Multiport Interaction Nets and Concurrency
(Extended Abstract)

Damiano Mazza

Institut de Mathématiques de Luminy
mazza@iml.univ-mrs.fr
http://iml.univ-mrs.fr/~mazza

Abstract. We consider an extension of Lafont's Interaction Nets, called Multiport Interaction Nets, and show that they are a model of concurrent computation by encoding the full π-calculus in them. We thus obtain a faithful graphical representation of the π-calculus in which every reduction step is decomposed in fully local graph-rewriting rules.

1 Introduction

Lafont's Interaction Nets [1] are a model of sequential computation inspired by proof-nets for Multiplicative Linear Logic that can be seen as distributed Turing machines: as in these latter, transitions are local, but may be performed in parallel. They have been explicitly designed to be strongly deterministic, so no truly concurrent behavior can be expressed within them.

In this paper, we consider a non-deterministic extension[1] of Interaction Nets, called *Multiport Interaction Nets*, and show that they are an expressive model of concurrent computation by encoding the full π-calculus in them.

A considerable number of graphical representations of the π-calculus (or other process calculi) can be found in the existing literature. Let us mention for example Milner's π-nets [2], Parrow's Interaction Diagrams [3], and Fu's Reaction Graphs [4]. All these approaches succeed in describing concurrent dynamics as graph rewriting, but the treatment of prefixing is not very natural (in π-nets and Reaction Graphs, some form of "guarded box" is used, while Interaction Diagrams use polyadicity to encode causal dependency), and they all need boxes to represent replication, so that duplication is seen as a synchronous, global operation. It must also be observed that none of the existing graphical representations is ever shown to cover the π-calculus in all of its features, including sums and match prefix.

More recently, Laneve et al. proposed Solo Diagrams [5] as a graphical presentation of the *solos calculus* [6]. They too use replication boxes, but show that these can be limited to certain configurations which ensure constant-time reductions, and thus locality.

[1] It would actually be fairer to put it the other way around: Lafont intentionally restricted to one principal port because this yields systems with very nice properties.

Much closer to the spirit of Interaction Nets, a nice graphical representation of (an extension of) the *fusion calculus* has been given by Beffara and Maurel [7], in which nevertheless replication must be accommodated using boxes. In view of our results, it does not seem unlikely that Multiport Interaction Nets can provide both an alternative, "box-less" graphical encoding for the solos calculus and a purely local version of Beffara and Maurel's Concurrent Nets.

It is worth mentioning the comparison between Interaction Nets and concurrent systems done by Yoshida [8], who found that, when seen from the point of view of Interaction Nets, the graphical representation for her concurrent combinators amounts more or less to allow *hyperwires* connecting cells, i.e., wires that link together more than two ports. This is also explicitly seen in Beffara and Maurel's Concurrent Nets. As a matter of fact, our approach "internalizes" these hyperconnections, extending Lafont's systems not with respect to the topology of the connections between cells but to the nature of the cells themselves.

Multiport Interaction Nets have already been considered by Vladimir Alexiev in his Ph.D. thesis [9] as one of several possible non-deterministic extensions of Interaction Nets; they are obtained from these latter by allowing cells to have more than one principal port.[2] Alexiev proved that this extension is as expressive as the "hyperwire" extension mentioned above, and defined in it a graphical encoding of the finite π-calculus, leaving open the problem of extending it to replication. These systems have also been the object of Lionel Khalil's Ph.D. thesis [10], in which he proved that it is actually sufficient to add to Lafont's Interaction Nets a single cell with two principal ports and two auxiliary ports, called amb, to obtain the full power of Multiport Interaction Nets. In spite of Khalil's result, it is still useful from the point of view of conciseness to consider cells with an arbitrary number of principal ports, as we shall do in this paper.

Our encoding (quite different from Alexiev's one, even in the finite case) covers every single feature of the π-calculus, in particular replication, which is crucial in terms of expressiveness. Compared to the aforementioned graphical formalisms, ours has an exceptional advantage: no "box" or other global notion is needed, i.e., the dynamics is fully local. In other words, our encoding may be seen as the equivalent of *sharing graphs* for the λ-calculus. In perspective, this opens the possibility for a new semantical study of concurrency, as the algebraic semantics enjoyed by Lafont's original systems (the Geometry of Interaction [11]) might be extended to Multiport Interaction Nets (this is not developed in the paper though).

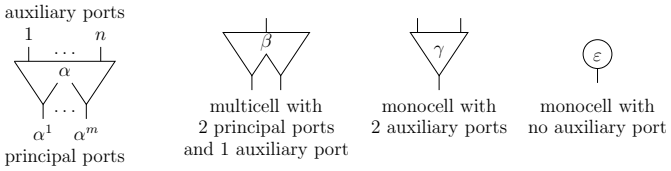
We also stress the fact that, unlike virtually any other graphical system proposed for concurrency, Multiport Interaction Nets *are not* built around the π-calculus, or any other process calculus. On the contrary, they must be seen as an independent, alternative model of concurrency, which is shown here to be equivalent to the π-calculus; our result ought to be read more in this sense than as "yet-another-graphical-representation-of-π".

[2] Alexiev calls them **INMPP**, *Interaction Nets with Multiple Principal Ports*.

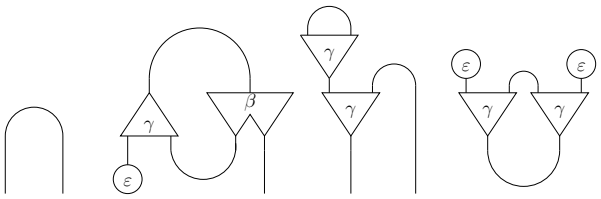
Another advantage of Multiport Interaction Nets lies in their logical roots: as extensions of multiplicative proof-nets, they can be endowed with a very natural type discipline, which makes programming much easier and more robust.

2 Multiport Interaction Net Systems

Cells. The basic elements of (multiport) interaction nets are *cells*. A cell has a distinct symbol identifying it (usually ranged over by $\alpha, \beta, \ldots$), and a finite number of *ports*, partitioned into *auxiliary ports*, the number of which is called the *arity* of the cell, and *principal ports*, the number of which is called the *co-arity* of the cell. Cells whose co-arity is 1 will be called *monocells*; cells with greater co-arities will instead be called *multicells*. Here is how we usually represent cells:



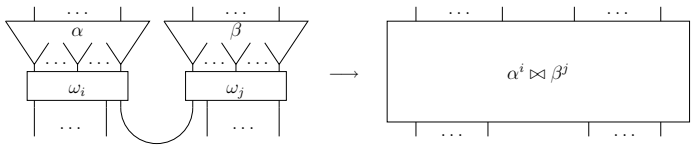
Nets. Given several occurrences[3] of cells, we use *wires* to connect their ports, and build a *net*. For example, here is a net that uses the three cells drawn above:



Nets are usually ranged over by $\mu, \nu, \ldots$ Notice that wires may be left "dangling", with one or both of their extremities not connected to any cell. In particular, a single wire is itself a net. These dangling wires are called *free ports* of the net. A free port can be principal or auxiliary, or neither, as in the case of an isolated wire. For example, the net above has 5 free ports, 2 of which are principal and 1 auxiliary. The free ports of a net form what is said to be its *interface*, since each of them is a "branching point" that can be used to connect the net to other nets, in a compositional/modular way.

Interaction. Nets become computational objects through *graph rewriting*. The fundamental *interaction principle* is that rewriting can occur only when two principal ports of two *distinct* occurrences of cells are connected; such a connection is called a *cut*. As a consequence, all rewriting rules will be of the form

[3] This is one of the very few times we will be pedantic about the distinction between cells and their occurrences; unless when strictly needed, we will usually make systematic confusion between the two concepts.



where, for graphical convenience, we have used two permutations ω_i and ω_j that "isolate" resp. the i-th principal port of α and the j-th principal port of β, i.e.

The left member is an *active pair*, and the right member its reduct. The reduct must be a net respecting the interface of the active pair, i.e., there must be a bijection between the free ports of the right and left members of each rule; this bijection will always be clear from the graphical representation.

The interface being respected, the active pair can be "disconnected" from the net and be replaced by its reduct. This is the basic rewriting step, and it is written $\mu \to \mu'$ (net μ *reduces* to μ'). We denote by $\to^*$ the reflexive and transitive closure of $\to$, and if $\mu \to^* \mu'$, we say that μ' is a *reduct* of μ. Notice that a multicell can in general be involved in several active pairs; the choice of which one is reduced is non-deterministic.

There are some additional constraints on interaction rules: first of all, there may be *at most* one rule for each pair of principal ports; allowing non-deterministic rules only complicates the definition without adding expressive power. We also observe that active pairs intrinsically lack an orientation, so the reduct $\beta^j \bowtie \alpha^i$ must be essentially the same as $\alpha^i \bowtie \beta^j$, just "flipped over"; we write this as $\beta^j \bowtie \alpha^i = \overline{\alpha^i \bowtie \beta^j}$. Moreover, reducts cannot contain active pairs (this does not prevent the possibility of infinite reductions).[4]

The formal definition. We are now ready to introduce the formal definition of a multiport Interaction Net System (mINS):

Definition 1 (Multiport Interaction Net System (mINS)). *A* multiport Interaction Net System $\mathcal{S}$ *is a couple* $(\Sigma, \bowtie)$*, where:*

- Σ *is a (possibly denumerably infinite) set of cells, called the* alphabet *of* $\mathcal{S}$*;*
- $\bowtie$ *is a (partial) function taking an active pair and yielding a cut-free net with the same interface, such that, if* $\alpha^i \bowtie \beta^j$ *is defined, then* $\beta^j \bowtie \alpha^i = \overline{\alpha^i \bowtie \beta^j}$*;*

A mINS is said to be finite *or* infinite *according to the cardinality of its alphabet.*

Since interaction is local, any rule of a finite mINS can be performed in constant time. However, in the rest of the paper we shall use only infinite systems, which make the presentation more readable. Nevertheless, everything we do can be done in finite mINS's; this will be detailed in the full version of the paper.

[4] Actually, reducts should be *reduced nets*, i.e., cut-free and *vicious-circle-free*. The definition of vicious circle is irrelevant for our present purposes, so we shall content ourselves with cut-free reducts.

Types. mINS's can be provided with a *type discipline*: given a system $\mathcal{S}$, we consider a set of *constant types*, ranged over by T, and to each port of the cells of $\mathcal{S}$ we assign an *input type* (T^-) or an *output type* (T^+). We say that a net is *well typed* if inputs are connected to outputs of the same type; a rule is well typed if both its left and right members are well typed, and the typing of the interface is preserved. If all rules of $\mathcal{S}$ are well typed, and if every well typed cut has a corresponding rule, we say that $\mathcal{S}$ is a *typed* mINS.

In a typed mINS, it is sometimes useful to have *overloaded* cells, i.e., cells which admit more than one typing for their ports; the typical example is a *duplicator cell*, which can duplicate no matter what and must therefore be capable of interacting with any cell of any type (see Fig. 3 and 4 below).

The type discipline can be useful to guarantee certain correctness properties of the system, mainly that "unreasonable" cuts never arise through reduction, like, say, that a cell representing integer addition never interacts with a string constructor.

3 mINS's and the π-calculus

3.1 The finite π-calculus

Our first step will be to find a mINS which implements the *finite π-calculus*, or Fπ. By finite π-calculus we mean the simplest subcalculus of π, modeling only name-passing and name-restriction, without any other construct (in particular without either replication or recursion). The prefixes and processes of Fπ are resp. generated by the following grammars:

$$\pi ::= \overline{x}y \mid x(z)$$

$$P, Q ::= \mathbf{0} \mid \pi.P \mid P \mid Q \mid \boldsymbol{\nu}(z)P \ .$$

The basic Fπ reduction rule is

$$\overline{x}y.P \mid x(z).Q \longrightarrow P \mid Q\{y/z\} \ .$$

The set of free names of a process P is denoted $\mathsf{fn}(P)$. Structural congruence is defined, as usual, by the axioms making the set of processes a commutative monoid with respect to parallel composition (the neutral element being $\mathbf{0}$), plus the three standard axioms concerning name restriction: $\boldsymbol{\nu}(z)\boldsymbol{\nu}(w)P \equiv \boldsymbol{\nu}(w)\boldsymbol{\nu}(z)P$, $\boldsymbol{\nu}(z)\mathbf{0} \equiv \mathbf{0}$, and, if $z \notin \mathsf{fn}(P_1)$, $z(P_1 \mid P_2) \equiv P_1 \mid \boldsymbol{\nu}(z)P_2$. The observability predicate is written $\downarrow_\mu$, where μ is either a name x (input action), or a co-name $\overline{x}$ (output action); the invisible transition relation is written $\xrightarrow{\tau}$, and its reflexive and transitive closure $\Rightarrow$.

Now consider the infinite typed mINS $\mathcal{F}_\infty$ whose alphabet and rules are given resp. in Fig. 1 and Fig. 2. Types have been omitted in the rules, but the reader can check that they are all well typed. The first rule of Fig. 2 is an example of "template rule": for a fixed $m \geq 0$, it actually condenses $2(m+1)$ rules. Template rules, the fact that ϵ ranges over $\{+, -\}$, and the permutations σ^ϵ will be notational conventions constantly adopted throughout the rest of the paper.

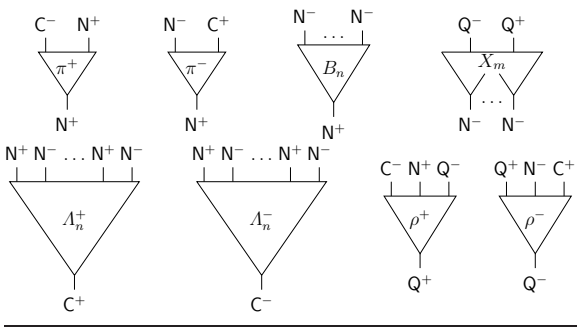


Fig. 1. The alphabet of $\mathcal{F}_\infty$

- Types N, C, and Q represent resp. *names*, *continuations*, and *queues*.
- π^+ and π^- cells will implement resp. output and input prefixes; each of them is ready to make a request on a name, and bears a continuation and another name (either the name being sent or the place-holder for substitution).
- B_n is a monocell with n auxiliary ports, with $n \geq 0$; this family of cells will be needed to bind the occurrences of the name used by an input prefix.
- X_m is a cell with m principal ports ($m \geq 1$) and 2 auxiliary ports. We stipulate that X_0 is just a wire of type Q. The cells belonging to this family will implement names: they are capable of concurrently handling several requests coming from π^+ and π^- cells, and they have two FIFO queues (one for inputs, the other for outputs) that will be used to store prefixes waiting for interaction. They also handle requests from B_n cells; the interaction with these cells models name-passing and the associated substitution.
- Λ_n^+ and Λ_n^- are monocells of arity $2n$, $n \geq 0$. These two families will implement the blocking function of prefixes: they "suspend" a connection until they interact.
- ρ^+ and ρ^- cells will be needed to represent resp. output and input queues on channels; they bear the same information as π^ϵ cells, plus a pointer to the rest of the queue. Their interaction synchronizes two prefixes: the name being sent is connected to the place-holder for substitution, and their two continuations are connected, so that they can be unblocked.

We shall now define a translation $\langle\cdot\rangle$ of Fπ processes into $\mathcal{F}_\infty$ nets. This encoding enjoys a clear correspondence with the interactive structure of processes, but accounts only for certain kinds of transitions. The free ports of $\langle P\rangle$ will be labelled with names: there will be one free port labelled x for each free occurrence of x in P. In particular, the presence of a free principal port labelled by x will mean that x is the subject of a prefix, i.e., $P \downarrow_{\overline{x}}$ or $P \downarrow_x$. In our graphical representations, we will always collect principal and auxiliary free ports resp. to the bottom and to the top of the net, so if P is a process with k observables, $\langle P\rangle$ will be pictured as a net with k free ports on the bottom.

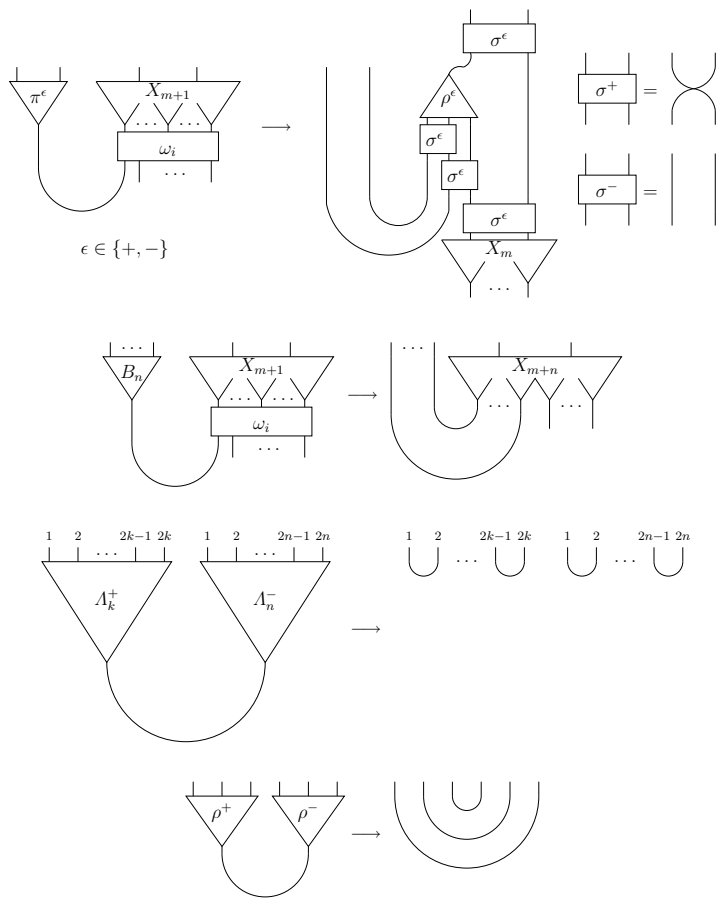


Fig. 2. The rules of $\mathcal{F}_\infty$

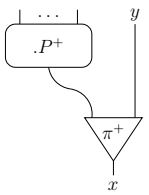
$\langle P \rangle$ might as well contain cuts; if we need to translate $\pi.P$, we must "inhibit" such cuts to correctly represent prefixing. So we introduce the nets $.P^\epsilon$ as follows:

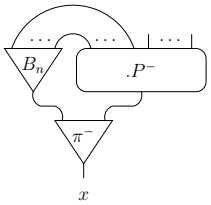
An important case is $.\mathbf{0}^\epsilon$, which is just a single 0-ary Λ_0^ϵ cell.

Definition 2 (Translation $\langle\cdot\rangle$ for $\mathrm{F}\pi$). *We define $\langle P \rangle$ by induction on P:*

– $\langle \mathbf{0} \rangle$ *is the empty net.*

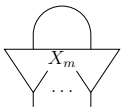
– $\langle \pi.P \rangle$ *is the following net, depending on the nature of* π*:*

$\langle \overline{x}y.P \rangle =$ 

$\langle x(z).P \rangle =$ 

In the encoding of the input prefix, the n *free ports of* $.P^-$ *labelled by* z *are connected to the* B_n *cell.*

– $\langle P \mid Q \rangle$ *is the net obtained by juxtaposing* $\langle P \rangle$ *and* $\langle Q \rangle$*.*
– *If* $\langle P \rangle$ *has* m *free ports labelled by* z*, then* $\langle \boldsymbol{\nu}(z)P \rangle$ *is the net obtained from* $\langle P \rangle$ *by connecting all such free ports to the free ports of the following net:*



Notice that this is the only case in which cuts may be introduced.

If $\langle P \rangle$ *has a free port labelled by* x *which is the principal port of a* π^+ *(resp.* π^-*) cell, we write* $\langle P \rangle \downarrow_{\overline{x}}$ *(resp.* $\langle P \rangle \downarrow_x$*).*

We have not mentioned types, but the reader can check that all the nets of Definition 2 are well typed. Also, the encoding is defined modulo the ordering of the connections to the ports of B_n, Λ^{ϵ}_n, and X_m cells, which is irrelevant.

The translation $\langle \cdot \rangle$ has already some interesting properties:

Proposition 1. *If* $P \equiv Q$*, then* $\langle P \rangle = \langle Q \rangle$*.*

Definition 3 (Fully invisible actions). *We say that a process* P *is capable of evolving to* Q *through a* fully invisible action*,* $P \xrightarrow{\tilde{\tau}} Q$*, if* $P \xrightarrow{\tau} Q$ *and the subject name used in the transition is under the scope of a restriction.*

Theorem 1 (Weak completeness of the encoding). *Let* P *be a process.*

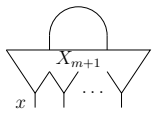
1. *If* $P \downarrow_\mu$*, then* $\langle P \rangle \downarrow_\mu$*.*
2. *If* $P \xrightarrow{\tilde{\tau}} Q$*, then* $\langle P \rangle \rightarrow^* \langle Q \rangle$*.*

Notice that the converse of Proposition 1 is false; in fact, whenever z does not appear in the prefix π, $\langle \boldsymbol{\nu}(z)\pi.P \rangle = \langle \pi.\boldsymbol{\nu}(z)P \rangle$, but the two processes are not structurally congruent (they are strong full bisimilar though).

To prove part 2 of Theorem 1 (part 1 is trivial), one just observes that $P \xrightarrow{\tilde{\tau}} Q$ means that $P \equiv \boldsymbol{\nu}(z, \widetilde{w})(\overline{z}x.R_1 \mid z(y).R_2 \mid S)$ and $Q \equiv \boldsymbol{\nu}(z, \widetilde{w})(R_1 \mid R_2\{x/y\} \mid S)$ (this is the Harmony Lemma [12]), so, using Proposition 1, $\langle P \rangle$ contains a π^+ and a π^- cell cut to the same X_m cell; knowing this, one easily finds a chain of 5 reductions leading to $\langle Q \rangle$.

Another translation, noted $[\cdot]$, is needed if we want to account for τ transitions which are not fully invisible, i.e., which are due to synchronization on free channels. Basically, $[P]$ is a sort of "closure" of $\langle P\rangle$, i.e., $[P]$ is practically identical to $\langle \boldsymbol{\nu}(\widetilde{x})P\rangle$, where $\widetilde{x}$ are the free names of P, the only difference being that we want to remember the names we artificially bound:

Definition 4 (Translation $[\cdot]$ for $\mathrm{F}\pi$). *Let x range over $\mathsf{fn}(P)$; if in $\langle P\rangle$ there are m free ports labelled by x, we define $[P]$ as the net obtained from $\langle P\rangle$ by connecting all such ports to a X_{m+1} cell, which will be left with one free port labelled by x:*



Hence, in general, $[P]$ has as many free ports as the number of free names in P. Notice that Proposition 1 transfers to $[\cdot]$ without any problem.

Now, free ports are stable under reduction, while free names are not (some might disappear); therefore, a statement like

$$\text{if } P \xrightarrow{\tau} Q, \text{ then } [P] \to^* [Q]$$

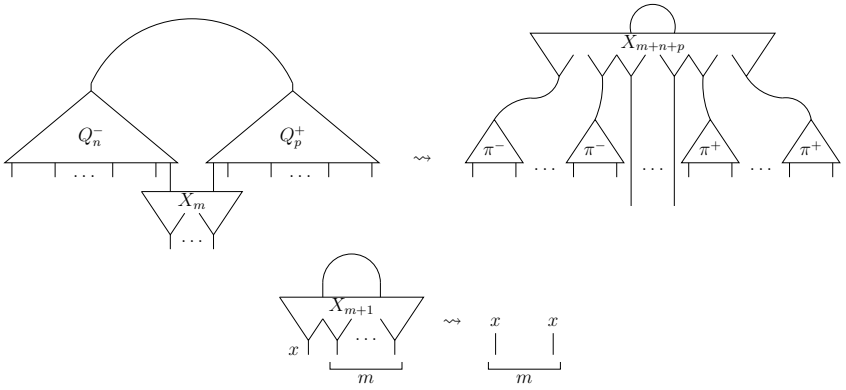
might fail for trivial reasons. In order to cope with this, and because it will be useful in other circumstances, we introduce the notion of *readback*.

Definition 5 (Bureaucratic cuts). *Cuts between B_n and X_m cells and between Λ^{ϵ}_n cells are called* bureaucratic*, and so are their respective reductions (which are resp. the second and third from the top in Fig. 2). We call* bureau-free *a net which contains no bureaucratic cut.*

The following is immediate:

Lemma 1. *Bureaucratic reduction is (strongly) confluent; hence, any net μ has a unique associated* bureau-free *form μ^{b}.*

Definition 6 (Readback). *Let μ be any reduct of a net of the form $[P]$ for some process P. The* readback *of μ, noted $\widehat{\mu}$, is the net obtained by taking μ^{b} and applying, until no longer possible, the following replacements:*



where Q^{ϵ}_k, for $k \geq 1$, is a tree of k ρ^{ϵ} cells (built in the only possible way induced by the typing). Notice that, in the second substitution, we start with 1 free port labelled by x and we end up with m free ports all labelled by x as well; as in Definition 4, x ranges over $\mathsf{fn}(P)$.

Basically, the readback procedure "undoes" the choices made in queuing up prefixes and removes the artificial closure of free names. It is evident from Definition 6 that $\widehat{[P]} = \langle P \rangle$. Now we can state the following, which follows immediately from Theorem 1 and the definition of readback:

Theorem 2 (Completeness of the encoding). *If $P \xrightarrow{\tau} Q$, then $[P] \rightarrow^* \nu$ such that $\widehat{\nu} = \langle Q \rangle$.*

The converse also holds:

Theorem 3 (Soundness of the encoding). *Let P be a process.*

1. *If $\langle P \rangle \downarrow_{\mu}$, then $P \downarrow_{\mu}$.*
2. *If $[P] \rightarrow^* \nu$, then $P \Rightarrow Q$ such that $\langle Q \rangle = \widehat{\nu}$.*

While part 1 is trivial, part 2 requires some work. We can reason by induction on the number s of reduction steps applied to get from $[P]$ to ν. If $s = 0$, then the statement follows from the above remark that $\widehat{[P]} = \langle P \rangle$. If $s > 0$, we call μ the reduct of $[P]$ after $s - 1$ steps, and we analyze the reduction $\mu \rightarrow \nu$. If this last transition results from applying a bureaucratic rule, then μ is not bureau-free, and (by Lemma 1) $\nu^{\mathsf{b}} = \mu^{\mathsf{b}}$, hence $\widehat{\nu} = \widehat{\mu}$ and we conclude using the induction hypothesis. If the last step is a π^{ϵ}/X_m rule (top of Fig. 2), the readback "undoes" the reduction and we have again $\widehat{\nu} = \widehat{\mu}$. The only case where something really happens, i.e., $\widehat{\nu} \neq \widehat{\mu}$, is when the last step is a ρ^{ϵ} rule (bottom of Fig. 2). We need here the following lemmas, the (not difficult) proofs of which are omitted:

Lemma 2. *Let P be a process, and μ a reduct of $[P]$. If μ contains a cut between a ρ^+ and a ρ^- cell, then the corresponding π^+ and π^- cells in $\widehat{\mu}$ are either cut to the same X_m multicell, or have their principal ports free and labelled with the same name.*

Lemma 3. *The reduction relation consisting of bureaucratic reductions and the ρ^{ϵ} reduction is (strongly) confluent.*

By the induction hypothesis, we know that $\widehat{\mu} = \langle Q \rangle$ for some Q such that $P \Rightarrow Q$; by Lemma 2, we also know that this Q contains an output and an input prefix acting on the same channel, i.e., $Q \equiv \boldsymbol{\nu}(\widetilde{w})(\overline{x}y.R_1 \mid x(z).R_2 \mid S)$. The ρ^{ϵ} reduction leading to ν introduces (at most) two bureaucratic cuts; by Lemma 3, we can assume these to be the only bureaucratic cuts in ν, for if μ was not bureau-free, reducing its cuts before or after the application of the ρ^{ϵ} rule has no effect on ν^{b} (and thus on $\widehat{\nu}$). It is then just a matter of applying a few rewriting rules to check that $\widehat{\nu} = \langle \boldsymbol{\nu}(\widetilde{w})(R_1 \mid R_2\{y/z\} \mid S) \rangle$, as needed to prove our statement. The typing discipline followed by $\mathcal{F}_{\infty}$ assures us that no cut other than those considered can arise through reduction, so we are done.

Of course the Soundness Theorem has a weaker version, stating that if $\langle P \rangle \rightarrow^* \nu$, then there are a process Q such that $\langle Q \rangle = \widehat{\nu}$ and a number of fully invisible transitions (including zero) leading from P to Q.

3.2 Adding replication

The fact that mINS's are able to faithfully encode Fπ is already meaningful from the point of view of concurrent computation, but is extremely poor in terms of expressive power. In this section we shall give a stronger result by showing that the mINS $\mathcal{F}_\infty$ can be extended into a mINS $\mathcal{C}_\infty$ that encodes a fragment of the π-calculus, called here the "core" π-calculus, or Cπ, which adds the replication operator to Fπ. One could see Cπ basically as a synchronous and extended version of the Pict language [13].

A well known fact is that the replication operator is not needed everywhere in the definition of processes: *replicated prefixes* suffice to give universal computational power to the π-calculus. This is why we introduce *extended prefixes*

$$\kappa ::= \overline{x}(z) \;\big|\; \pi \qquad \text{(where } \pi \text{ is a prefix of F}\pi\text{)} \;,$$

which add the bound-output prefix to the "standard" prefixes of Fπ, and we define the processes of Cπ to be those generated by the following grammar:

$$P, Q ::= \mathbf{0} \;\big|\; \pi.P \;\big|\; P \mid Q \;\big|\; \boldsymbol{\nu}(z)P \;\big|\; \kappa \rhd P \;.$$

In the traditional syntax, if π is an Fπ prefix, we would write $\pi \rhd P$ as $!\pi.P$, while $\overline{x}(z) \rhd P$ would be written as $!\boldsymbol{\nu}(z)\overline{x}z.P$. Here, we choose this alternative syntax since we *do not* consider the standard axiom for structural congruence $!P \equiv P \mid !P$; structural congruence on Cπ processes is thus the same relation we defined on Fπ (see page 5). To recover the adequate notion of transition relation, we add the following rules:

$$\frac{}{\overline{x}y \rhd P \overset{\overline{x}y}{\to} \overline{x}y \rhd P \mid P} \qquad \frac{}{x(z) \rhd P \overset{xy}{\to} x(z) \rhd P \mid P\{y/z\}} \qquad \frac{}{\overline{x}(z) \rhd P \overset{\overline{x}(z)}{\to} \overline{x}(z) \rhd P \mid P}$$

The reduction relation is defined in the same way; so, for example, we have

$$\overline{x}y.P \mid x(z) \rhd Q \longrightarrow P \mid Q\{y/z\} \mid x(z) \rhd Q \;.$$

The alphabet of the (infinite) typed mINS $\mathcal{C}_\infty$ is defined by adding to the alphabet of $\mathcal{F}_\infty$ the cells of Fig. 3, whose interactions are given by the rules of Fig. 4:

- There is an additional type, R, which represents *name restrictions*.
- $!\pi^\epsilon$ and $!\rho^\epsilon$ cells play the same role resp. as π^ϵ and ρ^ϵ cells: the first represent replicated prefixes, the second *enqueued* replicated prefixes. They carry two additional pieces of information: another name, and a restriction. The first is a *potential occurrence* of the subject of the prefix, which is needed since a replicated prefix whose subject is x potentially generates a new occurrence of x after replication. The second is a sort of pointer to the restricted names which are under the scope of the replication; these names are not usable until replication takes place.
- A cell belonging to the $!X_m$ family (m auxiliary ports, $m \geq 1$) represents a restricted name with m occurrences blocked under a replicated prefix.

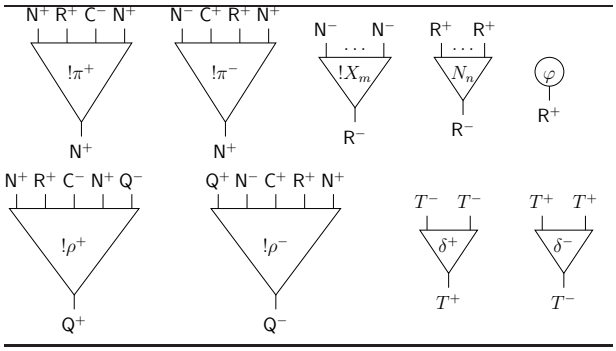
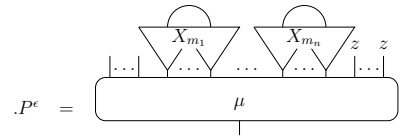


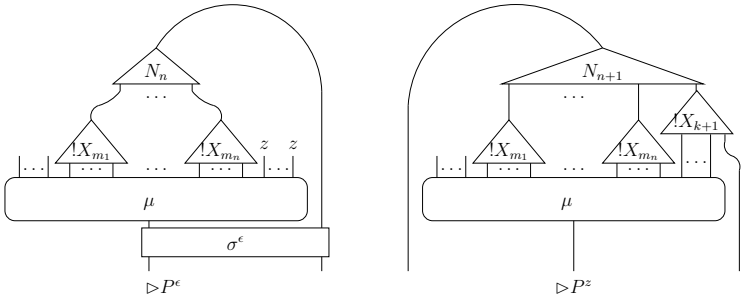
Fig. 3. Additional cells for $\mathcal{C}_\infty$.

– Cells belonging to the N_n family (n auxiliary ports, $n \geq 0$) "collect" $!X_m$ cells and pass them to $!\pi^\epsilon$ cells.
– The φ cell "unblocks" restricted names whenever a copy of a replicated process is activated.
– δ^ϵ cells are *duplicators*: they implement (local) replication of processes. They are *overloaded* cells, i.e., their ports can be typed with any type T, provided the typing respects the input/output specifications of Fig. 3.

Definition 7 (Translations $\langle\cdot\rangle$ and $[\cdot]$ for Cπ). *We extend the translation $\langle\cdot\rangle$ of Definition 2 to Cπ processes in the following way. Suppose that $.P^\epsilon$ (as always, $\epsilon \in \{+,-\}$) is a net containing n X-cells and (among others) k free ports labelled with the name z:*



Then, we define $\triangleright P^\epsilon$ and $\triangleright P^z$ as follows:



where, as usual, σ^- is the identity permutation and σ^+ is the "twist" permutation. We can now define $\langle\kappa \triangleright P\rangle$:

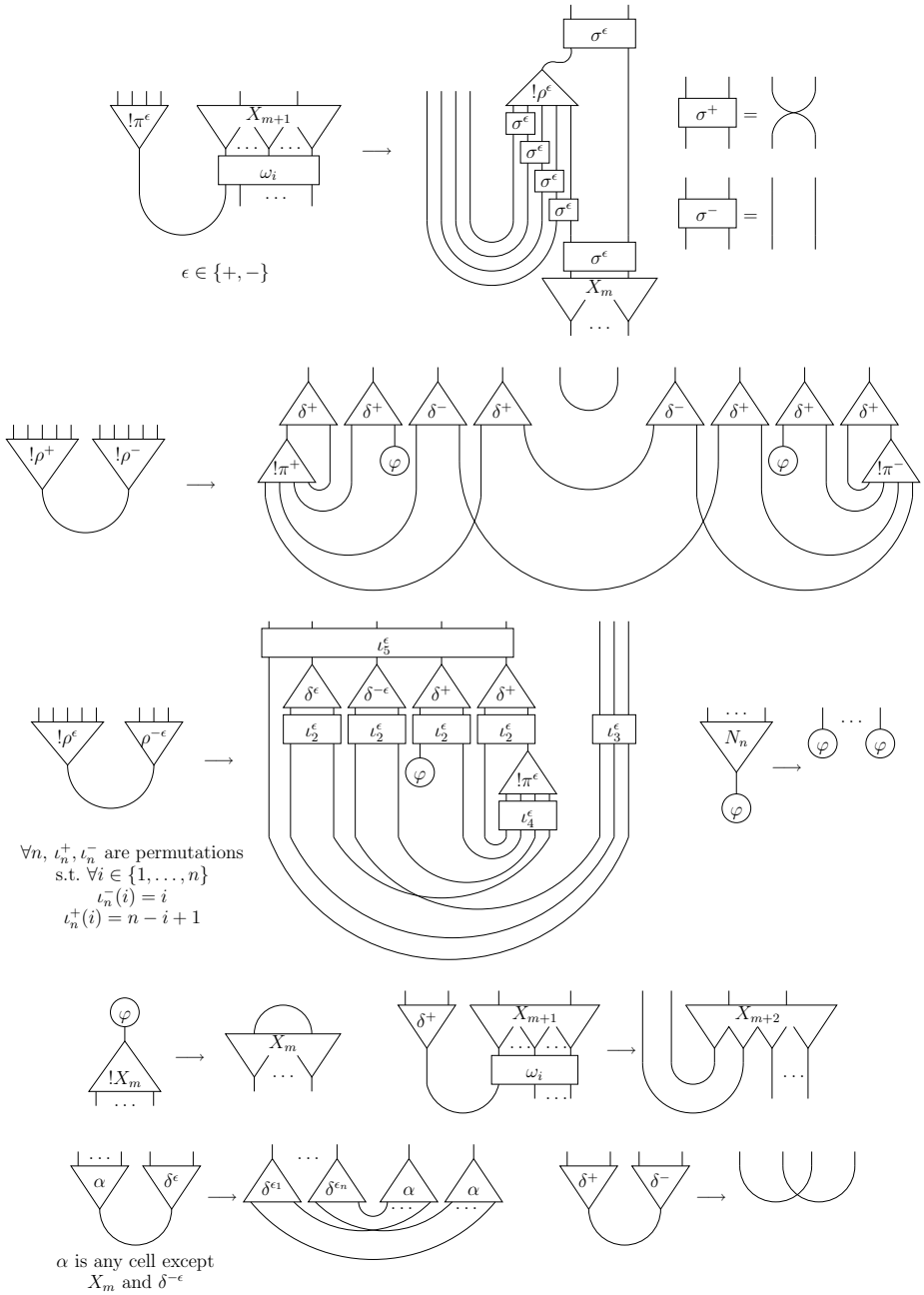
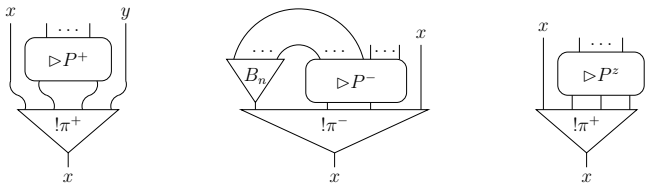


Fig. 4. Rules for the additional cells of $\mathcal{C}_\infty$.



If $\langle P \rangle$ has a free port labelled by x which is the principal port of a π^+ or $!\pi^+$ (resp. π^- or $!\pi^-$) cell, we write $\langle P \rangle \downarrow_{\overline{x}}$ (resp. $\langle P \rangle \downarrow_x$).

The translation $[P]$ is obtained from $\langle P \rangle$ exactly as in Definition 4.

Notice that it is no longer the case that each free occurrence of x in P corresponds to a free port labelled by x in $\langle P \rangle$; here, a free occurrence of x as subject of a replicated prefix generates *two* free ports. It is still the case though that the free *principal* ports of $\langle P \rangle$ are in bijection with the observables of P. We also remark that Proposition 1 holds trivially for both of the extended translations; this would of course be impossible if we admitted that $!P \equiv P \mid !P$.

Definition 5 can be extended to $\mathcal{C}_\infty$ by considering as bureaucratic the last 5 cuts of Fig. 4, i.e., all cuts involving φ and δ^ϵ cells; it is immediate to verify that Lemma 1 still holds. We can then define the readback $\widehat{\mu}$ of a $\mathcal{C}_\infty$ net μ which is the reduct of a net $[P]$: simply take its bureau-free form $\mu^\flat$, and apply the substitutions of Definition 6, where this time the queues might contain $!\rho^\epsilon$ cells, which need to be replaced by the corresponding $!\pi^\epsilon$ cells.

With all the definitions extended to $\mathcal{C}_\infty$, it is not hard to prove the following:

Theorem 4 (Faithfullness of the encoding of $\mathrm{C}\pi$). *Let P be a $\mathrm{C}\pi$ process.*

1. *$P \downarrow_\mu$ iff $\langle P \rangle \downarrow_\mu$.*
2. *If $P \xrightarrow{\tau} Q$, then $[P] \rightarrow^* \nu$ and $\widehat{\nu} = \langle Q \rangle$.*
3. *If $[P] \rightarrow^* \nu$, then $P \Rightarrow Q$ and $\langle Q \rangle = \widehat{\nu}$.*

The proof follows a similar argument to that given for Theorem 2 and Theorem 3. The fundamental issue concerning replication is that neither cuts, nor X_m or δ^ϵ cells can be duplicated by δ^ϵ cells. This is why $\triangleright P^\epsilon$ and $\triangleright P^z$ are introduced: such nets are cut-free, and do not contain either X_m or δ^ϵ cells, so they can be safely duplicated. Then, φ cells extract P from $\triangleright P^\epsilon$ or $\triangleright P^z$. We also observe that, thanks to the encoding, nested replication poses no problem. The other two important points are that duplication is completely bureaucratic (therefore strongly confluent), and that it stops on free channels; this assures us that the replication process does not interfere with prefix synchronization.

4 Conclusions

We have seen how one can find an infinite typed mINS $\mathcal{C}_\infty$ which is able to faithfully encode a quite expressive fragment of the π-calculus, equivalent to a synchronous and extended version of the Pict language. As a matter of fact,

much more can be done: one can enrich $\mathcal{C}_\infty$ in order to represent any other feature of the π-calculus, in particular guarded choice and match prefix. This will be shown in the full version of the paper.

We also remark that the systems introduced in Sect. 3 can easily be adapted to encode any kind of *typed* π-calculus. To see this, just notice, for example, that X_m cells can be overloaded by uniformly instantiating the type N into any type V belonging to the grammar $V ::= B \mid \sharp V$, where B ranges over some set of basic types. We can then proceed to overload π^ϵ cells so that if their two auxiliary ports have types V^ϵ and $\mathsf{C}^{-\epsilon}$, then the principal port has type $\sharp V^+$. If we apply similar changes to $!\pi^\epsilon$, ρ^ϵ and $!\rho^\epsilon$ cells, we obtain a mINS for the simply typed π-calculus [12]. Of course, the original system $\mathcal{C}_\infty$ is retrievable by typing everything with a fixpoint type N such that $\mathsf{N} = \sharp\mathsf{N}$.

Acknowledgments. We would like to thank Laurent Regnier and the anonymous referees for their useful comments and suggestions.

References

1. Lafont, Y.: Interaction Nets. In: Conference Record of POPL'90, ACM Press (1990) 95–108
2. Milner, R.: Pi-nets: A graphical form of π-calculus. In: Proceedings of ESOP'94. Volume 788 of Lecture Notes in Computer Science., Springer (1994) 26–42
3. Parrow, J.: Interaction diagrams. Nordic Journal of Computing **2** (1995) 407–443 A previous version appeared in *Proceedings of A Decade in Concurrency*, LNCS 803: 477–508, 1993.
4. Fu, Y.: Reaction Graph. Journal of Computer Science and Technology **13** (1998) 510–530
5. Laneve, C., Parrow, J., Victor, B.: Solo Diagrams. In: Proceedings of TACS'01. Volume 2215 of Lecture Notes in Computer Science., Springer-Verlag (2001) 127–144
6. Laneve, C., Victor, B.: Solos in Concert. In: Proceedings of ICALP'99. Volume 1644 of LNCS., Springer-Verlag (1999) 513–523
7. Beffara, E., Maurel, F.: Concurrent nets: a study of prefixing in process calculi. In: Proceedings of EXPRESS 2004. Volume 128 of ENTCS., Elsevier (2005) 67–86
8. Yoshida, N.: Graph Notation for Concurrent Combinators. In: Proceedings of TPPP'99. Volume 907 of LNCS., Springer (1995) 393–412
9. Alexiev, V.: Non-deterministic Interaction Nets. Ph.D. Thesis, University of Alberta (1999)
10. Khalil, L.: Généralisation des Réseaux d'Interaction avec amb, l'agent de McCarthy: propriétés et applications. Ph.D. Thesis, École Normale Supérieure de Paris (2003)
11. Lafont, Y.: Interaction combinators. Information and Computation **137** (1997) 69–101
12. Sangiorgi, D., Walker, D.: The π-calculus — A Theory of Mobile Processes. Cambridge University Press (2001)
13. Pierce, B., Turner, D.: Pict: A Programming Language Based on the Pi-Calculus. CSCI Technical Report 476, Indiana University (1997)