On differential interaction nets and the pi-calculus

Thomas Ehrhard, Olivier Laurent

On Differential Interaction Nets and the Pi-Calculus

Thomas Ehrhard and Olivier Laurent
Preuves, Programmes & Systèmes

September 11, 2006

Abstract

We propose a translation of a finitary (that is, replication-free) version of the pi-calculus into promotion-free differential interaction net structures, a linear logic version of the *differential lambda-calculus* (or, more precisely, of a *resource lambda-calculus*).

For the sake of simplicity only, we restrict our attention to a monadic version of the pi-calculus, so that the differential interaction net structures we consider need only to have exponential cells.

We prove that the nets obtained by this translation satisfy an acyclicity criterion weaker than the standard Girard (or Danos-Regnier) acyclicity criterion, and we compare the operational semantics of the pi-calculus, presented by means of an environment machine, and the reduction of differential interaction nets.

Differential interaction net structures being of a logical nature, this work provides a Curry-Howard interpretation of processes.

Introduction

Since the introduction of Linear Logic by Girard, it was clear to many logicians and computer scientists that some deep connection between this new logical setting and concurrency should show up. Indeed, linear logic proofs admit a *proof net* representation which has a very asynchronous and local reduction procedure. This impression has been enforced by the introduction of *interaction nets* by Lafont in [Laf95], where this kind of local and asynchronous interaction is generalized, showing that general recursion can be represented in such a setting, thanks to the *interaction combinators* of [Laf95].

In the same line of ideas, many semantical investigations of linear logic have insisted on the parallel flavour of the tensor connective of linear logic: one should mention here the Petri net interpretation of linear logic by Engbberg and Winskel [EW97], and many other works such as the concurrent games of Abramsky and Melliès [AM99] or the linear logic analyzis of processes by Beffara [Bef05]. The recent syntactic investigations of Curien and Faggian [CF06] on L-nets are going in the same direction, proposing a wide spectrum of "degrees of concurrency", ranging from the extreme sequentiality of *ludics* [Gir00] to the asynchrony of proof nets.

We think however that all these attempts towards "concurrent" interpretations of linear logic missed a crucial point of true concurrency, such as modelled by process calculi like Milner's π-calculus (see [Mil93, SW01]), namely its intrinsic *non-determinism*.

This failure is easily understandable since there is an apparent contradiction between non-determinism and the Curry-Howard approach to computation consisting in identifying proofs and programs. Indeed, one of the main properties that one expects from a well-behaved proof system is not only that it possesses a cut-elimination procedure, but also that this procedure enjoys a confluence property similar to the Church-Rosser property of the lambda-calculus. But confluence is a way of expressing determinism in a rewriting setting: typically, it implies that a closed proof of boolean type cannot reduce to *true* and also to *false*.

For instance, it has been one of the main achievements of linear logic to allow representations of classical logic which have the same provability power as standard classical logic, but with a Church-Rosser cut-elimination procedure, whereas the standard cut-elimination of classical Gentzen sequent calculus is

essentially non-deterministic (any two proofs of the same formula are identified by the corresponding equivalence relation). Thanks to linear logic and also to Parigot's lambda-mu calculus, classical logic is now understood as the logical side of a Curry-Howard correspondence whose computer science side corresponds to functional languages extended with `call-cc` like programming constructs.

But one can advocate that non-determinism is not an absolute concept, and that the non-determinism of classical cut-elimination, where all the elements of the same type are identified, is an extreme situation which is not desirable, even in concurrent settings.

In "static" – as opposed to game-theoretic – denotational semantics, determinism is modelled by means of the notion of *coherence*, which can be a combinatorial graphical concept as in coherence spaces [Gir87] or hypercoherences [Ehr93], or defined in terms of a norm on a vector space as in [Gir99, Gir04] (in that case, a vector is "coherent" if its norm is less than 1). In both cases, the effect of coherence is to prevent the formation of arbitrary unions (in the first case) or sums (in the second case).

But one knows since the introduction of powerdomains by Plotkin in [Plo76] that denotational semantics can be extended with a reasonable amount of non-determinism, corresponding for instance to a non-deterministic choice operator – non-deterministic extensions of the lambda-calculus and of PCF have been designed, with this kind of operational features, and powerdomain-based denotational semantics. Even more drastically, if one renounces to the domain-theoretic viewpoint on semantics, or more precisely, to the fact that the domain interpreting the types should have some kind of built-in coherence, or compatibility notion, then there are no obstacles to define models of lambda-calculi, or of linear logic, which admit non-determinism under the guise of the possibility of defining arbitrary joins (or unions, or sums) of points.

Such a model of the lambda-calculus has been first designed by Girard: this is the *quantitative* semantics of [Gir88], where types are interpreted by sets and a morphisms from a set S to a set T is a *normal functor* from the category $\mathbf{Set}^S$ ($\mathbf{Set}$ being the category of sets and arbitrary functions) to the category $\mathbf{Set}^T$, that is, a functor preserving all directed limits and binary pullbacks. Such functors can be represented as powerseries whose coefficients are sets (they turn out to be a special case of Joyal's analytic functors, see [Has02]).

Using vector spaces [Ehr02, Ehr05], the first author designed *finitary*[1] versions of quantitative semantics: the corresponding algebraic constructions are very natural in linear logic which has, at least at an intuitive level, strong connections with multilinear algebra. Types being interpreted as vector spaces, it becomes very natural to add proofs and multiply them by scalars, since proofs are interpreted by vectors. Other operations, which were absent from linear logic (and of course from classical and intuitionistic logic or lambda-calculus) such as *differentiation* become quite natural as well, and strongly use the possibility of adding vectors, that is, the non-determinism of the model: think of Leibniz laws $(uv)' = u'v + uv'$.

Fortunately, this extended semantical framework has a nice proof-theoretic counterpart, which corresponds to a simple extension of the rules that linear logic associates with the exponentials, recovering, at the exponential level, a symmetry that linear logic possesses for its multiplicative and additive connectives. 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; see [ER04] where an intuitive explaination of the connection between these nets and the elementary differential calculus is provided.

Cocontraction is a particularly interesting rule. Remember that, when representing the lambda-calculus in proof-nets, following the most natural translation proposed by [Gir87], the argument of type A, say, of an application has first to be promoted and turned into a net of type $!A$. This has the effect of making this argument duplicable by the function which will use it. Now cocontraction allows to take two (or more) such promoted arguments, and to put them together into a kind of compound argument, that the function will use, picking non-deterministically one or the other of the various terms (or nets) which have been promoted and then glued together by this operation. More precisely, this non-deterministic choice will occur when the compound argument will arrive in head position, and it is the role of the dereliction rule of linear logic to perform this choice, and then to open the "promotion box" of the branche of the cocontraction which will have been chosen — in usual proof-nets, no such choice has to be performed, and the role of the dereliction

[1] Where coefficients are finite numbers, instead of being arbitrary sets.

rule is simply to open the box.

In a joint work with Kohei Honda [HL06], 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. The basic idea consists in interpreting the parallel composition as a cut between a contraction link (to which several *emitters* 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 fondamental idea of this translation consists in using linear logic polarities for making the difference between emitters (negative) and receivers (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 behaves quite well, in the sense that π-calculus reduction is faithfully simulated by the reduction of linear logic proof-nets and therefore has to be considered as the first really convincing Curry-Howard interpretation of processes. It has however two features which can be considered as slight defects. First, it does not host very naturally linear receivers, since receivers must be promoted[2] for getting the right exponential type, and then they become indefinetely replicable. Second, this translation is not really modular, in the sense for instance that the interpretation of the parallel composition of two processes can hardly be described by connecting together the corresponding proof-nets through their conclusions (by cut links): some surgery has to be performed on the nets for extending the arity of their contraction and cocontraction trees.

Principle of our translation of processes to differential net structures

The purpose of the present paper is to continue this line of ideas, using more systematically the new structures introduced by differential interaction 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 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.). However these settings are not clearly related to a "logical" interpretation, whereas differential interaction nets have a straightforward denotational semantics[3] and can also be seen as an asynchronous notation for a sequent calculus, just as the proof-nets of linear logic.

The first key decision we made, guided by the structure of the typical cocontraction/contraction cut intended to interpret parallel composition, was of associating to 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 and will have to be considered as the *input port* of the corresponding name for this process, and the other will have a ?-type and will be considered as an *output port*.

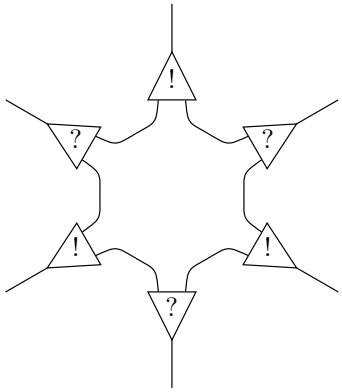
Therefore, for interpreting parallel composition, a simple cut implemented as a wire able to connect only pairs of ports was no more sufficient. More complex structures, able to connect pairs of wires, and not only wires, became necessary. We discovered such structures and called them *broadcast areas*: they are obtained by combining in a completely symmetric way generalized[4] cocontraction and contraction cells. There are broadcast areas of any "arity" (number of pairs of wires connected to it): the broadcast area of arity 1 is made of a coweakening cell and a weakening cell, the area of arity 2 is made of two wires.

The broadcast area of arity 3 can be pictured as an hexagon where each vertex is equipped with a binary cocontraction or a contraction cell (in an alternated way), whose auxiliary ports are connected to the auxiliary ports of its two neighbours. In the next picture, cocontraction cells are pictured as !-labelled triangles and contraction cells as ?-labelled triangles.

[2] This promotion has also the effect, by putting the net corresponding to the receiver sub-process into a box, of preventing this net to interact with the rest of the world before the box is opened by a dereliction corresponding to a process emitting on the channel on which the receiver is listening. This trick allows to simulate the sequentiality of prefix nesting in the pi-calculus, but one can advocate that replication has nothing to do with this sequentiality.

[3] The relational semantics of course, but also, in the simply typed case, algebraic semantics as the one of [Ehr05] or, also in the "pure" case, the predicate transformer semantics of [Hyv04].

[4] Cocontraction and contraction cells are binary cells, but by combining them, we obtain n-ary cocontraction and contraction cells called generalized (co)contraction cells.



The ports corresponding to the same pairs are the principal ports of antipodic cells. The area of arity 4 admits a similar description, but the cocontraction and contraction cells are now of arity 3, and the structure is drawn on a cube instead of an hexagon, etc.

The interpretation of the input and output prefix of the π-calculus is guided by polarities: input must be positive and output negative. The main ingredient for interpreting an output prefix is therefore a dereliction cell, since it turns a positive premise into a negative conclusion. Dually, the main ingredient for interpreting an input prefix is a codereliction cell. It turns out that, when interacting, codereliction and dereliction reduce to a simple wire connecting their auxiliary ports, which corresponds to the expected behaviour of the interaction of an input and an output prefix having the same subject, in the π-calculus.

Another essential construction in process algebras is the restriction operation which allows to make a name private to a process π, in such a way that other processes cannot communicate with π on the corresponding channel. This construction is simply interpreted by plugging a coweakening cell and a weakening cell (that is, a unary broadcast area) on the two ports corresponding to the name on which the restriction has to be performed.

Content

We first introduce the finitary π-calculus and specify an operational semantics for this calculus by means of an abstract machine similar to the well known Krivine's machine for interpreting the λ-calculus. The reason for this rather non standard choice is that in such an abstract machine, no substitution of names have to be performed in processes during the reduction. Indeed, in differential interaction net structures, name substitution is a rather critical operation involving the introduction of broadcast areas, see [EL06].

The operational semantics of this abstract machine is described by a transition system where nodes are "canonical states" of the machine, and arrows correspond to interactions between input and output prefixes of processes (and are labelled by the occurrences of these prefixes, specified by pairs of labels taken in a set $\mathcal{L}$; each prefix of our processes is labelled with an element of $\mathcal{L}$, all these labels being distinct within a state).

Then we introduce differential interaction net structures. We first define a differential linear logic, which is a propositional sequent calculus system having "!" and "?" as only logical connectives (these are unary connectives). This system could easily be extended with multiplicative connectives which would be necessary for interpreting the polyadic π-calculus. Since this extension is straightfroward, we prefer to stay in the purely exponential system. As in usual linear logic, there is a weakening and a dereliction rule (which introduce the "?" connective) and a contraction rule, which has a geometry similar to that of the "par" rule of multiplicative linear logic. But, whereas ordinary linear logic has only the promotion rule[5] for introducing the "!" connective, differential linear logic has two finitary ways of introducing the "!" connective: coweakening and codereliction. It has also a binary rule, cocontraction, which has a geometry similar to that of the "tensor" rule of multiplicative linear logic.

We also provide a relational denotational semantics for this sequent calculus and we introduce differential interaction net structures: these are graphical structures, similar to proof nets, and built using six kinds of

[5]This rule is fundamentally infinitary since it allow for the promoted proof to be arbitrarily copied.

cells corresponding to the rules of differential linear logic. More precisely, these structures are called "simple" and a net structure is a finite set of simple net structures, exactly as additive proof nets are sets of slices, see [Gir96, vGH05].

We explain how to type these net structures (with formulae of differential linear logic), and how to translate sequent calculus proofs as differential interaction net structures. A correctness criterion, analogous to the Danos-Regnier criterion for multiplicative proof-nets, allows to characterize the differential interaction net structures which represent sequent calculus proofs. This criterion however is too restrictive for our purpose, since some well formed processes (or states of the abstract machine) will translate to differential interaction net structures which will not satisfy the correctness criterion.

Then we define an equivalence relation on net structures, corresponding to the associativity and commutativity of cocontraction and contraction. We shall consider net structures up to this equivalence relation. We then define the reduction rules for these net structures. They split naturally in four categories.

- The *communicating rule*, which reduces a redex consisting of a codereliction and a dereliction cell through their principal ports.
- The *non-deterministic rules*, which reduce redexes consiting of a dereliction cell connected to contraction or weakening cell, or dually.
- The *structural rules*, which reduce redexes consiting of a coweakening or cocontraction cell connected to a weakening or contraction cell.
- And last, the *neutrality rules* (or Rétoré rules) which are not standard interaction net reduction rules[6]. These rules express that weakening is the neutral element of contraction, and dually for coweakening and cocontraction.

It has to be noticed that when reducing a non-deterministic redex in a simple net structure s, one does not get a simple net structure, but a finite set of simple net structures $\{s_1, \ldots, s_n\}$ (or linear combination, if we were working with coefficients, which is not the case in the present paper) of simple net structures. This motivates the terminilogy of "non-deterministic reduction" since we can consider that, in this situation, s reduces non-deterministically to one of the net structures $s_1, \ldots, s_n$. We consider then the net structure $\{s_1, \ldots, s_n\}$ as the non-deterministic superimposition of $s_1, \ldots, s_n$.

We show that the rewriting system defined in that way is confluent, this is not completely trivial because of the neutrality rules which create critical pairs.

We then introduce the notion of *active* net structure, which is a weakened version of the standard correctness criterion, and we prove a normalisation result for the non-deterministic, structural and neutrality reduction (called SND reduction) on the net structures enjoying this property. Since this SND reduction is also confluent, this shows that any active net structure has a unique normal form for this SND reduction.

We therefore can describe the operational semantics of differential interaction net structures by another transition system. The vertices of this transition system are the active differential interaction nets which are normal for the SND reduction; the only redexes of such net structures are therefore communication redexes. These differential net structures are moreover assumed to be labelled in the sense that each codereliction and dereliction cell bears a label belonging to $\mathcal{L}$ (all these labels being distinct, in each simple net structure belonging to the considered net structure). There is a transition from such an active, labelled and SND-normal net structure to another one if one can pass from the first to the second by firing all the communication redexes labelled by a given pair of labels (and then this transition is labelled by this pair of labels), and then by applying SND reductions only.

Next, we present basic modules built with the various cells of differential interaction nets: broadcast areas and input and output prefixes. We explain how these modules interact when one applies the SND reduction rules:

- when connecting together broadcast areas, one gets larger broadcast areas;

[6]One of the nicest features of interaction nets is that redexes consist of pairs of cells connected through their principal ports; in such a setting, critical pairs cannot appear and the reduction is trivially Church-Rosser.

- when connecting the principal port of an input or of an output prefix to one of the ports of a broadcast area, one obtains the non deterministic superimposition of all the ways of transfering this prefix to one of the other ports of the broadcast area: broadcast areas actually broadcast prefixes to the various agents to which they are connected[7];
- when connecting together prefixes by their principal ports, they can "cross eachother" or produce a communication redex (more precisely, one gets the non-deterministic superimposition of these two possibilities).

Using these modules, we translate processes (or more generally, states of our abstract machine) to differential interaction net structures, applying the principle explained in the previous section. We show that the net structures obtained in that way are active. Therefore, we can define a translation map Φ from the vertices of the transition system of states to those of the transition system of SND-normal active differential interaction net structures.

Last we show that the transitions of the transition system of process states are faithfully simulated by the transitions of the differential net structure transition system. More precisely, we show that if there is a transition labelled by (l, m) from a state E to a state F, then there is a transition labelled by (l, m) from differential interaction net structure $\Phi(E)$ to the differential interaction net structure $\Phi(F)$, and converesely, that if there is an (l, m)-transition from $\Phi(E)$ to some differential interaction net structure t, then $t = \Phi(F)$ for a state F such that $\Phi(F) = t$ and such that there is an (l, m)-transition from E to F.

For obtaining the second part of this result, we need to endow the labels of differential interaction net structures with an order relation and to restrict the communication reduction rule to fire only pairs of codereliction/dereliction cells whose labels are minimal in this order relation: this allows to simulate in differential interaction net structures the sequentiality which is due, in processes, to the fact that a prefix which is below another prefix cannot interact with the outer world before the outer prefix is fired. In differential net structures, the fact that two prefixes connected by their principal ports can cross eachother means that, at least in the present translation of processes to net structures, this kind of communication with "hidden" prefixes can occur.

This fact suggests to consider other process algebras, such as the calculus of *solos* [LPV01], where prefixing of communication agents is not possible, but which has nevertheless essentially the same expressive power as the π-calculus. This direction seems quite promising.

Contents

[7] In the sense of a replication-free broadcasting: non-deterministically, one of the connecteds agent will get the prefix and the others will still wait for a communication.

1 A finitary and monadic π-calculus

We restrict ourselves to this core process calculus (with a locality property, as we shall see in Section 3.5) because it is sufficient for developping our translation. However, extending our constructions to the polyadic case is easy, and requires simply to add the two multiplicative connectives of Linear Logic (tensor and par) to our interaction nets (as in the differential interaction nets of [ER04]). Replicable input processes can be interpreted as well, using exponential boxes, along the lines of [HL06].

Let $\mathcal{N}$ be a countable set of names.

1.1 Processes: syntax and reduction

General processes are defined as follows.

- $*$ is the *empty process*.
- $\pi \mid \pi'$ is the *parallel composition* of the two processes π and π'.
- $\nu a \cdot \pi$ is the process π where the name a has been made private. In this construction, the name a is bound and this operation is called *restriction*.
- $a(b) \cdot \pi$ is the process π prefixed by the action which reads name b on channel a. The name b is bound in this process and a is free and therefore a and b must be distinct. One says that a is the *subject* and that b is the *object* of this input action. In such a process, $a(b)$ is called an *input prefix*.
- $\overline{a}\langle b\rangle \cdot \pi$ is the process π prefixed by the action which writes name b on channel a (a and b must be distinct names[8]) – notice that the name b is *not* bound in this process. One says that a is the *subject* and that b is the *object* of this output action. In such a process, $\overline{a}\langle b\rangle$ is a called an *output prefix*.

The set $\mathsf{FV}(\pi)$ of free names of a process π is defined in the obvious way (the only binders are the restriction and the input prefix constructions).

General processes are considered up to α-equivalence and up to a structural congruence $\sim$ which is the least one such that

$$\begin{array}{rcll} * \mid \pi & \sim & \pi & \\ (\pi_1 \mid \pi_2) \mid \pi_3 & \sim & \pi_1 \mid (\pi_2 \mid \pi_3) & \\ \pi_1 \mid \pi_2 & \sim & \pi_2 \mid \pi_1 & \\ \nu a \cdot \nu b \cdot \pi & \sim & \nu b \cdot \nu a \cdot \pi & \\ \nu a \cdot (\pi_1 \mid \pi_2) & \sim & (\nu a \cdot \pi_1) \mid \pi_2 & \text{if } a \notin \mathsf{FV}(\pi_2); \text{ this rule is called } \textit{scope extrusion}. \end{array}$$

We can now define the reduction relation on processes. There is only one reduction rule

$$\overline{a}\langle b\rangle \cdot \pi_1 \mid a(c) \cdot \pi_2 \quad \rightsquigarrow \quad \pi_1 \mid (\pi_2\,[b/c])$$

which should be now extended to suitable contexts by means of the following deduction rules.

$$\frac{\pi \rightsquigarrow \pi'}{\pi \mid \rho \rightsquigarrow \pi' \mid \rho} \qquad \frac{\pi \rightsquigarrow \pi'}{\nu a \cdot \pi \rightsquigarrow \nu a \cdot \pi'} \qquad \frac{\pi \sim \rho \quad \rho \rightsquigarrow \rho' \quad \rho' \sim \pi'}{\pi \rightsquigarrow \pi'}$$

But we prefer to obtain the same result by means of an environment machine for processes.

[8] This restriction is non standard from the process calculus viewpoint, and by the way, it is not preserved during process reduction. In our presentation, the identification between the names a and b will be possible, by means of the environment functions of our abstract machine.

1.2 An environment machine for the π-calculus

We introduce a simple environment machine for evaluating processes of our version of the π-calculus, which is based on the ideas presented in [AC98]. It bears some similarities with Berry and Boudol's Chemical Abstract Machine [BB90]. This machine can easily be extended to larger fragments of the π-calculus.

A *closure* is a pair (π, e) where π is a process and e is a finite partial function from $\mathcal{N}$ to $\mathcal{N}$, such that $\mathsf{FV}(\pi) \subseteq \mathsf{Dom}(e)$. This function e is called the *environment* of the closure; the environments of the various closures of a soup (multiset of closures, see below) express the identifications to be performed between the free names of the various processes of that soup: these indentifications determine the possible communications between processes. The *codomain* $\mathsf{Codom}(c)$ of a closure (π, e) is the codomain of the function e.

A *soup* is a finite multiset $[c_1, \ldots, c_n]$ of closures. We use multiplicative notations for denoting multiset operations on soups and we use "c" for denoting the soup whose only element is the closure c. So for instance cS is the soup which has the same elements as the soup S, plus one instance of the closure c. The *codomain* $\mathsf{Codom}(S)$ of the soup $S = c_1 \ldots c_n$ is the union of the codomains of the c_i's. The essential property of soups is that there is no order on their elements: for any permutation f, the soups $c_1 \ldots c_n$ and $c_{f(1)} \ldots c_{f(n)}$ are the same.

A *state* is a pair $(S, \mathcal{P})$ where S is a soup and $\mathcal{P}$ is a finite set of names, called the set of *private names* of the state S. States are identified up to α-conversion, that is, up to renaming of their private names.

The *public names* of $(S, \mathcal{P})$ are the elements of $\mathsf{Codom}(S) \setminus \mathcal{P}$.

We define a non-deterministic reduction relation $\rightsquigarrow$ on states, and a sub-relation $\rightsquigarrow_{\mathsf{can}}$ of $\rightsquigarrow$, as follows.

$$
\begin{array}{rcll}
((*, e)S, \mathcal{P}) & \rightsquigarrow_{\mathsf{can}} & (S, \mathcal{P}) & \\
((\pi_1 \mid \pi_2, e)S, \mathcal{P}) & \rightsquigarrow_{\mathsf{can}} & ((\pi_1, e)(\pi_2, e)S, \mathcal{P}) & \\
((\nu a \cdot \pi, e)S, \mathcal{P}) & \rightsquigarrow_{\mathsf{can}} & ((\pi, e[a \mapsto \alpha])S, \mathcal{P} \cup \{\alpha\}) & \text{with } \alpha \in \mathcal{N} \setminus (\mathsf{Codom}(S) \cup \mathsf{Codom}(e)) \\
((\overline{a_1}\langle b\rangle \cdot \pi_1, e_1)(a_2(c) \cdot \pi_2, e_2)S, \mathcal{P}) & \rightsquigarrow & ((\pi_1, e_1)(\pi_2, e_2[c \mapsto e_1(b)])S, \mathcal{P}) & \text{if } e_1(a_1) = e_2(a_2)
\end{array}
$$

Remark 1 The right hand state of the third rule is uniquely defined, up to α-conversion of states, ie. renaming of private names.

Remark 2 The set of public names of states decreases or remains constant along the reduction (it can decrease strictly because of the last rule). This is certainly an expected property, since it holds for free names in processes, and it explains the importance of the set of private names in a state, although this set plays no role for the reduction itself.

We say that a state is *canonical* if all the closures appearing in its soup are *guarded*, that is, are of the shape (π, e) with the process π starting with an output or an input prefix (one says then that π is guarded). This means that the state is normal for the relation $\rightsquigarrow_{\mathsf{can}}$. Since this reduction relation is confluent[9] and (strongly) normalizing[10] as easily checked, any state has a unique *canonical form*: its normal form for $\rightsquigarrow_{\mathsf{can}}$.

Given a process π, we define a state $\mathsf{St}(\pi) = ((\pi, \mathrm{Id}), \mathsf{FV}(\pi))$.

Remark 3 Observe that there is no reason why the codomains of environments should be subsets of $\mathcal{N}$: it suffices to have a infinite countable set as codomain of these functions. We shall use greek letters $\alpha, \alpha_i, \beta \ldots$ for ranging over these symbols.

Remark 4 In the soup $S = (\pi_1, e_1) \cdots (\pi_p, e_p)$ of a state $(S, \mathcal{P})$, one can assume without loss of generality that the domains of the environment partial functions e_i (and thus the sets of free names of the processes π_i) are pairwise disjoint. We shall always make implicitely this assumption.

[9] There are no critical pairs for $\rightsquigarrow_{\mathsf{can}}$, so it enjoys the diamond property.

[10] Indeed, $\rightsquigarrow$ itself is strongly normalizing, since the "size" of soups decreases along the reduction, but this is due to the fact that we do not admit replication here. In a calculus with *guarded replications*, $\rightsquigarrow_{\mathsf{can}}$ would normalize nevertheless.

1.3 An associated transition system

Let $\mathcal{L}$ be an infinite set of *labels* ranged over by the letters $l, l_1, l', m \dots$. A *labelled state* is a state where each *subject occurrence* of each name (free or bound) has been individuated by means of a label taken in $\mathcal{L}$. We assume that all the labels occurring in a labelled state are pairwise distinct. When we want to specify the label carried by a subject occurrence of a name, we put it as a superscript to this occurrence. We denote by $\mathcal{L}(S)$ the set of all labels occurring in the state S.

We define the labelled transition system $\mathbb{S}_{\mathcal{L}}$ of states as follows.

The vertices of $\mathbb{S}_{\mathcal{L}}$ are the triples $(S, \mathcal{P}, \mathcal{A})$ where $(S, \mathcal{P})$ is a labelled canonical states and $\mathcal{A}$ is a set of names, disjoint from $\mathcal{P}$ and wich contains all the free names of $(S, \mathcal{P})$. This set $\mathcal{A}$ of names is here just for technical reasons: it simplifies a bit the definition of the translation map from states to differential interaction net structures.

There is a transition

$$(S, \mathcal{P}, \mathcal{A}) \stackrel{l/\overline{m}}{\to} (T, \mathcal{Q}, \mathcal{B})$$

in $\mathbb{S}_{\mathcal{L}}$ if the following conditions are satisfied:

- S is a canonical soup of the shape $S = (a_1^l(b) \cdot \pi_1, e_1)(\overline{a_2^m}\langle c\rangle \cdot \pi_2, e_2)S'$ with $e_1(a_1) = e_2(a_2)$
- $(T, \mathcal{Q})$ is the canonical form of the state $((\pi_1, e_1[b \mapsto e_2(c)])(\pi_2, e_2)S', \mathcal{P})$
- and $\mathcal{B} = \mathcal{A}$.

Observe that the labels occurring in T are pairwise distinct since this property holds for S.

2 Minimal differential linear logic

We provide a short introduction to a purely exponential and finitary system of differential linear logic (with a sequent calculus formalism), to the corresponding differential interaction nets and to their relational denotational semantics.

2.1 A polarized exponential linear logic

We introduce a polarized linear logic where the only connectives are the exponentials. So the formulae are given as follows, the atoms being ranged over $\xi, \xi_1 \dots$ and being assumed to be positive.

- A positive formula is an atom ξ, a formula $!N$ where N is a negative formula, or the formula ι.
- A negative formula is a negated atom $\xi^\perp$ or a formula $?P$ where P is a positive formula.

Linear negation is defined by means of the usual De Morgan laws. Moreover, formulae are considered up to the equivalence relation generated by the following equation:

$$\iota = !(\iota^\perp)\,.$$

Observe that this equation is compatible with polarities (it identifies two positive formulae).

We use o as a shorthand for $\iota^\perp$, so that $o = ?\iota$ and $\iota = !o$.

The positive formula ι, subject to the recursive equation above, will allow to define net structures which are "pure" in the sense of the "pure" (that is, untyped) lambda-calculus. It is similar to the type ι of the *pure nets* of [DR99].

2.1.1 Relational denotational semantics: interpreting formulae

Together with this sequent calculus, we provide its denotational semantics in the well known relational denotational model of linear logic, which is based on the star-autonomous category of sets and relations (see [BE01] for a description of this model).

If E is a set, then $\mathcal{M}_{\mathrm{fin}}(E)$ is the set of all finite multisets of elements of E.

Let $\mathsf{D} = \bigcup_{i=0}^{\infty} \mathsf{D}_i$ where $\mathsf{D}_0 = \emptyset$ and $\mathsf{D}_{i+1} = \mathcal{M}_{\mathrm{fin}}(\mathsf{D}_i)$ (D this is the set of all finitely branching unordered trees, with unlabelled leaves), which is the least set satisfying $\mathsf{D} = \mathcal{M}_{\mathrm{fin}}(\mathsf{D})$.

A *valuation* is a map from atoms to sets.

Given a valuation $\mathcal{I}$, one associates a set $[A]_{\mathcal{I}}$ to each formula of our system by setting $[\xi]_{\mathcal{I}} = [\xi^{\perp}]_{\mathcal{I}} = \mathcal{I}(\xi)$ and $[!A]_{\mathcal{I}} = [?A]_{\mathcal{I}} = \mathcal{M}_{\mathrm{fin}}([A]_{\mathcal{I}})$, and by setting $[\iota]_{\mathcal{I}} = \mathsf{D}$ (of course, this interpretation does not depend on $\mathcal{I}$), so that $[\iota]_{\mathcal{I}} = [!(\iota^{\perp})]_{\mathcal{I}}$.

The finite sequence $\Gamma = (A_1, \ldots, A_n)$ of formulae will be interpreted as the cartesian product $[\Gamma]_{\mathcal{I}} = [A_1]_{\mathcal{I}} \times \cdots \times [A_n]_{\mathcal{I}}$, to be understood as an n-ary *par* of linear logic.

2.1.2 A sequent calculus and its denotational semantics

We define a sequent calculus for this logic. As usual, sequents are expressions of the form $\vdash A_1, \ldots, A_n$ where each A_i is a formula. A *simple deduction* is a tree whose leaves and nodes are built using the deduction rules we describe now.

We also assume a valuation $\mathcal{I}$ to be given, and, simultaneously to the deduction system, we provide a denotational semantics of proofs: the semantics of a proof π of a sequent $\vdash \Gamma$ will be a subset $[\pi]_{\mathcal{I}}$ of $[\Gamma]_{\mathcal{I}}$.

The following one node tree is a proof π

$$\frac{}{\vdash A, A^{\perp}}$$

and one sets $[\pi]_{\mathcal{I}} = \{(x, x) \mid x \in [A]_{\mathcal{I}}\}$.

This axiom link cannot be restricted to hold in the case where A is an atom as one often does because the present system does not contain the promotion rule of linear logic, and therefore does not validate η-expansion for the exponentials: axioms cannot be η-expanded.

The cut rule is standard:

$$\frac{\begin{array}{cc} \vdots\,\lambda & \vdots\,\rho \\ \vdash \Gamma_1, A^{\perp} & \vdash A, \Gamma_2 \end{array}}{\vdash \Gamma_1, \Gamma_2}$$

and the semantics of this proof π is $[\pi]_{\mathcal{I}} = \{(\vec{y}, \vec{z}) \mid \exists x \in [A]_{\mathcal{I}}\ (\vec{y}, x) \in [\lambda]_{\mathcal{I}} \text{ and } (x, \vec{z}) \in [\rho]_{\mathcal{I}}\}$.

The coweakening rule is similar to a "tensor unit" rule of MLL

$$\frac{}{\vdash !N}$$

and the semantics of this proof π is $[\pi]_{\mathcal{I}} = \{[]\}$.

The weakening rule is standard:

$$\frac{\begin{array}{c} \vdots\,\lambda \\ \vdash \Gamma \end{array}}{\vdash \Gamma, ?P}$$

and the semantics of this proof π is $[\pi]_{\mathcal{I}} = \{(\vec{y}, []) \mid \vec{y} \in [\lambda]_{\mathcal{I}}\}$.

The cocontraction rule is similar to a tensor rule of MLL

$$\frac{\begin{array}{cc} \vdots\,\lambda & \vdots\,\rho \\ \vdash \Gamma_1, !N & \vdash \Gamma_2, !N \end{array}}{\vdash \Gamma_1, \Gamma_2, !N}$$

and the semantics of this proof π is $[\pi]_{\mathcal{I}} = \{(\vec{y}, \vec{z}, x + x') \mid (\vec{y}, x) \in [\lambda]_{\mathcal{I}} \text{ and } (\vec{z}, x') \in [\rho]_{\mathcal{I}}\}$ where "+" denotes the addtition of multisets.

The contraction rule is standard:

$$\dfrac{\begin{array}{c}\vdots\,\lambda\\ \vdash \Gamma, ?P, ?P\end{array}}{\vdash \Gamma, ?P}$$

and the semantics of this proof π is $[\pi]_{\mathcal{I}} = \{(\vec{y}, x + x') \mid (\vec{y}, x, x') \in [\lambda]_{\mathcal{I}}\}$.

The codereliction turns a negative formula into a positive one.

$$\dfrac{\begin{array}{c}\vdots\,\lambda\\ \vdash \Gamma, N\end{array}}{\vdash \Gamma, !N}$$

and the semantics of this proof π is $[\pi]_{\mathcal{I}} = \{(\vec{y}, [x]) \mid (\vec{y}, x) \in [\lambda]_{\mathcal{I}}\}$.

The dereliction does the converse.

$$\dfrac{\begin{array}{c}\vdots\,\lambda\\ \vdash \Gamma, P\end{array}}{\vdash \Gamma, ?P}$$

and the semantics of this proof π is $[\pi]_{\mathcal{I}} = \{(\vec{y}, [x]) \mid (\vec{y}, x) \in [\lambda]_{\mathcal{I}}\}$.

Though not essential, we admit the *mix rule* of linear logic (see [Gir87]) because it simplifies the correctness criterion for differential interaction nets.

$$\dfrac{\begin{array}{cc}\vdots\,\lambda & \vdots\,\rho\\ \vdash \Gamma_1 & \vdash \Gamma_2\end{array}}{\vdash \Gamma_1, \Gamma_2}$$

and the semantics of this proof π is $[\pi]_{\mathcal{I}} = \{(\vec{y}, \vec{z}) \mid \vec{y} \in [\lambda]_{\mathcal{I}} \text{ and } \vec{z} \in [\rho]_{\mathcal{I}}\}$.

We also admit the 0-ary case of the mix rule, namely

$$\dfrac{}{\vdash}$$

and the semantics of this proof π is $[\pi]_{\mathcal{I}} = \{*\}$ where $*$ is the unique element of the "empty cartesian product" (the cartesian product of an empty list of sets, which is a singleton).

A *proof* of a sequent $\vdash \Gamma$ is a set of simple proofs of $\vdash \Gamma$ and the relational semantics of such a proof is the union of the semantics of its elements. The elements of this set have to be considered as *slices* in the sense of [Gir96].

One could define a cut elimination procedure, which transforms simple proofs into finite cut-free proofs (and then proofs into cut-free proofs), but since this sequent calculus procedure is not essential to our purpose, we prefer to define cut elimination on differential interaction nets only where it is simpler to describe.

2.1.3 The pure system

The subsystem of this sequent calculus where the only positive formula is ι is called the *pure system*. It subsumes in some sense the general system, since, by replacing any atom ξ by ι in a proof, one gets a proof of the pure system.

2.2 Differential interaction nets

Very much like terms are made of function symbols, interaction nets [Laf95] are made of cells. Each cell has an arity n and has $n + 1$ *ports*, among which one *principal port* and n *auxiliary ports*. Cells are pictured as triangles, and ports are drawn on the border of these triangles: the principal port is located on one of the angles of the triangle, and the other ports, on the opposite side of the triangle.

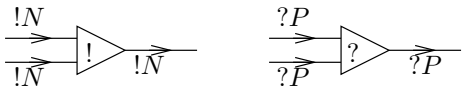
2.2.1 The cells

In purely exponential finitary differential interaction nets that we consider here, there are six kinds of cells that we give here with their typing rules (the types are formulae of the system of section 2.1):

- codereliction and dereliction, of arity 1

- coweakening and weakening, of arity 0

- cocontraction and contraction, of arity 2



2.2.2 Simple net structures and net structures

A *simple net structure* is a combination of such cells, connected with each other by *wires*, as described in [Laf95, ER04] (see these articles for a formal definition).

More precisely, any simple net structure has a finite set of *free ports* (which can be empty) and possesses therefore a set of *ports* made of its free ports and of the ports of its cells (these sets of ports are assumed to be pairwise disjoint). The *wiring* of the net structure can then be seen as a partition of this set of ports into sets of cardinality 2 or 0 (these latter wires are loops, they can appear during the reduction of a net structure, or when connecting two net structures).

We call *unit net structure* the net structure which has no cells and no free ports.

In this paper, a *net structure* is a set of simple net structures which have all the same set of free ports (in general, it would be a linear combination, but we only consider qualitative aspects here).

2.2.3 Typed net structures and interfaces

A *typing* of a simple net structure t is a mapping from the *oriented wires*[11] of t to the formulae of the linear logic of section 2.1 in such a way that the constraints of section 2.2.1 (typing rules for cells) be satisfied and in such a way that, if an oriented wire w is mapped to a formula A, then the oriented wire w' obtained by reversing the orientation of w be mapped to $A^\perp$.

Let t be a typed simple net structure (that is, a simple net structure equipped with a typing). Each free port of t is equipped with a type: the type associated to the wire connected to this port, this wire being considered as oriented towards the port under consideration. If $p_1, \ldots, p_n$ are the free ports of t, when t is typed, each port p has a type A and we call *interface* of t the corresponding mapping $p \mapsto A$.

An interface will often be written as a list $(p_1 : A_1, \ldots, p_n : A_n)$ where the p_i's are pairwise distinct.

Typing a net structure t consists in typing each of its elements, in such a way that all the elements of t have the same interface, and then this common interface is called the interface of the net structure t.

Remark 5 For any given interface, there is an empty net structure having this interface. One should absolutely avoid the confusion between the unit simple net structure, which has an empty interface, and the empty net structure, which admits all possible interfaces. The first one is empty in the multiplicative sense, the second one is empty in the additive sense; this is the same distinction as the between the neutral elements for multiplication (1) and for addition (0) in a ring.

[11] An oriented wire is a wire equipped with an orientation, that is, an order between its ending ports.

A net structure which is typed using only the formulae ι and $?\iota = o$ will be called a *pure net structure*. Any typed net structure can be turned into a pure net structure by replacing all the propositional atoms by ι.

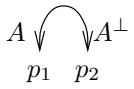
Convention : All the net structures we consider from now on will be assumed to be typed. The point of this convention is that, in any typed net structure, when two cells are connected by their principal ports, one knows that one of these two cells is a !-cell and the other one is a ?-cell. This convention is not restrictive, because recursive types are quite expressive.

2.3 Net structure associated with a proof and correctness criterion

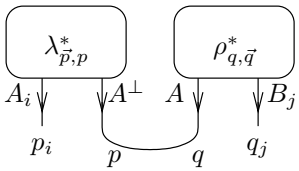
2.3.1 From simple proofs to simple net structures

Given a simple proof π of a sequent $\vdash A_1, \ldots, A_n$ in the sequent calculus of section 2.1 and a repetition-free list of ports $\vec{p} = (p_1, \ldots, p_n)$, we define a simple net structure $\pi^*_{\vec{p}}$ with free ports $p_1 : A_1, \ldots, p_n : A_n$. We give the definition, by induction on π, sticking to the notations of section 2.1.2.

If π consists of an axiom, then $\pi^*_{p_1,p_2}$ is



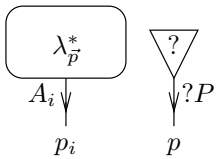
If π ends with a cut rule, then $\pi^*_{\vec{p},\vec{q}}$ is



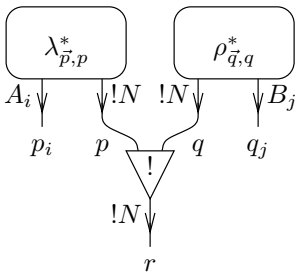
where we have set $\Gamma_1 = (A_1, \ldots, A_n)$, $\Gamma_2 = (B_1, \ldots, B_m)$ and where i ranges over $\{1, \ldots, n\}$ and j ranges over $\{1, \ldots, m\}$.

If π consists of a coweakening rule, then π^*_p is

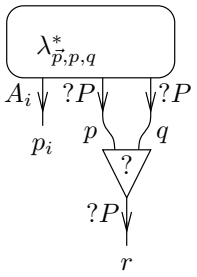
If π ends with a weakening rule, then $\pi^*_{\vec{p},p}$ is



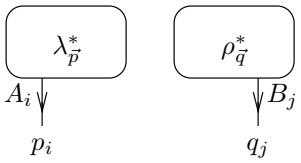
If π ends with a cocontraction rule, then $\pi^*_{\vec{p},\vec{q},r}$ is



If π ends with a contraction rule, then $\pi^*_{\vec{p},r}$ is



If π ends with a mix rule, then $\pi^*_{\vec{p},\vec{q}}$ is



Last, if π consists of a 0-ary mix rule, then $\pi^*_\emptyset$ is the unit net structure.

If π is a (not necessarily simple) proof of the sequent $\vdash \Gamma$, then one defines $\pi^*_{\vec{p}} = \{\lambda^*_{\vec{p}} \mid \lambda \in \pi\}$, and the obtained net structure is typed.

Just as in the case of multiplicative linear logic, one can characterize those net structures which are the translations of sequent calculus proofs.

2.3.2 Switching paths and correctness

Given a simple net structure t, we define a *switching path* (or simply *path*) of this structure as a sequence $\tau = (p_1, \ldots, p_n)$ of ports of t such that

- $i \neq j \Rightarrow p_i \neq p_j$;
- if $|i - j| = 1$ then p_i is the principal port and p_j is an auxiliary port of a cell of t, or conversely, or $\{p_i, p_j\}$ is a wire of t.
- τ does not contain the three ports of any contraction cell of t.

A *switching cycle* (or simply *cycle*) of t is a switching path $(p_1, \ldots, p_n)$ of t such that $n \geq 3$ and $(p_n, p_1, \ldots, p_{n-1})$ is also a switching path.

Theorem 6 *Let t be a simple net structure with interface $(p_1 : A_1, \ldots, p_n : A_n)$. The two following conditions are equivalent.*

- *There is a proof π of the sequent $\vdash A_1, \ldots, A_n$ such that $t = \pi^*_{\vec{p}}$.*

- *There are no switching cycles in t.*

The result extends obviously to (not necessarily simple) net structures. A simple net structure without switching cycles is called a *simple net*. A *net* is a net structure all the elements of which are simple nets.

We do not give the proof of this theorem, as it is completely similar to the proof of the corresponding result for MLL (see [BvdW95]), and can even easily be derived from that result (replace each contraction cell by a *par* cell and each cocontraction cell by a *tensor* cell).

2.3.3 Active cycles and active net structures

As we shall see, the translation of processes gives rise to net structures which contain cycles. However, these cycles will have the essential property of being active.

A cycle is *active* if it passes through a codereliction or dereliction cell. A net structure is *active* if all its cycles are active.

2.4 Experiments and the denotational semantics of net structures

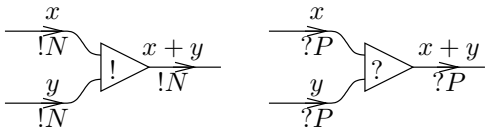
It is crucial to observe that the semantics of a proof depends only on the associated net. For this, we adapt the concept of *experiment* of [Gir87] to the present setting. Let $\mathcal{I}$ be a valuation.

Let t be a net structure. An experiment for t is a mapping which associates to each (unoriented) wire w of t an element of $[A]_{\mathcal{I}}$ where A is the formula associated to w by the typing of t (for this to make sense, one should mention an orientation for w, but observe that the set $[A]_{\mathcal{I}}$ does not depend on this orientation, since A becomes $A^{\perp}$ when the orientation is reversed).

Just as for typing[12], to each cell is associated a constraint on experiments. For codereliction and dereliction:

For coweakening and weakening:

And last, for cocontraction and contraction:



We denote by $\mathsf{exper}_{\mathcal{I}}(t)$ the set of all the experiments of t for the valuation $\mathcal{I}$.

Let $(p_1 : A_1, \ldots, p_n : A_n)$ be the interface of the net structure t, then each experiment ε of t induces an element $\mathsf{res}(\varepsilon)$ of $[A_1]_{\mathcal{I}} \times \cdots \times [A_n]_{\mathcal{I}}$ (by restricting ε to the wires of t ending on one of the free ports of t). We denote as $[t]^{\vec{p}}_{\mathcal{I}}$ the set of these restrictions and call this set the semantics of t. When t is not simple, one defines $[t]^{\vec{p}}_{\mathcal{I}}$ as the union of the sets $[s]^{\vec{p}}_{\mathcal{I}}$ for $s \in t$.

Then the following result is easy to prove, by simple inspection of the definitions of experiments, of the semantics of proofs and of the traduction of proofs into nets.

Theorem 7 *Let t be a net with interface $(p_1 : A_1, \ldots, p_n : A_n)$. Then for any proof π of the sequent $\vdash A_1, \ldots, A_n$ such that $\pi^*_{\vec{p}} = t$, the set $[\pi]_{\mathcal{I}}$ coincides with $[t]^{\vec{p}}_{\mathcal{I}}$.*

[12] Experiments can actually be considered as a kind of typing discipline, the elements of the relational model being considered as types in an *intersection* type system similar to those considered first by [CDCV80] and then by various authors.

2.5 Reduction rules for net structures

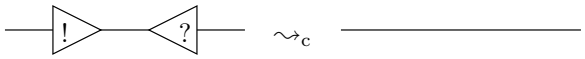
We introduce now the reduction rules for net structures. We refer to [ER04] for a simple mathematical interpretation of these rules, in terms of elementary differential calculus.

As usual in interaction nets, a redex is a sub-net structure consisting of two cells connected by their principal ports. Reducing a simple net structure consists in replacing this sub-net structure by a net structure which has the same free ports.

The reduction relation on net structures is denoted as $\rightsquigarrow$. We divide the rules into three groups.

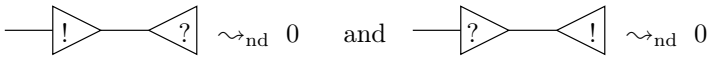
2.5.1 The communication reduction rule

This rule corresponds to the interaction between a dereliction and a codereliction cell.

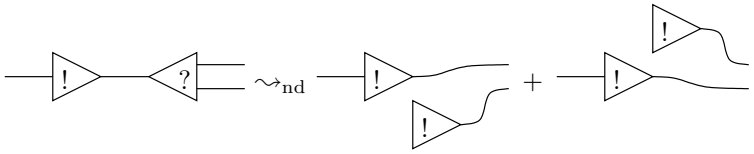


2.5.2 The non-deterministic reduction rules

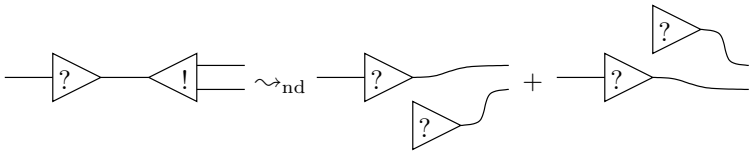
They correspond to the interaction between codereliction (resp. dereliction) and weakening and contraction (resp. coweakening and cocontraction). So there are four non-deterministic rules.



In the present qualitative setting[13], this means that whenever the left-hand pattern occurs within a simple net structure, the whole simple net can be replaced by the empty net structure (that is, the empty set of simple net structures), with the same free ports.

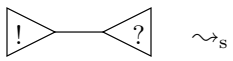


In the present qualitative setting, this means that whenever the left-hand pattern occurs within a simple net structure t, we can replace this simple net structure by the set of two simple net structures obtained by replacing in t the pattern under consideration by each of the two nets of the right-hand sum above. And symmetrically, we have



2.5.3 The structural reduction rules

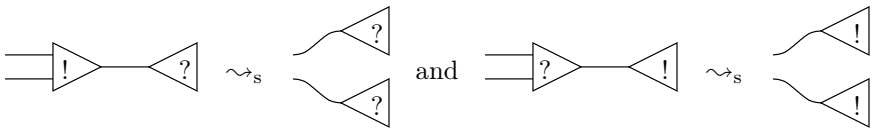
They correspond to the interaction between the coweakening, cocontraction, weakening and contraction cells. So there are four structural reduction rules.



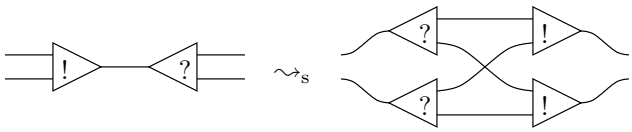
[13] Again, this means that, in the present paper, we interpret 0 as the empty set and addition as union.

where the right-hand side of this reduction rule is the unit net structure. This means that if one encounters this coweakening/weakening redex in a simple net structure, one can simply erase it, without modifying the rest of the net structure.

The cocontraction-weakening and contraction-coweakening rules read as follows.



And last the cocontraction-contraction reduction is the standard bialgebra rule.



2.5.4 Rétoré neutrality reduction rules

These are not exactly standard interaction nets reduction rules. In the present setting, they express the neutrality of coweakening and weakening for cocontraction and contraction respectively, on the left and on the right.

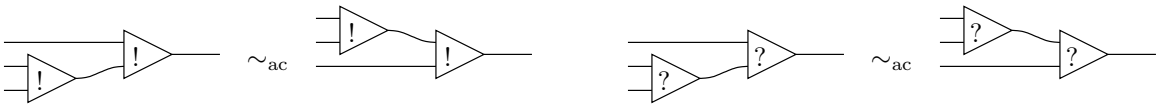
and

2.6 Equivalence of net structures

2.6.1 Associative and commutative equivalence

The last notion of equality of net structures we shall need corresponds to the fact that cocontraction and contraction are both associative and commutative operations. Associativity is absolutely crucial; commutativity can probably be dropped, we assume it because it holds in the relational model as well as in process algebras (corresponding to the fact that parallel composition is a commutative operation on processes). We denote by $\sim_{\mathrm{ac}}$ the corresponding equivalence relation on simple net structures (extended to arbitrary net structures in the obvious way: $u \sim_{\mathrm{ac}} u'$ iff $u = \{s_1, \ldots, s_n\}$ and $u' = \{s'_1, \ldots, s'_n\}$ with $s_i \sim_{\mathrm{ac}} s'_i$ for $i = 1, \ldots, n$).

This equivalence is generated by the following basic equations:



and

The crucial property of this equivalence relation is that it does not interact with the above defined rewriting relations: let $\mathcal{R} \in \{\rightsquigarrow_{\mathrm{c}}, \rightsquigarrow_{\mathrm{nd}}, \rightsquigarrow_{\mathrm{s}}, \rightsquigarrow_{\mathrm{n}}\}$ and let s, s' and u be net structures such that $s \sim_{\mathrm{ac}} s'$ (these two net structures being simple), and moreover $s \mathrel{\mathcal{R}} u$. Then there is a net structure u' such that $s' \mathrel{\mathcal{R}} u'$ and $u \sim_{\mathrm{ac}} u'$. Therefore, as far as reduction is concerned, it is harmless to consider net structures up to associativity and commutativity equivalence, what we shall do in the sequel.

2.6.2 Reducing non simple net structures

We have presented these reduction rules as a rewriting relations $\mathcal{R}$ from simple net structures to net structures. We extend these relations to relations from net structures to net structures, saying that $t \mathcal{R} t'$ if there is a non-empty family $(s_i)_{i\in I}$ of simple net structures, a family $(s'_i)_{i\in I}$ of net structures such that $s_i \mathcal{R} s'_i$ for each $i \in I$, and a net structure u such that

$$t = u \cup \{s_i \mid i \in I\} \quad \text{and} \quad t' = u \cup \bigcup_{i\in I} s'_i$$

Observe that, even on non simple net structures, $\mathcal{R}$ has still to be considered as a *one step* reduction relation, although it can reduce redexes in many elements of the net structure. The point is that, within each element of these simple net structures, only one redex can be reduced by this relation and that the non simple net structure has to be considered as a non-deterministic superposition.

2.7 Generalized reduction

Using (co)contraction trees instead of (co)contraction and (co)weakenging cells, the reduction rules involving these structural cells generalize in a natural way, as we explain now.

2.7.1 Cocontraction and contraction trees

We call *contraction trees* the simple net structures generated by the following inductive definition (we define in the same induction the principal port and the auxiliary ports of a contraction tree).

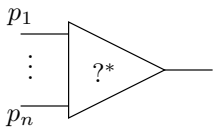
- A weakening cell is a contraction tree, whose principal port is the principal port of the cell and which has no auxiliary ports.
- A simple wire between two ports p and q typed by saying that the oriented wire (p, q) is of type o is a contraction tree whose principal port is q and which has one auxiliary port q. Such a structural tree will be said to be *trivial*.
- If t_1 and t_2 are contraction trees with disjoint sets of ports, and with principal ports p_1 and p_2 respectively, then the net obtained by plugging p_1 and p_2 to the auxiliary ports of a contraction cell c is a contraction tree, whose principal port is the principal port of c, and whose auxiliary ports are those of t_1 and those of t_2.

In a completely similar way, one defines the notion of cocontraction tree.

We use the term *structural tree* for refering to both notions.

Observe that, by applying $\rightsquigarrow_{\mathrm{n}}$ reductions only, any unary (co)contraction tree reduces to a wire. More generally, any structural tree whose arity is larger than 1 reduces to a structural tree which has no cocontraction and no contraction cells by $\rightsquigarrow_{\mathrm{n}}$ reduction.

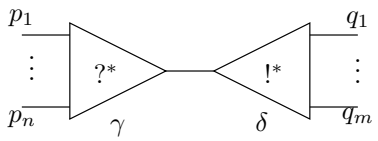
A contraction tree τ with auxiliary ports $p_1, \ldots, p_n$ will typically be pictured as follows.



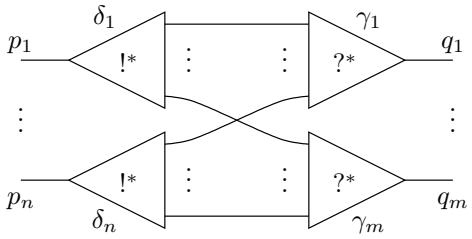
Such a contraction tree will be said to be *n-ary*. We adopt similar conventions for cocontraction trees.

2.7.2 Generalized structural reduction

A *generalized structural redex* is a simple net structure of the following shape



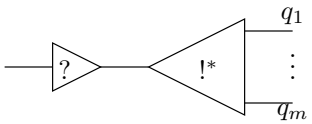
where both structural trees are non trivial. An easy computations shows that it reduces in several steps of structural reduction to the following simple net structure, with the same interface:



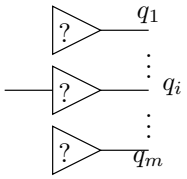
where the δ_i's are cocontraction trees isomorphic to δ and the γ_j's are contraction trees isomorphic to γ.

2.7.3 Generalized non-deterministic reduction

Similarly, the following *generalized non-deterministic redex*



(where the structural tree must be non trivial) reduces as easily checked, in several steps of non-deterministic reduction, to the set (or the sum, if we were working with coefficients) of the following simple nets



for $i = 1, \ldots, m$. A similar reduction occurs for a generalized redex consisting of a codereliction and a non trivial generalized contraction tree, connected throught their principal ports.

2.8 Confluence of the reduction of finite net structures

Let $\rightsquigarrow_{\mathrm{snd}}$ be the union of the rewriting relations $\rightsquigarrow_{\mathrm{nd}}$, $\rightsquigarrow_{\mathrm{s}}$ and let $\rightsquigarrow$ be the union of the rewriting relations $\rightsquigarrow_{\mathrm{c}}$, $\rightsquigarrow_{\mathrm{nd}}$, $\rightsquigarrow_{\mathrm{s}}$.

Given a binary relation $\mathcal{R}$, we denote by $\mathcal{R}^*$ its transitive closure, by $\mathcal{R}^-$ its "reflexive closure" (the union of $\mathcal{R}$ and of the identity relation) and by $\mathcal{R}^+$ its "strict transitive closore", that is, $\mathcal{R}^* \setminus \mathrm{Id}$.

As in [ER04], one can show that both relations $\rightsquigarrow^*_{\mathrm{snd}}$ and $\rightsquigarrow^*$ are confluent on finite net structures. This confluence follows easily from the next lemma, which states a slightly weakened form of "diamond property".

Lemma 8 *Let $\mathcal{R}_1, \mathcal{R}_2 \in \{\rightsquigarrow_{\mathrm{c}}, \rightsquigarrow_{\mathrm{nd}}, \rightsquigarrow_{\mathrm{s}}\}$ and let t, t_1 and t_2 be finite net structures such that $t\ \mathcal{R}_1\ t_1$ and $t\ \mathcal{R}_2\ t_2$. Then there is a finite net structure t' such that $t_1\ \mathcal{R}_2^-\ t'$ and $t_2\ \mathcal{R}_1^-\ t'$. The same holds if $\mathcal{R}_1$ and $\mathcal{R}_2$ are both the $\rightsquigarrow_{\mathrm{n}}$ reduction relation.*

This lemma is essentially trivial: within any simple net structure, two communicating, non-deterministic or structural redexes cannot have any cell in common. And the same holds for two neutrality redexes.

The $\rightsquigarrow_{\mathrm{n}}$ reduction is not a standard interaction net reduction relation (it involves two cells which are not connected through their principal ports), and so critical pairs between this reduction and the other standard interaction net reductions appear. We have to take them into account for proving confluence of the reduction.

2.8.1 A commutation property

We first prove that the $\rightsquigarrow_{\mathrm{n}}^*$ reductions can always be postponed. This requires a few preliminary lemmas.

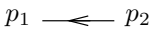
We call *pre-path* in a simple net structure u a sequence $p_1, \ldots, p_n$ of pairwise distinct ports such that, for each $i < n$, p_i and p_{i+1} are either principal and auxiliary ports of a cell of u, or the two ending ports of a wire of u. We say that a simple net structure u is pre-connected if any two ports of u are related by a pre-path in u. If u is not pre-connected, then it can be written in a unique way as a multiplicative juxtaposition of pre-connected simple net structures.

Lemma 9 *Let u be simple net structure and let p and q two free ports of u. If $u \rightsquigarrow_{\mathrm{n}} u'$, then there is a pre-path in u between p and q iff there is a pre-path in u' between p and q.*

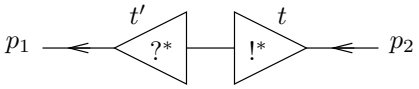
The proof is straightforward.

Lemma 10 *Let u be a simple net structure.*

1. *If u has two free ports p_1 and p_2 and reduces by a finite number of $\rightsquigarrow_{\mathrm{n}}$ steps to a wire w between p_1 and p_2 with the following typing:*

   then there is a unary cocontraction tree t and a unary contraction tree t' such that $u \rightsquigarrow_{\mathrm{s}}^* u'$ where u' is the following net structure:

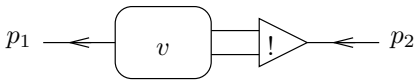
   

2. *If u has one free port p and reduces by a finite number of $\rightsquigarrow_{\mathrm{n}}$ steps to a coweakening cell connected to p, then u reduces to a 0-ary cocontraction tree with free port p by a finite number of $\rightsquigarrow_{\mathrm{s}}$ steps.*

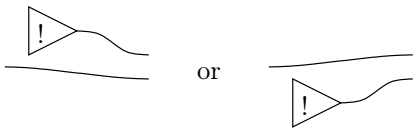
3. *If u has one free port p and reduces by a finite number of $\rightsquigarrow_{\mathrm{n}}$ steps to a weakening cell connected to p, then u reduces to a 0-ary contraction tree with free port p by a finite number of $\rightsquigarrow_{\mathrm{s}}$ steps.*

Proof. By induction on the number of cells of u. Observe that u cannot contain any codereliction or dereliction cells, since these cells are never erased by $\rightsquigarrow_{\mathrm{n}}$ reduction.

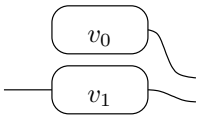
Case 1. Assume first that u is of the shape



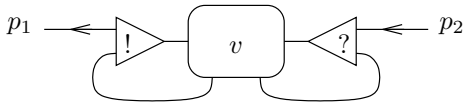
Then the $\rightsquigarrow_{\mathrm{n}}$-normal form of v must be



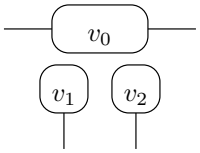
and hence by Lemma 9, v must be of the shape



where v_0 reduces to a coweakening cell and v_1 reduces to a wire. We conclude by applying the inductive hypothesis to v_0 and v_1. The reasoning is similar if p_1 is connected to the principal port of a contraction cell. We are left with the case where u is of the shape (up to commutativity of contraction and cocontraction)



As above, v must be of the shape



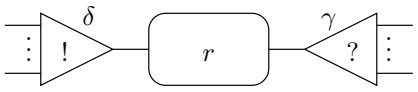
where v_0 reduces to a wire, v_1 to a coweakening cell and v_2 to a weakening cell, by $\rightsquigarrow_{\mathrm{n}}$ reductions only. We can apply the inductive hypothesis to each of these substructures and we conclude, using the generalized structural reduction of Section 2.7.2.

Case 2 and case 3. Similar reasoning. □

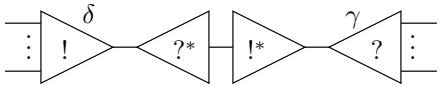
Lemma 11 *Let $\mathcal{R} \in \{\rightsquigarrow_{\mathrm{c}}, \rightsquigarrow_{\mathrm{nd}}, \rightsquigarrow_{\mathrm{s}}\}$ and let t, t_1 and t' be finite net structures. If $t \rightsquigarrow_{\mathrm{n}}^* t_1 \; \mathcal{R} \; t'$ then there is a net structure t_2 such that $t \; \mathcal{S}^+ \; t_2 \rightsquigarrow_{\mathrm{n}}^* t'$, where $\mathcal{S} = \mathcal{R} \cup \{\rightsquigarrow_{\mathrm{nd}}, \rightsquigarrow_{\mathrm{s}}\}$.*

Proof. It suffices to consider the case where t is simple (and therefore, t_1 is also simple).

The only non trivial case is the situation where, in t, the $\mathcal{R}$ redex δ, γ is "frozen" by a subnet r which reduces to a wire by a sequence of $\rightsquigarrow_{\mathrm{n}}$ reductions, in a configuration t_0 of the shape[14]

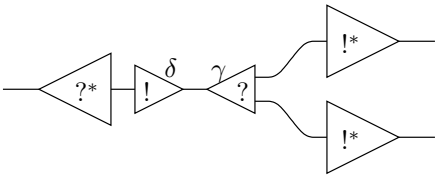


Applying Lemma 10, we first perform a series of $\rightsquigarrow_{\mathrm{s}}$ reductions to r, transforming it into a sub-net structure r_0 which is of the shape described at the end of Lemma 10, statement 1, so that t_0 becomes the following simple net structure t_1.



[14] A priori, r could have more than 2 free ports, and have a $\rightsquigarrow_{\mathrm{n}}$-normal form which contains a wire between its two free ports connected to the principal ports of γ and δ. But by Lemma 9, it contains then a sub-structure which reduces to this wire.

Then we apply the generalized structural and non-deterministic reductions to these two generalized redexes; assume for instance that δ is a codereliction cell and that γ is a contraction cell (so $\mathcal{R}$ is the $\rightsquigarrow_{\mathrm{nd}}$ reduction). We get, after a finite number of $\rightsquigarrow_{\mathrm{snd}}$ steps, the following simple net structure t_2



We conclude by firing the δ, γ redex, and then by reducing the unary (co)contraction trees to wires, using only $\rightsquigarrow_{\mathrm{n}}$ reductions. The other cases are similar. □

We can generalize this property to the transitive closures of the reduction relations, and we get a result which states that $\rightsquigarrow_{\mathrm{n}}$ can always been postponed.

Theorem 12 *Let $\mathcal{R} \in \{\rightsquigarrow_{\mathrm{snd}}, \rightsquigarrow\}$ and let t, t_1 and t' be finite net structures such that $t \rightsquigarrow_{\mathrm{n}}^* t_1\ \mathcal{R}^*\ t'$. Then there is a finite net structure t_2 such that $t\ \mathcal{R}^*\ t_2 \rightsquigarrow_{\mathrm{n}}^* t'$.*

Moreover, if t is $\mathcal{R}$-normal and if $t \rightsquigarrow_{\mathrm{n}} t_1$, then t_1 is also $\mathcal{R}$-normal.

Proof. The first statement is obtained by induction on the length of the $\mathcal{R}^*$ reduction, applying Lemma 11 in the inductive step.

The second statement is obtained first by reduction to the case where t is simple and then by a case analysis similar to that of Lemma 11. □

2.8.2 Confluence

Let $\rightsquigarrow_{\mathrm{sndw}}$ the sub-reduction relation of $\rightsquigarrow_{\mathrm{snd}}$ where one fires only redexes involving coweakening and weakening cells.

Lemma 13 *Let $\mathcal{R} \in \{\rightsquigarrow_{\mathrm{snd}}, \rightsquigarrow\}$. Let t, t_1 and t_2 be finite net structures, and assume that $t \rightsquigarrow_{\mathrm{n}} t_1$ and that $t\ \mathcal{R}\ t_2$. Then*

- *either there is a structure t' such that $t_1\ \mathcal{R}\ t'$ and $t_2 \rightsquigarrow_{\mathrm{n}} t'$*
- *or there is a structure t_3 such that $t_2 \rightsquigarrow_{\mathrm{sndw}} t_3 \rightsquigarrow_{\mathrm{n}}^* t_1$.*

Proof. It suffices to consider the case where t is simple. The first case arises when the $\rightsquigarrow_{\mathrm{n}}$ redex and the $\mathcal{R}$ redex are disjoint. The second case arises when these two redexes share a coconctraction or contraction cell. □

Let $\rightsquigarrow_{\mathrm{nw}}$ be the union of the $\rightsquigarrow_{\mathrm{n}}$ and of the $\rightsquigarrow_{\mathrm{sndw}}$ reduction relations. One can summarize the two-fold conclusion of Lemma 13 by saying that there exists a structure t' such that $t_1\ \mathcal{R}^-\ t'$ and $t_2 \rightsquigarrow_{\mathrm{nw}}^* t'$. Therefore we get the following result.

Lemma 14 *Let $\mathcal{R} \in \{\rightsquigarrow_{\mathrm{snd}}, \rightsquigarrow\}$. Let t, t_1 and t_2 be finite net structures, and assume that $t \rightsquigarrow_{\mathrm{n}}^* t_1$ and that $t\ \mathcal{R}^*\ t_2$. Then there is a finite net structure t' such that $t_1\ \mathcal{R}^*\ t'$ and $t_2 \rightsquigarrow_{\mathrm{nw}}^* t'$.*

Proof. One first deal with the case where $t\ \mathcal{R}\ t_2$, using the diamond property of $\rightsquigarrow_{\mathrm{n}}$ and of $\rightsquigarrow_{\mathrm{snd}}^- \cup \mathcal{R}^-$. Then one concludes by induction on the length of the $\mathcal{R}$ reduction. □

Lemma 15 *The reduction relation $\rightsquigarrow_{\mathrm{nw}}$ is confluent on finite net structures.*

Proof. Observe first that this reduction relation transforms simple net structures into simple or empty net structures. So it suffices to prove the result for such net structures. But $\rightsquigarrow_{\mathrm{nw}}$ is locally confluent (that is, if $t \rightsquigarrow_{\mathrm{nw}} t_1$ and $t \rightsquigarrow_{\mathrm{nw}} t_2$ then there exists t' such that $t_1 \rightsquigarrow_{\mathrm{nw}}^* t'$ and $t_2 \rightsquigarrow_{\mathrm{nw}}^* t'$). Moreover, $\rightsquigarrow_{\mathrm{nw}}$ is strongly normalizing on simple or empty net structures: if t and t' are simple and $t \rightsquigarrow_{\mathrm{nw}} t'$, then $(f(t'), g(t')) < (f(t), g(t))$ for the lexicographic order (defined by $(p, q) < (p', q')$ if $p < p'$ or $p = p'$ and $q < q'$), where $f(t)$ is the number of cocontraction and contraction cells in t and $g(t)$ is the number of coweakening and weakening cells in t. We conclude by Newmann's lemma. □

Theorem 16 *Let $\mathcal{R} \in \{\rightsquigarrow_{\mathrm{snd}}, \rightsquigarrow\}$. Then the relation $\mathcal{R} \cup \rightsquigarrow_{\mathrm{n}}$ is confluent on finite net structures.*

Proof. Let $\mathcal{S} = \mathcal{R} \cup \rightsquigarrow_{\mathrm{n}}$ and let t, t_1 and t_2 be finite net structures such that $t\ \mathcal{S}^*\ t_1$ and $t\ \mathcal{S}^*\ t_2$. By Theorem 12, there are finite net structures u_1 and u_2 such that $t\ \mathcal{R}^*\ u_i \rightsquigarrow_{\mathrm{n}}^* t_i$ for $i = 1, 2$. By confluence of $\mathcal{R}$, there is a finite net structure u such that $u_i\ \mathcal{R}^*\ u$ for $i = 1, 2$. By Lemma 14, for $i = 1, 2$, there is a finite net structure v_i such that $t_i\ \mathcal{R}^*\ v_i$ and $u \rightsquigarrow_{\mathrm{nw}}^* v_i$. By Lemma 15, there is a finite net structure t' such that $v_i \rightsquigarrow_{\mathrm{nw}}^* t'$ for $i = 1, 2$. For $i = 1, 2$, we have $t_i\ \mathcal{R}^*\ v_i \rightsquigarrow_{\mathrm{nw}}^* t'$ and hence $t_i\ \mathcal{S}^*\ t'$. □

Just as in [ER04], one can prove that the $\rightsquigarrow$ reduction relation is normalizing on finite *nets* — that is, net structures satisfying the acyclicity property — (the proof is not completely straightforward, because the cocontraction/contraction redex reduces to a structure which is larger than the redex itself). But strong normalization does not hold (if $t \rightsquigarrow t'$, one has $\{t, t'\} \rightsquigarrow \{t, t'\}$).

This normalization property is not essential for our purpose because the translation of processes produces cyclic net structures, as we shall see.

2.8.3 Duality of net structures

There are many notions of duality one might want to define on net structures, following the ideas of Girard [Gir87, Gir01] or of Beffara [Bef05]. Here we introduce the most basic one, which is simply based on types.

Let s and t be simple net structures. We shall say that s and t are *in duality* and write $s \perp t$ if the following properties hold: for any p which is a free port of s and of t, the type of p in the interface of s is the orthogonal of the type of p in the interface of t. In that case, one can define a net structure $s \cdot t$ by identifying the free ports of s and of t which have the same names[15].

More generally, if s and t are (not necessarily simple) net structures, we say that s and t are in duality if each element of s is in duality with each element of t. In that situation, one sets accordingly $s \cdot t = \{s' \cdot t' \mid s' \in s \text{ and } t' \in t\}$.

We just observe that computing the denotational semantics of a net structure is a "modular" operation.

Lemma 17 *Let s and t be net structures with interfaces $(p_1 : A_1, \ldots, p_l : A_l, q_1 : B_1, \ldots, q_m : B_m)$ and $(q_1 : {B_1}^\perp, \ldots, q_m : {B_m}^\perp, p_{l+1} : A_{l+1}, \ldots, p_n : A_n)$, with the p_i's pairwise distinct and assume that s and t are in duality. Let $\mathcal{I}$ be a valuation.*

Then $[s \cdot t]_{\mathcal{I}}^{\vec{p}}$ is the set of all $(x_1, \ldots, x_n) \in [A_1]_{\mathcal{I}} \times \cdots \times [A_n]_{\mathcal{I}}$ such that there exists $(y_1, \ldots, y_m) \in [B_1]_{\mathcal{I}} \times \cdots \times [B_m]_{\mathcal{I}}$ with $(x_1, \ldots, x_l, y_1, \ldots, y_m) \in [s]_{\mathcal{I}}^{p_1, \ldots, p_l, \vec{q}}$ and $(y_1, \ldots, y_m, x_{l+1}, \ldots, x_n) \in [t]_{\mathcal{I}}^{\vec{q}, p_{l+1}, \ldots, p_n}$.

One proves first the result when s and t are assumed to be simple, and then it simply follows from the definition of experiments. One gets the general result by observing that

$$[s \cdot t]_{\mathcal{I}}^{\vec{p}} = \bigcup_{\substack{s' \in s \\ t' \in t}} [s' \cdot t']_{\mathcal{I}}^{\vec{p}}.$$

[15] This operation is not as simple as it seems: it can produce sequences of wires connected to eachother just like electric extensions – and these sequences have to be turned in a single wire – and even loops.

2.8.4 Invariance properties

Typing, correctness and the denotational semantics of net structures are preserved under reduction.

Theorem 18 *Let t be a net structure with interface $(p_1 : A_1, \ldots, p_n : A_n)$, let t' be a net structure and assume that $t \rightsquigarrow t'$ (here t' is not assumed to be typed).*

- *t' admits a typing with interface $(p_1 : A_1, \ldots, p_n : A_n)$.*
- *If t is correct (that is, each of its elements satisfy the correctness criterion), then so is t'.*
- *For any valuation $\mathcal{I}$, one has $[t]^{\vec{p}}_{\mathcal{I}} = [t']^{\vec{p}}_{\mathcal{I}}$.*

The first two statements are proven in [ER04].

The last statement is proven by observing first that it holds for the two-cells redex nets, and then by applying Lemma 17.

2.9 Reduction of active net structures

Unfortunately, the translation of processes (or, more generally, states) into net structures gives rise to cyclic structures, and such net structures can have infinite reductions. But we shall take benefit of the fact that the net structures associated to states are active (that is, all their cycles pass through dereliction or codereliction cells).

2.9.1 Confluence and normalization for finite active net structures

Let $\rightsquigarrow_{\mathrm{Can}}$ be the union of the $\rightsquigarrow_{\mathrm{n}}$ and of the $\rightsquigarrow_{\mathrm{snd}}$ reductions on finite net structures. We have seen that this reduction relation is confluent.

Lemma 19 *Let t and t' be finite net structures and assume that $t \rightsquigarrow_{\mathrm{Can}} t'$. If t is active then t' is active.*

It suffices to prove the result for t simple, and it follows essentially from the fact that the $\rightsquigarrow_{\mathrm{Can}}$ reduction never cancels codereliction or dereliction cells. The proof requires a study of each possible reductions from t to t', similar to that of [ER04], where it is shown that correctness is preserved by reduction (the only non trivial case is that of a cocontraction/contraction reduction).

We want to prove that $\rightsquigarrow_{\mathrm{Can}}$ is weakly normalizing on active finite net structures. For this, by Theorem 12 and since $\rightsquigarrow_{\mathrm{n}}$ is weakly normalizing on arbitrary finite net structures, it suffices to prove that $\rightsquigarrow_{\mathrm{snd}}$ is weakly normalizing on active finite net structures.

For this, we introduce a reduction strategy, reducing generalized redexes rather than basic ones.

Let t be a simple net structure. A *maximal contraction tree* of t is a sub-net structure u of t which is a contraction tree such that no auxiliary port of u is connected to the principal port of a contraction or weakening cell of t. One defines dually the notion of maximal cocontraction tree of t.

Let t be a simple net structure and t' be a finite net structure, we write $t \rightsquigarrow^{\max}_{\mathrm{snd}} t'$ if t reduces to t' by reduction of a generalized structural redex whose cocontraction and contraction trees are maximal (called a *maximal structural redex*), or by reduction of a non-deterministic redex whose structural tree is maximal (called a *maximal structural redex*).

If $t = \{t_1, \ldots, t_n\}$ is a finite net structure, we write $t \rightsquigarrow^{\max}_{\mathrm{snd}} t'$ if $t' = t'_1 \cup \cdots \cup t'_n$ where, for each i, either t_i is normal for the SND (structural and non-deterministic) reduction and $t'_i = \{t_i\}$, or $t_i \rightsquigarrow^{\max}_{\mathrm{snd}} t'_i$, and this later situation occurs for at least one index i.

Given a simple net structure t, we denote by $\mathsf{N}_{\mathrm{str}}(t)$ the number of maximal structural redexes of t, and by $\mathsf{N}_{\mathrm{ND}}(t)$ its number of maximal non-deterministic redexes.

Lemma 20 *Let t and t' be simple net structures, and assume that t reduces to t' by reducing a maximal structural redex. If t is active, then $\mathsf{N}_{\mathrm{str}}(t') = \mathsf{N}_{\mathrm{str}}(t) - 1$.*

Proof. Consider a maximal structural redex of t, as pictured in Section 2.7.2. None of the ports $p_1,\ldots,p_n$ can be identical (or directly connected) to one of the ports q_j, since we know that each switching cycle of t must pass through a codereliction or a dereliction cell. Therefore, after reduction of this generalized redex, none of the ports p_i can be directly connected to the principal port of a generalized contraction tree (otherwise this contraction tree would have been present in t, and γ would not have been maximal). Similarly, none of the ports q_j can be directly connected to the principal port of a generalized cocontraction tree. Therefore, in t', the redex (γ,δ) vanishes and no new structural redex is created. Of course, new non-deterministic redexes can appear. □

Lemma 21 *Let t be a simple net structure and let $t' = \{t'_1,\ldots,t'_n\}$ be a net structure. If t reduces to t' by reduction of a maximal non-deterministic redex, then $\mathsf{N}_{\mathrm{ND}}(t'_i) = \mathsf{N}_{\mathrm{ND}}(t) - 1$ for $i = 1,\ldots,n$ and $\mathsf{N}_{\mathrm{str}}(t'_i) = \mathsf{N}_{\mathrm{str}}(t)$.*

Proof. Consider a maximal non-deterministic redex of t, as pictured in Section 2.7.3. One simply observes that none of the ports q_j can be directly connected to the principal port of a cocontraction tree, by maximality of the cocontraction tree of the generalized non-deterministic redex. □

If t is a simple net structure, we set $\mathsf{N}(t) = (\mathsf{N}_{\mathrm{str}}(t), \mathsf{N}_{\mathrm{ND}}(t))$, and order these pairs under the lexicographic order. If $t = \{t_1,\ldots,t_n\}$, we define $\mathsf{N}(t)$ as the maximum of the pairs $\mathsf{N}(t_1),\ldots,\mathsf{N}(t_n)$, for the lexicographic order. Combining Lemmas 19, 20 and 21, we get the following result.

Theorem 22 *Let t and t' be finite net structures. Assume that t is active and that $t \leadsto^{\max}_{\mathrm{snd}} t'$. Then $\mathsf{N}(t') < \mathsf{N}(t)$. There are no infinite sequences of net structures $t_1, t_2, \ldots$ such that $t_1 = t$ and $t_i \leadsto^{\max}_{\mathrm{snd}} t_{i+1}$.*

2.9.2 Purely Communicating Normal Form of an active net structure

Therefore, any finite active net structure t has a normal form t_1 for the $\leadsto^{\max}_{\mathrm{snd}}$ reduction, which is also a normal form for the $\leadsto_{\mathrm{snd}}$ reduction. This finite net structure t_1 in turn has a normal form t' for the $\leadsto_{\mathrm{n}}$ reduction, which is also a normal form for the $\leadsto_{\mathrm{snd}}$ reduction (see Theorem 12, second statement).

By Theorem 12, this finite net structure t' does not depend on the order in which the $\leadsto_{\mathrm{snd}}$ and $\leadsto_{\mathrm{n}}$ reductions have been performed. We denote it as $\mathsf{Com}(t)$. It is a finite set of active simple net structures whose only redexes are communication redexes (codereliction/dereliction redexes), that we call the *purely communicating normal form (PCNF)* of t.

The following observation results again directly from Theorem 12, but since it will be used quite often, we give it the status of a theorem.

Theorem 23 *If t and t_1 are finite net structures, if t is active and reduces to t_1 by $\leadsto_{\mathrm{Can}}$ reduction steps, then t_1 is active and $\mathsf{Com}(t) = \mathsf{Com}(t_1)$.*

2.9.3 Labelled nets

Just as we did for states of the π-calculus environment machine of Section 1.2, we need to introduce some labelling of net structures, for being able to trace their reduction. We define then a labelled transition system of labelled net structures.

A *labelled simple net structure* is a simple net structure u where all codereliction and dereliction cells have been equipped with a label taken in $\mathcal{L}$, all these labels being pairwise distinct. We denote by $\mathcal{L}(u)$ the set of all labels occurring in u.

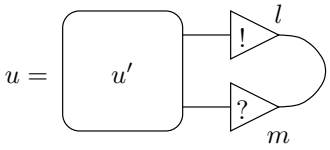
In a labelled simple net structure u, we assume moreover that the set of labels $\mathcal{L}(u)$ is equipped with a partial order relation $\leq$.

A *labelled net structure* is a set of labelled simple net structures which have all the same interface and all the same set of labels, equipped with the same order relation. Therefore we can speak of *the* partial order $\mathcal{L}(u)$ of a non-empty[16] labelled net structure u.

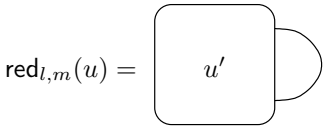
Remark 24 Let u be a labelled simple net structure. It is crucial to observe, by examining the $\rightsquigarrow_{\mathrm{snd}}$ and $\rightsquigarrow_{\mathrm{n}}$ reduction rules, that no codereliction or dereliction cells are created or suppressed during such reductions applied to u. More precisely, assume that u reduces by $\rightsquigarrow_{\mathrm{Can}}$ reduction to a set of simple net structures $\{u_1, \ldots, u_n\}$. Then for each $i = 1, \ldots, n$, there is a canonical bijection between the codereliction cells of u_i and those of u, and between the dereliction cells of u_i and those of u. Therefore, if u is active and has $\mathsf{Com}(u) = \{u_1, \ldots, u_n\}$ as PCNF, we have $\mathcal{L}(u_i) = \mathcal{L}(u)$ for each i, and we extend this identification to the order relation on labels, defining $\mathsf{Com}(u)$ as a labelled net structure.

Given two labels $l, m \in \mathcal{L}$, we denote by $\Delta_{l,m}$ the set of all simple net structures which contain a communication redex whose codereliction cell is labelled by l and whose dereliction cell is labelled by m.

Let t be a finite labelled net structure which is in PCNF, and let $l, m \in \mathcal{L}(t)$. We say that t is (l, m)-*reducible* if, for any element u of $t \cap \Delta_{l,m}$, that is, which is of the shape



the labelled simple net structure



whose set of labels is $\mathcal{L}(u) \setminus \{l, m\}$ equipped with the restricted order, is active.

We define a transition system $\mathbb{D}_{\mathcal{L}}$ as follows.

- The vertices of $\mathbb{D}_{\mathcal{L}}$ are the finite labelled net structures which are in PCNF.
- Let $s, t \in \mathbb{D}_{\mathcal{L}}$. There is a transition from s to t in $\mathbb{D}_{\mathcal{L}}$ labelled by l (as receiver) and m (as emitter), written $s \rightarrow^{l/\overline{m}} t$, if the following conditions are satisfied
  1. l and m are minimal in the poset $\mathcal{L}(s)$;
  2. s is (l, m)-reducible;
  3. the set $s \cap \Delta_{l,m}$ is non-empty
  4. and last
$$t = \bigcup_{u \in s \cap \Delta_{l,m}} \mathsf{Com}(\mathsf{red}_{l,m}(u)) \,.$$

Observe that this transition system is deterministic in the sense that, for any $s \in \mathbb{D}_{\mathcal{L}}$ and any $l, m \in \mathcal{L}$, there is at most one transition $s \rightarrow^{l/\overline{m}} t$ to another $t \in \mathbb{D}_{\mathcal{L}}$.

[16] Being pedantic, one can avoid this restriction by defining a labelled net structure as a pair (u, L) where u is a set of labelled simple net structures which have all the same poset of labels, this poset being L if it is defined, that is if u is non-empty.

3 From states to nets

We define a translation of states as defined in section 1.2 to finite *pure* net structures. Remember that in this system, there are only two types, which are ι and $?\iota = \iota^{\perp}$ (also denoted as o).

Convention : When drawing a pure net structure s we indicate the types of its wires simply by orienting them, the type of these oriented wires being always o.

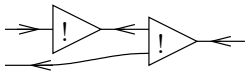
Quite often, such a pure net structure s will be a subnet of a larger structure and will be drawn as a box with rounded corners or a disk.

3.1 Primitives

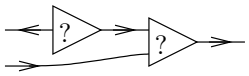
We define first the elementary simple net structures which will be used over and over when translating the π-calculus to differential interaction nets. They are of two kinds: *broadcast areas* which account for the parallel composition of processes, and *prefixes* which will be used for interpreting input and output prefixes of the π-calculus.

3.1.1 Input and output prefixes.

We call input prefix a net of the shape

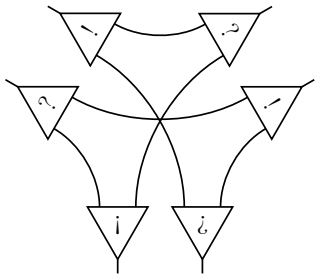


and output prefix a net of the shape

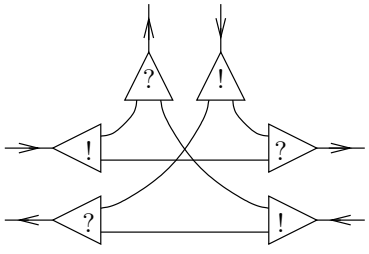


3.1.2 Broadcast areas.

The simplest non trivial *broadcast area* is the following net.



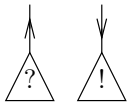
It will be used for interpreting the parallel composition operation. It will be usually pictured less symetrically, but more conveniently, as



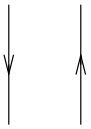
The picture above corresponds more precisely to the 3-ary broadcast area, denoted as Br_1 for reasons which will become clear soon. This net has three pairs of free ports.

One defines more generally Br_n for each integer $n \geq -1$. This net 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 auxiliray port of γ_i^+ to an auxiliary port of γ_j^- and a wire from an auxiliray port of γ_i^- to an auxiliary port of γ_j^+.

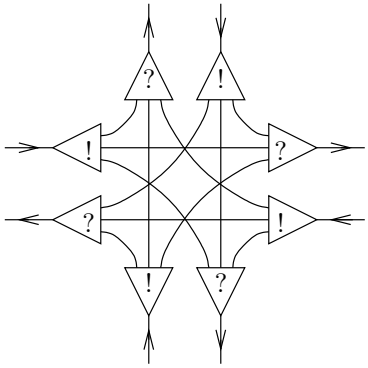
So Br_{-1} is



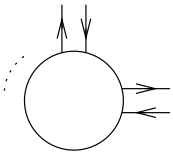
and Br_0 is a pair of parallel wires



As a last example, here is Br_2.

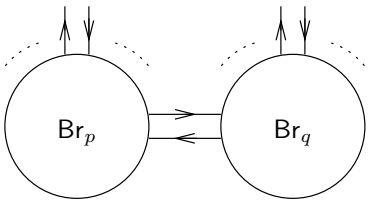


Broadcast areas will be pictured as disks, as for instance



3.1.3 Combining broadcast areas.

One can combine broadcast areas by connecting them through their free ports as follows:



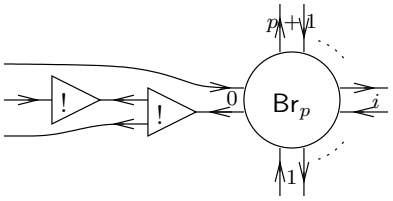
As easily checked, such a combination of broadcast areas reduces, using only $\rightsquigarrow_{\mathrm{s}}$ and $\rightsquigarrow_{\mathrm{n}}$ reductions, to the broadcast area Br_{p+q}. Observe by the way that the net above has $(p-1)+(q-1)=p+q-2$ pairs of free ports, which is exactly the number of pair of free ports of Br_{p+q} (this explains our notation for broadcast areas).

This crucial property will be called *associativity of bradcast areas*.

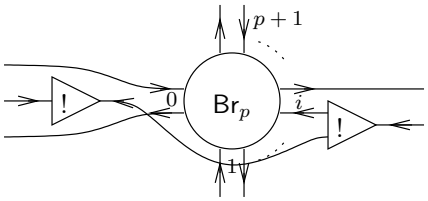
In particular, when one plugs a Br_{-1} net on a broadcast area Br_p, one reduces its number of pairs of free ports by 1, getting Br_{p-1}. When $p=-1$, one gets the empty net which we could also consistently denote as Br_{-2}.

3.1.4 Combining prefixes and broadcast areas.

Another very important configuration occurs when a prefix is connected to one of the free ports of a broadcast area, as follows (we have indexed the pairs of free ports of the broadcast area, from 0 to $p+1$, deciding conventionally that the input prefix is plugged on the negative wire of the pair numbered 0):

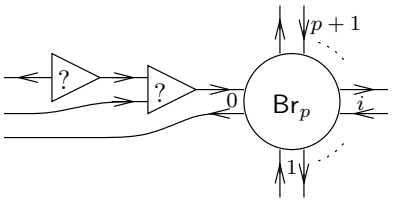


An easy graphical computation shows that the net above reduces to the following sum of p simple nets whose i-th term is

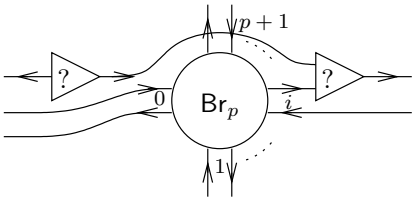


using only $\rightsquigarrow_{\mathrm{snd}}$ and $\rightsquigarrow_{\mathrm{n}}$ reductions (that is, $\rightsquigarrow_{\mathrm{Can}}$ reductions).

There is of course a completely symmetric equation concerning an output prefix connected to a broadcast area: the net



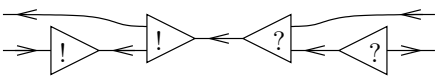
reduces to the sum of p terms whose i-th element is



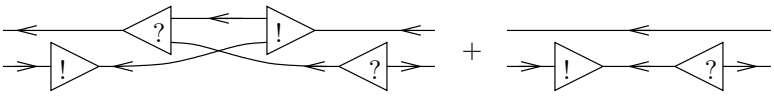
using only $\rightsquigarrow_{\mathrm{Can}}$ reductions.

3.1.5 Combining prefixes.

The following combination of prefixes



reduces by $\rightsquigarrow_{\mathrm{Can}}$ reductions to the following sum



and the second term of this sum reduces to a pair of wires, by a dereliction/codereliction reduction.

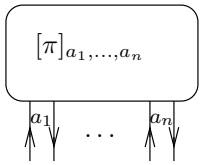
Remark 25 The first term of this sum explains one of the main difference between π-calculus processes and their translation into differential interaction net structures: in this latter setting, two prefixes communicating on the same channel can cross each other without communicating (first term of the sum), or decide to establish a communication between their object channels (second term of the sum).

3.2 Interpretation of states

3.2.1 Interpreting processes

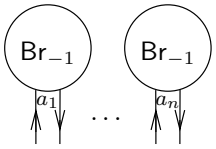
Given a labelled process π and a finite set A of n names containing all the free names of π, we define a simple labelled pure net structure $[\pi]_A$ with $2n$ free ports, and with the same set of labels: to each element a of the list is associated a pair of free ports (a^+, a^-) (input port typed by ι and output port typed by o respectively, in the interface of the pure net structure $[\pi]_A$). Very often this set A will be considered as enumerated as a list $\vec{a} = (a_1, \ldots, a_n)$ of pairwise distinct names and in that case we use indifferently A or $\vec{a}$ for denoting this set of names.

The definition is by induction on processes. For representing the interpretation $[\rho]_A$ of a process ρ, we use boxes with rounded corners (we keep angle corner boxes for the exponential boxes of linear logic) as follows



Simultaneously to this inductive definition, one checks by induction that trivially $\mathcal{L}([\pi]_A) = \mathcal{L}(\pi)$ and that the defined simple net structure is well typed in the pure type system.

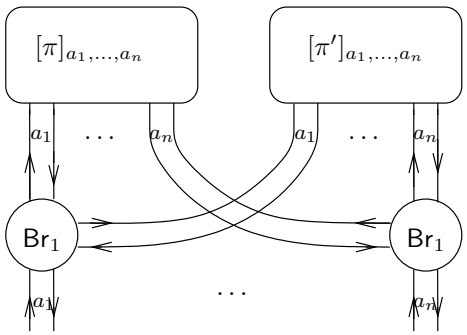
$[*]_A$ is the net structure



The set of labels of this net structure is empty.

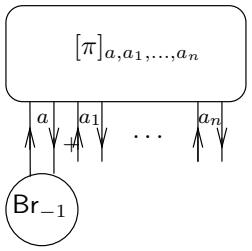
$[\pi \mid \pi']_{\vec{a}}$ is the net structure[17]

[17]This net structure seems to contain two occurrences of each of the ports a_i^+ and a_i^-; these ports are assumed to be distinct although we gave them the same names for the sake of readability.



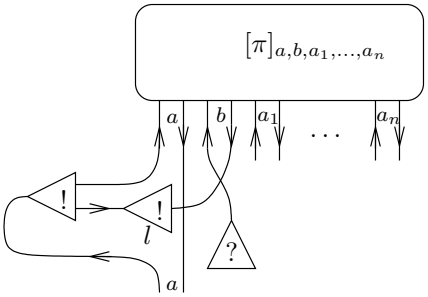
Observe that we introduce as many broadcast areas as there are free names under consideration. The set of labels of this net structure is the union $\mathcal{L}([\pi]_{\vec{a}}) \cup \mathcal{L}([\pi']_{\vec{a}})$ which is a disjoint union by inductive hypothesis (we know by inductive hypothesis that $\mathcal{L}([\pi]_{\vec{a}}) = \mathcal{L}(\pi)$ and $\mathcal{L}([\pi']_{\vec{a}}) = \mathcal{L}(\pi')$ and these sets are disjoint because labels occurring in processes are pairwise distinct). The order relation we endow this disjoint union with is the parallel composition of orders ($l \leq m$ if l and m belong to the same component of this union and if $l \leq m$ holds in that component).

$[\nu a \cdot \pi]_{\vec{a}}$ is the net structure



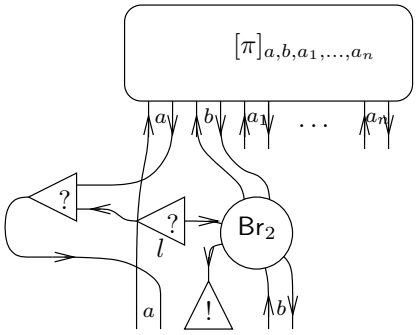
This net structure has the same set of labels as $[\pi]_{a,\vec{a}}$, and the same order relation on these labels.

For interpreting prefixed processes, we use crucially the hypothesis that a and b are distinct names. $[a^l(b) \cdot \pi]_{a,\vec{a}}$ is the net structure

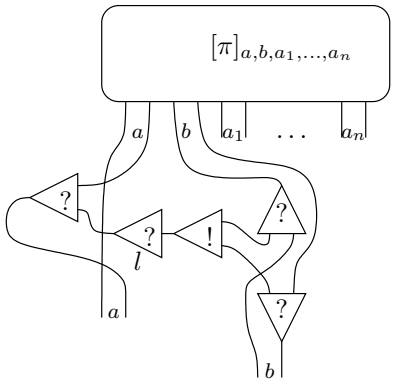


The set of labels of this net structure is $\mathcal{L}([\pi]_{\vec{a}}) + \{l\}$ (where, by inductive hypothesis, $\mathcal{L}([\pi]_{\vec{a}}) = \mathcal{L}(\pi)$ does not contain the label l) and the order relation on this set is defined by setting $l < m$ for all $m \in \mathcal{L}([\pi]_{\vec{a}})$.

Last, $[\overline{a^l}\langle b \rangle \cdot \pi]_{a,b,\vec{a}}$ is the net structure



or equivalently (leaving the types implicit)



These two net structures are equivalent up to $\rightsquigarrow_{\mathrm{Can}}$ reduction rules, but the first expression is often more convenient because of the general associativity property of broadcast areas.

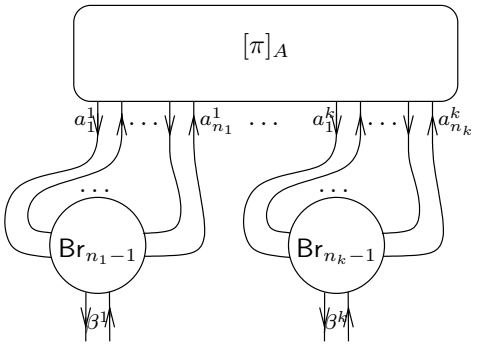
The order on the set of labels of this net is defined as in the case above of an input prefix: l is taken as new unique least element.

3.2.2 Interpreting closures

Remember that a closure is a pair $c = (\pi, e)$ where π is a process and e is a finite function from $\mathcal{N}$ to $\mathcal{N}$ whose domain contains all the free names of π. Given a finite set $\mathcal{B}$ of names which contains the codomain of the function e, we define the net structure $[c]_{\mathcal{B}}$ which has two free ports β^+ and β^- for each $\beta \in \mathcal{B}$ (these are the only free ports of this simple net structure).

This net structure $[c]_{\mathcal{B}}$ is built as follows. Let A be the domain of e. For each element β of $\mathcal{B}$, let $a_1, \ldots, a_n$ be the elements of A which are mapped to β by e, we introduce a Br_{n-1} broadcast area with one pair of free ports connected to (β^+, β^-), and the n other pairs connected to the pairs of free ports (a_i^+, a_i^-) of the net structure $[\pi]_A$.

Pictorially, this construction can be represented as



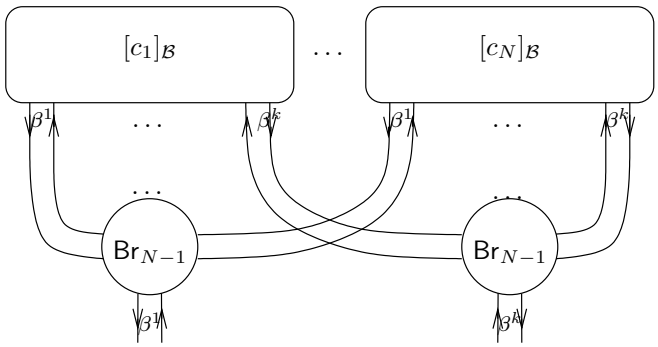
where $\beta_1, \ldots, \beta_k$ are the elements of $\mathcal{B}$ and, for each $j = 1, \ldots, k$, $a^j_1, \ldots, a^j_{n_j}$ are the elements of A which are mapped to β_j by e.

The poset of labels of this net structure is that of $[\pi]_A$.

3.2.3 Interpreting soups

Remember that a soup is a finite multiset of closures, $S = c_1 \ldots c_N$. Given a finite subset $\mathcal{B}$ of $\mathcal{N}$ which contains the codomain of all the environments of the closures $c_1, \ldots, c_N$, we define a simple net structure $[S]_{\mathcal{B}}$ which has two free ports β^+ and β^- for each $\beta \in \mathcal{B}$ (these are the only free ports of this simple net structure).

The simple nets $[c_1]_{\mathcal{B}}$,...,$[c_N]_{\mathcal{B}}$ are combined as in an N-ary parallel composition of processes, using again broadcast areas. Pictorially, denoting by β_1,...,β_k the elements of $\mathcal{B}$, this gives



3.2.4 Interpreting states

Let $(S, \mathcal{P})$ be a state and let $\mathcal{A}$ be a finite set of names containing all the free names of $(S, \mathcal{P})$ (we assume that $\mathcal{A} \cap \mathcal{P} = \emptyset$, if this is not the case, perform first an α-conversion on $(S, \mathcal{P})$). Let $\mathcal{B} = \mathcal{A} \cup \mathcal{P}$.

Then the interpretation $[S, \mathcal{P}]_{\mathcal{A}}$ of $(S, \mathcal{P})$ is obtained by plugging Br_{-1} broadcast areas on the pair of free ports of $[S]_{\mathcal{B}}$ corresponding to the elements of $\mathcal{P}$.

Lemma 26 *Let $(S, \mathcal{P})$ and $(S', \mathcal{P}')$ be states and let $\mathcal{A}$ be a set of names containing all the free names of $(S, \mathcal{P})$. If $(S, \mathcal{P}) \rightsquigarrow_{\mathsf{can}} (S', \mathcal{P}')$ then $[S, \mathcal{P}]_{\mathcal{A}} \rightsquigarrow^*_{\mathrm{Can}} [S', \mathcal{P}']_{\mathcal{A}}$ (actually, one needs to use only $\rightsquigarrow_{\mathrm{s}}$ and $\rightsquigarrow_{\mathrm{n}}$ reductions).*

In particular, if $(S', \mathcal{P}')$ is the canonical form of $(S, \mathcal{P})$, one has $\mathsf{Com}([S, \mathcal{P}]_{\mathcal{A}}) = \mathsf{Com}([S', \mathcal{P}']_{\mathcal{A}})$.

The proof is straightforward.

3.3 Activity of the net structure interpreting a state

As already mentioned, when translating processes, one obtains net structures which contain switching cycles, but fortunately, these net structures are active.

Lemma 27 *Let $(S, \mathcal{P})$ be a state and let $\mathcal{A}$ be a finite set of names containing all the free names of $(S, \mathcal{P})$. Then the finite net structure $[S, \mathcal{P}]_{\mathcal{A}}$ is active.*

Proof. We give just a sketch of the proof, because the result is quite clear, but the details are rather tedious.

One observes first that, in a broadcast area Br_n with pairs of free ports (p^+_1, p^-_1),..., (p^+_{n+2}, p^-_{n+2}), there is no switching path between p^+_i and p^-_i, for any $i = 1, \ldots, n+2$.

Then one shows by induction on the process π that for any free ports p and q of the finite net structure $[\pi]_A$ interpreting π, any switching path from p to q in $[\pi]_A$ passes through a codereliction or a dereliction cell (the most interesting case is of course the situation where π starts with an output prefix).

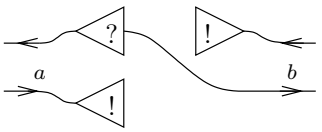
Last, by the first observation, any switching cycle of $[S, \mathcal{P}]_{\mathcal{A}}$ must pass through the interpretation of a process of the soup S, and therefore must contain a codereliction or dereliction cell. □

3.4 Examples of process interpretations

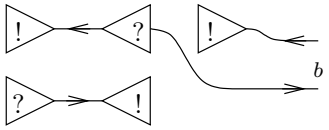
Before starting to explore the operational properties of this interpretation, we have to study a few examples in order to get some intuition about it.

3.4.1 Solos

As a first example, consider the "emitting solo[18]" $\overline{a}\langle b\rangle \cdot *$, also written $\overline{a}\langle b\rangle$.

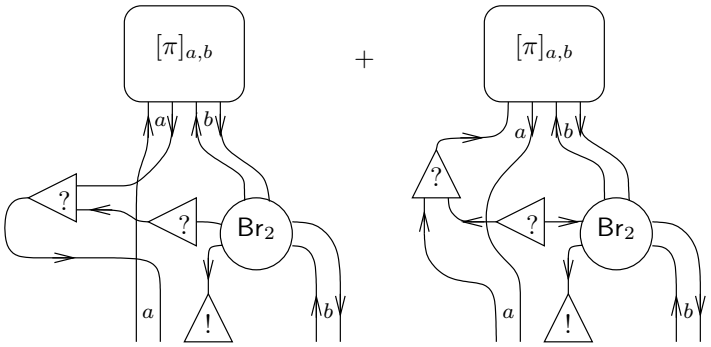


The process $\nu a \cdot \overline{a}\langle b\rangle$ is equal to



and therefore reduces to 0. This "0" value can be seen as a kind of error as it means that the emission action that the prefix wants to perform is impossible, because the name a has no access to the outer world, because of the ν binding.

Let π be a process (for simplifying the notations, assume that it has no other free names but a and b). Then the net $[\overline{a}\langle b\rangle \mid \pi]_{a,b}$ reduces to the following sum:

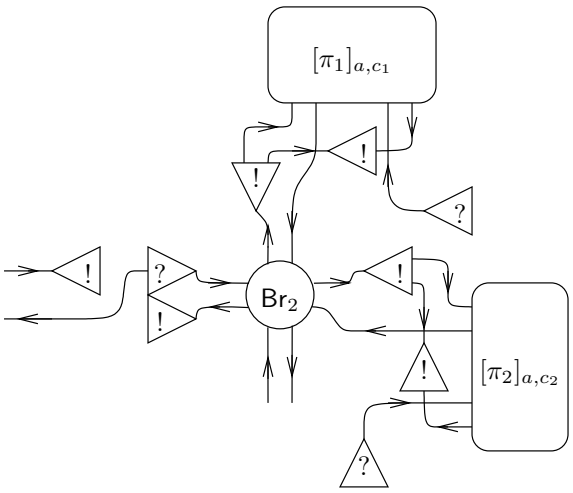


where the first term is identical to $[\overline{a}\langle b\rangle \cdot \pi]_{a,b}$ and the second term corresponds to the possibility that π perform a reception of name b on channel a.

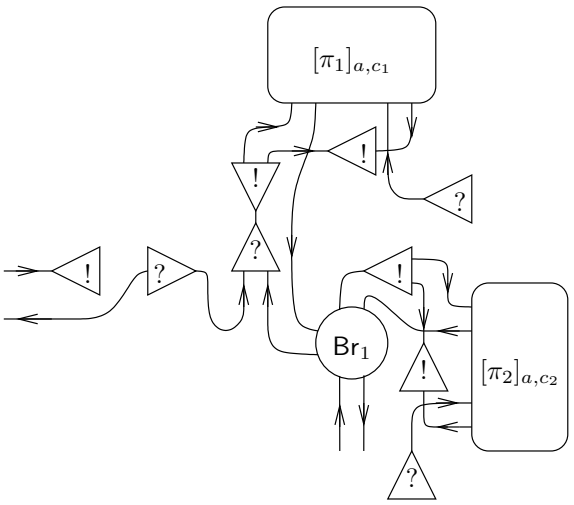
3.4.2 Competition

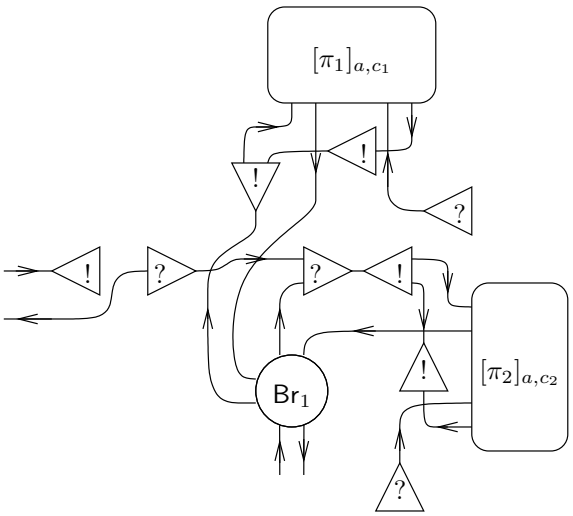
Let us now consider a situation where we have a competition between two emitters for one receiver on channel a, as in the following process: $\pi = \overline{a}\langle b\rangle \mid a(c_1) \cdot \pi_1 \mid a(c_2) \cdot \pi_2$. The processes π_1 and π_2 can have other free names than the ones mentioned above, but we don't mention them (the corresponding wires are simply connected to broadcast areas as explained above).

[18] Actually, the interpretation of the calculi of solos will use a more sophisticated interpretation of solos.

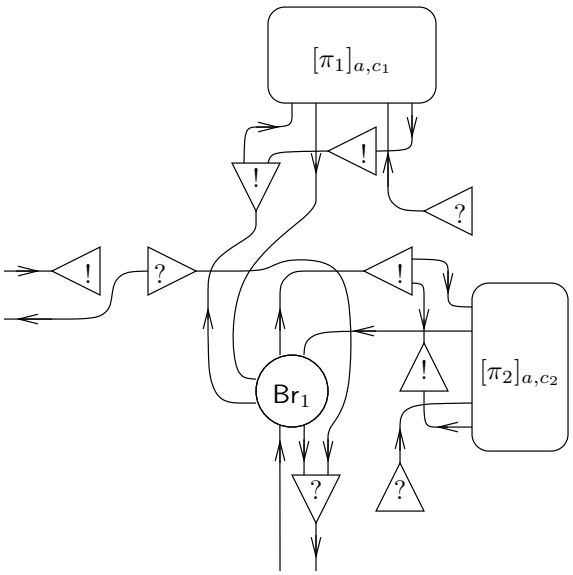


Upon applying the prefix/broadcast interaction principle of Section 3.1.4, one obtains a sum made of the following three nets. The two first nets are symmetric of each other:



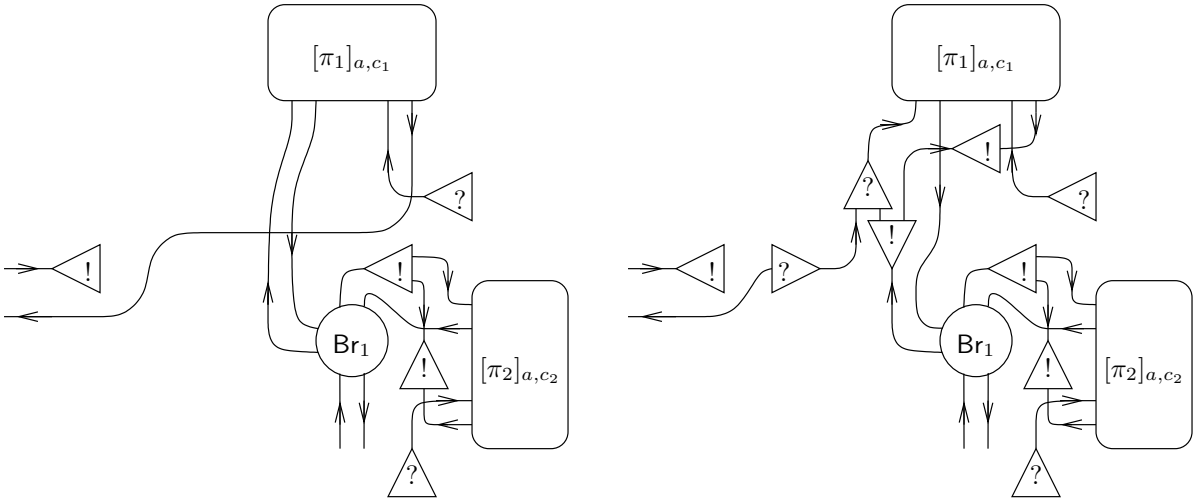


and the last one is



This latter net corresponds to the situation where the output solo $\overline{a}\langle b\rangle$ will communicate with the outer world, the two first nets correspond to the case where the prefix communicate with the two input prefixed processes.

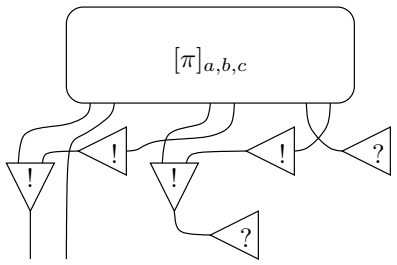
Let us consider the first simple net structure. It reduces to the following sum of two simple net structures.



The first net structure corresponds to the reduction where the name b is passed to the process π_1 as c_1, whereas the second net corresponds to the passing of the name b to the process π_1 through channel a; such reduction could not occur in the standard π-calculus, unless the external input prefix of $a(c_1) \cdot \pi_1$ were reduced by reception of another name through channel a. Such synchonization is not required by differential interaction nets, but we shall see how it is implemented by the order relation with which we have endowed the label sets of processes.

3.4.3 Non localized process

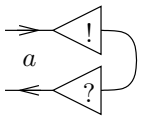
Next, consider the process $a(b) \cdot b(c) \cdot \pi$ where the object of the outermost input prefix is the subject of the innermost input prefix. Here is the interpretation of this process (again we do not mention the other names, but there can be some).



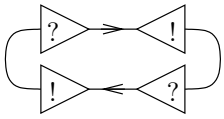
So this process reduces to 0, due to the interaction between codereliction and weakening. This explains why we shall restrict to the localized π-calculus.

3.4.4 Cyclic process

The interpretation of the process $a(b) \cdot \overline{a}\langle b\rangle \cdot *$ reduces (using only structural reduction rules) to



and so the interpretation of $\nu a \cdot (a(b) \cdot \overline{a}\langle b\rangle \cdot * \mid a(b) \cdot \overline{a}\langle b\rangle \cdot *)$ reduces (using only structural reduction rules) to



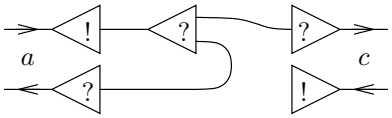
which, by reducing the two communication redexes, leads to the following loop:

3.4.5 Non-terminating cyclic process

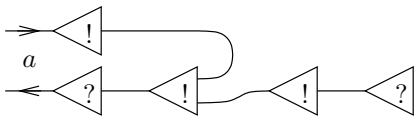
The above process was cyclic but had nevertheless a finite reduction. But some processes can lead to cyclic net structures which have a non-terminating reduction. Consider the process

$$\pi = \nu a \cdot (a(b) \cdot \overline{a}\langle b\rangle \cdot \overline{c}\langle b\rangle \cdot * \mid a(b) \cdot \overline{a}\langle b\rangle \cdot b(d) \cdot *)\,.$$

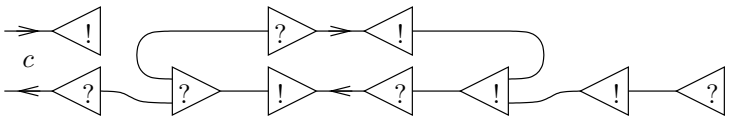
The sub-process $a(b) \cdot \overline{a}\langle b\rangle \cdot \overline{c}\langle b\rangle \cdot *$ translates to (up to structural reductions)



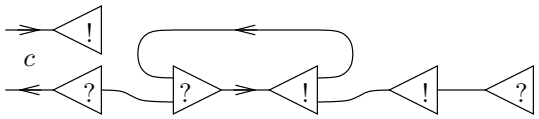
and the sub-process $a(b) \cdot \overline{a}\langle b\rangle \cdot b(d) \cdot *$ translates to



and therefore, π translates to



which, after reducing the two communication redexes, leads to the following net structure.

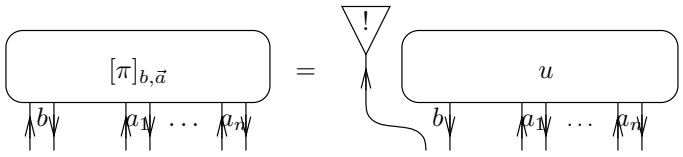


Reducing the cocontraction/contraction redex in that structure leads to a larger net structure which contains again a cocontraction/contraction redex, as easily checked.

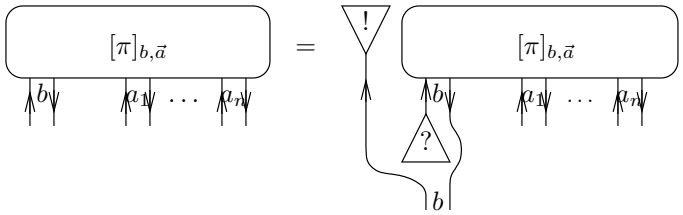
3.5 The localized π-calculus

A π-term is *localized* if for any sub-term of π of the shape $a(b) \cdot \rho$, the name b does not occur in ρ as the subject of an input prefix. See [SW01] for more informations on this concept.

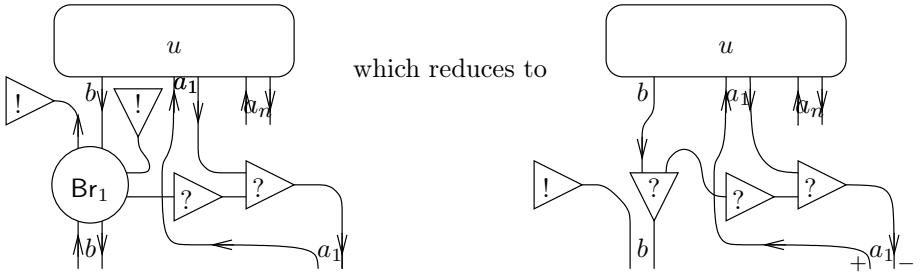
Lemma 28 *Let $b, \vec{a}$ be a repetition-free sequence of names containing all the free names of a process π. If the name b does not occur in the process π as the subject of an input prefix, then there is a simple net u with free ports $b^-, {a_1}^+, {a_1}^-, \ldots, {a_n}^+, {a_n}^-$, such that $[\pi]_{b,\vec{a}}$ satisfies (up to $\rightsquigarrow_{\mathrm{s}}$ and $\rightsquigarrow_{\mathrm{n}}$ reductions)*



and hence (still up to $\rightsquigarrow_{\mathrm{s}}$ and $\rightsquigarrow_{\mathrm{n}}$ reductions)



Proof. Simple induction on the structure of π, the most interesting case being the one where $\pi = \overline{a_i}\langle b \rangle \cdot \rho$. Without loss of generality, we assume $i = 1$. Applying the inductive hypothesis, $[\pi]_{b,\vec{a}}$ can be written



by $\rightsquigarrow_{\mathrm{s}}$ and $\rightsquigarrow_{\mathrm{n}}$ reductions. The right hand net has the required shape. The other cases are simpler and left to the reader. □

4 Comparing the reductions

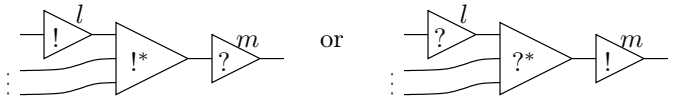
We define a map Φ from the vertices of the LTS $\mathbb{S}_{\mathcal{L}}$ to the vertices of the LTS $\mathbb{D}_{\mathcal{L}}$:

$$\Phi(S, \mathcal{P}, \mathcal{A}) = \mathsf{Com}([S, \mathcal{P}]_{\mathcal{A}})$$

which is well defined by Theorem 22 since we know that $[S, \mathcal{P}]_{\mathcal{A}}$ is an active labelled net structure by Lemma 27.

4.1 Persistency of prefixes

Lemma 29 *Let u be an active simple net structure which contains a subnet of the shape*



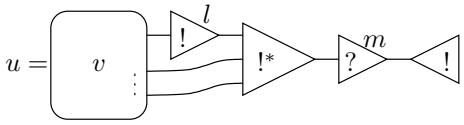
We call (l, m)-garded input prefix *any net of the first kind and* (l, m)-garded output prefix *any net structure of the second kind.*

Then any element of $\mathsf{Com}(u)$ *contains an* (l, m)*-garded input prefix in the first case and* (l, m)*-garded output prefix in the second case.*

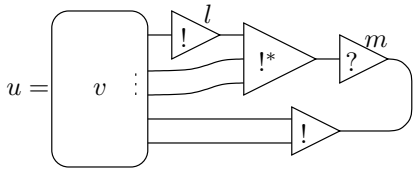
Proof. The hypothesis that u is active is used only for ensuring that $\mathsf{Com}(u)$ is well defined.

We consider the first subnet (call it u_0), the other one being dual. It suffices to show that, if u is a pure net structure which contains u_0 as a subnet, and if $u \rightsquigarrow_{\mathrm{snd}} u'$ or $u \rightsquigarrow_{\mathrm{n}} u'$, then any element of u' contains an (l, m)-guarded input prefix as a subnet. The case of a $\rightsquigarrow_{\mathrm{n}}$ reduction is straightforward.

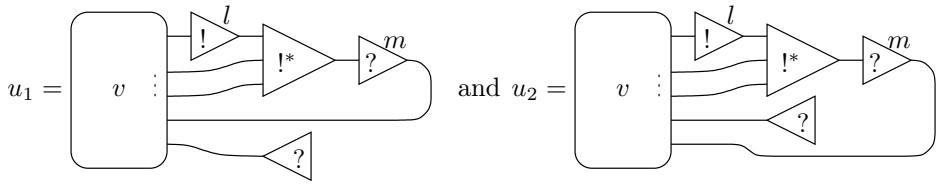
So choose a structural or non-deterministic redex in u and let u' be the net obtained by reducing this redex. If this redex does not contain the dereliction cell labelled by m, each element of u' will clearly contain the subnet under consideration. Otherwise, we are in one of the two following situations: either



and then $\mathsf{Com}(u) = \emptyset$ and there is nothing to say, or



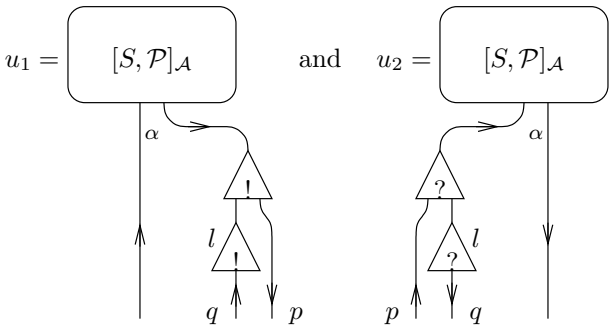
and in that case, $\mathsf{Com}(u) = \mathsf{Com}(u_1) \cup \mathsf{Com}(u_2)$ where



but both u_1 and u_2 contain the subnet under consideration. □

From this lemma, we deduce the following property which expresses that a prefix which enters a process will never exit from this process, if we only perform structural and non-determininstic reductions.

Lemma 30 *Let $(S, \mathcal{P})$ be a state and let $\mathcal{A}$ be a set of names containing all the free names of $(S, \mathcal{P})$. Let $\alpha \in \mathcal{A}$ and let $l \in \mathcal{L}$ which does not occur in S. Let u_1 and u_2 be the following net structures (where we only mention the ports associated to the name α)*



Then any element of $\mathsf{Com}(u_1)$ contains a communication redex where the codereliction cell is labelled by l and the dereliction cell is labelled by some $m \in \mathcal{L}(S)$, or an (l, m)-guarded input prefix, with $m \in \mathcal{L}(S)$. Similarly, any element of $\mathsf{Com}(u_2)$ contains a communication redex where the dereliction cell is labelled by l and the codereliction cell is labelled by some $m \in \mathcal{L}(S)$, or an (l, m)-guarded output prefix with $m \in \mathcal{L}(S)$.

Proof. In view of the definition of the interpretation of a state $(S, \mathcal{P})$ in Section 3.2.4, we are reduced to proving the same statement for the interpretation of a soup.

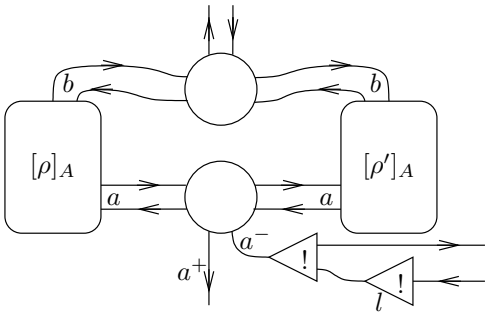
Using the definition of the interpretation of a soup in Section 3.2.3 as well as the prefix/broadcast reduction of Section 3.1.4, we are reduced to proving the same statement for the interpretation of a closure. We use the fact that this prefix/broadcast reduction uses only $\rightsquigarrow_{\mathrm{Can}}$ steps, and we also use Theorem 23 for arguing that these reduction steps can be performed immediately.

By the same argument, we are reduced to proving the statements of the theorem for the interpretation of a process $[\pi]_A$ of a process π (we use a instead of α and A instead of $\mathcal{A}$ to be consistent with our notational conventions).

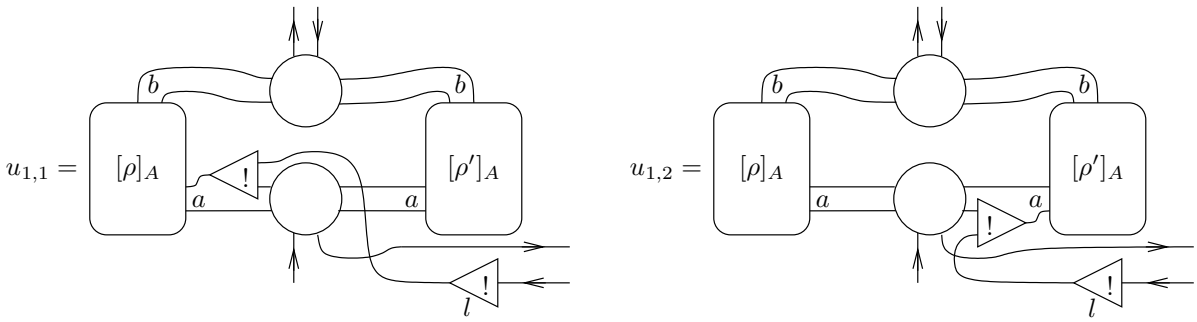
We proceed by induction on π.

Case 1. Assume first that $\pi = *$. Then $\mathsf{Com}(u_1) = \mathsf{Com}(u_2) = \emptyset$ and there is nothing to prove.

Case 2. Assume next that $\pi = \rho \mid \rho'$. Then, mentioning the free ports associated to a and to a generic element b of A, u_1 has the following structure, where we have made explicit the label l of the codereliction cell of the input prefix under consideration:



Then, applying the prefix/broadcast equation of section 3.1.4, which uses only structural and non-deterministic reductions, we get the net structure $\{u_{1,1}, u_{1,2}\}$ where



and we have $\mathsf{Com}(u_1) = \mathsf{Com}(u_{1,1}) \cup \mathsf{Com}(u_{1,2})$.

Let us consider the simple net structure $u_{1,1}$. In this net structure, consider the subnet structure v which consists of $[\rho]_A$ together with the input prefix attached to its output port a^-, whose codereliction cell bears label l. This net structure v is active since $[\rho]_A$ is active. Let $\{v_1, \ldots, v_p\}$ be $\mathsf{Com}(v)$ and let $u^j_{1,1}$ be the net structure $u_{1,1}$ where the subnet structure v has been replaced by v_j (for $j = 1, \ldots, p$). By Theorem 23, we know that

$$\mathsf{Com}(u_{1,1}) = \bigcup_{j=1}^{p} \mathsf{Com}(u^j_{1,1})\,.$$

By inductive hypothesis, for each $j = 1, \ldots, p$, two situations may occur.

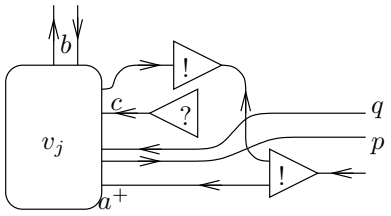
- Either v_j contains a communication redex whose codereliction cell is labelled by l, and in that case, we know that this redex will be present in each element of $\mathsf{Com}(u^j_{1,1})$ since the structural and non-deterministic reduction does not affect such redexes.
- Or v_j contains an (l, m)-garded input prefix and then, by Lemma 29, this guarded input prefix will still appear in each element of $\mathsf{Com}(u^j_{1,1})$.

The same holds of course for $u_{1,2}$ by symmetry. The statement of the lemma concerning u_2 (still in the case where $\pi = \rho \mid \rho'$) is proven similarly, using the equation of section 3.1.4 in the case of an output prefix now.

The case $\pi = \nu b \cdot \rho$ is similar and simpler.

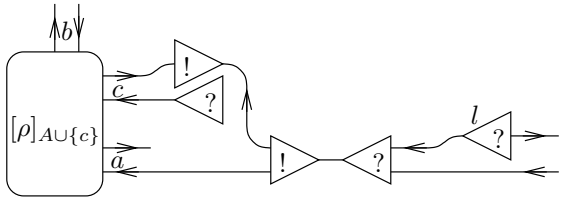
Now we must consider the situations where π starts with an input or output prefix. If none of the names involved by these prefixes is equal to a, the inductive hypothesis applies simply (with the help of Lemma 29). So the remaining situations we must consider are the following ones.

Case 3: $\pi = a(c) \cdot \rho$. Let us first consider u_1. In this simple net structure, the input prefix whose dereliction cell is labelled by l is directly connected to the port a^- of $[\rho]_{A \cup \{c\}}$. Consider the subnet of u_1 which consists of $[\rho]_{A \cup \{c\}}$ together with the input prefix whose dereliction cell is labelled by l connected to its port a^-. Let $\{v_1, \ldots, v_p\}$ be $\mathsf{Com}(v)$ and let u^j_2 be the net u_2 where the subnet v has been replaced by v_j (for $j = 1, \ldots, p$), so that u^j_1 is the following simple net

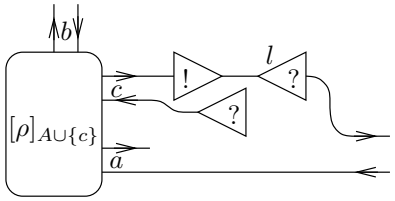


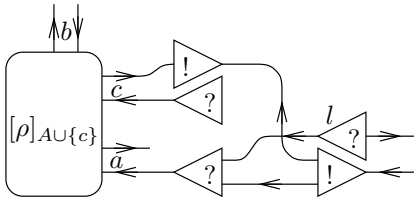
where we have specified the "free ports" p and q of the input prefix, as in the statement of the lemma we are proving. We conclude as before, since by inductive hypothesis $\mathsf{Com}(v_j)$ contains a communication redex involving l or a guarded input prefix whose codereliction cell bears the label l. This will still be the case of $\mathsf{Com}(u^j_1)$ (apply Lemma 29).

Now let us consider u_2, which is



Applying the prefix/prefix reduction of section 3.1.5, which uses only $\rightsquigarrow_{\mathrm{Can}}$ reductions, we get the net structure $\{u_{2,1}, u_{2,2}\}$ where

$u_{2,1} =$ 

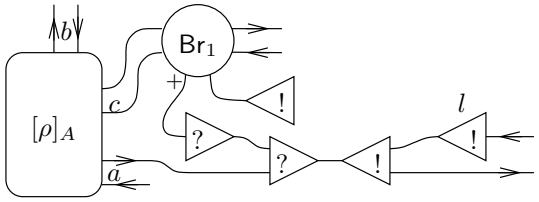
$u_{2,2} =$ 

and therefore $\mathsf{Com}(u_2) = \mathsf{Com}(u_{2,1}) \cup \mathsf{Com}(u_{2,2})$. But $u_{2,1}$ contains a communication redex involving l and this redex will not be affected by the structural and non-deterministic reduction, so each element of $\mathsf{Com}(u_{2,1})$ will contain the same redex. On the other hand, let v be the subnet of $u_{2,2}$ which consists of $[\rho]_{A\cup\{c\}}$ together with the output prefix whose dereliction cell is labelled by l connected to the port a^+. Let $\{v_1, \ldots, v_p\}$ be $\mathsf{Com}(v)$ and let $u^j_{2,2}$ be the net structure $u_{2,2}$ where the subnet v has been replaced by v_j (for $j = 1, \ldots, p$). By inductive hypothesis, either v_j contains a communication redex whose dereliction cell is labelled by l, and this redex will still be present in each element of $\mathsf{Com}(u^j_{2,2})$, or it will contain a guarded output prefix whose dereliction cell is labelled by l, and this guarded output prefix will still be present in each element of $\mathsf{Com}(u^j_{2,2})$ by Lemma 29. We conclude for this case because

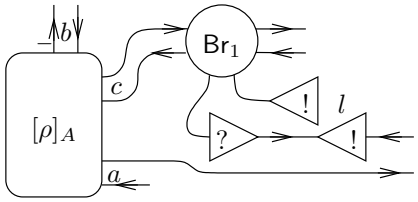
$$\mathsf{Com}(u_{2,2}) = \bigcup_{j=1}^{p} \mathsf{Com}(u^j_{2,2})$$

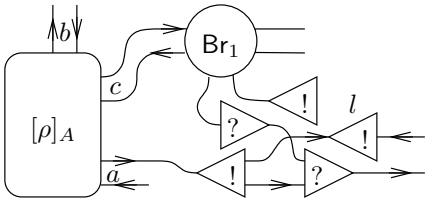
by Theorem 23.

Case 4: $\pi = \overline{a}\langle c\rangle \cdot \rho$. Observe that we have $c \in A$ since the name c is free in π. In that case, u_2 is handeled exactly as u_1 in the previous case, so let us consider only u_1, which is



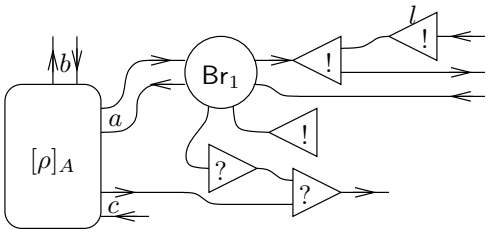
By $\rightsquigarrow_{\mathrm{Can}}$ reductions, this simple net structure reduces to $\{u_{1,1}, u_{1,2}\}$ where

$u_{1,1} =$ 

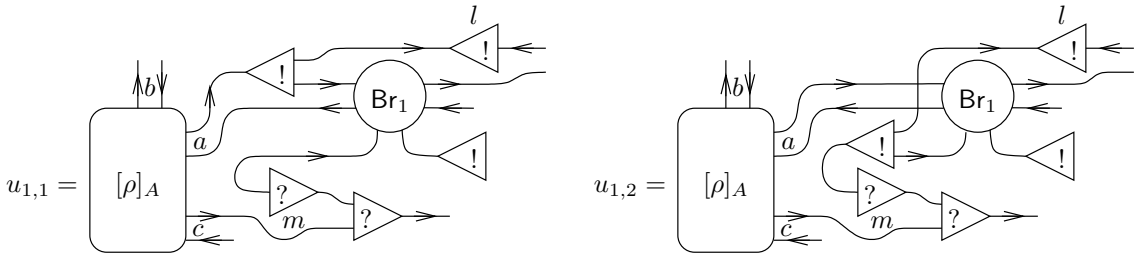
$u_{1,2} =$ 

The first of these simple net structures contains a communication redex whose codereliction cell is labelled by l, and this redex will still be present in $\mathsf{Com}(u_{1,1})$. As to the second of these nets, we first apply the inductive hypothesis to the subnet consisting of $[\rho]_A$ together with the input prefix whose codereliction cell is labelled by l connected to the port a^-, and then apply Lemma 29 as in the previous case.

Case 5: $\pi = \overline{c}\langle a\rangle \cdot \rho$. In that case, u_1 is the following simple net.

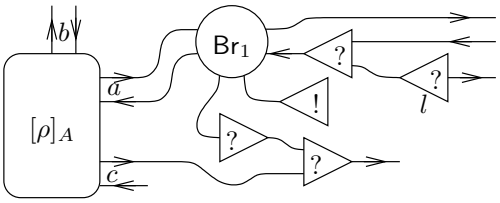


Applying the prefix/broadcast reduction of Section 3.1.4, which uses only $\rightsquigarrow_{\mathrm{Can}}$ reductions, we get the net structure $\{u_{1,1}, u_{1,2}\}$ where

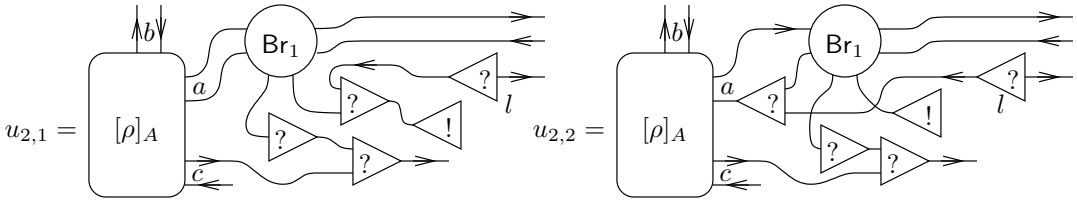


Concerning $u_{1,1}$, we can proceed as before, applying the inductive hypothesis to $[\rho]_A$ combined with the input prefix under consideration, and then Lemma 29. As to $u_{1,2}$, we see that it contains a guarded input prefix whose codereliction cell is labelled by l and whose dereliction cell is labelled by some $m \in \mathcal{L}(\pi)$. By Lemma 29, this guarded input prefix will still be present in all the elements of $\mathsf{Com}(u_{1,2})$ (we do not need the lemma in the present situation, indeed).

Last, we consider u_2, which is the following simple net structure.



Applying again the prefix/broadcast reduction of Section 3.1.4, which uses only $\rightsquigarrow_{\mathrm{Can}}$ reductions, we get the net structure $\{u_{2,1}, u_{2,2}\}$ where



We conclude the proof of this lemma by observing that $\mathsf{Com}(u_{2,1}) = \emptyset$ and by applying the same reasoning as before to $u_{2,2}$. □

4.2 A simulation theorem

Theorem 31 *Let $(S, \mathcal{P}, \mathcal{A}) \in \mathbb{S}_{\mathcal{L}}$ (so that $(S, \mathcal{P})$ is a canonical state) and l, m be two labels.*

1. *Let $(T, \mathcal{Q}, \mathcal{B}) \in \mathbb{S}_{\mathcal{L}}$, and assume that $(S, \mathcal{P}, \mathcal{A}) \rightarrow^{l/\overline{m}} (T, \mathcal{Q}, \mathcal{B})$ (so that $\mathcal{B} = \mathcal{A}$). Then $\Phi(S, \mathcal{P}, \mathcal{A}) \rightarrow^{l/\overline{m}} \Phi(T, \mathcal{Q}, \mathcal{B})$.*

2. *Let t be a net structure in PCNF and assume that $\Phi(S, \mathcal{P}, \mathcal{A}) \rightarrow^{l/\overline{m}} t$. Then there is a canonical state $(T, \mathcal{Q})$ such that $\Phi(T, \mathcal{Q}, \mathcal{A}) = t$ and $(S, \mathcal{P}, \mathcal{A}) \rightarrow^{l/\overline{m}} (T, \mathcal{Q}, \mathcal{A})$.*

Proof.

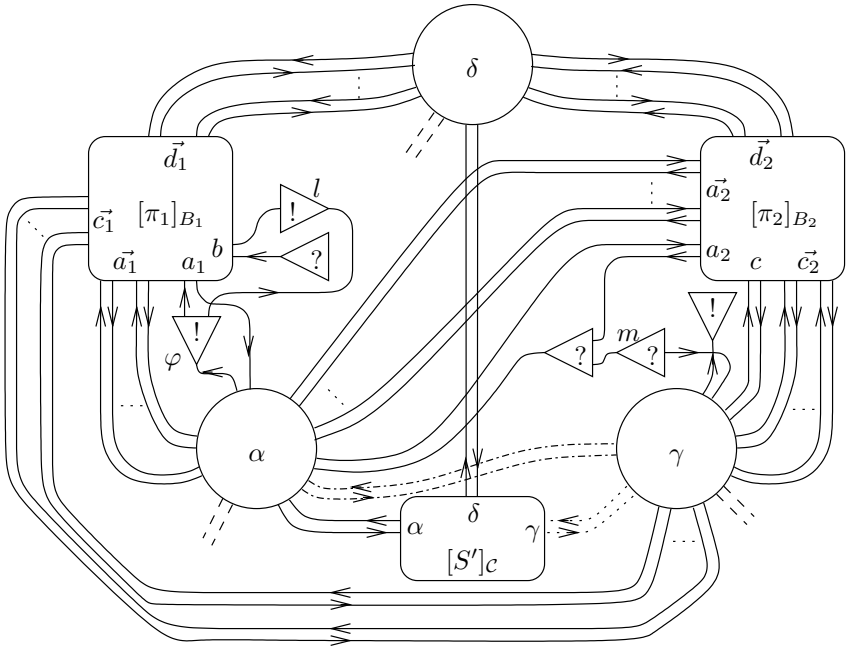
First statement. Since $(S, \mathcal{P}, \mathcal{A}) \rightarrow^{l/\overline{m}} (T, \mathcal{Q}, \mathcal{B})$, we have $\mathcal{A} = \mathcal{B}$ (remember that we assume that $\mathcal{A} \cap \mathcal{P} = \emptyset$ and $\mathcal{B} \cap \mathcal{Q} = \emptyset$, which is possible by α-conversion),

$$S = (a_1^l(b) \cdot \pi_1, e_1)(\overline{a_2^m}\langle c\rangle \cdot \pi_2, e_2)S' \tag{1}$$

with $e_1(a_1) = e_2(a_2) = \alpha \in \mathcal{N}$, for some processes π_1 and π_2 and some soup S', and with these notations, $(T, \mathcal{Q})$ is the canonical form of the state $(T', \mathcal{P})$ where

$$T' = (\pi_1, e_1[b \mapsto e_2(c)])(\pi_2, e_2)S'\,. \tag{2}$$

We set $\gamma = e_2(c)$. Notice that one could possibly have $\gamma = \alpha$.
Therefore, $\mathsf{Com}(S, \mathcal{P}, \mathcal{A})$ is the PCNF of the following net structure:



where we have adopted the following notations and conventions:

- B_1 is the domain of e_1 and B_2 is the domain if e_2;
- for $i = 1, 2$, $\vec{a_i}$ is a repetition-free list of all the elements of B_i different from a_i and mapped to α by e_i;
- $\vec{c_1}$ is a repetition-free list of all the elements of B_1 mapped to γ by e_1;
- $\vec{c_2}$ is a repetition-free list of all the elements of B_2 different from c and mapped to γ by e_2;
- δ is a generic element of $\mathcal{P} \cup \mathcal{A}$ and $\vec{d_i}$ is a repetition-free list of the elements of B_i mapped to δ by e_i;
- the broadcast areas introduced in the interpretation of closures and soups (see Section 3.2) are decorated with the corresponding names;

- $\mathcal{C} = \mathcal{P} \cup \mathcal{A}$ contains all the free names of the soup S';
- the dashed pairs of wires are absent if the corresponding names belong to $\mathcal{P}$ (and then these names are bound in the state), and present otherwise;
- if $\gamma \neq \alpha$, the dashed-dotted pair of wires is absent and the dotted line is present (in that case, α and γ correspond to two different pairs of free ports of $[S']_{\mathcal{C}}$) and conversely if $\alpha = \gamma$ (in that case, there is exactly one pair of free ports corresponding to $\gamma = \alpha$ in $[S']_{\mathcal{C}}$). If $\alpha = \gamma$, the list $\vec{c_1}$ is empty, as well as the list $\vec{c_2}$, but we must require that all the elements of $\vec{a_2}$ be different from c as well.

We can now apply the prefix/broadcast area interaction of Section 3.1.4, for instance to the input prefix whose codereliction cell is labelled by l.

Assume first that $\alpha \neq \gamma$, so that the dashed-dotted pair of wires is absent and the dotted pair of wires is present.

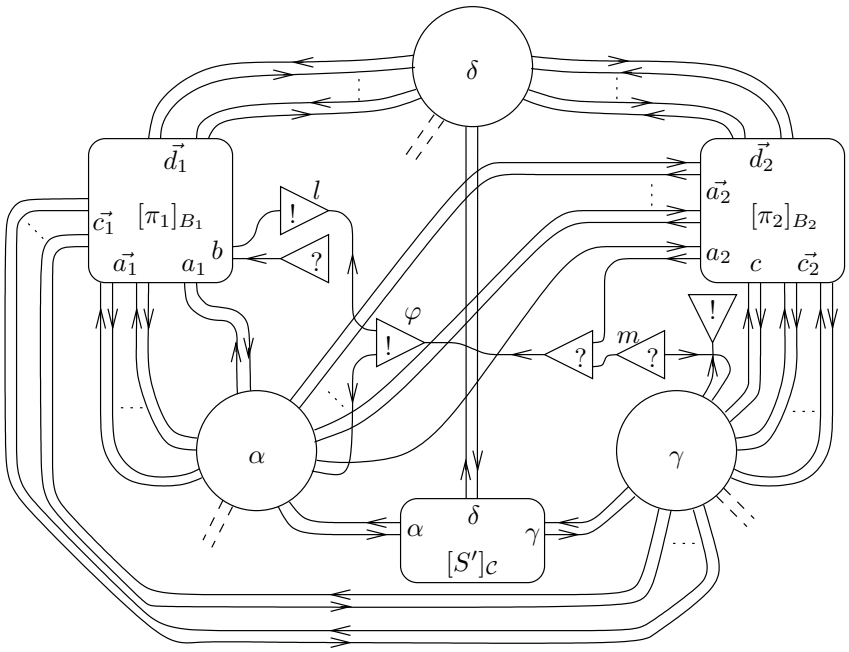
We obtain that $\Phi(S, \mathcal{P}, \mathcal{A})$ can be written as a union of finite net structures

$$\Phi(S, \mathcal{P}, \mathcal{A}) = \mathsf{Com}(v) \cup \bigcup_{i=1}^{N} \mathsf{Com}(u_i)$$

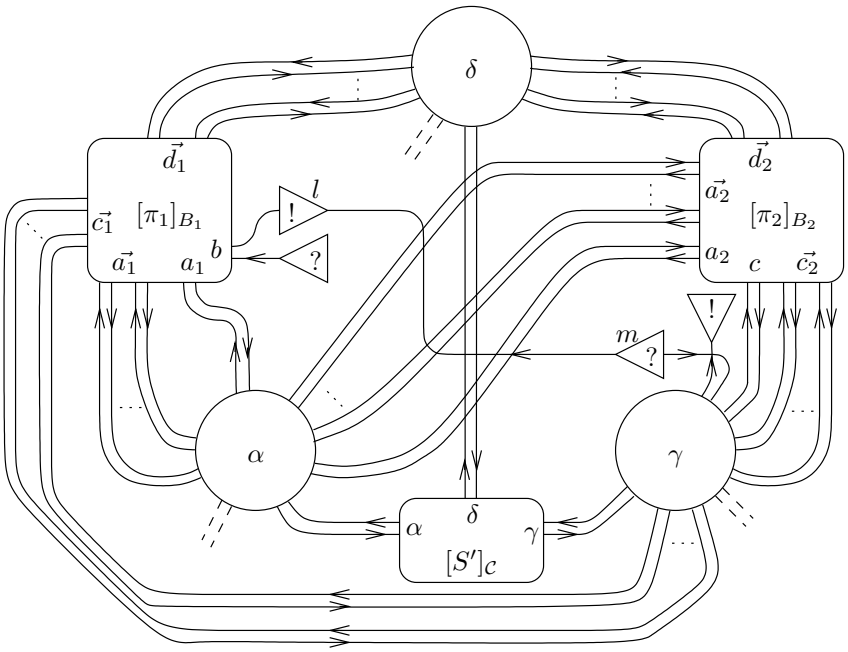
where, in each of the u_i's, the principal port of the input prefix consisting of the codereliction cell labelled by l and the cocontraction cell φ (that is, the principal port of φ) is connected to one of the free ports of $[\pi_1]_{B_1}$, $[\pi_2]_{B_2}$ or $[S']_{\mathcal{C}}$, or to one of the dashed outputs of α, if present. By Lemma 30, in each of the net structures $\mathsf{Com}(u_i)$,

- either the codereliction cell l is part of a communication redex whose label m' belongs to $\mathcal{L}(\pi_1)$, $\mathcal{L}(\pi_2)$ or $\mathcal{L}(S')$, and so $m' \neq m$,
- or the input prefix (l, φ) is guarded in u_i by some dereliction cell whose label m' belongs to $\mathcal{L}(\pi_1)$, $\mathcal{L}(\pi_2)$ or $\mathcal{L}(S')$.

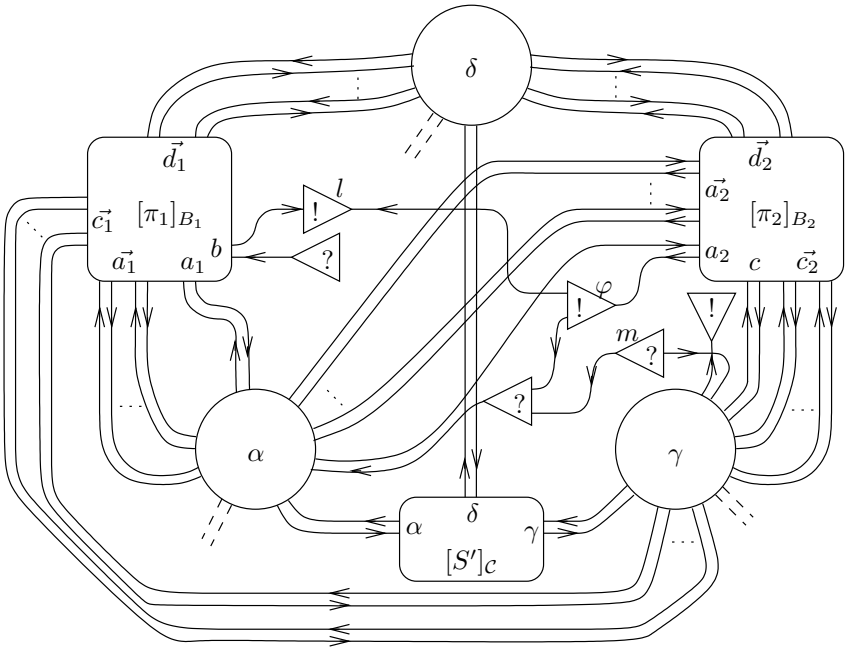
In both cases, it is clear that none of the elements of $\mathsf{Com}(u_i)$ can belong to $\Delta_{l,m}$. The net structure v is



and therefore, applying the prefix/prefix reduction (using only the $\rightsquigarrow_{\mathrm{Can}}$ reduction), we have $\mathsf{Com}(v) = \mathsf{Com}(v_1) \cup \mathsf{Com}(v_2)$ where v_1 is



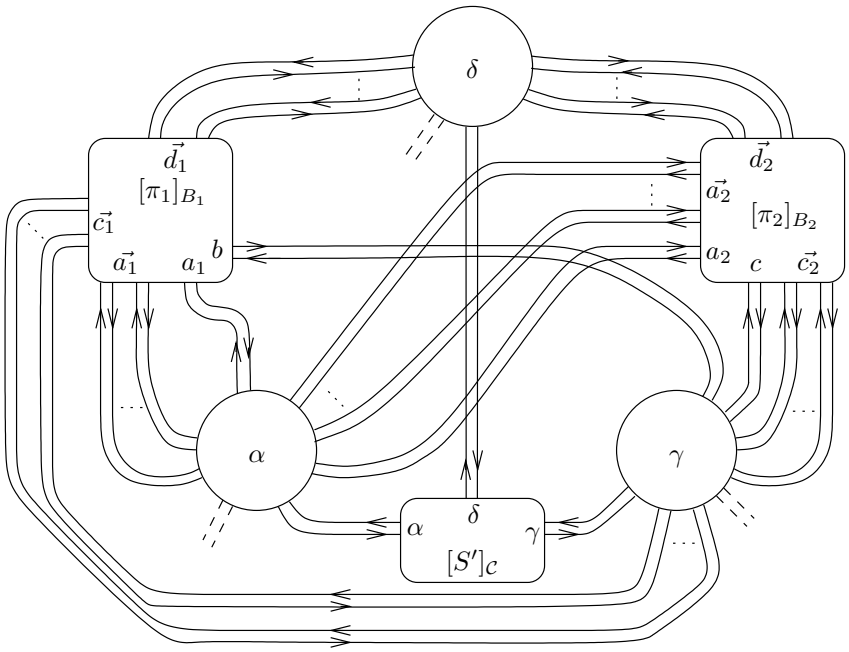
and v_2 is



None of the elements of $\mathsf{Com}(v_2)$ can belong to $\Delta_{l,m}$, by the same argument that we applied to the u_i's, and clearly $\mathsf{Com}(v_1) \subseteq \Delta_{l,m}$. So we have shown that

$$\Phi(S, \mathcal{P}, \mathcal{A}) \cap \Delta_{l,m} = \mathsf{Com}(v_1)\,.$$

But on the other hand, by Equation (2), the net structure $\Phi(T, \mathcal{Q}, \mathcal{B})$ is the PCNF of the following net structure

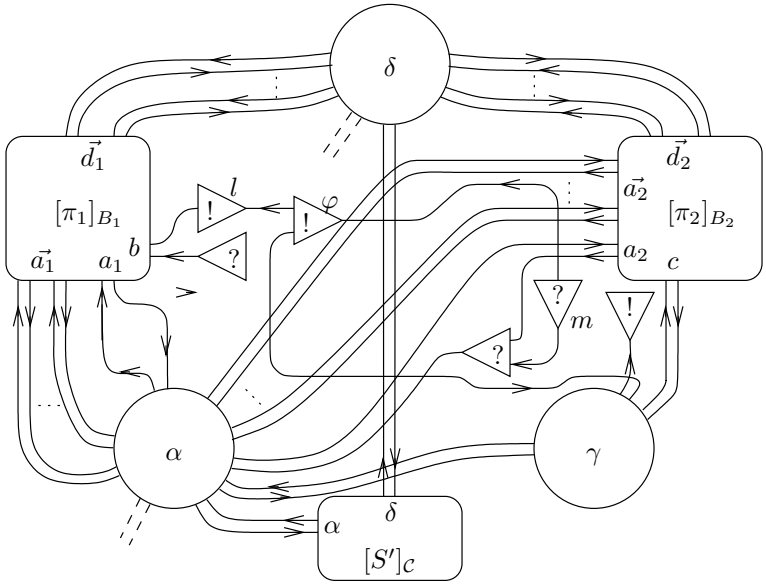


Remember that all the processes under consideration are assumed to be localized, and therefore, by Lemma 28, $\Phi(T, \mathcal{Q}, \mathcal{B})$ is equal to $\mathsf{Com}(v_1)$ (indeed, b cannot occur in π_1 as the subject of an input prefix). So we have a transition $\Phi(S, \mathcal{P}, \mathcal{A}) \rightarrow^{l/\overline{m}} \Phi(T, \mathcal{Q}, \mathcal{B})$ as announced.

Assume then that $\alpha = \gamma$, so that the dashed-dotted pair of wires is present and the dotted pair of wires is absent (there is exactly one pair of free ports of $[S']_{\mathcal{P}}$ corresponding to $\alpha = \gamma$). The reasoning is essentially the same as above. The main difference is that now

$$\Phi(S, \mathcal{P}, \mathcal{A}) = \mathsf{Com}(v) \cup \mathsf{Com}(w) \cup \bigcup_{i=1}^{N} \mathsf{Com}(u_i)$$

where v and the u_i's are similar to the homonymous net structures in the case $\gamma \neq \alpha$, and w is the following net structure



In w, the cells labelled by l and m constitute, together with the cocontraction cell φ, a guarded input prefix which, by Lemma 29, will be present is each of the elements of $\mathsf{Com}(w)$. Therefore, $\mathsf{Com}(w) \cap \Delta_{l,m} = \emptyset$ and we conclude as in the case $\gamma \neq \alpha$.

Second statement. Using the same notations as above, l and m must be the labels in $[S, \mathcal{P}]_{\mathcal{A}}$ of a codereliction cell and of a dereliction cell respectively and moreover l and m must be minimal in the order relation on $\mathcal{L}([S, \mathcal{P}]_{\mathcal{A}})$. An inspection of the definition of this poset shows that S must be of the shape given by Equation (1) and then the first statement of the theorem provides a proof of the second statement for the canonical form $(T, \mathcal{Q})$ of $(T', \mathcal{P})$ where T' is given by Equation (2), because the LTS $\mathbb{D}_{\mathcal{L}}$ is deterministic. □

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 17th ACM Symposium on Principles of Programming Languages (POPL)*, pages 81–94. ACM Press, January 1990.

[BE01] Antonio Bucciarelli and Thomas Ehrhard. On phase semantics and denotational semantics: the exponentials. *Annals of Pure and Applied Logic*, 109(3):205–241, 2001.

[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.

[BvdW95] Gianluigi Bellin and Jacques van de Wiele. Subnets of proofnets in MLL. In Jean-Yves Girard, Yves Lafont, and Laurent Regnier, editors, *Advances in Linear Logic*, volume 222 of *London Mathematical Society Lecture Note Series*, pages 249–270. Cambridge University Press, 1995.

[CDCV80] M. Coppo, M. Dezani-Ciancaglini, and B. Venneri. Principal type schemes and lambda-calculus semantics. In J. R. Hindley and J. P. Seldin, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda-Calculus and Formalism*. Academic Press, 1980.

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

[DR99] Vincent Danos and Laurent Regnier. Reversible, irreversible and optimal lambda-machines. *Theoretical Computer Science*, 227(1-2):273–291, 1999.

[Ehr93] Thomas Ehrhard. Hypercoherences: a strongly stable model of linear logic. *Mathematical Structures in Computer Science*, 3:365–385, 1993.

[Ehr02] Thomas Ehrhard. On Köthe sequence spaces and linear logic. *Mathematical Structures in Computer Science*, 12:579–623, 2002.

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

[EL06] Thomas Ehrhard and Olivier Laurent. Embedding the finitary pi-calculus in differential interaction nets. In *Proceedings of the Higher Order Rewriting workshop (HOR 2006)*, 2006. Electronic publication.

[ER04] Thomas Ehrhard and Laurent Regnier. Differential interaction nets. In *Proceedings of WoLLIC'04*, volume 103 of *Electronic Notes in Theoretical Computer Science*, pages 35–74. Elsevier Science, 2004.

[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.

[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.

[Gir96] Jean-Yves Girard. Proof-nets: The parallel syntax for proof-theory. In P. Agliano and A. Ursini, editors, *Logic and Algebra*, pages 97–124. Marcel Dekker, New York, 1996.

[Gir99] Jean-Yves Girard. Coherent Banach spaces: a continuous denotational semantics. *Theoretical Computer Science*, 227:275–297, 1999.

[Gir00] Jean-Yves Girard. On the meaning of logical rules II: multiplicative/additive case. In F. L. Bauer and R. Steinbrüggen, editors, *Foundation of Secure Computation*, Amsterdam, 2000. IOS Press. NATO series F 175.

[Gir01] Jean-Yves Girard. Locus Solum. *Mathematical Structures in Computer Science*, 11(3):301–506, 2001.

[Gir04] Jean-Yves Girard. Between logic and quantic: a tract. In Thomas Ehrhard, Jean-Yves Girard, Paul Ruet, and Philip Scott, editors, *Linear Logic in Computer Science*, volume 316 of *London Mathematical Society Lecture Note Series*, pages 346–381. Cambridge University Press, 2004.

[Has02] Ryu Hasegawa. Two applications of analytic functors. *Theoretical Computer Science*, 272(1-2):113–175, 2002.

[HL06] Kohei Honda and Olivier Laurent. Processes and polarized proof-nets. Technical report, Preuves, Programmes et Systèmes, 2006.

[Hyv04] Pierre Hyvernat. Predicate transformers and linear logic: yet another denotational model. In Jerzy Marcinkowski and Andrzej Tarlecki, editors, *Proceedings of the 18th Annual Conference of the European Association for Computer Science Logic (CSL'04)*, volume 3210 of *Lecture Notes in Computer Science*, pages 115–129. Springer-Verlag, September 2004.

[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.

[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.

[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.

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

[vGH05] Rob van Glabbeek and Dominic Hughes. Proof nets for unit-free multiplicative additive linear logic. In *ACM transactions on computational logic*, pages 784–842. ACM Press, 2005.