are connected to pipes when they are composed with other invariant nets.

A first application is to implement binary word constants.

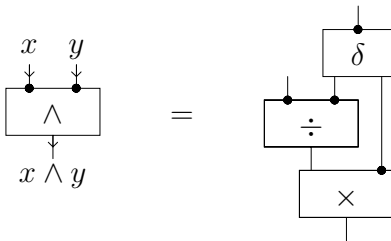
### Definition-Lemma 5.3 (constant)



**Remark 5.9** For clarity, constants are defined with non-reduced nets. We can verify that we can reduce them and by the confluence property, we can use the reduced form.

Let us give an invariant net for boolean and.

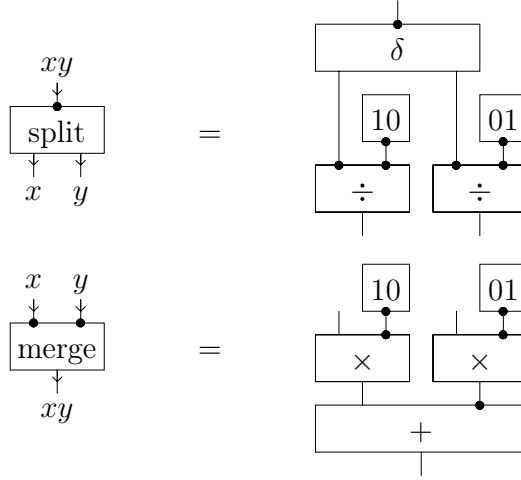
### Definition-Lemma 5.4 (boolean and)



**Proof** We consider two cases:  $y = 0$  and  $y = 1$  and apply composition. □

**Remark 5.10**  $x \wedge y = x \times y$  so the difference between sequential product and boolean and is that boolean and always uses its two inputs.

In the same way, we can define invariant nets with several inputs and outputs for vectorial boolean functions on several inputs. Eventually, those invariant nets can be used to build the corresponding functions on binary words. To that purpose, the nets spit and merge can be composed to build some kind of parallel/serial adaptators.

**Definition-Lemma 5.5 (split and merge)**

**Proof** By composition. □

## 6 The Translation

Now we are ready to translate a given hard interaction system into the system of hard combinators presented in section 3. Symbols are numbered and represented by binary words of a fixed length  $N$ . A first idea is to represent the set of rules that we want to encode by a partial function  $\varphi : \mathbb{B}^N \times \mathbb{B}^N \rightarrow \mathbb{B}^N \times \mathbb{N}$  where  $\varphi(p, q) = (p', k)$  if  $p$  interacts with  $q$ , becomes  $p'$  and turns  $k$  times. Let us remark that we need the values of  $\varphi(p, q)$  and  $\varphi(q, p)$  to compute the reduction between  $p$  and  $q$ .

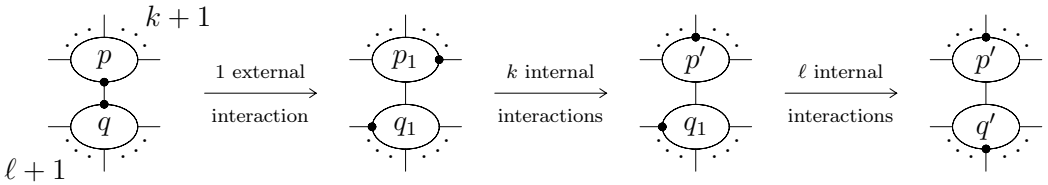
In fact, we choose a slightly different representation and introduce *stable cells* that interact with another (stable) cell and *unstable cells* that interact internally reaching eventually a stable state. Each interaction is decomposed into one *external interaction* between two *stable cells* followed by several (possibly zero) *internal interactions* inside each *unstable cell*. This way we can impose that a cell turns (uniformly !) exactly once at each (external or internal) interaction. Consequently, the set of rules is represented by a partial function  $\psi : \mathbb{B}^N \times \mathbb{B}^N \rightarrow \mathbb{B}^N$  where  $\psi(p, q) = p'$  if  $p$  interacts with  $q$ , becomes  $p'$  and turns exactly once.

Let us define  $\psi$  from  $\varphi$ . For each couple of (stable) symbols  $p$  and  $q$  such that  $\varphi(p, q) = (p', k + 1)$ <sup>5</sup> we introduce  $k$  new (unstable) symbols  $p_1, \dots, p_k$  and set,

<sup>5</sup> If the principal port remains unchanged after reduction, we say that it turns  $a + 1$  times where  $a$  is the arity of the cell.

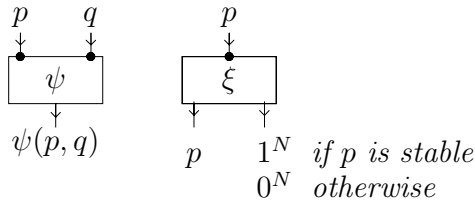
$$\begin{cases} \psi(p, q) &= p_1 \\ \psi(p_1, 0^N) &= p_2 \\ \vdots & \\ \psi(p_{k-1}, 0^N) &= p_k \\ \psi(p_k, 0^N) &= p' \end{cases}$$

Since unstable cells do not interact with another one, we arbitrarily fix the value of the second argument of  $\psi$  to  $0^N$ . Here is the graphical representation of an interaction between  $p$  and  $q$  where  $\varphi(p, q) = (p', k + 1)$  and  $\varphi(q, p) = (q', \ell + 1)$ ,



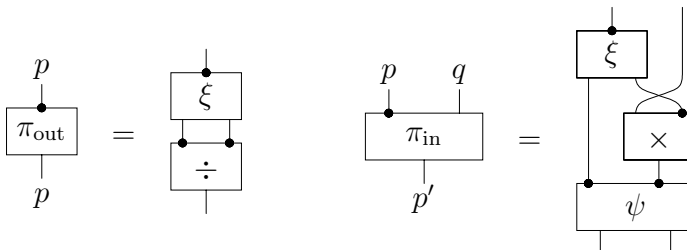
Let us introduce two invariant nets. The first one corresponds to the function  $\psi$  that computes the new symbol after an (internal or external) interaction. The second one called discriminant  $\xi$ , says if a cell is stable or not.

**Definition 6.1** [transition and discriminant]

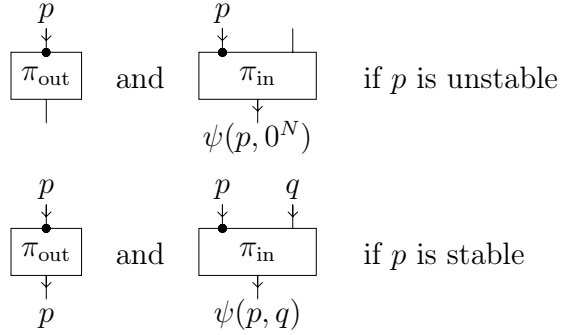


Now we can give the translation of the port of a cell into two parts:  $\pi_{\text{in}}$  and  $\pi_{\text{out}}$ . The important idea is that  $\pi_{\text{in}}$  computes the next symbol  $p'$  without any interaction with  $q$  in the case  $p$  is not stable. In the same way  $\pi_{\text{out}}$  gives the current symbol  $p$  only if  $p$  is stable.

**Definition 6.2**

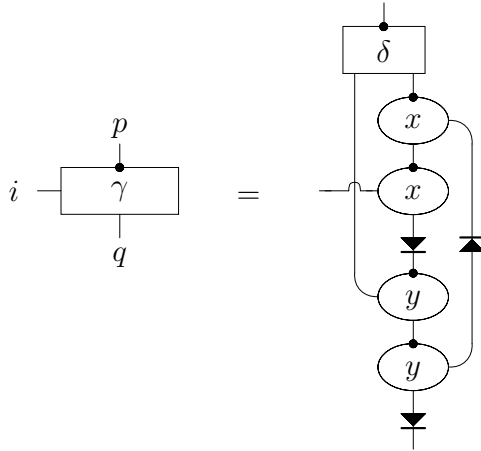


**Lemma 6.3**



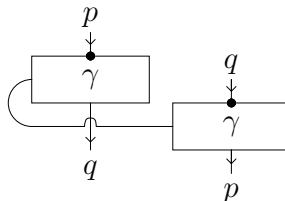
Invariant nets are easy to use and compose because we feel “at home” with inputs/outputs. However this notion is not mandatory for general interaction nets. Indeed, in the translation of a port, we need some kind of “full/duplex” connection since a cell outputs its current symbol to another cell but also inputs the symbol of the cell with whom it is interacting ! This is exactly what is done by the net  $\gamma$ .

**Definition 6.4** [gamma]



Port  $p$  corresponds to an input, port  $q$  to an output and  $i$  to the “full/duplex” interface. Each port of a cell corresponds to a  $\gamma$ -cell; when two cells interact, the input of a  $\gamma$ -cell is reproduced on the output of the other  $\gamma$ -cell. This property is summed up in the following lemma.

**Lemma 6.5**

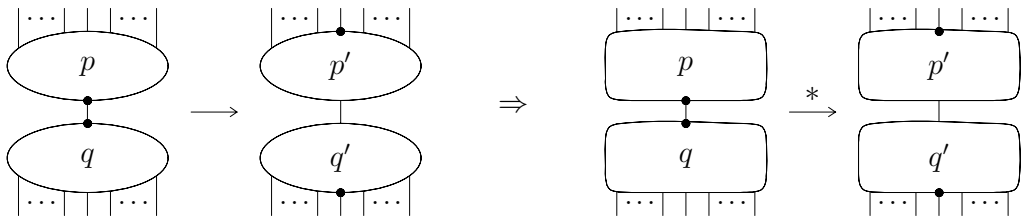




and Logical Unit (ALU) and pipe to a register. Then this basic architecture (a net  $\pi$  composed with a pipe) is repeated for the translation of each port of the cell. Another possibility is to “centralize” the transition function for the whole cell. The advantage is we do not have to introduce unstable cells but on the other side we have to implement a more complicated component for the interface part.

Finally, it is now easy to verify that our translation simulates the rules of a given hard system.

### Theorem 6.10



**Proof** Apply lemma 6.8 and definition 6.9. See appendix A for the detailed reduction.  $\square$

## 7 Conclusion

The system we propose seems to be a good candidate for a universal hard system. However this work is a first step in the domain of hard interaction nets. Indeed many questions related to fundamentals as well as applications remain still open.

- The first one concerns the minimality of such a system; is it possible to give a simpler universal system with fewer symbols or rules? For instance, it is not easy to know whether three symbols would be sufficient. We only know that a system composed only of uniform rules cannot be universal.
- There is a correctness criterion for interaction nets imported from linear logic to prevent deadlocks. It is important to reformulate this criterion for the particular case of hard interaction nets since it is an opportunity to simplify and perhaps to refine it.
- Although (general) interaction nets cannot be translated into hard interaction nets, it is interesting to see if there could be a compilation process for some subclass of interaction nets. Interaction nets would be the high level programming language whereas hard interaction nets would be the target (low level) language. In the same spirit, interpreters have been developed for interaction nets. Would it be possible to physically implement components for hard combinators? In other words, we can consider hard combinators as components for electronic asynchronous circuits?
- As interaction nets can be compared to graph rewriting systems, hard interaction nets can be compared to graph relabeling. These techniques have been particularly successful in the study of graph election algorithms [2]. It would be interesting to



implement such algorithms with hard interaction nets and this way take benefit from the confluence property! More generally, it would be interesting to compare hard interaction nets with other existing rewriting techniques.

- The fixed geometry of hard interaction nets gives them a very similar flavour to cellular automata, or a generalization of cellular automata to non-rectangular grids and there are universality results for cellular automata so it should be interesting to compare those rewriting systems.

## References

- [1] Cecile Germain, Daniel Etiemble. *Architecture des Ordinateurs*. Cours de Licence Informatique, Université d'Orsay, 2005.
- [2] Emmanuel Godard, Yves Métivier, M. Mosbah, and A. Sellani. Termination detection of distributed algorithms by graph relabelling systems. In *proceedings of the first Conference on Graph Transformation*, 2002.
- [3] Yves Lafont. Interaction nets. In *proceedings of the 17th Annual ACM Symposium on Principles of Programming Languages, Orlando (Fla., USA)*, pages 95–108, 1990.
- [4] Yves Lafont. Interaction combinators. *Information and Computation*, 137(1):69–101, 1997.
- [5] Sylvain Lippi. Encoding left reduction in the lambda-calculus with interaction nets. *Mathematical Structures in Computer Science*, 12(6), December 2002.
- [6] 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.
- [7] Sylvain Lippi. Asynchronous Components with Hard Combinators. In Preparation.
- [8] Ian Mackie. Yale: Yet another lambda evaluator based on interaction nets. In *Proceedings of the 3rd ACM SIGPLAN International Conference on Functional Programming (ICFP'98)*. ACM Press, 1998.
- [9] Damiano Mazza. Multiport Interaction Nets and Concurrency. In *proceedings of CONCUR 05, LNCS 3653*, 2005.
- [10] Jorge Sousa Pinto. *Implantation parallèle avec la logique linéaire (applications des réseaux d'interaction et de la géométrie de l'interaction)*. PhD thesis, Ecole Polytechnique, 2001.

## A Simulation of hard interaction rules

We detail the proof of theorem 6.10. We consider the interaction between a cell  $p$  and  $q$  where  $p$  becomes  $p'$  and turns  $k + 1$  times and  $q$  becomes  $q'$  and turns  $\ell + 1$  times.

