Interpreting a Finitary Pi-Calculus in Differential Interaction Nets[☆]

Thomas Ehrhard[a], Olivier Laurent[b]

[a] *Preuves, Programmes et Systèmes, CNRS and Université Paris Diderot – Paris 7*
[b] *Laboratoire de l'Informatique du Parallélisme, Université de Lyon, ENS Lyon - CNRS - UCBL - INRIA*

Abstract

We propose and study a translation of a pi-calculus without sums nor recursion into an untyped version of differential interaction nets. We define a transition system of labeled processes and a transition system of labeled differential interaction nets. We prove that our translation from processes to nets is a bisimulation between these two transition systems. This shows that differential interaction nets are sufficiently expressive for representing concurrency and mobility, as formalized by the pi-calculus.

Our study will concern essentially a replication-free fragment of the pi-calculus, but we shall also give indications on how to deal with a restricted form of replication.

Key words: linear logic, interaction nets, concurrency, pi-calculus

Introduction

Linear Logic proofs [Gir87] admit a *proof net* representation which has a very asynchronous and local reduction procedure, suggesting strong connections with parallel computation. This impression has been enforced by the introduction of *interaction nets* and *interaction combinators* by Lafont in [Laf95].

But the attempts at relating concurrency with linear logic (e.g. [EW97], [AM99], [Mel06], [Bef05], [CF06] based on [FM05]...) missed a crucial feature of concurrency, such as modeled by process calculi like Milner's π-calculus [Mil93], [SW01]: its intrinsic *non-determinism.* Indeed, all known logical systems had either an essentially deterministic reduction procedure – this is the case of intuitionistic and linear logic, and of classical systems such as Girard's LC or Parigot's $\lambda\mu$ – or an excessively non-deterministic one, as Gentzen's classical sequent calculus LK, which equates all proofs of the same formula.

However, many denotational models of the lambda-calculus and of linear logic admit some form of non-determinism (e.g. [Plo76, Gir88b]), showing that

[☆]This work has been partially funded by the ANR project BLAN07-1 189926 TIF *Curry-Howard for Concurrency* (CHOCO).

a non-deterministic proof calculus is not necessarily trivial. The first author introduced such models, based on vector spaces (see e.g. [Ehr05]), which have a nice proof-theoretic counterpart, corresponding to a simple extension of the rules that linear logic associates with the exponentials.

In this *differential linear logic (DiLL)*, the weakening rule has a mirror image rule called *coweakening*, and similarly for dereliction and for contraction, and the reduction rules have the same mirror symmetry[1]. The corresponding formalism of *differential interaction nets (DIN)* has been introduced in a joint work by the first author and Regnier [ER06]. In DiLL, two proofs of the same formula can be added and there is a 0-proof of any formula, which is neutral for this addition. So the set of proofs of any formula is a commutative monoid and this is necessary because the reductions associated with the dereliction/cocontraction and codereliction/contraction cuts of DiLL lead to such non trivial sums of proofs: in that sense, DiLL is a non-deterministic logic. As it is well known in a categorical setting, this possibility of adding proofs is equivalent to the identification of the two additive connectives $\oplus$ and &.

In a joint work with Kohei Honda [HL08], the second author proposed a translation of a version of the π-calculus in proof-nets for a version of linear logic extended with the cocontraction rule (as we now understand). The basic idea consists in interpreting the parallel composition as a cut between a contraction link (to which several *outputs* are connected, through dereliction links) and a cocontraction link, to which several promoted receivers are connected. Being promoted, these receivers are replicable, in the sense of the π-calculus. The other fundamental idea of this translation consists in using linear logic polarities for making the difference between outputs (negative) and inputs (positive), and of imposing a strict alternation between these two polarities. This allows to recast in a polarized linear logic setting a typing system for the π-calculus previously introduced by Berger, Honda and Yoshida in [BHY04]. This translation has two features which can be considered as slight defects: it accepts only replicable receivers and it is not really modular (the parallel composition of two processes cannot be described as a combination of the corresponding nets).

One should mention here that translations of the π-calculus into nets of various kinds, subject to local reduction relations, have been provided by several authors (cf. the work of Laneve, Parrow and Victor on *solo diagrams* [LPV01], of Beffara and Maurel [BM06], of Milner on *bigraphs* [JM03], of Mazza [Maz05] on *multiport interaction nets* etc.). One should also mention the early work of Honda and Yoshida [HY94] which introduces a system of combinators for interpreting a process algebra. These combinators have connections with Lafont's interaction nets; just like multiport interaction nets and solo diagrams, this system seems however to lack the main feature of interaction nets, namely (strong) confluence. Moreover, as far as we know, these approaches have no clear logical grounds nor simple denotational semantics. Indeed, the fact that DINs have

[1]The only non symmetric rule of DiLL is promotion. Finding a symmetric version thereof seems to be a rather challenging task!

a denotational semantics, together with the translation we propose, suggest to interpret the π-calculus in DINs' denotational models and to study the induced equivalence of processes. This approach will be developed in further work. It should be observed moreover that the denotational models of DINs' are also models of the lambda-calculus, suggesting natural combinations between concurrent programming (as modeled in DINs) and functional programming.

Principle of the translation. The purpose of the present paper is to continue this line of ideas, using more systematically the new structures introduced by DINs.

The first key decision we made, guided by the structure of the typical co-contraction/contraction cut intended to interpret parallel composition, was of associating with each free name of a process not one, but *two* free ports in the corresponding differential interaction net. One of these ports will have a !-type (positive type) and will have to be considered as the *input port* of the corresponding name for this process, and the other one will have a ?-type (negative type) and will be considered as an *output port*.

We discovered structures which allow one to combine these pairs of wires for interpreting parallel composition and called them *communication areas*: they can be seen as complete graphs between vertices made of pairs of contraction cells (marked by a "?" symbol) and cocontraction cells (marked by a "!" symbol), connected by edges which are pairs of wires. An example of such a structure, with 3 vertices, is given in figure 1. Output and input prefixes will be interpreted using dereliction and codereliction, as well as the multiplicative connectives.

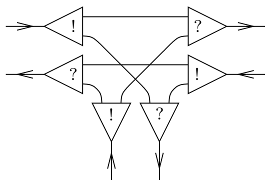


Figure 1: Communication area

Content. We first introduce differential linear logic, presented as a sequent calculus, and then differential interaction nets. These nets are typed with the recursive typing system introduced by Danos and Regnier in [Reg92] (which corresponds to the untyped lambda-calculus) for avoiding the appearance of non reducible configurations. To simplify the presentation, these nets use only a restricted form of the promotion rule of linear logic, which is sufficient for interpreting a replication-free version of the π-calculus, as well as a restricted form of replication. In this setting, we define a "toolbox", a collection of nets that we shall combine for interpreting processes, and a few associated reductions, derived from the basic reduction rules of differential interaction nets.

We organize reduction rules of nets as a labeled transition system, whose vertices are nets, and where the transitions correspond to dereliction/codereliction reductions. Then we define a process algebra which is a polyadic π-calculus, without replication and without sums. We specify the operational semantics of this calculus by means of an abstract machine inspired by the machine presented in [AC98, Chapter 16]. We define a transition system whose vertices are the states of this machine, and transitions correspond to input/output reductions.

And we define a "translation" relation from machine states to nets and show that this translation relation is a bisimulation between the two transition systems.

Last, we sketch the extension of this translation to a version of our π-calculus augmented with a restricted form of replication (input-guarded replication where the only free name of the replicated process is the subject of the input prefix, and moreover, this name is not free in the continuation of the replicated input prefix). We conclude the paper with several concrete examples, showing how various operational features of the π-calculus are modeled in differential interaction nets.

1. Differential interaction nets

1.1. Differential linear logic

In the fragment of linear logic we use, there are two constants 1 and $\perp$ and 4 connectives: $\otimes$ and $\parr$ which are binary (the multiplicative connectives) and ! and ? (the exponentials), which are unary. Given a formula A, its dual (or linear negation) $A^\perp$ is defined by induction: $(A \otimes B)^\perp = A^\perp \parr B^\perp$ etc. We present the logical system in a sequent calculus style, with unilateral sequents (all formulae are on the right side of the turnstyle symbol). The identity rules are the axiom and the cut rule:

$$\frac{}{\vdash A^\perp, A} \qquad \frac{\vdash \Gamma, A \qquad \vdash \Delta, A^\perp}{\vdash \Gamma, \Delta}$$

The multiplicative rules are:

$$\frac{\vdash \Gamma, A \qquad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \otimes B} \qquad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B} \qquad \frac{}{\vdash 1} \qquad \frac{\vdash \Gamma}{\vdash \Gamma, \perp}$$

The "standard" exponential rules are the weakening, contraction and dereliction rules:

$$\frac{\vdash \Gamma}{\vdash \Gamma, ?A} \qquad \frac{\vdash \Gamma, ?A, ?A}{\vdash \Gamma, ?A} \qquad \frac{\vdash \Gamma, A}{\vdash \Gamma, ?A}$$

The exponential rules which are new in differential linear logic are the coweakening, cocontraction and codereliction rules:

$$\frac{}{\vdash !A} \qquad \frac{\vdash \Gamma, !A \qquad \vdash \Delta, !A}{\vdash \Gamma, \Delta, !A} \qquad \frac{\vdash \Gamma, A}{\vdash \Gamma, !A}$$

The promotion rule is a standard rule of ordinary linear logic. It allows to turn a proof into a duplicable object:

$$\frac{\vdash ?A_1, \dots, ?A_n, B}{\vdash ?A_1, \dots, ?A_n, !B}$$

Because the reduction rules for the dereliction/cocontraction and codereliction-/contraction redexes produce formal sums of proofs, we have to introduce a rule for such sums.

$$\frac{\vdash \Gamma \quad \cdots \quad \vdash \Gamma}{\vdash \Gamma}$$

There is one such rule for each $n \in \mathbb{N}$ (the number of premises), and in particular for $n = 0$, so that each sequent is provable in this logic by a 0 proof: this means that our proofs should be considered as *partial* objects, just as Böhm trees in the lambda-calculus, which are partial lambda-terms (in this analogy, the Ω symbol of Böhm trees corresponds to the 0 proof).

The graphical formalism of interaction nets is much more convenient for representing this system, in particular when one wants to deal with the cut elimination rules (the reduction of the contraction/cocontraction cut is particularly unnatural in the sequent calculus presentation).

1.2. The general formalism of interaction nets

We recall now the general syntax of interaction nets, as introduced in [Laf95]. See also [ER06] for more details. Assume we are given a set of *symbols* and that an arity (a non-negative integer) and a typing rule is associated with each symbol. This typing rule is a list $(A_0, A_1, \ldots, A_n)$ of types, where n is the arity associated with the symbol. Types are formulae of some system of linear logic. A *net* is made of *cells*. With each cell γ is associated exactly one symbol and therefore an arity n and a typing rule $(A_0, A_1, \ldots, A_n)$. Such a cell γ has one *principal port* p_0 and n *auxiliary ports* $p_1, \ldots, p_n$. A net has also a finite set of *free ports*. All these ports (the free ports and the ports associated with cells) have to be pairwise distinct and a set of *wires* is given. This wiring is a set of pairwise disjoint sets of ports of cardinality 2 (ordinary wires) or 0 (loops[2]), and the union of these wires must be equal to the set of all ports of the net. In other words, each port of the net (free or associated with a cell) is connected to exactly one other port (free or associated with a cell) through a wire, and each such wire connects exactly two ports: ports cannot be shared. The free ports of the net are those which are not associated with a cell.

An *oriented wire* of the net is an ordered pair (p_1, p_2) where $\{p_1, p_2\}$ is a wire. In a net, a type is associated with each oriented wire, in such a way that if A is associated with (p_1, p_2), then $A^\perp$ is associated with (p_2, p_1). Last, the typing rules of the cells must be respected in the sense that for each cell γ of arity n, whose ports are $p_0, p_1, \ldots, p_n$ and typing rule is $(A_0, A_1, \ldots, A_n)$, denoting by $p'_0, p'_1, \ldots, p'_n$ the ports of the net uniquely defined by the fact that the sets $\{p_i, p'_i\}$ are wires (for $i = 0, 1, \ldots, n$), then the oriented wires (p_0, p'_0), (p'_1, p_1),…,(p'_n, p_n) have types A_0, A_1,…,A_n respectively.

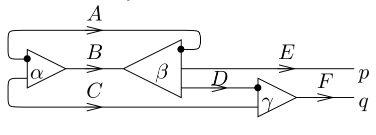


Figure 2: An interaction net

The free ports of the net constitute its *interface*. With each free port p can be associated the type of the unique oriented wire whose endpoint is p: this is the type of p in the interface of the net. Figure 2

[2]To be more precise, one has to specify the number of loops in the net, but this will not play any role in the sequel.

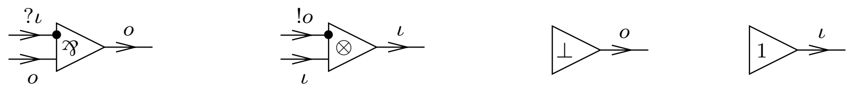
shows a typical example of a typed interaction net, with cells of symbols α, β and γ, of respective types $(B, A^\perp, C^\perp)$, $(B^\perp, A, E^\perp, D^\perp)$ and (F, D, C). The interface is $(p : E, q : F)$. Cells are represented as triangles, with principal port located at one of the angles and other ports on the opposite edge. We often draw a black dot to locate the auxiliary port number 1.

1.3. Presentation of the cells

Our nets will be typed using a type system which corresponds to the untyped lambda-calculus. This system is based on a single type symbol o (the type of outputs), subject to the recursive equation $o = ?o^\perp ⅋ o$. We set $\iota = o^\perp$, so that $\iota = !o \otimes \iota$ and $o = ?\iota ⅋ o$. The tensor connective is used only with premises $!o$ and ι and dually for the par, and therefore, the only types we actually need are o, ι, $!o$ and $?\iota$ for typing our nets.

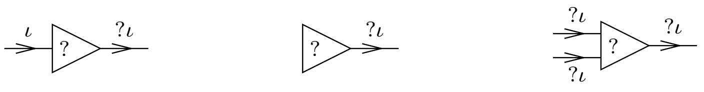
In the present setting, there are eleven symbols: par (arity 2), bottom (arity 0), tensor (arity 2), one (arity 0), dereliction (arity 1), weakening (arity 0), contraction (arity 2), codereliction (arity 1), coweakening (arity 0), cocontraction (arity 2) and closed promotion (arity 0). We present now the various cell symbols, with their typing rules, in a pictorial way.

1.3.1. Multiplicative cells. The *par* and *tensor* cells, and their "nullary" versions *bottom* and *one* are as follows:

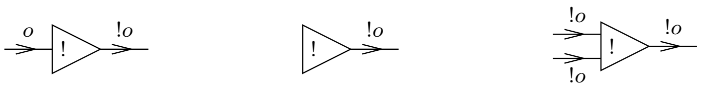


The first two cells are graphical representations of the ⅋ and $\otimes$ rules of Section 1.1. The last two cells are similar to the $\perp$ and 1 rules.

1.3.2. Exponential cells. They are typed according to a strictly polarized discipline. Here are first the *why not* cells, which are called *dereliction*, *weakening* and *contraction*:



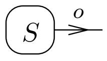
and then the *bang* cells, called *codereliction*, *coweakening* and *cocontraction*:

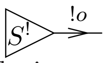


1.3.3. Closed promotion cells and the definition of nets. The notion of net is then defined inductively, together with *closed promotion* cells.

- A *simple differential net* is a typed interaction net, which uses the multiplicative and exponential cells introduced above as well as the closed promotion cells we are defining now.

- A *differential net* is a finite formal sum $S = s_1 + \cdots + s_n$ of simple differential nets having all the same interface, and this interface is then considered as the interface of S. A particular case is the net $S = 0$ (the empty sum), and this net has to be given together with its interface: there is a 0 net for each interface.

- Given a differential net S with only one free port 

 we introduce the *closed promotion* cell 

. This corresponds to the promotion box construction of linear logic nets, restricted here to the case where the resulting box has no "auxiliary ports". We say that s is the subnet of this promotion cell. There would be of course no difficulties in introducing more general promotion cells, with auxiliary ports, but we shall not use them in the present work.

In the sequel, since no confusion with other kinds of interaction nets will be possible, we shall use "net" for "differential net".

1.3.4. Logical correctness. It is easy to transform any[3] proof of the sequent calculus of Section 1.1 into a net made of these cells. The nets which result from this translation are exactly those which satisfy one of the various equivalent *correctness criteria* [Gir87, DR89, ...]: one says that such nets can be *sequentialized*[4]. One of the most remarkable features of interaction nets is that they allow to compute (using the forthcoming reduction rules), even on structures which cannot be sequentialized.

1.3.5. Labeled nets. We now introduce labels and labeled nets, which are nets where particular cells are equipped with labels. The labeled transition system of differential nets will be defined using these labels in Section 2.3. We shall also use these labels in Section 4 for defining a version of the π-calculus where prefixes are labeled, and for defining a transition system for this π-calculus. The main result of the paper will be a comparison between these two systems. These labels are not used for representing the names of the π-calculus, but just for identifying the various occurrences of names.

Let $\mathcal{L}$ be a countable set of labels containing a distinguished element τ (to be understood as the absence of label). A *labeled simple net* is a simple net where all dereliction, codereliction and promotion cells are equipped with labels belonging to $\mathcal{L}$.

All the nets we consider in this paper are labeled. In our pictures, the labels of dereliction, codereliction and box cells will be indicated, when this label is

[3]Not exactly any actually, because we consider only a restricted form of promotion in our differential interaction nets, but the general promotion rule can be translated as well, with more general nets.

[4]The criteria have to be extended to the differential setting. This is straightforward: cocontraction is handled like the tensor rule.

different from τ. When its label is τ, a (co)dereliction or box cell will be drawn without any label.

2. Reduction rules

We denote by $\mathbf{\Delta}$ the collection of all simple nets, ranged over by the letters s, t, u, with or without subscripts or superscripts, and by $\mathbb{N}\langle\mathbf{\Delta}\rangle$ the collection of all nets (finite sums of simple nets with the same interface), ranged over by the letters S, T, U, with or without subscripts or superscripts. We consider $\mathbf{\Delta}$ as a subset of $\mathbb{N}\langle\mathbf{\Delta}\rangle$ ($s \in \mathbf{\Delta}$ being identified with the sum made of exactly one copy of s).

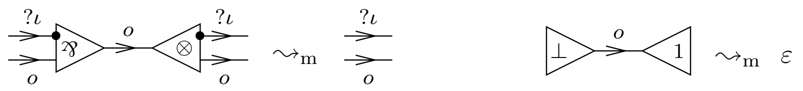
A *reduction rule* is a subset $\mathcal{R}$ of $\mathbf{\Delta} \times \mathbb{N}\langle\mathbf{\Delta}\rangle$ consisting of pairs (s, S) where s is a simple net made of two cells connected by their principal ports and S is a net that has the same interface as s. There are actually reduction rules which transform simple nets in non simple ones, see 2.1.3.

This set $\mathcal{R}$ can be finite or infinite. Such a relation is easily extended to arbitrary simple nets ($s\ \mathcal{R}\ T$ if there is $(s_0, u_1 + \cdots + u_n) \in \mathcal{R}$ where s_0 is a subnet of s, each u_i is a simple net and $T = t_1 + \cdots + t_n$ where t_i is the simple net resulting from the replacement of s_0 by u_i in s). This relation is extended to nets (sums of simple nets): $s_1 + \cdots + s_n$ (where each s_i is simple) is related to T by this extension $\mathcal{R}^\Sigma$ if $T = T_1 + \cdots + T_n$ where, for each i, $s_i\ \mathcal{R}\ T_i$ or $s_i = T_i$. Last, $\mathcal{R}^*$ is the transitive closure of $\mathcal{R}^\Sigma$ (which is reflexive).

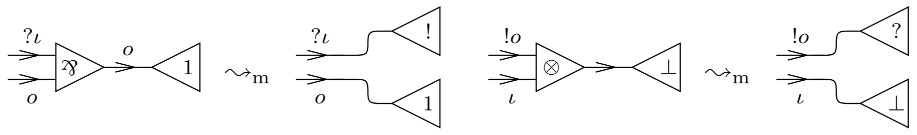
2.1. Defining the reduction

We give now the reduction rules of differential interaction nets. They correspond to the cut elimination rules of the differential linear logic of Section 1.1.

2.1.1. Multiplicative reduction. The first two rules concern the interaction of two multiplicative cells of the same arity.

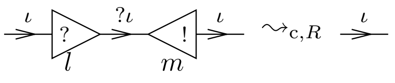


where ε stands for the empty simple net (not to be confused with the net $0 \in \mathbb{N}\langle\mathbf{\Delta}\rangle$, the empty sum, which is not a simple net). The next two rules concern the interaction between a binary and a nullary multiplicative cell.

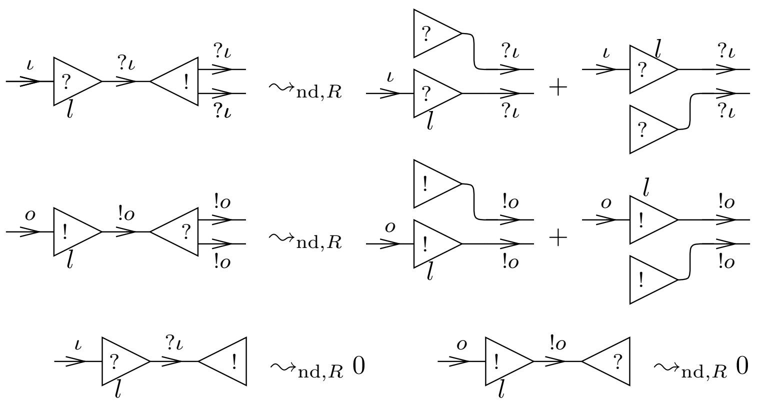


2.1.2. Communication reduction. This is in some sense the most fundamental reduction of the system: from the process calculus viewpoint, it corresponds to a communication between an input and an output prefix which have the same subject.

Let $R \subseteq \mathcal{L}$. We have the following reductions if $l, m \in R$.



2.1.3. Non-deterministic reduction. These rules will be used for implementing the non-determinism of the process calculus. Let $R \subseteq \mathcal{L}$. We have the following reductions if $l \in R$.

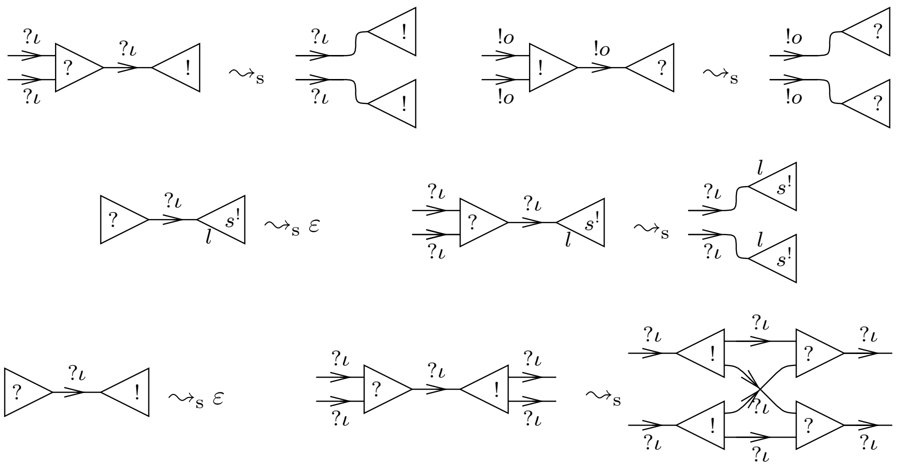


Remark 1 One can consider a sum $s_1 + \cdots + s_n$ of several simple nets as a non-deterministic superposition, and then a reduction $s \rightsquigarrow s_1 + \cdots + s_n$ can be interpreted as meaning that all the reductions $s \rightsquigarrow s_1$,…,$s \rightsquigarrow s_n$ are possible, but that the various outcomes s_i correspond to semantically distinct computations. In that case, there is an essential conflict between these various choices, as it should be clear in the rules above: in the two terms of the sums, we establish completely different connections in the net.

On the other hand, by reducing various redexes in s, it is also possible to obtain various results: $s \rightsquigarrow T_1$,…,$s \rightsquigarrow T_p$, but these choices of redexes in s commute with each other (this is the main content of Theorem 2), and the resulting nets T_1,…,T_p are semantically equivalent.

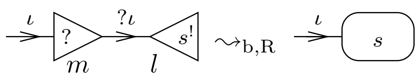
One of the main features of differential interaction nets is that they reify this distinction in the rewriting rules: in the first case s reduces to the net $s_1 + \cdots + s_n$ whereas in the second case, s reduces to each of the nets T_1,…,T_p. Moreover, this reification is compatible with (and actually, comes from) the denotational semantics of differential linear logic (see e.g. [Ehr05]), where these "non-deterministic sums" are interpreted as algebraic sums. Of course this distinction between two kinds of reduction is not new (it is pervasive in rewriting theory, in concurrency etc), what is new is its formalization in the present setting, using formal sums.

2.1.4. Structural reduction. From the process calculus viewpoint, these rules implement the associativity and commutativity laws of parallel composition which are implicit in the Chemical Abstract Machine [BB90], and in the abstract machine of Section 4.2. They also implement some of the laws associated with name restriction (scope extrusion in particular). They are called "structural" because they correspond to the interaction between the structural and the costructural rules of differential linear logic.



We use $\sim_s$ for the symmetric and transitive closure of $\rightsquigarrow_s$.

2.1.5. Box reduction. Let $R \subseteq \mathcal{L}$. We have the following reductions if $l, m \in R$.



Observe that the reduction rules are compatible with the identification of the coweakening cell with a promotion cell containing the 0 net. Observe also that the only rules which do not admit a "symmetric" rule are those which involve a promotion cell. Indeed, promotion is the only asymmetric rule of differential linear logic.

2.1.6. Completeness of the reduction. One can check that we have provided reduction rules for all redexes compatible with our typing system: for any simple net s made of two cells connected through their principal ports, there is a reduction rule whose left member is s. This rule is unique, up to the choice of a set of labels, but this choice has no influence on the right member of the rule.

2.1.7. Conditions on labeled nets. We say that a simple net s satisfies the *condition on labels for simple nets* if two labels associated with distinct cells[5] of s are either distinct or equal to τ. As such, this condition will not preserved under reduction, due to the fact that promotion cells are duplicated. Therefore, we reinforce this condition by requiring also that all the promotion cells of s be labeled by τ and all the labels occurring in subnets of promotion cells of s be equal to τ. We shall refer to the conjunction of these conditions as to the CLB (*condition on labels and boxes*).

One can also check, by simple inspection of the rules that, if t is a simple net which satisfies the CLB and if $t \rightsquigarrow t_1 + \cdots + t_n$ by one of our reduction rules, then all the simple nets t_i satisfy the CLB.

2.2. Confluence

Theorem 2 *Let $R, R', R'' \subseteq \mathcal{L}$. Let $\mathcal{R} \subseteq \mathbf{\Delta} \times \mathbb{N}\langle\mathbf{\Delta}\rangle$ be the union of some of the reduction relations $\rightsquigarrow_{\mathrm{c},R}$, $\rightsquigarrow_{\mathrm{nd},R'}$, $\rightsquigarrow_{\mathrm{m}}$, $\rightsquigarrow_{\mathrm{s}}$ and $\rightsquigarrow_{\mathrm{b,R''}}$. The relation $\mathcal{R}^*$ is confluent on $\mathbb{N}\langle\mathbf{\Delta}\rangle$.*

The proof is essentially trivial since the rewriting relation has no critical pair (see [ER06]). Given $R \subseteq \mathcal{L}$, we consider in particular the following reduction: $\rightsquigarrow_R = \rightsquigarrow_{\mathrm{m}} \cup \rightsquigarrow_{\mathrm{c},\{\tau\}} \cup \rightsquigarrow_{\mathrm{s}} \cup \rightsquigarrow_{\mathrm{b},\{\tau\}} \cup \rightsquigarrow_{\mathrm{nd},R}$. We set $\rightsquigarrow_{\mathrm{d}} = \rightsquigarrow_{\emptyset}$ ("d" for "deterministic") and denote by $\sim_{\mathrm{d}}$ the symmetric and transitive closure of this relation. Observe that, if s and S are nets with s simple and if $s \rightsquigarrow_{\mathrm{d}} S$, then S is also simple.

Some of the reduction rules we have defined depend on a set of labels. This dependence is clearly monotone in the sense that the relation becomes larger when the set of labels increases.

2.3. A transition system of simple nets

2.3.1. Restriction on simple nets. From now on, and until Section 6, we assume that all simple nets satisfy the CLB; remember that, together, these conditions are preserved under reduction. This will be sufficient for dealing with replication-free processes. The reason for this restriction is that the useful Lemmata 3 and 4 seem to depend on the uniqueness of label occurrences.

2.3.2. $\{l, m\}$-neutrality. Let l and m be distinct elements of $\mathcal{L} \setminus \{\tau\}$. We call (l, m)*-communication redex* a communication redex whose codereliction cell is labeled by l and whose dereliction cell is labeled by m.

The following is a simple, but quite useful remark.

Lemma 3 *Let s_0 be a simple net which contains an (l, m)-communication redex. If $s_0 \rightsquigarrow^*_{\{l,m\}} T_0$, then T_0 is a simple net t_0 which contains an (l, m)-communication redex and one has actually $s_0 \rightsquigarrow^*_{\mathrm{d}} t_0$. Moreover, if s is the*

[5]This means that they can also occur in subnets associated with promotion cells, at any depth.

simple net resulting from the reduction of the (l,m)-communication redex in s_0, then $s \rightsquigarrow_{\mathrm{d}}^{*} t$ where t is the simple net resulting from the reduction of the (l,m)-communication redex in t_0.

We say that a simple net s is $\{l,m\}$*-neutral* if, whenever $s \rightsquigarrow_{\{l,m\}}^{*} S$, none of the simple summands of S contains an (l,m)-communication redex.

Lemma 4 *Let s be a simple net. If $s \rightsquigarrow_{\{l,m\}}^{*} S$ where all the simple summands of S are $\{l,m\}$-neutral, then s is also $\{l,m\}$-neutral.*

The converse implication clearly holds, but we do not use it.

Proof. Assume, towards a contradiction, that $s \rightsquigarrow_{\{l,m\}}^{*} T = s_1 + \cdots + s_n$ where each s_i is simple and where s_1 contains an (l,m)-communication redex. By the Church-Rosser property of $\rightsquigarrow_{\{l,m\}}^{*}$, there is S' such that $T \rightsquigarrow_{\{l,m\}}^{*} S'$ and $S \rightsquigarrow_{\{l,m\}}^{*} S'$. By Lemma 3 applied to s_1, S' must have a summand containing an (l,m)-communication redex, contradicting our hypothesis on S. □

2.3.3. The transition system. We define a labeled transition system $\mathbb{D}_{\mathcal{L}}$ whose objects are simple nets, and transitions are labeled by pairs of distinct elements of $\mathcal{L} \setminus \{\tau\}$. Let s and t be simple nets, we have $s \xrightarrow{l\overline{m}} t$ if the following holds: $s \rightsquigarrow_{\{l,m\}}^{*} s_0 + s_1 + \cdots + s_n$ where s_0 is a simple net which contains an (l,m)-communication redex and becomes t when one reduces this redex, and each s_i (for $i > 0$) is $\{l,m\}$-neutral.

Remark 5 The simple nets $s_1, \ldots, s_n$ correspond to other possible communications, where typically the codereliction labeled by l will meet a dereliction labeled by some $m' \neq m$, and similarly for the dereliction labeled by m. So these terms are not garbage but correspond to the branches of the non-deterministic reductions which do not lead to a communication between l and m. There are two restrictions in our definition which deserve further comments:

- The non-deterministic steps allowed in the reduction from s to $s_0 + s_1 + \cdots + s_n$ can involve only the codereliction and dereliction labeled by l and m respectively. In process algebras, prefixes communicate in one step through a parallel composition. This single step becomes here a sequence of many elementary steps and our restriction allows to avoid considering the steps which have nothing to do with the communication we are interested in.

- The second restriction consists in requiring the s_is to be (l,m)-neutral for $i > 0$ and seems to potentially prune out relevant (l,m) communications from the LTS $\mathbb{D}_{\mathcal{L}}$, and therefore to weaken Proposition 11 and hence Theorem 12.

We think that Theorem 12 would hold even without these restrictions in the definition of $\mathbb{D}_{\mathcal{L}}$, which are here only for making the proofs tractable. In the final remark of the Conclusion, we shortly argue that the second restriction is not essential. The first one can probably be weakened as well.

Lemma 6 *The relation $\sim_{\mathrm{d}} \subseteq \mathbf{\Delta} \times \mathbf{\Delta}$ is a strong bisimulation on $\mathbb{D}_{\mathcal{L}}$.*

Proof. Let $s, s' \in \mathbf{\Delta}$ and assume that $s \sim_{\mathrm{d}} s'$. Assume moreover that $s \xrightarrow{l\overline{m}} t$, which means that $s \rightsquigarrow^*_{\{l,m\}} s_0 + s_1 + \cdots + s_n$ where each s_i is simple, s_0 contains an (l,m)-communication redex, each s_i is $\{l,m\}$-neutral for $i \geq 1$ and t results from the reduction of the (l,m)-communication redex of s_0. By the Church-Rosser property of $\rightsquigarrow^*_{\{l,m\}}$ (remember that $\rightsquigarrow_{\mathrm{d}} \subseteq \rightsquigarrow^*_{\{l,m\}}$), there exists $U \in \mathbb{N}\langle\mathbf{\Delta}\rangle$ such that $s_0 + s_1 + \cdots + s_n \rightsquigarrow^*_{\{l,m\}} U$ and $s' \rightsquigarrow^*_{\{l,m\}} U$. But by Lemmata 3 and 4, we have $U = u_0 + u_1 + \cdots + u_m$ with $s_0 \rightsquigarrow^*_{\mathrm{d}} u_0$, u_0 contains an (l,m)-communication redex, and if we reduce this redex, we obtain a net t' such that $t \rightsquigarrow^*_{\mathrm{d}} t'$. □

3. A toolbox for process calculi interpretation

We introduce now a few families of simple nets, which are built using the previously introduced basic cells. They will be used as basic modules for interpreting processes. All of these nets, but the communication areas, can be considered as *compound cells*: in reduction, they behave in the same way as cells of interaction nets. We advise the reader acquainted with the π-calculus to have simultaneously a look at Section 4.3 in order to figure out how these various structures will be used.

3.1. Compound cells

3.1.1. Generalized contraction and cocontraction. A *generalized contraction cell* or *contraction tree* is a simple net γ (with one principal port and a finite number of auxiliary ports) which is either a wire or a weakening cell or a contraction cell whose auxiliary ports are connected to the principal port of other contraction trees, whose auxiliary ports become the auxiliary ports of γ. Generalized cocontraction cells (cocontraction trees) are defined dually.

We use the same graphical notations for generalized (co)contraction cells as for ordinary (co)contraction cells, with a "$*$" in superscript to the "!" or "?" symbols to avoid confusions. Observe that there are infinitely many generalized (co)contraction cells of any given arity. Figure 3 gives an example of a ternary generalized cocontraction cell.

3.1.2. The dereliction-tensor and the codereliction-par cells. Let n be a non-negative integer. We define an n-ary $?\otimes$ compound cell as in Figure 4. It will be decorated by the label of its dereliction cell (if different from τ). The number of tensor cells in this compound cell is equal to n. We define dually the $!\invamp$ compound cell.

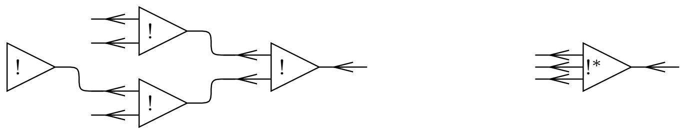


Figure 3: A ternary generalized cocontraction cell and its graphical representation (all oriented wires are typed with $?\iota$)

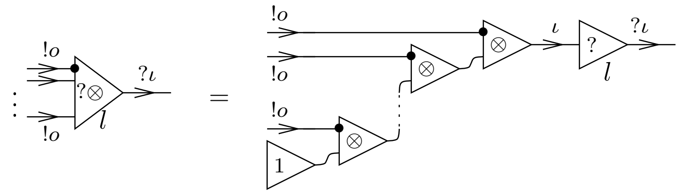


Figure 4: Dereliction-tensor compound cell (the codereliction-par is dual)

3.1.3. The prefix cells. Now we can define the compound cells which will play the main role in the interpretation of prefixes of the π-calculus. Thanks to the above defined cells, all the oriented wires of the nets we shall define will have type $?\iota$ or $!o$. Therefore, we adopt the following graphical convention: the wires will have an orientation corresponding to the $?\iota$ type.

The *n-ary input cell* and the *n-ary output cell* are defined in Figure 5, they have n pairs of auxiliary ports. In Section 6, we shall also use a version of the input prefix where the codereliction cell has been removed. The main port of this *pre-input cell* has therefore type o (when oriented towards the outside) instead of $!o$. We use the same notation as for the input cell (Figure 5), with the only difference that the symbol "!" will be replaced by the symbol "$\overline{\parr}$". See an example in Figure 6.

Prefix cells are labeled by the label carried by their outermost $?\otimes$ or $!\parr$ compound cell, if different from τ, the other $?\otimes$ or $!\parr$ compound cells being unlabeled (that is, labeled by τ).

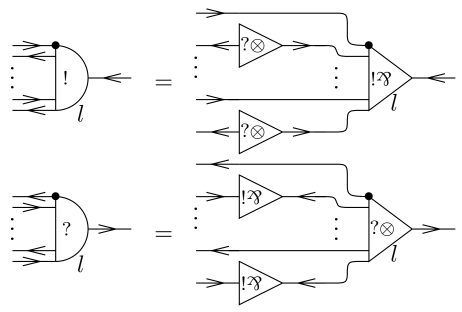


Figure 5: Input and output compound cells

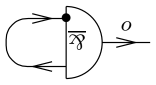
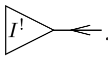


Figure 6: Identity

3.1.4. Transistors and boxed identity. In order to implement the sequentiality corresponding to sequences of prefixes in the π-calculus, we shall use the unary output prefix cell defined above as a kind of transistor, that is, as a kind of switch that one can put on a wire, and which is controlled by another wire. This idea is strongly inspired by the translation of the π-calculus in the solos calculus [6].

These switches will be closed by "boxed identity cells", which are the unique use we make of promotion in the present work (apart from the extension sketched in Section 6). Let I be the "identity" net of Figure 6, which uses a pre-input compound cell. Then we shall use the closed promotion cell labeled by $I^!$: .

3.2. Communication tools

3.2.1. The communication areas. Let $n \geq -2$. We define a family of nets with $2(n+2)$ free ports, called *communication areas of order* n, that we shall draw using rectangles with beveled angles. Figure 7 shows how we picture a communication area of order 3.

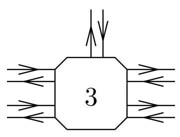


Figure 7: Area of order 3

A communication area of order n is made of $n+2$ pairs of $(n+1)$-ary generalized cocontraction and contraction cells (γ_1^+, γ_1^-) ,..., $(\gamma_{n+2}^+, \gamma_{n+2}^-)$, with, for each i and j such that $1 \leq i < j \leq n+2$, a wire from an auxiliary port of γ_i^+ to an auxiliary port of γ_j^- and a wire from an auxiliary port of γ_i^- to an auxiliary port of γ_j^+.

So the communication area of order -2 is the empty net ε, and communication areas of order -1, 0, 1 and 2 are the structures shown in Figure 8.

3.2.2. Identification structures. Let $n, p \in \mathbb{N}$ and let $f : \{1, \ldots, p\} \rightarrow \{1, \ldots, n\}$ be a function. An *f-identification net* is a structure with $p+n$ pairs of free ports (p pairs correspond to the domain of f and, in our pictures, will be attached to the non beveled side of the identification structure, and n pairs correspond to the codomain of f, attached to the beveled side of the structure) as in Figure 9(a). Such a net is made of n communication areas, and on the j-th area, the j-th pair of wires of the codomain is connected, as well as the pairs of wires of index i of the domain such that $f(i) = j$. For instance, if $n = 4$,

[6]It is shown in [LV03] that one can encode the π-calculus sequentiality induced by prefix nesting in the completely asynchronous solos formalism: the idea of such translations is to observe that, in a solos process like $P = \nu y\,(u(x,y) \mid y(\ldots)) \mid Q$, the second solo cannot interact with the environment Q before the first one.

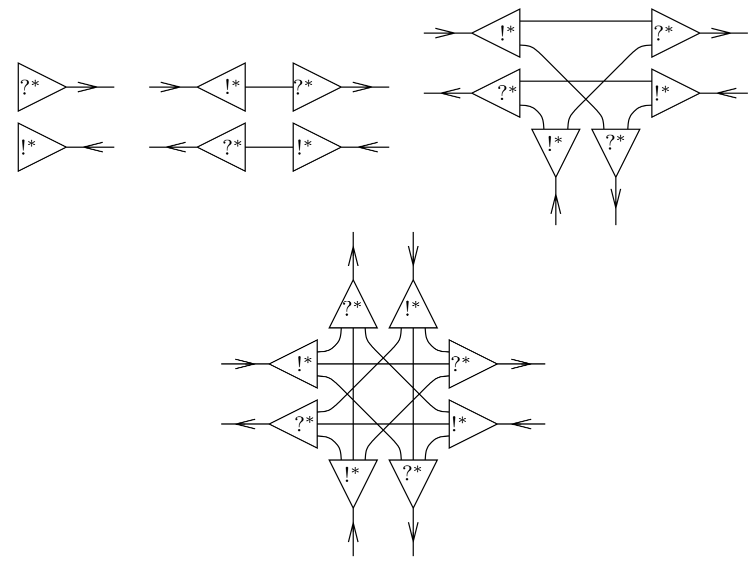


Figure 8: Communication areas of order -1, 0, 1 and 2

$p = 3$, $f(1) = 2$, $f(2) = 3$ and $f(3) = 2$, a corresponding identification structure is made of four communication areas, two of order -1, one of order 0 and one of order 1, as in Figure 9(b).

When we want to mention a particular communication area of such a structure, we refer to it as to the j-th communication area (where j is the corresponding element of $\{1, \dots, n\}$).

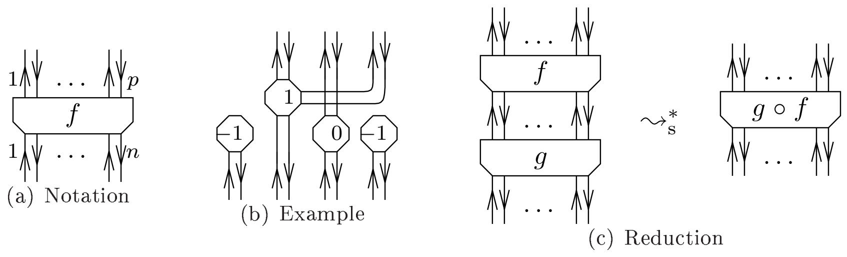


Figure 9: Identification structures

3.3. Useful reductions.

3.3.1. Aggregation of communication areas. One of the nice properties of communication areas is that, when one connects two such areas through a pair of wires, one gets another communication area; if the two areas are of respective orders $p \geq -1$ and $q \geq -1$, the resulting area is of order $p + q$, see Figure 10.

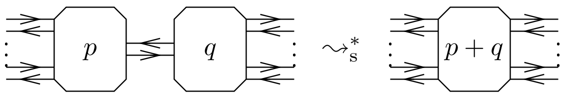


Figure 10: Aggregation, with $p, q \geq -1$

3.3.2. Composition of identification structures. In particular, we get the reduction of Figure 9(c).

3.3.3. Port forwarding in a net. Let t be a net and p be a free port of t. We say that p *is forwarded in* t if there is a free port q of t such that t is of one of the two shapes given in Figure 11. When a port is forwarded in a net, we mark this port with a small triangle, as in Figures 12 and 13.

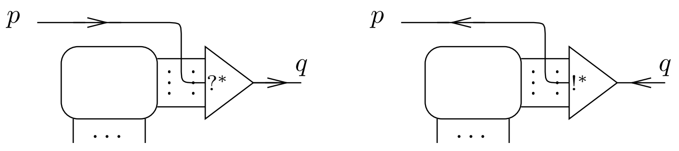


Figure 11: Port forwarding

3.3.4. Communication and forwarding of derelictions and coderelictions in communication areas. The reduction of Figure 12 shows that derelictions and coderelictions can meet each other, when connected to a common communication area. More precisely, let $l, m \in \mathcal{L}$, then we have the reduction of Figure 12, where N is a non-negative integer (actually, $N = (p+2)^2$) and, in

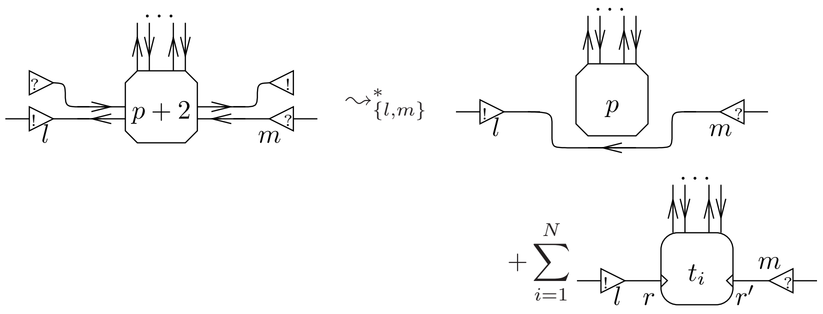


Figure 12: Dereliction and codereliction communicating through a communication area. The forwarded ports are indicated by small triangles.

each simple net t_i, both ports r and r' are forwarded.

3.3.5. General forwarding. Let $l \in \mathcal{L}$. The more general but less informative property shown in Figure 13 will also be used, where in each simple net

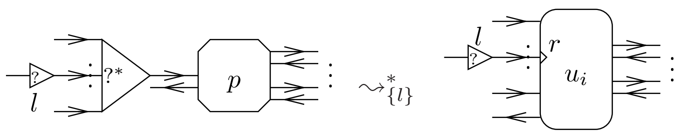
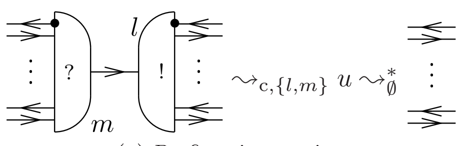


Figure 13: General forwarding

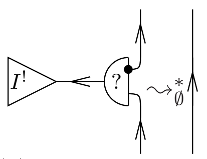
u_i, the port r is forwarded (see 3.3.3). Of course one also has a dual reduction (where the dereliction is replaced by a codereliction, and the generalized contraction by a generalized cocontraction).

3.3.6. Reduction of prefixes. Let $l, m \in \mathcal{L}$. If we connect an n-ary output prefix labeled by m to a p-ary input prefix labeled by l, we obtain a net which reduces by $\rightsquigarrow_{\mathrm{c},\{l,m\}}$ to a net u which reduces by $\rightsquigarrow^*_{\{\tau\}}$ to 0 if $n \neq p$ and to simple wires by $\rightsquigarrow^*_{\emptyset}$, as in Figure 14(a), if $n = p$.

3.3.7. Transistor triggering. A boxed identity connected to the principal port of a unary output cell used as a "transistor" turns it into a simple wire as in Figure 14(b).


?
m
l
!
$\rightsquigarrow_{\mathrm{c},\{l,m\}}$ u $\rightsquigarrow^*_{\emptyset}$

(a) Prefixes interaction



(b) Transistor triggering

Figure 14: Prefixes and transistors

4. A polyadic finitary π-calculus and its encoding

The process calculus we consider is a fragment of the π-calculus where we have suppressed the following features: sums, replication, recursive definitions, match and mismatch. This does not mean of course that differential interaction nets cannot interpret these features. We shortly discuss this point in the Conclusion.

It is well known that the monadic π-calculus is as expressive as the polyadic one. We nevertheless consider a polyadic version of the π-calculus because our encoding can easily be adapted to other process algebras, and in particular to asynchronous ones (such as the solos calculus), where polyadicity is essential for expressiveness. Moreover, polyadic calculi are more natural and widely used in the process algebra community.

Let $\mathcal{N}$ be a countable set of names. Our processes are defined by the following syntax. We use the same set $\mathcal{L}$ of labels as before.

- nil is the empty process.
- If P_1 and P_2 are processes, then $P_1 \mid P_2$ is a process.
- If P is a process and $a \in \mathcal{N}$, then $\nu a \cdot P$ is a process. The name a is bound in this process.
- If P is a process, $a, b_1, \dots, b_n \in \mathcal{N}$, the b_is being pairwise distinct and if $l \in \mathcal{L}$, then $Q = [l]a(b_1 \dots b_n) \cdot P$ is a process (prefixed by an input action, whose subject is a and whose objects are the b_is; the name a is free and each b_i is bound in Q and hence a is distinct from each b_i).
- If P is a process, $a, b_1, \dots, b_n \in \mathcal{N}$ and $l \in \mathcal{L}$, then $\overline{[l]a}\langle b_1 \dots b_n\rangle \cdot P$ is a process (prefixed by an output action, whose subject is a and whose objects are the b_is). This construction does not bind the names b_i, and we do not require the b_is to be distinct. The name a can be equal to some of the b_is.

We introduce this labeling of prefixes to distinguish the various occurrences of names as subject of prefixes; these labels do not play any active role in the reduction of processes, they are here only for tracing purposes. The set $\mathsf{FV}(P)$ of free names of a process P is defined in the obvious way. The α-equivalence relation on processes is defined as usual.

A *labeled process* is a process where all prefixes are labeled, by pairwise distinct labels, all these labels being different from τ. If P is a labeled process, $\mathcal{L}(P)$ denotes the set of all labels occurring in P. Observe that this set has a natural poset (forest actually) structure ($l < m$ if, in P, l labels a prefix μ and m occurs in the process prefixed by μ).

All the processes we consider in this paper are labeled.

4.1. Arity typing of processes.

Although not strictly necessary, it is convenient to assume that our processes are "typed" (one often speaks rather of "sorting" in this context) in the sense that each name is given with an arity, which is a possibly empty list of arities. When a name of arity $(\rho_1, \dots, \rho_n)$ occurs as subject, it is always assumed that it has n objects $b_1, \dots, b_n$, the arity of b_i being ρ_i. This guarantees that, during the reduction, when an input prefix communicates with an output prefix, the numbers of objects of the two involved prefixes coincide. Since this is a standard π-calculus notion (see [SW01, Part III]), we shall not say more about it, and we shall simply assume that, during the reduction of processes and states, the arities of communicating prefixes always coincide.

4.2. An execution model

Rather than considering a rewriting relation on processes as one usually does, we prefer to define an "environment machine", similar to the machine introduced in [AC98, Chapter 16][7], which itself is based on the Chemical Abstract Machine of Berry and Boudol [BB90]. It is not difficult to show that this presentation of the π-calculus is equivalent to more standard ones.

An *environment* is a function e from a finite subset $\mathsf{Dom}\, e$ of $\mathcal{N}$ to a finite subset $\mathsf{Codom}\, e$ of $\mathcal{N}$. A *closure* is a pair (P, e) where P is a process and e is an environment such that $\mathsf{FV}(P) \subseteq \mathsf{Dom}(e)$. A *soup* is a multiset $\Gamma = (P_1, e_1) \cdots (P_N, e_N)$ of closures (denoted by simple juxtaposition). The set $\mathsf{FV}(\Gamma)$ of free names of a soup Γ is the union of the codomains of the environments of Γ. The soup Γ is labeled if all the P_is are labeled, with pairwise disjoint sets of labels. A *state* is a pair (Γ, L) where Γ is a soup and L is a set of names (the names which have to be considered as local to the state) and we set $\mathsf{FV}(\Gamma, L) = \mathsf{FV}(\Gamma) \setminus L$. The state (Γ, L) is labeled if the soup Γ is labeled.

All the states we consider are labeled. We define the poset $\mathcal{L}(\Gamma, L)$ of all labels of the state (Γ, L) in the straightforward way, as the parallel composition of the posets associated with the processes of the closures of Γ.

4.2.1. α-equivalence of states. Given a partial function $f : \mathcal{N} \to \mathcal{N}$ and a process P, we denote by $f \cdot P$ the process where each free name a has been replaced by $f(a)$ (if $a \in \mathsf{Dom}\, f$) — this construction is not part of the syntax, it is a meta-operation like substitution in the lambda-calculus. Of course, bound names have to be renamed to avoid name clashes.

Two closures (P_1, e_1) and (P_2, e_2) are α-equivalent (written $(P_1, e_1) \sim_\alpha (P_2, e_2)$) if there is a bijection on names f such that $f \cdot P_1$ and P_2 are α-equivalent, and $e_2 \circ f = e_1$. Two soups Γ and Δ are α-equivalent if $\Gamma = \gamma_1 \ldots \gamma_N$ and $\Delta = \delta_1 \ldots \delta_N$ with $\gamma_i \sim_\alpha \delta_i$ for each i. Let $f : \mathcal{N} \to \mathcal{N}$ be a function. If $\gamma = (P, e)$ is a closure, one sets $f \cdot \gamma = (P, f \circ e)$. And last, $f \cdot (\gamma_1 \ldots \gamma_N) = (f \cdot \gamma_1) \cdots (f \cdot \gamma_N)$.

Two states (Γ, L) and (Δ, M) are α-equivalent if there is a bijection on names f which is the identity on $\mathcal{N} \setminus L$ and satisfies $f(L) = M$ and $f \cdot \Gamma \sim_\alpha \Delta$.

4.2.2. Canonical form of a state. We say that a process is *guarded* if it starts with an input prefix or an output prefix. We say that a soup $\Gamma = (P_1, e_1) \cdots (P_N, e_N)$ is *canonical* if each P_i is guarded, and that a state (Γ, L) is canonical if the soup Γ is canonical. We define a rewriting relation $\rightsquigarrow_{\mathsf{can}}$ which

[7]The reason for this choice is that the rewriting approach uses an operation which consists in replacing a name by another name in a process. The corresponding operation on nets is rather complicated and we prefer not to define it here.

turns any state into a canonical one.

$$\begin{array}{rcl} ((\mathsf{nil}, e)\Gamma, L) & \rightsquigarrow_{\mathsf{can}} & (\Gamma, L) \\ ((\nu a \cdot P, e)\Gamma, L) & \rightsquigarrow_{\mathsf{can}} & ((P, e[a \mapsto a'])\Gamma, L \cup \{a'\}) \\ ((P \mid Q, e)\Gamma, L) & \rightsquigarrow_{\mathsf{can}} & ((P, e)(Q, e)\Gamma, L) \end{array}$$

where, in the second rule, $a' \in \mathcal{N} \setminus (L \cup \mathsf{Codom}(e) \cup \mathsf{FV}(\Gamma))$. It is easy to show that, up to α-equivalence, this reduction relation is confluent, and it is clearly strongly normalizing. We denote by $\mathsf{Can}(\Gamma, L)$ the normal form of the state (Γ, L) for this rewriting relation. Observe that if $(\Gamma, L) \rightsquigarrow_{\mathsf{can}} (\Delta, M)$ then $\mathsf{FV}(\Delta, M) \subseteq \mathsf{FV}(\Gamma, L)$.

4.2.3. Transitions. Next, we define a labeled transition system $\mathbb{S}_{\mathcal{L}}$. The objects of this system are labeled canonical states and the transitions, labeled by pairs of labels, are defined as follows.

$$\begin{array}{l} (([l]a(b_1 \dots b_n) \cdot P, e)(\overline{[m]a'}\langle b'_1 \dots b'_n\rangle \cdot P', e')\Gamma, L) \\ \qquad \xrightarrow{l\overline{m}} \mathsf{Can}((P, e[b_1 \mapsto e'(b'_1), \dots, b_n \mapsto e'(b'_n)])(P', e')\Gamma, L) \end{array}$$

if $e(a) = e'(a')$. Observe that if $(\Gamma, L) \xrightarrow{l\overline{m}} (\Delta, M)$ then $\mathsf{FV}(\Delta, M) \subseteq \mathsf{FV}(\Gamma, L)$.

4.3. Translation of processes to differential interaction nets

Since we do not work up to associativity and commutativity of contraction and cocontraction, it does not make sense to define this translation as a function from processes to nets. For each repetition-free list of names $a_1, \dots, a_n$, we define a relation $\mathcal{I}_{a_1,\dots,a_n}$ from processes whose free names are contained in $\{a_1, \dots, a_n\}$ to simple nets t which have $2n+1$ free ports $a_1^\iota, a_1^o, \dots, a_n^\iota, a_n^o$ and $\mathbf{c}$ as in Figure 15(a). The additional port $\mathbf{c}$ will be used for controlling the sequentiality of the reduction, thanks to transistors. Reducing the translation of a process will be possible only when a boxed identity cell is connected to its control port. This is completely similar to the additional control free name in the translation of the π-calculus in solos, in [LV03][8].

It will be possible to check that, if P and P' are α-equivalent, then $P\ \mathcal{I}_{a_1,\dots,a_n}\ s$ iff $P'\ \mathcal{I}_{a_1,\dots,a_n}\ s$. We define now the translation relation, by induction on processes. And next we define the translation relation for states.

4.3.1. Empty process. One has $\mathsf{nil}\ \mathcal{I}_{b_1,\dots,b_n}\ t$ if t is as in Figure 15(b).

[8]There is a simple interpretation of solo diagrams into differential interaction nets, which uses only our toolbox without promotion so that solo diagrams can be seen as an intermediate graphical language which can be implemented in the low level differential syntax. Our translation of the π-calculus results from an analysis and a simplification of the composed translation "π-calculus $\to$ solo diagrams $\to$ differential nets". The simplification results from some rewiring and from the use of the boxed identity cells which are easily replicable. The translation of solos into differential nets leads to cycles (which appear when a name is identified with itself) which are avoided in the present direct translation. Well behaved conditions on solos for avoiding such cycles are introduced and studied in [EL08].

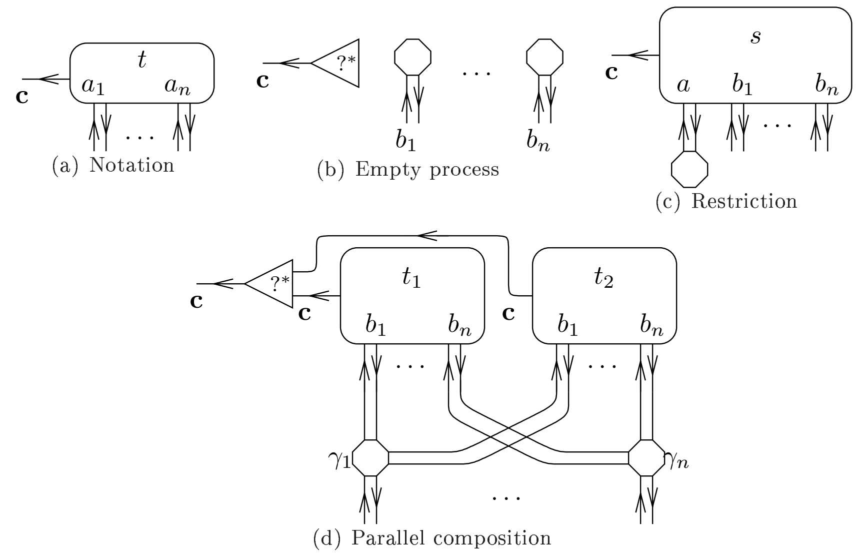


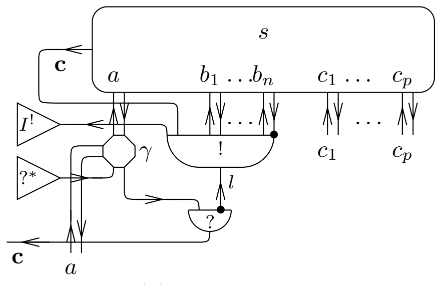
Figure 15: Translation of processes: structural constructions, see Section 4.3

4.3.2. Name restriction. One has $\nu a \cdot P \, \mathcal{I}_{b_1,\dots,b_n} \, t$ iff t is as in Figure 15(c), with s satisfying $P \, \mathcal{I}_{a,b_1,\dots,b_n} \, s$.

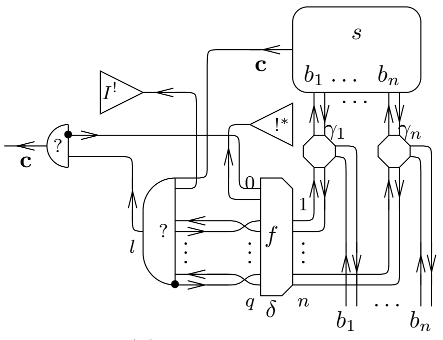
4.3.3. Parallel composition. One has $P_1 \mid P_2 \, \mathcal{I}_{b_1,\dots,b_n} \, t$ iff the simple net t is as in Figure 15(d), where $P_1 \, \mathcal{I}_{b_1,\dots,b_n} \, t_1$, $P_2 \, \mathcal{I}_{b_1,\dots,b_n} \, t_2$ and $\gamma_1, \dots, \gamma_n$ are communication areas of order 1.

4.3.4. Input prefix. Let $l \in \mathcal{L}$. Assume that $a, b_1, \dots, b_n, c_1, \dots, c_p$ are pairwise distinct names and let $Q = [l]a(b_1 \dots b_n) \cdot P$. One has $Q \, \mathcal{I}_{a,c_1,\dots,c_p} \, t$ if t is as in Figure 16(a), where γ is a communication area of order 1 and where s is a simple net which satisfies $P \, \mathcal{I}_{a,b_1,\dots,b_n,c_1,\dots,c_p} \, s$. The communication area γ is required to endow the channel a with a further input communication capability and making it available to the environment.

4.3.5. Output prefix. Let $l \in \mathcal{L}$. Let $b_1, \dots, b_n$ be a list of pairwise distinct names and let $Q = \overline{[l]b_{f(0)}}\langle b_{f(1)} \dots b_{f(q)} \rangle \cdot P$, where $f : \{0, 1, \dots, q\} \to \{1, \dots, n\}$ is a function (this function is uniquely determined by Q and by the enumeration $b_1, \dots, b_n$). So $b_1, \dots, b_n$ is a list of pairwise distinct names containing all the names of the prefix we want to translate and the function f says where each name occurs in the prefix; observe that some names of the list can be omitted in the prefix (f is not necessarily surjective). One has $Q \, \mathcal{I}_{b_1,\dots,b_n} \, t$ if t is as in Figure 16(b), where $\gamma_1, \dots, \gamma_n$ are communication areas of order 1, δ is an f-identification structure and where s is a simple net which satisfies $P \, \mathcal{I}_{b_1,\dots,b_n}$ s. This identification structure and the additional communication areas are required because the names occurring in the output prefix are not necessarily



(a) Input prefix



(b) Output prefix

Figure 16: Translation of processes: prefix constructions, see Section 4.3

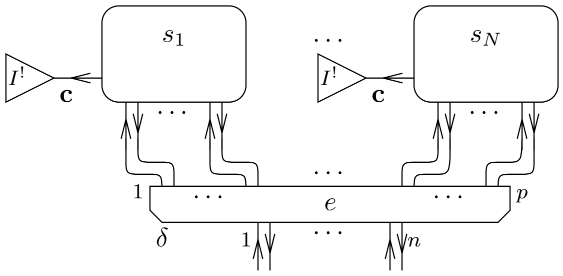


Figure 17: State translation, see Section 4.3

distinct from each other, and the object names are not bound by the output prefix: the identification structures implement these equalities between names and the communication areas make the corresponding communication channels available to the environment. These structures are not required in an input prefix because, in such a prefix, the object names are bound, pairwise distinct and distinct from the subject name which is free in the prefix.

4.3.6. States. Let $\Gamma = (P_1, e_1) \dots (P_N, e_N)$ be a soup and $b_1, \dots, b_n$ be a repetition-free list of names containing all the codomains of the environments $e_1, \dots, e_N$ (that is, containing $\mathsf{FV}(\Gamma)$). We assume that the domains of the environments e_i are pairwise disjoint, which is possible up to α-equivalence. Let $a_1, \dots, a_p$ be a repetition-free enumeration of the elements of $\bigcup_{i=1}^{N} \mathsf{Dom}\, e_i$, such that there is a list of non-negative integers $0 = h_0 \leq h_1 \leq \dots \leq h_N = p$ such that, for $i = 1, \dots, N$, the list $a_{h_{i-1}+1}, \dots, a_{h_i}$ is a repetition-free enumeration of the elements of $\mathsf{Dom}(e_i)$. Let $e : \{1, \dots, p\} \to \{1, \dots, n\}$ be the map which is uniquely defined by the fact that, for each $i = 1, \dots, N$ and each j such that $h_{i-1} + 1 \leq j \leq h_i$, one has $e_i(a_j) = b_{e(j)}$.

Then one has $\Gamma\, \mathcal{I}_{b_1,\dots,b_n}\, t$ if t is a simple net of the shape shown in Figure 4.3, where $s_1, \dots, s_N$ are simple nets such that $P_i\, \mathcal{I}_{a_{h_{i-1}+1},\dots,a_{h_i}} s_i$ and δ is an e-identification structure.

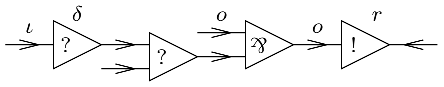


Figure 18: A guarding path from the dereliction δ to the codereliction r.

Last, if we are moreover given $L \subseteq \mathcal{N}$ and a repetition-free list of names $b_1, \ldots, b_n$ containing all the free names of the state (Γ, L), one has $(\Gamma, L)\ \mathcal{I}_{b_1,\ldots,b_n}\ u$ if one has $\Gamma\ \mathcal{I}_{b_1,\ldots,b_n,c_1,\ldots,c_p}\ t$ for some repetition-free enumeration $c_1, \ldots, c_p$ of L (assumed of course to be disjoint from $b_1, \ldots, b_n$, which is always possible up to α-equivalence), and u is the simple net t with additional communication areas of order -1 plugged on its pairs of free ports corresponding to the c_js.

A simple inspection of the translation above shows that, if $(\Gamma, L)\ \mathcal{I}_{b_1,\ldots,b_n}\ u$, then the simple net u satisfies the CLB of 1.3.5.

Before reading the following technical developments, it might be a good idea to have a look at Section 7 where examples of our translation are given.

5. Comparing the transition systems

5.1. A diving lemma

We first introduce the auxiliary notions of guarded cell and of a dereliction or codereliction cell diving into a process. We then state and prove two lemmata which will be crucial in proving Propositions 10 and 11. These propositions express the two directions in the main bisimulation result of the paper, Theorem 12.

5.1.1. Guarded dereliction and codereliction cells. Let $l, r \in \mathcal{L}$ be distinct, $r \neq \tau$ and let $s \in \mathbf{\Delta}$. Let δ be a (co)dereliction cell labeled by l in s. We say that δ is *guarded by* (the dereliction or codereliction cell labeled by) r in s if there is a sequence $p_1, \ldots, p_n$ of pairwise distinct ports of s such that

- p_1 is the auxiliary port of δ and p_2 is its principal port;
- p_{n-1} is the auxiliary port of r and p_n is its principal port;
- and for each i with $1 < i < n-1$, either p_i and p_{i+1} are the two ports of a wire of s or there is a cell in s such that p_i is an auxiliary port of that cell and p_{i+1} is its principal port.

Such a sequence of ports will be called a *guarding path* from δ to r in s (observe that since $r \neq \tau$, there is no ambiguity on the (co)dereliction cell labeled by r in s, whereas l can be equal to τ and so there might be several (co)dereliction cells labeled by l in s). See Figure 18 for an example of such a path.

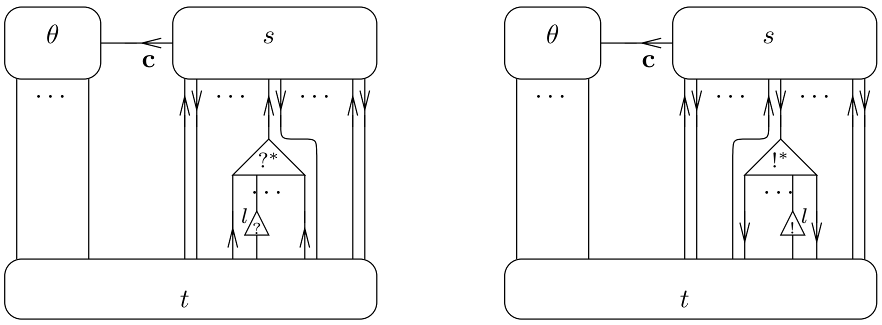


Figure 19: Diving of dereliction and codereliction: initial configurations

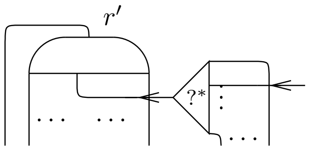


Figure 20: Possible shape for the subnet θ of Figure 19

5.1.2. Persistency.

Lemma 7 *Let s be a simple net, let $R \subseteq \mathcal{L}$, let l, r be labels which are distinct, with $r \neq \tau$. Let δ be an l-labeled (co)dereliction cell which is guarded by r in s and assume that $s \rightsquigarrow_R^* s_1 + \cdots + s_p$ where the s_i are simple. Then δ and r occur, and δ is guarded by r, in each of the simple nets s_i.*

Proof. The proof is straightforward: the (co)dereliction r can take part only in non-deterministic reductions during an $\rightsquigarrow_R$-reduction, and hence cannot disappear (more precisely, its only way of disappearing is by turning to 0 the whole simple net where it occurs). Hence the guarding path from δ to r is preserved during this reduction since its cells are not involved in any redex. □

5.1.3. Diving of derelictions and coderelictions. Let $l \in \mathcal{L} \setminus \{\tau\}$, let u be a simple net, let P be a process. We say that l *dives into P in u* if there is a repetition-free list of names $b_1, \dots, b_n$ and a simple net s such that $P \, \mathcal{I}_{b_1,\dots,b_n} \, s$ and u is of one of the shapes (according to whether l labels a dereliction or a codereliction cell) shown in Figure 19, where θ is either a boxed identity cell or a net of the shape shown in Figure 20, consisting of a labeled input or output prefix compound cell, with a label $r' \neq \tau$.

With these notations, our aim is here to prove the following property.

Lemma 8 (Diving) *Assume that $l \in \mathcal{L} \setminus \{\tau\}$ dives into P in the simple net u, and let $m \in \mathcal{L} \setminus \{\tau\}$ be a label which does not occur in P. Then u is $\{l, m\}$-neutral.*

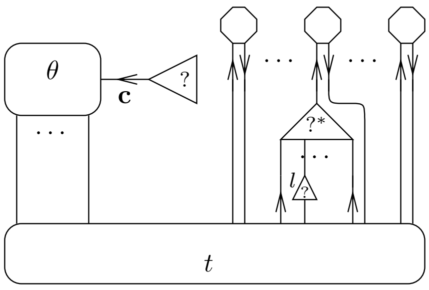


Figure 21: Proof of Lemma 8

The label m cannot occur in P, but it can occur in the remainder of u; the meaning of the lemma is that, during the reduction, "l cannot exit from P" or, more precisely, if it exits, it is by the control port $\mathbf{c}$. This lemma will be essential in the proofs of Propositions 10 and 11 and seems to be a crucial property of our translation.

Proof. By induction on P (and, in some cases, by contradiction: in these cases, we assume that $u \rightsquigarrow^*_{\{l,m\}} u_1 + U$ and that u_1 contains an (l,m)-communication redex).

Assume first that $P = \mathsf{nil}$. Assume that l is a dereliction. Then u has the shape shown in Figure 21. Thus $u \rightsquigarrow^*_{\{l,m\}} 0$ by 3.3.5. Hence by the Church-Rosser property of $\rightsquigarrow^*_{\{l,m\}}$, we must have $u_1 + U \rightsquigarrow^*_{\{l,m\}} 0$. But this is impossible by Lemma 3 since u_1 has an (l,m)-communication redex. The case where l is a codereliction is similar.

The case $P = P_1 \mid P_2$ is handled similarly: using 3.3.5 and the inductive hypothesis, one shows that $u \rightsquigarrow^*_{\{l,m\}} V$ where V is a sum of $\{l,m\}$-neutral simple nets, and hence u is $\{l,m\}$-neutral by Lemma 4.

If $P = \nu a \cdot Q$, one applies directly the inductive hypothesis.

To conclude, we consider the case where $P = \overline{[r]b_{f(0)}}\langle b_{f(1)} \ldots b_{f(p)}\rangle \cdot Q$. Assume first that l is a dereliction. Then u is of the shape shown in Figure 22 (without loss of generality, we assume that the dereliction is connected to a port corresponding to the name b_n), where s is a simple net satisfying $Q \; \mathcal{I}_{b_1,\ldots,b_n} \; s$. Then, aggregating first the communication area γ_n with the communication area of the f-identification structure to which it is connected, we see that we have $u \rightsquigarrow^*_{\{l,m\}} \sum_{i=1}^N u_i$ where u_i is a simple net which has the shape shown in Figure 23 and where, according to 3.3.5, in v_i, the principal port of l is forwarded (see the definition of this concept in 3.3.3 and remember that this is indicated pictorially by a small triangle)

1. to the port b_n^+ of s
2. or to the principal port of the coweakening cell γ, in the case where $f(0) = n$
3. or to one of the input auxiliary port of the compound cell φ, corresponding to an index $j \in \{1,\ldots,q\}$ such that $f(j) = n$.

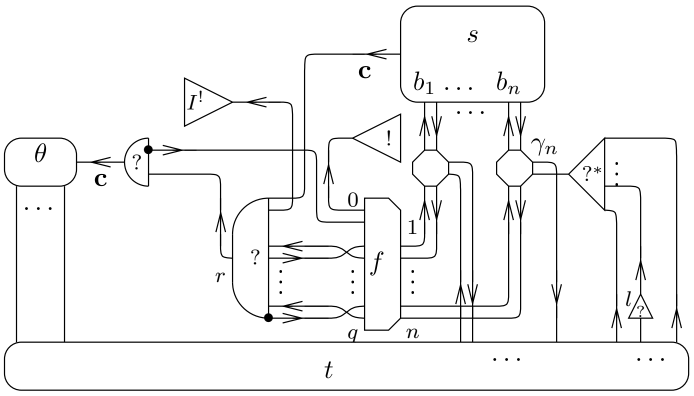


Figure 22: Proof of Lemma 8

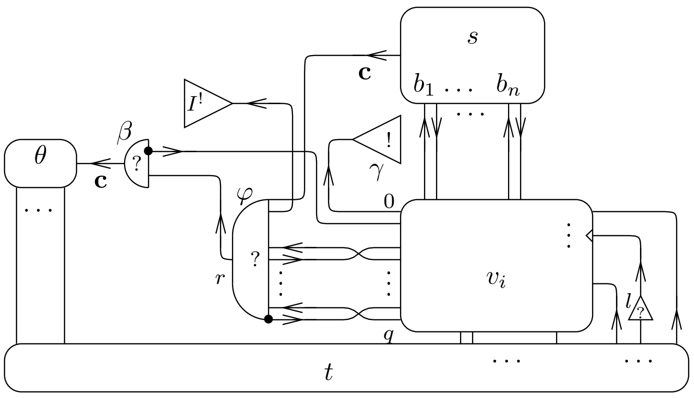


Figure 23: Proof of Lemma 8

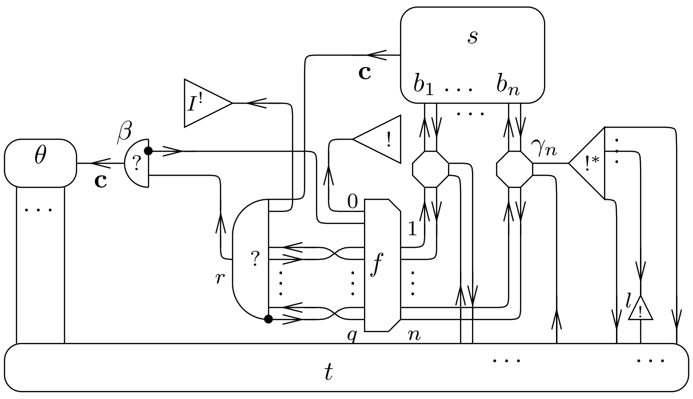


Figure 24: Proof of Lemma 8

For i satisfying (2), we have $u_i \leadsto^*_{\{l,m\}} 0$. For i satisfying (3), l is guarded by $r \neq \tau$ (the labeled dereliction cell of φ) in u_i, and so u_i is $\{l,m\}$-neutral by Lemma 7. For i satisfying (1), the inductive hypothesis applies, showing that u_i is $\{l,m\}$-neutral. Therefore u is $\{l,m\}$-neutral by Lemma 4.

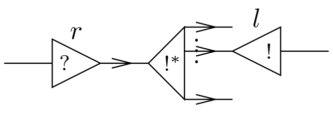
Assume now that l is a codereliction, so that u has the shape shown in Figure 24 (with the same notations as above).

As before, we have $u \leadsto^*_{\{l,m\}} \sum_{i=1}^N u_i$ where the u_is have the same shape as before. Using the same notations, in v_i, the principal port of l is forwarded

1. to the port b_n^- of s
2. or to the dotted auxiliary port of the transistor output compound cell β, in the case where $f(0) = n$
3. or to one of the input auxiliary ports of the compound cell φ, corresponding to an index $j \in \{1,\dots,q\}$ such that $f(j) = n$.

The cases (1) and (3) are handled as before. So consider an index i corresponding to case (2). There are two possibilities, depending on the value of the net θ.

If θ is a boxed identity cell, then $u_i \leadsto^*_{\{l,m\}} u'$ where u' is a simple net which contains the subnet shown aside.



Since we have $r \notin \{l,m\}$ (remember that we have assumed that m does not occur in P), this subnet has no $\leadsto^*_{\{l,m\}}$-redex, and therefore, it will still be present in any simple summand of a net U such that $u' \leadsto^*_{\{l,m\}} U$. So u' is $\{l,m\}$-neutral, and so is u by Lemma 4.

Assume last that θ consists of an r'-labeled output or input prefix compound cell (with $r' \neq \tau$) together with a generalized contraction cell (second possibility for θ in 5.1.3, see Figure 20). Here we can have $r' = m$, but l is guarded by r' in u, and hence u is $\{l,m\}$-neutral by Lemma 7 and Lemma 4.

The case where P starts with an input prefix is completely similar to that of an output prefix, and of course simpler. □

Lemma 9 *Let (Γ, L) be a state and let $b_1, \dots, b_n$ be a repetition-free enumeration of the free names of (Γ, L). Let (Δ, M) be the canonical form of (Γ, L) and let s be a simple net such that $(\Gamma, L)\, \mathcal{I}_{b_1,\dots,b_n}\, s$. Then there exists a simple net t such that $(\Delta, M)\, \mathcal{I}_{b_1,\dots,b_n}\, t$ and $s \sim_{\mathrm{s}} t$.*

The equivalence relation $\sim_{\mathrm{s}}$ is defined in 2.1.4. The proof is by simple inspection of the definition of the interpretation relation, using 3.3.1.

We establish now two results which are the main ingredients towards our bisimulation theorem.

Proposition 10 *Let (Γ, L) and (Δ, M) be canonical states and let $l, m \in \mathcal{L} \setminus \{\tau\}$. Assume that $(\Gamma, L) \xrightarrow{l\overline{m}} (\Delta, M)$. Let s be a simple net and assume that $(\Gamma, L)\, \mathcal{I}_{b_1,\dots,b_n}\, s$ where $b_1, \dots, b_n$ is a repetition-free list of names containing all the free names of (Γ, L). Then there are simple nets t_0 and t such that $(\Delta, M)\, \mathcal{I}_{b_1,\dots,b_n}\, t$, $s \xrightarrow{l\overline{m}} t_0$ and $t_0 \sim_{\mathrm{d}} t$.*

Proof. We know that Γ must be of the shape

$$([l]a(c_1 \dots c_p) \cdot P, e_1)(\overline{[m]d_{f(0)}}\langle d_{f(1)} \dots d_{f(p)}\rangle \cdot Q, e_2)(P_3, e_3) \cdots (P_N, e_N) \quad (1)$$

where we assume that the e_is have pairwise disjoint domains, that a, c_{p+1},..., c_{p+q} is a repetition-free enumeration of the domain of e_1 (these names are assumed to be distinct from the names $c_1, \dots, c_p$, which are bound in the first process of the soup (1)), that $d_1, \dots, d_r$ is a repetition-free enumeration of the domain of e_2, that $h_1, \dots, h_m$ is a repetition-free enumeration of the union of the domains of e_3,...,e_N, and $f : \{0, \dots, p\} \to \{1, \dots, r\}$ is a function, and we have $e_1(a) = e_2(d_{f(0)})$. And $(\Delta, M) = \mathsf{Can}(\Gamma', L)$ where

$$\Gamma' = (P, e_1[c_1 \mapsto e_2(d_{f(1)}), \dots, c_p \mapsto e_2(d_{f(p)})])(Q, e_2)(P_3, e_3) \cdots (P_N, e_N)\,.$$

Without loss of generality, we can assume that $f(0) = 1$. With these notations, the simple net s is of the shape shown in Figure 25, where s_1 is a simple net such that $P\, \mathcal{I}_{a,c_1,\dots,c_{p+q}}\, s_1$, s_2 is a simple net such that $Q\, \mathcal{I}_{d_1,\dots,d_r}\, s_2$ and s' stands for the juxtaposition of simple nets s_is such that $P_i\, \mathcal{I}_{\vec{h^i}}\, s_i$ (for $3 \le i \le N$) where $\vec{h^i}$ stands for an enumeration of the domain of e_i (so that the lists of names $\vec{h^i}$ are pairwise disjoint, and their concatenation is a repetition-free enumeration of the names $h_1, \dots, h_m$), with a boxed identity connected to the control ports of each s_i. In this net, e is the function $\{1, \dots, r + q + m + 1\} \to \{1, \dots, n\}$ which corresponds to the union of the functions e_i for $i = 1, \dots, N$. Observe that we have $e(1) = e(r + 1)$ since by hypothesis $e_1(a) = e_2(d_1)$.

We have omitted in Figure 25 the pairs of free ports corresponding to $b_1, \dots, b_n$, $b_{n+1}, \dots, b_{n+n'}$, the names b_i for $i > n$ corresponding to the elements of L; remember that they are there and that each pair of frees port corresponding to a b_i with $i > n$ is connected to a communication area of order -1.

Then we can reduce the net of Figure 25 along the following steps.

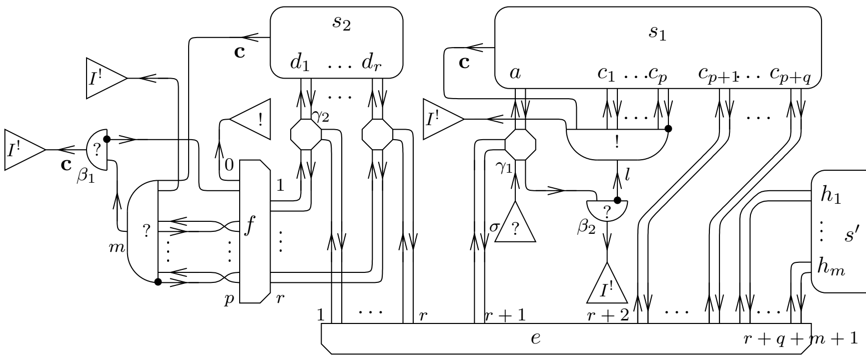


Figure 25: Translation of the state of Formula (1)

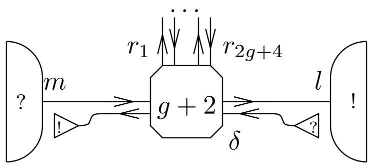
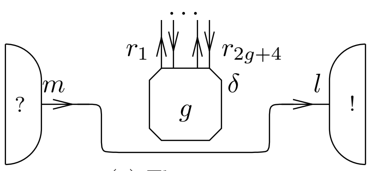
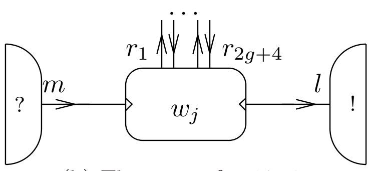


Figure 26: Proof of Proposition 10

- Observe first that the pairs of ports 1 and $r+1$ (attached to the domain of e) are connected to a common communication area δ_1 in the identification structure labeled by e (see 3.2.2) since $e(1) = e(r+1)$, and also that the codomain pair of ports 1 and the domain pair of ports 0 of the identification structure labeled by f are connected to a common communication area δ_2 in this identification structure, since $f(0) = 1$. We apply reduction 3.3.1 to aggregate the communication areas γ_1, δ_1, γ_2 and δ_2 in a unique communication area δ. Let u be the resulting simple net, we have $s \leadsto^*_{\{l,m\}} u$.
- Apply reduction 3.3.7 to both transistors β_1 and β_2 and let u' be the resulting simple net, we have $u \leadsto^*_{\{l,m\}} u'$.
- u' contains therefore the subnet v shown in Figure 26 where, for $i = -1, 0, \ldots, g$ the pair of ports (r_{2i+3}, r_{2i+4}) is connected either
  1. to the pair of ports a of s_1
  2. or to one of the pairs of ports $c_{p+1}, \ldots, c_{p+q}$ of s_1
  3. or to one of the pairs of ports $h_1, \ldots, h_m$ of s'
  4. or to a pair of ports of one of the communication areas connected to $d_2, \ldots, d_r$
  5. or to the pair of ports d_1
  6. or to one of the auxiliary pairs of ports of the output prefix compound cell labeled by m



(a) The net v_0



(b) The net v_j for $j \geq 1$

Figure 27: Proof of Proposition 10

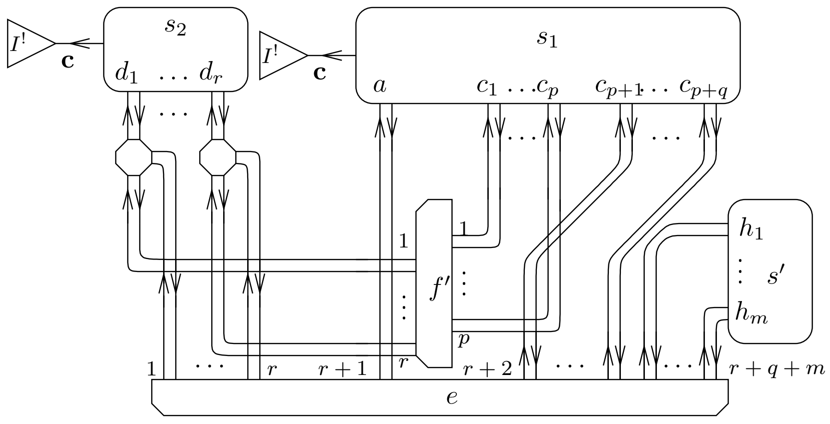


Figure 28: Proof of Proposition 10

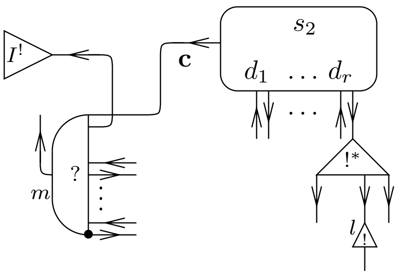
7. or to one of the pairs of ports b_h corresponding to codomain pairs of ports of the identification structure e; these pairs of ports are either free in s (and hence in u') or connected to a communication area of order -1.

To v, we can apply reduction 3.3.4. This subnet reduces by the $\rightsquigarrow^*_{\{l,m\}}$ reduction to a sum $v_0 + v_1 + \cdots + v_k$ where v_0 is shown in Figure 27(a) and the v_js ($j \geq 1$) are nets of the shape shown in Figure 27(b) where the principal port of l and m are forwarded to ports among $r_1, \dots, r_{2g+4}$. We have $u' \rightsquigarrow^*_{\{l,m\}} u'_0 + u'_1 + \cdots + u'_k$ where u'_j results from the replacement of the net v by the net v_j in u' ($j = 0, \dots, k$).

- We apply the (l,m)-communication reduction to u'_0, getting a simple net t_0 which is $\sim_{\mathrm{d}}$ equivalent to the simple net of Figure 28 where f' is the restriction of f to $\{1, \dots, p\}$. This net is $\sim_{\mathrm{s}}$ equivalent to a simple net t_1 with $(\Gamma', L)\ \mathcal{I}_{b_1,\dots,b_n}\ t_1$ (upon applying 3.3.1 to the communication areas of the identification structure f', the ones which are connected to the pairs of free ports d_i of s_2 and those belonging to the identification structure e). By Lemma 9, there is a simple net t such that $t_1 \sim_{\mathrm{s}} t$ and $(\Delta, M)\ \mathcal{I}_{b_1,\dots,b_n}\ t$.

To conclude, we must check that, for $j \geq 1$, u'_j is $\{l,m\}$-neutral. But, for each of the two labels l and m, we are in one of the seven cases (1) to (7) above. Consider for instance label l. If we are in case (1), (2), (3), (5), we can directly apply Lemma 8.

Assume that we are in case (4) and that, in u'_j, the codereliction labeled by l is forwarded to the communication area connected to d_r (so that $r \geq 2$), we can apply 3.3.5 and see that $u'_j \rightsquigarrow^*_{\{l,m\}} w_1 + w_2$ where w_1 and w_2 are simple, and w_1 contains a subnet of the shape shown aside. Hence by Lemma 8, w_1 is $\{l, m\}$-neutral.



On the other hand, in w_2, l is connected to the r-th communication area (in the sense of 3.2.2) of the identification structure labeled by f and the other pairs of ports of that communication area are connected to auxiliary ports of the output prefix compound cell labeled by m. Therefore, by Lemmata 7 and 4, w_2 is $\{l, m\}$-neutral. So, by Lemma 4, u'_j is $\{l, m\}$-neutral.

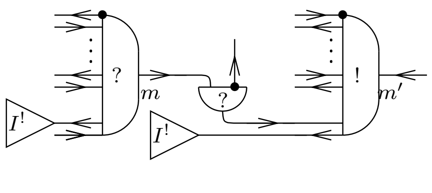
If we are in case (6) then, in u'_j, l is guarded by m and hence u'_j is $\{l, m\}$-neutral by Lemma 7. Last assume we are in case (7); in this case, l is connected to an auxiliary port of a generalized structural cell whose principal port is free, or is connected to a weakening cell. In both cases again it is clear that u'_j is $\{l, m\}$-neutral $\square$

We prove now a converse statement. We explain in 7.4.3 that this statement, and hence also Theorem 12, can be strengthened.

Proposition 11 *Let* (Γ, L) *be a canonical state and* $b_1, \ldots, b_n$ *be a repetition-free list of names containing all the free names of* (Γ, L)*. Let* s *be a simple net such that* $(\Gamma, L)\, \mathcal{I}_{b_1,\ldots,b_n}\, s$*. If* t'_0 *is a simple net such that* $s \xrightarrow{l\overline{m}} t'_0$*, then there is a canonical state* (Δ, M) *such that* $(\Gamma, L) \xrightarrow{l\overline{m}} (\Delta, M)$ *and there exists a simple net* t *such that* $(\Delta, M)\, \mathcal{I}_{b_1,\ldots,b_n}\, t$ *and* $t \sim_{\mathrm{d}} t'_0$*.*

Proof. We show first that both l and m must be minimal in the poset $\mathcal{L}(\Gamma, L)$ (see Section 4.2). Assume for instance that m is not minimal. Then the principal port of the dereliction cell labeled by m is connected to an auxiliary port of a transistor whose principal port is connected to an auxiliary port of an input or output prefix cell, labeled say by m', with $m' < m$ (actually, m' is the predecessor of m in the forest $\mathcal{L}(\Gamma, L)$). Say for instance that the prefix cell labeled by m' is an input prefix cell.

Hence s contains the subnet shown aside. So m is guarded by m' in s and so, whenever $s \rightsquigarrow^*_{\{l,m\}} s'$, no simple net appearing in s' can contain an (l, m)-communication redex, in contradiction with our hypothesis that $s \xrightarrow{l\overline{m}} t'_0$.



We have seen that l and m are minimal in the poset $\mathcal{L}(\Gamma, L)$ and this means that in Γ, the prefixes labeled by l and m are the outermost prefixes of P_1 and P_2 where $\Gamma = (P_1, e_1) \cdots (P_N, e_N)$ (and the choice of P_1 and P_2 is uniquely determined by l and m), that is, Γ is of the form described by Formula (1) in the proof of Proposition 10, P_1 denoting the first process in that expression, which is guarded by an l-labeled input prefix, and P_2 the second one, which is guarded by an m-labeled output prefix. Using the notations of Formula (1), we argue now that necessarily $e_1(a) = e_2(d_{f(0)})$ (we can refer to Figure 25 as describing s). But if this is not the case, an inspection of the interpretation of input prefixes 4.3.4, of states 4.3.6 and of the identification structure associated with the "global environment" e (see 3.2.2) shows that $s \rightsquigarrow^*_{\{l,m\}} S' = s'_1 + \cdots + s'_q$ where for each i, s'_i is simple and one of the following holds:

1. in s'_i, l is forwarded to a free port of S'
2. or l dives into P_j in s'_i for some $j = 1, \ldots, N$. We denote by t the subnet of s'_i such that $P_j \; \mathcal{I}_{c_1,\ldots,c_r} \; t$, where $c_1, \ldots, c_r$ is a repetition-free enumeration of the domain of e_j.

In case (1), s'_i is $\{l, m\}$-neutral. The same is true of s'_i in case (2) when the index j is different from 2 since then P_j cannot contain the label m and we can apply Lemma 8. In the case $j = 2$, using our assumption that $e_1(a) \neq e_2(d_{f(0)})$, we see that l dives into P_2 through a free port which does not correspond to $d_{f(0)}$ and from this (and from an inspection of the interpretation of output prefixes 4.3.5), we see that $s_i \rightsquigarrow^*_{\{l,m\}} S''$ where S'' is a sum of simple nets in which, either l is guarded by m, or l dives into Q in t where Q is the process guarded by the m-labeled output prefix of P_2 (and therefore, Q does not contain the label m). Applying Lemma 7 in the first case and Lemma 8 in the second case, we see that each simple summand of S'' is $\{l, m\}$-neutral and therefore s_i also is $\{l, m\}$-neutral by Lemma 4. Finally, by the same lemma, s itself is $\{l, m\}$-neutral, contradicting the hypothesis that $s \xrightarrow{l\overline{m}} t'_0$.

So we must have $e_1(a) = e_2(d_{f(0)})$ and since our processes and states are implicitly arity-typed (see 4.1), we know that the number of objects of the two involved prefixes coincide (the common value of these numbers is p, according to our notations).

Using the same notations as in Proposition 10, and the statement itself of that proposition, we have $(\Gamma, L) \xrightarrow{l\overline{m}} (\Delta, M)$ and there are simple nets t and t_0 such that $(\Delta, M) \; \mathcal{I}_{b_1,\ldots,b_n} \; t$, $t \sim_{\mathrm{d}} t_0$ and $s \xrightarrow{l\overline{m}} t_0$. This means more precisely that $s \rightsquigarrow^*_{\{l,m\}} S' = s_0 + s_1 + \cdots + s_p$, with the s_js simple, such that s_0 has an (l, m)-communication redex and each s_j (for $j \geq 1$) is $\{l, m\}$-neutral and t_0 is the simple net which results from the reduction of the (l, m)-communication redex in s_0.

We conclude by showing that $t_0 \sim_{\mathrm{d}} t'_0$.

We know from our hypothesis that $s \rightsquigarrow^*_{\{l,m\}} S'' = s'_0 + s'_1 + \cdots + s'_q$, where s'_0 has an (l, m)-communication redex and each s'_j (for $j \geq 1$) is $\{l, m\}$-neutral, and t'_0 is the simple net which results from the reduction of the (l, m)-communication redex in s'_0.

By the Church Rosser property of $\rightsquigarrow^*_{\{l,m\}}$, there is a net U such that $S' \rightsquigarrow^*_{\{l,m\}} U$ and $S'' \rightsquigarrow^*_{\{l,m\}} U$. By Lemma 3, we have $U = u_0 + U'$ with $s_0 \rightsquigarrow^*_{\mathrm{d}} u_0$ and $s'_0 \rightsquigarrow^*_{\mathrm{d}} u_0$, thanks also to the $\{l,m\}$-neutrality of s_j and s'_j for $j \geq 1$. Moreover (still by Lemma 3), u_0 contains an (l,m)-communication redex as well, and if v_0 is the net which results from the reduction of the (l,m)-communication redex in u_0, we have also $t_0 \rightsquigarrow^*_{\mathrm{d}} v_0$ and $t'_0 \rightsquigarrow^*_{\mathrm{d}} v_0$. So we have $t_0 \sim_{\mathrm{d}} t'_0$. □

We are now ready to state a bisimulation theorem. Given a repetition-free list $b_1, \ldots, b_n$ of names, we define a relation $\widetilde{\mathcal{I}}_{b_1,\ldots,b_n}$ between states and simple nets by: $(\Gamma, L)\, \widetilde{\mathcal{I}}_{b_1,\ldots,b_n}\, s$ if there exists a simple net s_0 such that $(\Gamma, L)\, \mathcal{I}_{b_1,\ldots,b_n}\, s_0$ and $s_0 \sim_{\mathrm{d}} s$.

Theorem 12 *The relation $\widetilde{\mathcal{I}}_{b_1,\ldots,b_n}$ is a strong bisimulation between the labeled transition systems $\mathbb{S}_{\mathcal{L}}$ and $\mathbb{D}_{\mathcal{L}}$.*

Proof. Let (Γ, L) be a canonical state and s_1 be a simple net, and assume that $(\Gamma, L)\, \widetilde{\mathcal{I}}_{b_1,\ldots,b_n}\, s_1$. So there is a simple net s such that $(\Gamma, L)\, \mathcal{I}_{b_1,\ldots,b_n}\, s$ and $s \sim_{\mathrm{d}} s_1$.

Assume first that $(\Gamma, L) \xrightarrow{l\overline{m}} (\Delta, M)$, with l, m two distinct elements of $\mathcal{L} \setminus \{\tau\}$. By Proposition 10, there are simple nets t_0 and t such that $(\Delta, M)\, \mathcal{I}_{b_1,\ldots,b_n}\, t_0 \sim_{\mathrm{d}} t$ and $s \xrightarrow{l\overline{m}} t$. By Lemma 6 ($\sim_{\mathrm{d}}$ is a bisimulation), there exists t_1 such that $t \sim_{\mathrm{d}} t_1$ and $s_1 \xrightarrow{l\overline{m}} t_1$. We have $(\Delta, M)\, \widetilde{\mathcal{I}}_{b_1,\ldots,b_n}\, t_1$.

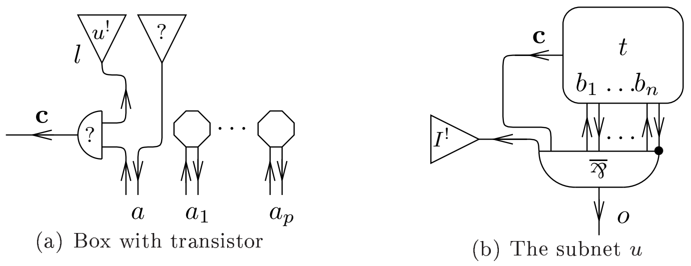
Conversely, assume that $s_1 \xrightarrow{l\overline{m}} t_1$. By Lemma 6, there exists t such that $t \sim_{\mathrm{d}} t_1$ and $s \xrightarrow{l\overline{m}} t$. By Proposition 11, there is a canonical state (Δ, M) and a simple net t_0 such that $(\Gamma, L) \xrightarrow{l\overline{m}} (\Delta, M)$ and $(\Delta, M)\, \mathcal{I}_{b_1,\ldots,b_n}\, t_0 \sim_{\mathrm{d}} t$. We have $(\Delta, M)\, \widetilde{\mathcal{I}}_{b_1,\ldots,b_n}\, t_1$. □

6. Dealing with replication

We extend our π-calculus with the following construction[9]: if $l \in \mathcal{L}$, if a and $b_1, \ldots, b_n$ are pairwise distinct names and if P is a process such that $\mathsf{FV}(P) \subseteq \{b_1, \ldots, b_n\}$, then $[l]!a(b_1, \ldots, b_n) \cdot P$ is a process, whose only free name is a. This process is guarded, in the sense of 4.2.2. This extension has no influence on the definition of the relation $\rightsquigarrow_{\mathsf{can}}$ on the states of our environment machine. The transition of the machine has to be extended with the following rule:

$$\begin{aligned}
&(([l]!a(b_1 \ldots b_n) \cdot P, e)(\overline{[m]a'}\langle b'_1 \ldots b'_n\rangle \cdot P', e')\Gamma, L) \\
&\quad \xrightarrow{l\overline{m}} \mathsf{Can}(([l]!a(b_1 \ldots b_n) \cdot P, e) \\
&\qquad\qquad (P, e[b_1 \mapsto e'(b'_1), \ldots, b_n \mapsto e'(b'_n)])(P', e')\Gamma, L)
\end{aligned}$$

[9]This is a restricted form of replication: all the free names of the replicable process have to be bound by the prefix.



(a) Box with transistor (b) The subnet u

Figure 29: Translation of input replication

We extend now the translation relation $\mathcal{I}$ to the replicated input process. Let P be a process whose free names are contained in the repetition-free list $b_1, \ldots, b_n$, and let $a, a_1, \ldots, a_p$ be a list of pairwise distinct names. We set $[l]!a(b_1 \ldots b_n) \cdot P \, \mathcal{I}_{a,a_1,\ldots,a_p} \, s$ if, for some simple net t such that $P \, \mathcal{I}_{b_1,\ldots,b_n} \, t$, s is of the shape given by Figure 29(a). The promotion cell of that net contains the net shown in Figure 29(b).

When $P \, \mathcal{I}_{b_1,\ldots,b_n} \, t$ for a process with replication P, the simple net t does not satisfy the CLB (see 1.3.5) in general since promotion cells will have labels $\neq \tau$, so that a bisimulation theorem will be harder to obtain (the transition system of simple nets is defined only for nets satisfying the CLB in Section 2.3). One should label in a different way the various copies of promotion cells, in the spirit of the geometry of interaction [Gir88a], with a similar discipline for processes as well.

7. Examples and conclusion

We give a few examples to illustrate some key features of communication in the π-calculus as represented in differential interaction nets.

7.1. Concurrent communication

Let P be the process (the restriction is here only to illustrate its interpretation in nets by a communication area of order -1):

$$\nu a \cdot \Big(\Big([l]a() \cdot \mathsf{nil} \mid \overline{[m]a}\langle\rangle \cdot \mathsf{nil} \Big) \mid \overline{[r]a}\langle\rangle \cdot \mathsf{nil} \Big) .$$

The simplest state containing P is $(\Gamma, L) = ((P, \emptyset), \emptyset)$. We have $(\Gamma, L) \, \mathcal{I} \, s$ where s is the simple net of Figure 30.

By applying aggregations of communication areas, we obtain the simple net s_1 of Figure 31. Thus $s \rightsquigarrow_{\mathrm{s}}^{*} s_1$. Since P is in fact a CCS process (namely $\nu a \cdot (a \mid \overline{a} \mid \overline{a})$), we can remark how the translation into differential interaction nets is given by first a tree (with nodes represented with dashed boxes) corresponding to the tree structure of the CCS process (built from sequential and parallel compositions), and second communication areas for the identification of names.

The simple net s_1 reduces to the net s_2 ($s_1 \rightsquigarrow_{\mathrm{d}}^{*} s_2$) of Figure 32, where

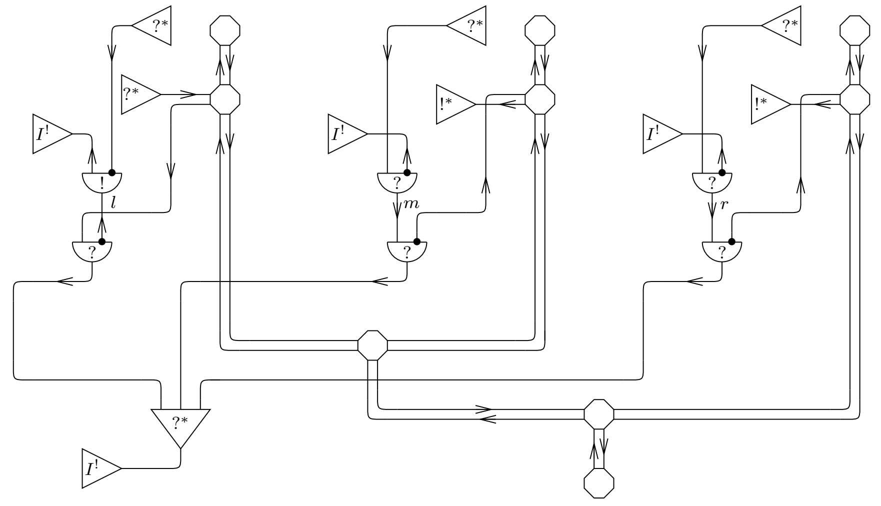


Figure 30: Concurrent communication in a CCS process

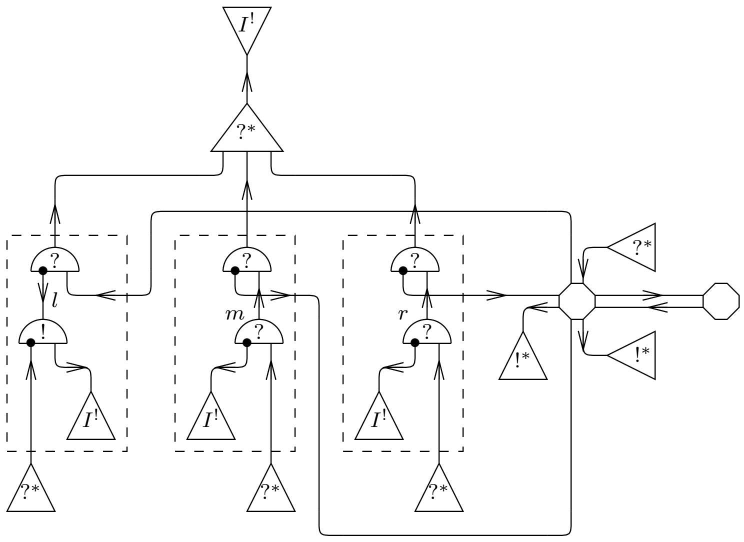


Figure 31: A CCS process: first step

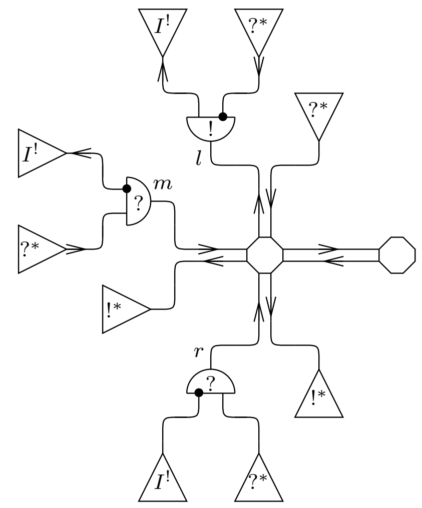


Figure 32: A CSS process: second step

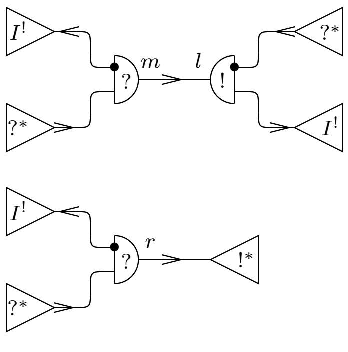


Figure 33: A CSS process: final state

the choice between actions ready to communicate will be done. This means that s_2 reduces to a sum of simple nets containing in particular the net s_3 ($s_2 \rightsquigarrow^*_{\{l,m\}} s_3 + \cdots$) of Figure 33. If t results from the reduction of the (l,m)-communication redex in s_3, we have $s \xrightarrow{l\overline{m}} t$. This corresponds to $(\Gamma, L) \rightsquigarrow_{\mathsf{can}}$ $(([l]a() \cdot \mathsf{nil}, e)(\overline{[m]a}\langle\rangle \cdot \mathsf{nil}, e)(\overline{[r]a}\langle\rangle \cdot \mathsf{nil}, e), \{a'\}) \xrightarrow{l\overline{m}} ((\overline{[r]a}\langle\rangle \cdot \mathsf{nil}, e), \{a'\})$ (with e defined only on $\{a\}$ by $e(a) = a'$) in the environment machine.

7.2. Sequentiality

Let P be the process:

$$[l]a() \cdot [l']b() \cdot \mathsf{nil} \mid \overline{[m']b}\langle\rangle \cdot \mathsf{nil} \mid \overline{[m]a}\langle\rangle \cdot \mathsf{nil}$$

The simplest state containing P is $(\Gamma, L) = ((P, e), \emptyset)$ (with e defined on $\{a, b\}$ by $e(a) = a'$ and $e(b) = b'$). We have $(\Gamma, L)\ \mathcal{I}_{a',b'}\ s$ with $s \rightsquigarrow^*_{\mathsf{s}} s_1$ (aggregations

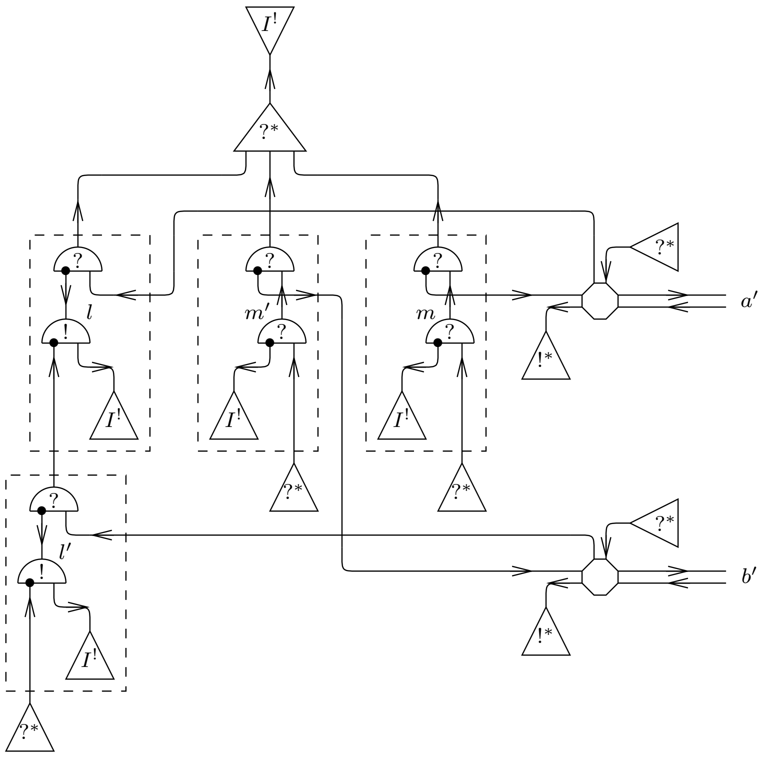


Figure 34: Sequentiality: s_1, translation of the process
$P = [l]a() \cdot [l']b() \cdot \mathsf{nil} \mid \overline{[m']b}\langle\rangle \cdot \mathsf{nil} \mid \overline{[m]a}\langle\rangle \cdot \mathsf{nil}$

of communication areas) and s_1 is the simple net of Figure 34; observe that there is a guarding path from l' to l which enforces sequentiality by preventing l' to interact with m. Since P is again a CCS process (namely $a \cdot b \mid \overline{b} \mid \overline{a}$), we can see its tree structure in the differential interaction net s_1 of Figure 34.

The simple net s_1 reduces to the simple net s_2 of Figure 35 ($s_1 \rightsquigarrow^*_{\mathrm{d}} s_2$), where the above mentioned guarding path is preserved.

Then there exists a simple net s_3 such that $s_2 \rightsquigarrow^*_{\{l,m\}} s_3 + \cdots$ and if t results from the reduction of the (l, m)-communication redex in s_3, we have $s \xrightarrow{l\overline{m}} t$. Moreover t reduces to the net of Figure 36. This corresponds to $\underline{(\Gamma, L) \rightsquigarrow_{\mathsf{can}}}$ $((([l]a() \cdot [l']b() \cdot \mathsf{nil}, e)(\overline{[m']b}\langle\rangle \cdot \mathsf{nil}, e)(\overline{[m]a}\langle\rangle \cdot \mathsf{nil}, e), \emptyset) \xrightarrow{l\overline{m}} ((([l']b() \cdot \mathsf{nil}, e)(\overline{[m']b}\langle\rangle \cdot \mathsf{nil}, e), \emptyset)$ in the environment machine.

7.3. Name passing

Let P, Q and R be processes such that the free names of P are a and z, the only free name of Q is y and the free names of R are x and b. Let P' be the process:

$$\nu z \cdot \left(\overline{[l]a}\langle z\rangle \cdot P \mid [l']z(y) \cdot Q\right) \mid [m]a(x) \cdot \overline{[m']x}\langle b\rangle \cdot R$$

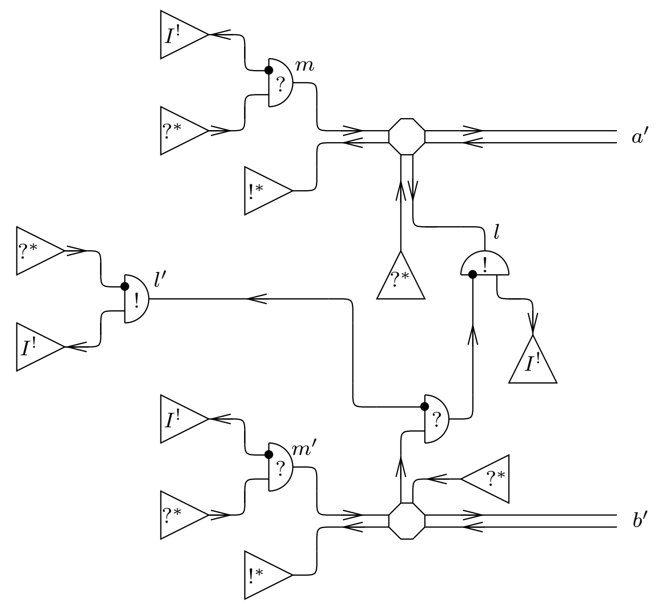


Figure 35: Sequentiality: simple net s_2, first step

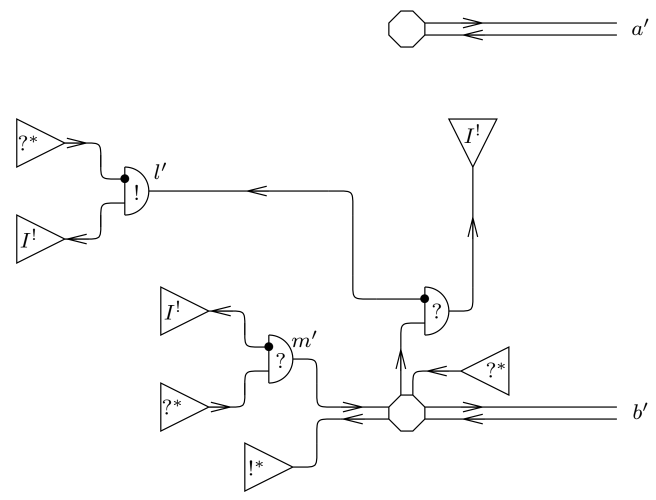


Figure 36: Sequentiality: simple net s_3, second step

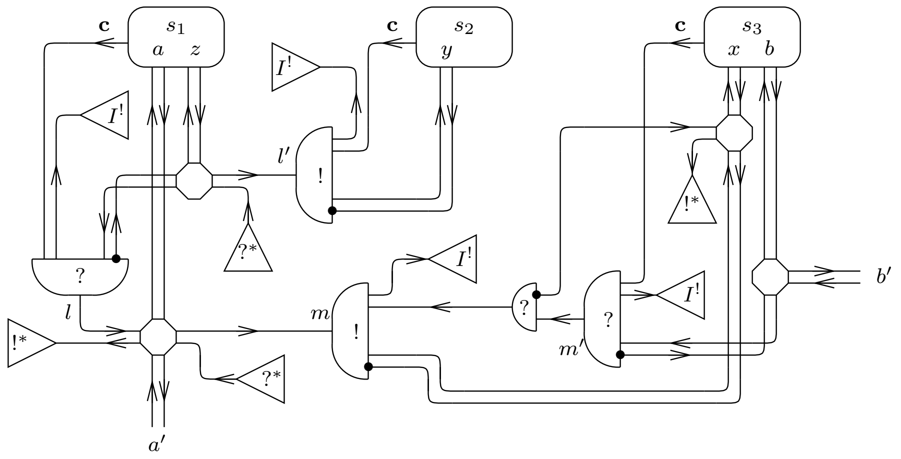


Figure 37: Name passing: simple net s_1', translation of the process P' (after a few structural reductions)

The simplest state containing P' is $(\Gamma, L) = ((P', e), \emptyset)$ (with e defined on $\{a, b\}$ by $e(a) = a'$ and $e(b) = b'$). If $P\ \mathcal{I}_{a,z}\ s_1$, $Q\ \mathcal{I}_y\ s_2$ and $R\ \mathcal{I}_{x,b}\ s_3$, we have $(\Gamma, L)\ \mathcal{I}_{a',b'}\ s'$ with $s' \rightsquigarrow_{\mathrm{s}}^* s_1'$ (aggregations of communication areas) and s_1' is the simple net of Figure 37.

We have $s' \xrightarrow{m\bar{l}} t$ with $t \rightsquigarrow_{\mathrm{d}}^* s_2'$ and s_2' is the simple net of Figure 38, where the identification of the names z and x corresponds to the connection of the associated communication areas.

Finally $t \xrightarrow{l'\overline{m'}} t'$ with $t' \rightsquigarrow_{\mathrm{d}}^* s_3'$ and s_3' is the simple net of Figure 39 where y and b are also identified.

This corresponds to

$$\begin{aligned}
(\Gamma, L) \quad \rightsquigarrow_{\mathsf{can}} \quad & ((\overline{[l]}a\langle z\rangle \cdot P, e[z \mapsto z'])([l']z(y) \cdot Q, e[z \mapsto z']) \\
& \qquad ([m]a(x) \cdot \overline{[m']}x\langle b\rangle \cdot R, e), \{z'\}) \\
\xrightarrow{m\bar{l}} \quad & ((P, e[z \mapsto z'])([l']z(y) \cdot Q, e[z \mapsto z']) \\
& \qquad (\overline{[m']}x\langle b\rangle \cdot R, e[x \mapsto z']), \{z'\}) \\
\xrightarrow{l'\overline{m'}} \quad & ((P, e[z \mapsto z'])(Q, e[z \mapsto z', y \mapsto b'])(R, e[x \mapsto z']), \{z'\})
\end{aligned}$$

in the environment machine.

7.4. Conclusion

The main goal of this work was not to define one more translation of the π-calculus into yet another exotic formalism. We wanted to illustrate by our bisimulation result that differential interaction nets are sufficiently expressive for simulating concurrency and mobility, as formalized in the π-calculus. We believe that differential interaction nets have their own interest and find a strong

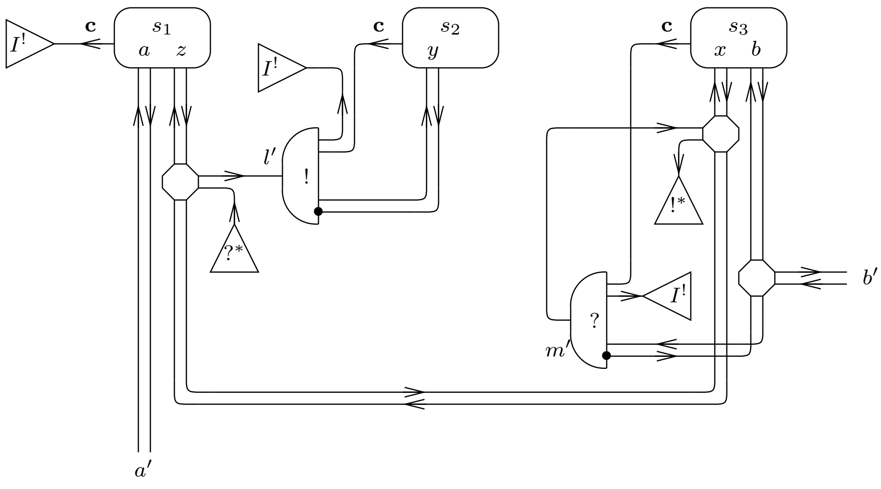


Figure 38: Name passing: simple net s'_2, first step

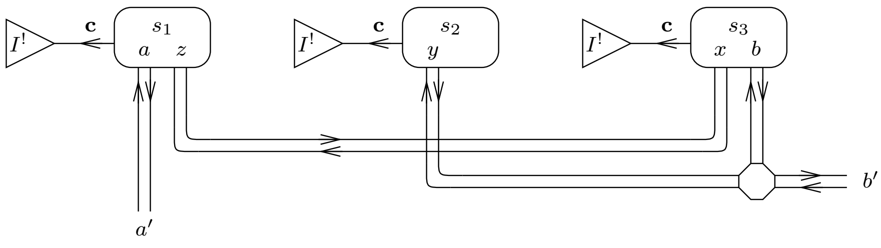


Figure 39: Name passing: simple net s'_3, final state

mathematical and logical justification in their connection with linear logic, in the existence of various denotational models and in the analogy between its basic constructs and fundamental mathematical operations such as differentiation and convolution product. The fact that differential interaction nets support concurrency and mobility suggests that they might provide more convenient mathematical and logical foundations to concurrent computing. In particular, this work suggests that differential linear logic might be the logical side of a Curry-Howard correspondence for concurrency and mobility, but there is still a lot of work to do for enforcing this idea. The following issues are crucial.

7.4.1. Logical correctness. The nets which result from our translation do not satisfy in general the Danos-Regnier acyclicity criterion, so they cannot always be sequentialized into proofs of the sequent calculus of Section 1.1. We think that the sequentializable nets are already quite expressive in terms of concurrency and mobility, but this claim has to be enforced by mathematical results. One research direction here would be to try to identify a well-behaved and sufficiently expressive fragment of the π-calculus, or of the solos calculus, whose processes are translated into sequentializable nets.

7.4.2. Typing. And then of course, there is the question of typing, which is orthogonal to the sequentializabilty issue. The nets presented here are "weakly typed": they are typed using a type o which satisfies the recursive equation $o = ?(o^{\perp})\parr o = (o \Rightarrow o)$. This is a typing system which accepts all untyped lambda-terms[10], and hence does not convey any information about terms, but has two effects when used in our setting. First it prevents "clashes" to appear during the reduction of nets (for instance, the principal port of a tensor cell connected to the principal port of another tensor cell). Second, it allows to interpret our nets in some denotational models of the untyped differential lambda-calculus, such as the relational model presented in [BEM07]. It is possible to adopt more informative typing disciplines, such as second order propositional linear logic. The question is then again to understand if such typed and logically correct differential nets are still sufficiently expressive in terms of concurrency and mobility and to design typed and logically correct process algebras associated with such nets. The success of this research program would lead to a true extension of the Curry-Howard correspondence to concurrency. Of course, many other issues have to be addressed as well. Let us mention only a few of them.

- How should we handle the sum of process algebras in our setting, and how is it related to the additive connectives of linear logic?
- What kind of replication can we encode in our nets, using more general

[10]Remember that the untyped lambda-calculus can be translated into nets of multiplicative-exponential linear logic, which are typed in this typing system, and satisfy the Danos-Regnier acyclicity criterion, see [Reg92].

instances of the promotion rule of linear logic than the closed promotion of Section 6?

- Since our nets belong to a differential extension of linear logic in which the lambda-calculus can be faithfully represented as well, does our setting suggest new ways of combining concurrent and functional programming?
- Our nets admit denotational models, such as the relational model introduced in [BEM07]. What kind of equivalence on processes do such interpretations induce through our translation?

7.4.3. Final remark. In the final revision process of this paper, we observed that Proposition 11 can be strengthened. Indeed, with the notations of that proposition, if $(\Gamma, L)\ \mathcal{I}_{b_1,\dots,b_n}\ s$ and if $s \rightsquigarrow^*_{\{l,m\}} s_0 + s_1 + \cdots + s_n$ where s_0 is a simple net which contains an (l, m)-communication redex, then s_i is $\{l, m\}$-neutral for each $i \geq 1$. This is actually a simple consequence of Theorem 2 and of Proposition 10, observing first that we must have $(\Gamma, L) \xrightarrow{l\overline{m}} (\Delta, M)$ for some (Δ, M), as we did at the beginning of the proof of Proposition 11. This indicates that Theorem 12 still holds if we remove the $\{l, m\}$-neutrality restriction in the definition of the transition system $\mathbb{D}_{\mathcal{L}}$ (see Remark 5 in 2.3.3).

References

[AC98] Roberto Amadio and Pierre-Louis Curien. *Domains and lambda-calculi*, volume 46 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1998.

[AM99] Samson Abramsky and Paul-André Melliès. Concurrent games and full completeness. In *Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science*. IEEE, 1999.

[BB90] Gérard Berry and Gérard Boudol. The chemical abstract machine. In *Proceedings of the 17h ACM Symposium on Principles of Programming Languages (POPL)*, pages 81–94. ACM Press, January 1990.

[Bef05] Emmanuel Beffara. *Logique, Réalisabilité et Concurrence*. PhD thesis, Université Denis Diderot, 2005.

[BEM07] Antonio Bucciarelli, Thomas Ehrhard, and Giulio Manzonetto. Not enough points is enough. In *Proceedings of the 21st Annual Conference of the European Association for Computer Science Logic (CSL'07)*, Lecture Notes in Computer Science. Springer-Verlag, September 2007.

[BHY04] Martin Berger, Kohei Honda, and Nobuko Yoshida. Strong normalisability in the pi-calculus. *Information and Computation*, 191:145–202, 2004.

[BM06] Emmanuel Beffara and François Maurel. Concurrent nets: a study of prefixing in process calculi. *Theoretical Computer Science*, 356(3):356–373, 2006.

[CF06] Pierre-Louis Curien and Claudia Faggian. An approach to innocent strategies as graphs. Technical Report hal-00155293, CCSD-HAL, 2006. Submitted for publication.

[DR89] Vincent Danos and Laurent Regnier. The structure of multiplicatives. *Archives for Mathematical Logic*, 28(3):181–203, 1989.

[Ehr05] Thomas Ehrhard. Finiteness spaces. *Mathematical Structures in Computer Science*, 15(4):615–646, 2005.

[EL08] Thomas Ehrhard and Olivier Laurent. Acyclic solos and differential interaction nets. Submitted to *Logical Methods in Computer Science*, October 2008.

[ER06] Thomas Ehrhard and Laurent Regnier. Differential interaction nets. *Theoretical Computer Science*, 364(2):166–195, 2006.

[EW97] Uffe Engberg and Glynn Winskel. Completeness Results for Linear Logic on Petri Nets. *Annals of Pure and Applied Logic*, 86(2):101–135, 1997.

[FM05] Claudia Faggian and François Maurel. Ludics nets, a game model of concurrent interaction. In *Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science*, pages 376–385. IEEE Computer Society, 2005.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[Gir88a] Jean-Yves Girard. Geometry of interaction II: Deadlock free algorithm. In Martin-Löf & Mints, editor, *Proceedings of COLOG'88*, volume 417 of *Lecture Notes in Computer Science*, pages 76–93. Springer-Verlag, 1988.

[Gir88b] Jean-Yves Girard. Normal functors, power series and the λ-calculus. *Annals of Pure and Applied Logic*, 37:129–177, 1988.

[HL08] Kohei Honda and Olivier Laurent. An exact correspondence between a typed pi-calculus and polarised proof-nets. Submitted to *Theoretical Computer Science*, November 2008.

[HY94] Kohei Honda and Nobuko Yoshida. Combinatory representation of mobile processes. In *Proceedings of the 21st ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 348–360. ACM, 1994.

[JM03] Ole Jensen and Robin Milner. Bigraphs and transitions. In *Proceedings of the 30th ACM SIGPLAN-SIGACT symposium on Principles of programming languages table*, pages 38–49. ACM, 2003.

[Laf95] Yves Lafont. From proof nets to interaction nets. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*, pages 225–247. Cambridge University Press, 1995. Proceedings of the Workshop on Linear Logic, Ithaca, New York, June 1993.

[LPV01] Cosimo Laneve, Joachim Parrow, and Björn Victor. Solo diagrams. In *Proceedings of the 4th conference on Theoretical Aspects of Computer Science, TACS'01*, number 2215 in Lecture Notes in Computer Science. Springer-Verlag, 2001.

[LV03] Cosimo Laneve and Björn Victor. Solos in concert. *Mathematical Structures in Computer Science*, 13(5):657–683, 2003.

[Maz05] Damiano Mazza. Multiport interaction nets and concurrency. In *Proceedings of CONCUR 2005*, number 3653 in Lecture Notes in Computer Science, pages 21–35. Springer-Verlag, 2005.

[Mel06] Paul-André Melliès. Asynchronous games 2: the true concurrency of innocence. *Theoretical Computer Science*, 358(2):200–228, 2006.

[Mil93] Robin Milner. The polyadic pi-calculus: a tutorial. In *Logic and Algebra of Specification*, pages 203–246. Springer-Verlag, 1993.

[Plo76] Gordon Plotkin. A powerdomain construction. *SIAM Journal of Computing*, 5(3):452–487, 1976.

[Reg92] Laurent Regnier. *Lambda-Calcul et Réseaux.* Thèse de doctorat, Université Paris 7, January 1992.

[SW01] Davide Sangiorgi and David Walker. *The pi-calculus: a Theory of Mobile Processes.* Cambridge University Press, 2001.