Interpreting a Finitary Pi-Calculus in Differential Interaction Nets

Thomas Ehrhard and Olivier Laurent

Preuves, Programmes & Systèmes
Université Denis Diderot and CNRS

Abstract. We propose and study a translation of a pi-calculus without sums nor replication/recursion into an untyped and essentially promotion-free 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.

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 true concurrency, such as modelled 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 determinitic 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-determinisms (e.g. [Plo76,Gir88]), showing that 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 setting, the weakening rule has a mirror image rule called *coweakening*, and similarly for dereliction and for contraction, and the reduction rules have the corresponding mirror symmetry. The corresponding formalism of *differential interaction nets* has been introduced in a joint work by the first author and Regnier [ER06][1].

[1] Note that, in this *differential linear logic*, the two additive connectives $\oplus$ and & are identified, but this does not prevent the system from having good logical properties,

In a joint work with Kohei Honda [HL07], 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 [BHY03]. This translation has two features which can be considered as slight defects: it accepts only replicable receivers and is not really modular (the parallel composition of two processes cannot be described as a combination of the corresponding nets).

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 differential interaction nets[2].

The first key decision we made, guided by the structure of the typical cocontraction/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*.

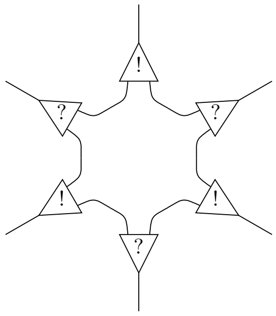


Fig. 1. Communication area

We discovered structures which allow to combine these pairs of wires for interpreting parallel composition and called them *communication areas*: they are obtained by combining in a completely symmetric way cocontraction and contraction cells. There are communication areas of any "arity" (number of pairs of wires connected to it). The communication area of arity 3 can be pictured as in Figure 1, where cocontraction cells are pictured as !-labeled triangles and contraction cells as ?-labeled triangles. The ports corresponding to the same pairs are the principal ports of antipodic cells.

---

and this identification – which results from non-determinism – does not extend to the multiplicative connectives: $\otimes$ and $⅋$ are distinct.

[2] One should mention here that translations of the π-calculus into nets of various kinds, subject to local reduction relations, have been provided by various authors (cf. the work of Laneve, Parrow and Victor on *solo diagrams* [LPV01], of Beffara and Maurel [BM05], of Milner on *bigraphs* [JM04], of Mazza [Maz05] on *multiport interaction nets* etc.). But these settings have no clear logical grounds nor simple denotational semantics.

Content. We first introduce differential interaction nets, typed with a recursive typing system (introduced by Danos and Regnier in [Reg92] and which corresponds to the untyped lambda-calculus) for avoiding the appearance of non reducible configurations. These nets are finitary in the sense that they use only a weak form of promotion. 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 reduction. 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. Last we define a "translation" relation from machine states to nets and show that this translation relation is a bisimulation between the two transition systems.

1 Differential interaction nets

Interaction nets have been introduced by Lafont [Laf95] as a generalization of linear logic proof nets. A *signature* of interaction nets is a set of *symbols*, each of them being given with an arity and a typing rule. A net is made of cells. In a net, each cell γ bears exactly one symbol, and has therefore an arity n; the cell γ must have n *auxiliary ports* (numbered from 1 to n) and one *principal port* (numbered 0). A net can also have *free ports*. Specifying the net consists last in giving its *wiring*, which is a partition of its ports in 2-elements sets (the wires). Typing the net means associating a formula of some linear logical system with each of its oriented wires in such a way that, when reversing the orientation of the wire, the formula be turned to its orthogonal. Of course, the typing rule attached to each cell of the net must also be respected by the typing.

See also [ER06] for an introduction to differential interaction nets.

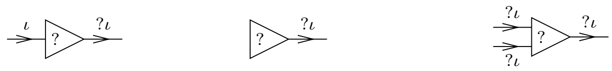
1.1 Presentation of the cells

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 following recursive equation $o = ?o^{\perp} ⅋ o$. We set $\iota = o^{\perp}$, so that $\iota = !o \otimes \iota$ and $o = ?\iota ⅋ o$.

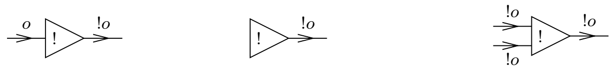
In the present setting, there are eleven symbols: par (arity 2), bottom (arity 0), tensor (arity 2), one (arity 0), dereliction (arity 1), weakening (arity 0), contraction (arity 2), codereliction (arity 1), coweakening (arity 0), cocontraction (arity 2) and closed promotion (arity 0). We present now the various cell symbols, with their typing rules, in a pictorial way. The principal port of a cell is located at one of angles of the triangle representing the cell, the other ports are located on the opposit edge. We put often a black dot to locate the auxiliary port number 1.

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

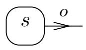
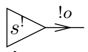
1.1.2 Exponential cells. They are typed according to a strictly polarized discipline. Here are first the *why not* cells, which are called *dereliction*, *weakening* and *contraction*:



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



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

Given a (non necessarily simple) net s with only one free port  we introduce a cell  .

A *simple net* is a typed interaction net, in the signature we have just defined.

A *net* is a finite formal sum of simple nets having all the same interface. Remember that the interface of a simple net s is the set of its free ports, together with the mapping associating to each free port the type of the oriented wire of s whose ending point is the corresponding port.

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 and codereliction cells are equipped with labels belonging to $\mathcal{L}$. We require moreover that, if two labels occurring in a labeled net are equal, they are equal to τ. All the nets we consider in this paper are labeled. In our pictures, the labels of dereliction and codereliction cells will be indicated, unless it is τ, in which case the (co)dereliction cell will be drawn without any label.

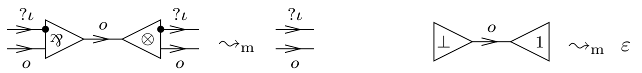
2 Reduction rules

We denote by Δ the collection of all simple nets and by $\mathbb{N}\langle\Delta\rangle$ the collection of all nets (finite sums of simple nets with the same interface).

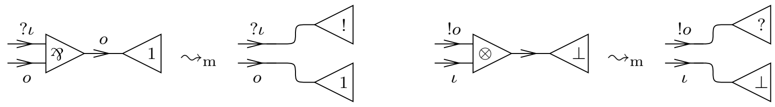
A *reduction rule* is a subset $\mathcal{R}$ of $\Delta \times \mathbb{N}\langle\Delta\rangle$ consisting of pairs (s, s') where s is made of two cells connected by their principal ports and s' has the same interface as s. This set can be finite or infinite. Such a relation is easily extended to arbitrary simple nets ($s \mathrel{\mathcal{R}} t$ 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 simple and $t = t_1 + \cdots + t_n$ where t_i is obtained by replacing 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 s' by this extension $\mathcal{R}^\Sigma$ if $s' = s'_1 + \cdots + s'_n$ where, for each i, $s_i \mathrel{\mathcal{R}} s'_i$ or $s_i = s'_i$. Last, $\mathcal{R}^*$ is the transitive closure of $\mathcal{R}^\Sigma$.

2.1 Defining the reduction

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

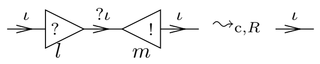


where ε stands for the empty simple net (not to be confused with the net $0 \in \mathbb{N}\langle\Delta\rangle$, the empty sum, which is not a simple net). The next two rules concern the interaction between a binary and a nullary multiplicative cell.



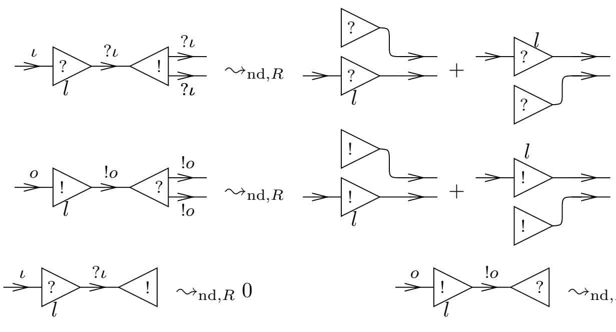
So here the reduction rule (denoted as $\rightsquigarrow_{\mathrm{m}}$) has four elements.

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

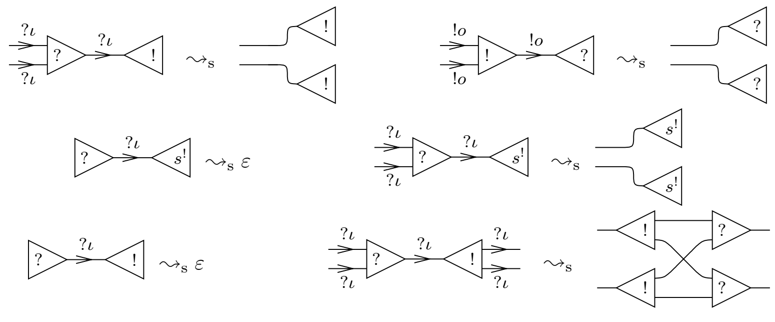


So the set $\rightsquigarrow_{\mathrm{c},R}$ is in bijective correspondence with the set of pairs (l, m) with $l, m \in R$ and $l = m \Rightarrow l = m = \tau$.

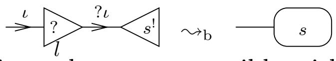
2.1.3 Non-deterministic reduction. Let $R \subseteq \mathcal{L}$. We have the following reductions if $l \in R$.



2.1.4 Structural reduction.



2.1.5 Box reduction.



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.

One can check that we have provided reduction rules for all possible 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.

2.2 Confluence

Theorem 1. *Let $R, R' \subseteq \mathcal{L}$. Let $\mathcal{R} \subseteq \Delta \times \mathbb{N}\langle\Delta\rangle$ be the union of some of the reduction relations $\rightsquigarrow_{\mathrm{c},R}$, $\rightsquigarrow_{\mathrm{nd},R'}$, $\rightsquigarrow_{\mathrm{m}}$, $\rightsquigarrow_{\mathrm{s}}$ and $\rightsquigarrow_{\mathrm{b}}$. The relation $\mathcal{R}^*$ is confluent on $\mathbb{N}\langle\Delta\rangle$.*

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: $\rightsquigarrow_R = \rightsquigarrow_{\mathrm{m}} \cup \rightsquigarrow_{\mathrm{c},\{\tau\}} \cup \rightsquigarrow_{\mathrm{s}} \cup \rightsquigarrow_{\mathrm{b}} \cup \rightsquigarrow_{\mathrm{nd},R}$. We set $\rightsquigarrow_{\mathrm{d}} = \rightsquigarrow_{\emptyset}$ ("d" for "deterministic") and denote by $\sim_{\mathrm{d}}$ the symmetric and transitive closure of this relation.

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 $\{l, m\}$-neutrality. Let l and m be distinct elements of $\mathcal{L} \setminus \{\tau\}$. We call *$(l, m)$-communication redex* a communication redex whose (co)dereliction cells are labeled by l and m. 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 1. *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.*

2.3.2 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: $s \rightsquigarrow^*_{\{l,m\}} s_1 + s_2 + \cdots + s_n$ where s_1 is a simple net which contains an (l, m)-communication redex (with dereliction labeled by m and codereliction labeled by l) and becomes t when one reduces this redex, and each s_i (for $i > 1$) is $\{l, m\}$-neutral.

Lemma 2. *The relation $\sim_{\mathrm{d}} \subseteq \Delta \times \Delta$ is a strong bisimulation on $\mathbb{D}_{\mathcal{L}}$.*

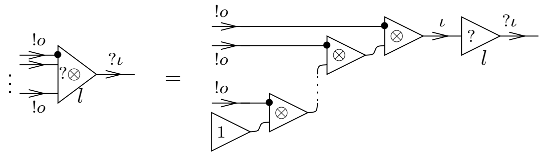
3 A toolbox for process calculi interpretation

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.

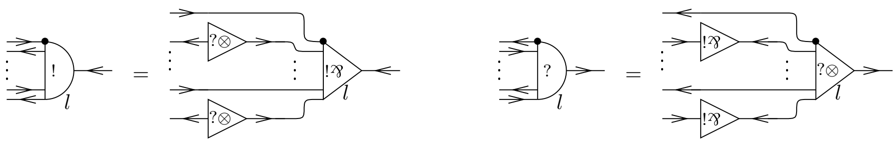
3.1.2 The dereliction-tensor and the codereliction-par cells. Let n be a non-negative integer. We define an n-ary cell as follows. It will be decorated by the label of its dereliction cell (if different from τ).



The number of tensor cells in this compound cell is equal to n. One defines dually the $!⅋$ compound cell.

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 bear type $?\iota$ or $!o$. Therefore we omit types and draw all wires with an orientation corresponding to the $?\iota$ type.

The *n-ary input cell* and the *n-ary output cell* are defined as



with n pairs of auxiliary ports.

Prefix cells are labeled by the label carried by their outermost dereliction-tensor or codereliction-par compound cell, if different from τ, the other codereliction-par or dereliction-tensor compound cells being unlabeled (that is, labeled by τ).

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 calculus of solos[3].

These switches will be closed by "boxed identity cells", which are the unique use we make of promotion in the present work. Let I be the "identity" net of Figure 2.

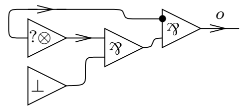
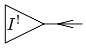


Fig. 2. Identity

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 3 shows how we picture a communication area of order 3.

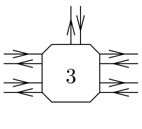
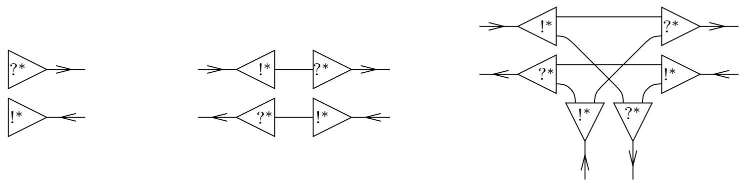


Fig. 3. Area of order 3

A communication area of order n is made of $n+2$ pairs of $(n+1)$-ary generalized cocontraction and contraction cells $(\gamma_1^+, \gamma_1^-), \ldots, (\gamma_{n+1}^+, \gamma_{n+1}^-)$, 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^+.

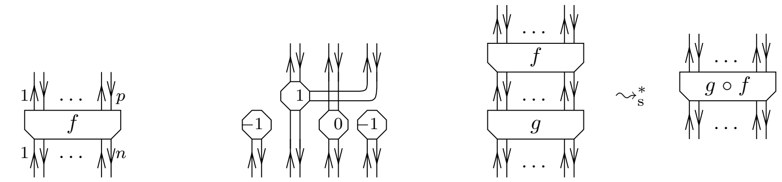
So the communication area of order -2 is the empty net ε, and communication areas of order -1, 0 and 1 are respectively of the shape



3.2.2 Identification structures. Let $n, p \in \mathbb{N}$ and let $f : \{1, \ldots, p\} \to \{1, \ldots, n\}$ 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 4(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 $n = 4$, $p = 3$, $f(1) = 2$, $f(2) = 3$ 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 4(b).

---

[3] It is shown in [LV03] that one can encode the π-calculus sequentiality induced by prefix nesting in the completely asynchronous solo formalism: the idea of such translations is to observe that, in a solo process like $P = \nu y\,(u(x,y) \mid y(\ldots)) \mid Q$, the first solo must interact before the second one with the environment Q.



(a) Notation (b) Example (c) Reduction

Fig. 4. 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 and q, the resulting area is of order $p+q$, see Figure 5.

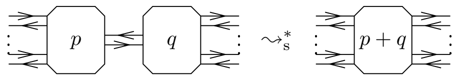
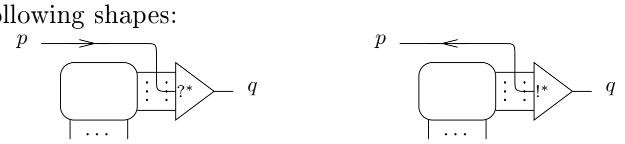


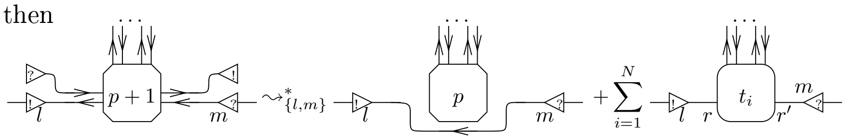
Fig. 5. Aggregation

3.3.2 Composition of identification structures. In particular, we get the reduction of Figure 4(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 following shapes:

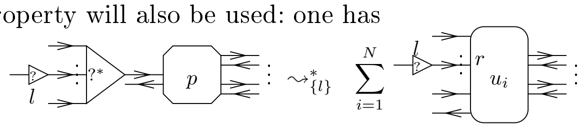


3.3.4 Forwarding of derelictions and coderelictions in communication areas. The following reduction shows that derelictions and coderelictions can meet eachother, when connected to a common communication areas. Let $l, m \in \mathcal{L}$, then



where N is a non-negative integer (actually, $N = (p+1)^2$) and, in each simple net t_i, both ports r and r' are forwarded.

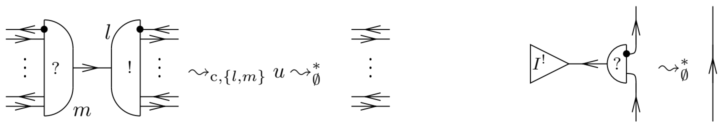
3.3.5 General forwarding. Let $l \in \mathcal{L}$. The following more general but less informative property will also be used: one has



where in each simple net 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, in Figure 6(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 6(b).



(a) Prefixes interaction (b) Transistor triggering

Fig. 6. Prefix reduction

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 that differential interaction nets cannot interpret these features[4]. Let $\mathcal{N}$ be a countable set of names. Our processes are defined by the following syntax. We use the same set 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 where a is bound.
- If P is a process, $a, b_1, \ldots, b_n \in \mathcal{N}$, the names b_i being pairwise distinct and if $l \in \mathcal{L}$, then $Q = [l]a(b_1 \ldots b_n) \cdot P$ 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, $a, b_1, \ldots, b_n \in \mathcal{N}$ and $l \in \mathcal{L}$, then $\overline{[l]a}\langle b_1 \ldots b_n\rangle \cdot P$ 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 one does not require the b_is to be distinct. The name a can be equal to some of the b_is.

The purpose of this labeling of prefixes is to distinguish the various occurrences of names as subject of prefixes. The set $\mathsf{FV}(P)$ of free names of a process P and the α-equivalence relation on processes are defined in the usual way.

[4] Replication can be interpreted using exponential boxes, sums are probably related to the unique additive connective of differential linear logic.

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 its labels. All the processes we consider in this paper are labeled.

4.1 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][5].

An *environment* is a function $e : \mathsf{Dom}\, e \to \mathsf{Codom}\, e$ between finite subsets 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 $S = (P_1, e_1) \cdots (P_N, e_N)$ of closures (denoted by simple juxtaposition). The set $\mathsf{FV}(S)$ of free names of a soup S is the union of the codomains of the environments of S. The soup S is labeled if all the P_is are labeled, with pairwise disjoint sets of labels. A *state* is a pair (S, L) where S 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 $\mathsf{FV}(S, L) = \mathsf{FV}(S) \setminus L$.

The state (S, L) is labeled if the soup S is labeled. All the states we consider are labeled. One defines the set $\mathcal{L}(S, L)$ of all labels of the state (S, L) as the disjoint union of the sets of labels associated to the processes of the closures of S.

4.1.1 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 $S = (P_1, e_1) \cdots (P_N, e_N)$ is *canonical* if each P_i is guarded, and that a state (S, L) is canonical if the soup S is canonical. One defines a rewriting relation $\rightsquigarrow_{\mathsf{can}}$ which allows to turn a state into a canonical one.

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

where, in the second rule, $a' \in \mathcal{N} \setminus (L \cup \mathsf{Codom}(e) \cup \mathsf{Codom}(S))$. One shows easily that, up to α-conversion, this reduction relation is confluent, and it is clearly strongly normalizing. We denote by $\mathsf{Can}(S, L)$ the normal form of the state (S, L) for this rewriting relation. Observe that if $(S, L) \rightsquigarrow_{\mathsf{can}} (T, M)$ then $\mathsf{FV}(T, M) \subseteq \mathsf{FV}(S, L)$.

4.1.2 Transitions. Next, one defines a labeled transition system $\mathbb{S}_{\mathcal{L}}$. The objects of this system are labeled canonical states and the transitions, labeled

[5] 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.

by pairs of labels, are defined as follows.

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

if $e(a) = e'(a')$. Observe that if $(S, L) \xrightarrow{lm} (T, M)$ then $\mathsf{FV}(T, M) \subseteq \mathsf{FV}(S, L)$.

4.2 Translation of processes

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 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 7(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 will be 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][6].

Clearly, if P and P' are α-equivalent, then $P\,\mathcal{I}_{a_1,\dots,a_n}\,s$ iff $P'\,\mathcal{I}_{a_1,\dots,a_n}\,s$.

4.2.1 Empty process. One has $\mathsf{nil}\,\mathcal{I}_{b_1,\dots,b_n}\,t$ if t is as in Figure 7(b).

4.2.2 Name restriction. One has $\nu a \cdot P\,\mathcal{I}_{b_1,\dots,b_n}\,t$ iff t is as in Figure 7(c), with s satisfying $P\,\mathcal{I}_{a,b_1,\dots,b_n}\,s$.

4.2.3 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 7(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.

4.2.4 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 all the free names of P are contained in $a, b_1, \dots, b_n, c_1, \dots, c_p$ and if t is as in Figure 7(e), where γ is a communication area of order 1 and where s is a simple net which satisfies $P\,\mathcal{I}_{a,b_1,\dots,b_n,c_1,\dots,c_p}\,s$.

[6] There is a simple interpretation of 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 $\rightarrow$ solo diagrams $\rightarrow$ differential nets". The simplification results from some rewiring and from the use of the boxed identity cells which is 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 [EL07].

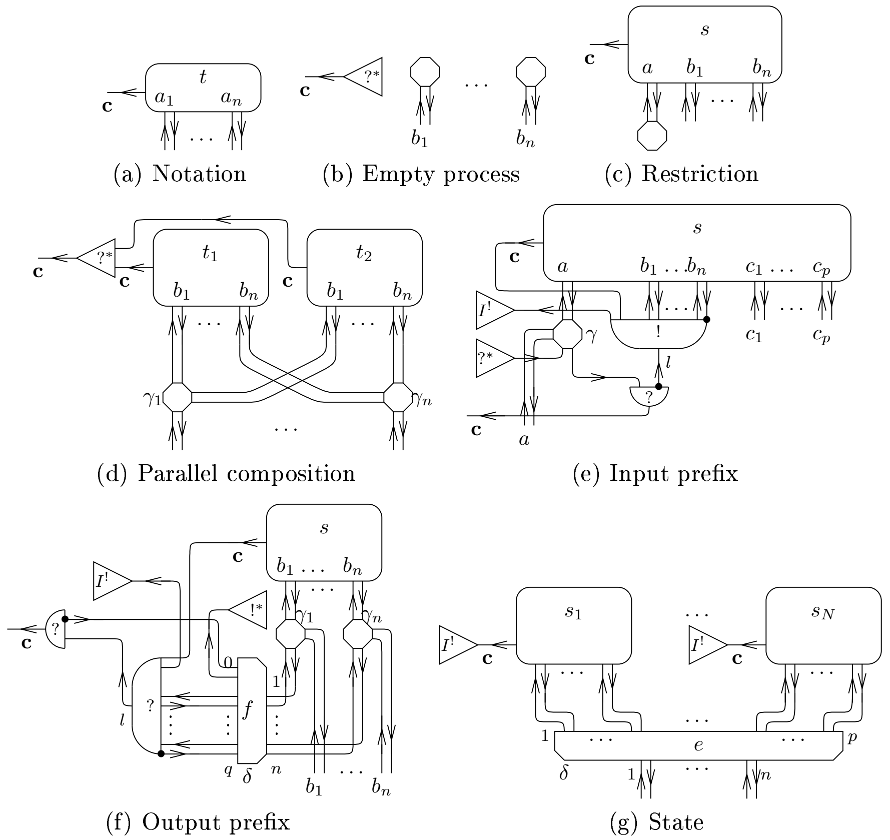


Fig. 7. Process and state translation

4.2.5 Output prefix. Let $l \in \mathcal{L}$. Let $b_1, \ldots, b_n$ be a list of pairwise distinct names and let $Q = \overline{[l]b_{f(0)}}\langle b_{f(1)} \ldots b_{f(q)}\rangle \cdot P$, where $f : \{0, 1, \ldots, q\} \to \{1, \ldots, n\}$ is a function. One has $Q\; \mathcal{I}_{b_1,\ldots,b_n}\; t$ if all the free names of P are contained in $b_1, \ldots, b_n$ and if t is as in Figure 7(f), where $\gamma_1, \ldots, \gamma_n$ are communication areas of order 1, δ is an f-identification structure and where s is a simple net which satisfies $P\; \mathcal{I}_{b_1,\ldots,b_n}\; s$.

4.2.6 States. Let $S = (P_1, e_1) \ldots (P_N, e_N)$ be a soup and $b_1, \ldots, b_n$ be a repetition-free list of names containing all the codomains of the environments $e_1, \ldots, e_N$. One has $S\; \mathcal{I}_{b_1,\ldots,b_n}\; t$ if, for some simple nets s_i ($i = 1, \ldots, N$) one has $P_i\; \mathcal{I}_{b^i_1,\ldots,b^i_{n_i}}\; s_i$ where $b^i_1, \ldots, b^i_{n_i}$ is a repetition-free enumeration of the domain of e_i, and t is obtained by connecting the pair of free ports of s_i associated to each b^i_k to the corresponding pair of free port of an identification structure associated to the function e defined by $e(b^i_k) = e_i(b^i_k)$, see Figure 7(g).

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 (S, L), one has $(S, L)\; \mathcal{I}_{b_1,\ldots,b_n}$ u if one has $S\; \mathcal{I}_{b_1,\ldots,b_n,c_1,\ldots,c_p}\; t$ for some repetition-free enumeration $c_1, \ldots, c_p$ of L (assumed of course to be disjoint from $b_1, \ldots, b_n$) and u is obtained by plugging communication areas of order -1 on the pairs of free ports of t corresponding to the c_js.

5 Comparing the transition systems

We are now ready to state a bisimulation[7] 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: $(S, L)\ \widetilde{\mathcal{I}}_{b_1,\ldots,b_n}\ s$ if there exists a simple net s_0 such that $(S, L)\ \mathcal{I}_{b_1,\ldots,b_n}\ s_0$ and $s_0 \sim_{\mathrm{d}} s$.

Theorem 2. *The relation $\widetilde{\mathcal{I}}_{b_1,\ldots,b_n}$ is a strong bisimulation between the labeled transition systems $\mathbb{S}_{\mathcal{L}}$ and $\mathbb{D}_{\mathcal{L}}$.*

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 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.

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.

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

[BHY03] Martin Berger, Kohei Honda, and Nobuko Yoshida. Strong normalisability in the pi-calculus. *Information and Computation*, 2003. To appear.

[BM05] Emmanuel Beffara and François Maurel. Concurrent nets: a study of prefixing in process calculi. *Theoretical Computer Science*, 356, 2005.

[CF06] Pierre-Louis Curien and Claudia Faggian. An approach to innocent strategies as graphs. Technical report, Preuves, Programmes et Systèmes, 2006. Submitted for publication.

[7] We are not using transition systems and bisimulation in the standard process theoretic way, for analyzing the possible interactions of processes with their environment. On the contrary, we use them for describing and comparing the internal reductions of processes and nets, thanks to labels.

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

[EL07] Thomas Ehrhard and Olivier Laurent. Acyclic solos. Submitted, 2007.

[ER06] Thomas Ehrhard and Laurent Regnier. Differential interaction nets. *Theoretical Computer Science*, 2006. To appear.

[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.

[Gir88] Jean-Yves Girard. Normal functors, power series and the λ-calculus. *Annals of Pure and Applied Logic*, 37:129–177, 1988.

[HL07] Kohei Honda and Olivier Laurent. An exact correspondence between a typed π-calculus and polarized proof-nets. In preparation, 2007.

[JM04] Ole Jensen and Robin Milner. Bigraphs and mobile processes (revised). Technical report, Cambridge University Computer Laboratory, 2004.

[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.