

Pi-nets: interaction nets for pi-calculus

Andrei Dorman

Damiano Mazza

November 1, 2010

Abstract

π -calculus is a framework that aims to describe concurrent calculations through a formal definition of processes. Originally, π -calculus is defined by a formal language and a set of reduction rules, much in the spirit of λ -calculus. Our aim is to provide a graphical representation of π -calculus using multi-wired interaction nets, in the spirit this time of proof-nets of linear logic.

1 π -calculus (without sums and promotion)

A typical system in π -calculus are multiple processes that can compute concurrently. Processes can interact with one another by sending and receiving information through a given set of channels. Many processes can “listen” on the same channel, but only one pair can communicate at a time on it. Therefore, the calculus is completely indeterministic in a strong way (e.g. not confluent), the choice of the pair being completely random.

More than simple parallelism, π -calculus describes mobile processes. The content of the messages are channel names that the recipient process can then use to communicate with other processes that have access to it, thus changing the “wiring” of the whole process-net.

Practically, a process is a sequence of three types of *actions* – sending, receiving and an invisible internal action, that can be combined by an operation of *concurrency*, putting two processes in parallel, the creation of *new names*, *replication* and *summation*, representing a definite indeterministic choice between two processes.

In this paper, we focus on a part of π -calculus without replication and summation.

We begin by defining the calculus formally.

The language of π -calculus Let $\mathcal{N}$ be a numerable set of *names* ranged over by lower case Latin letters. Names represent communication channels, and are also the values being transmitted in communications.

Definition 1.1 (Prefix). Actions are expressed by *prefixes*, of which there are three kinds:

$$\pi ::= \bar{x}y \mid (x)y \mid \tau$$

The first variable (here x in both cases) is called the *name* of the prefix, the second one is called its *co-name*.

Informally, $\bar{x}y$ corresponds to the fact that the process can send the name y through the channel x ; $(x)z$ means the process can receive a name on the channel x (the role

of z will be explained shortly); τ represent the possibility of an internal unobservable action.

Definition 1.2 (The language of π -calculus). π -calculus is given by the following grammar.

$$\begin{aligned} P, Q &::= M \mid P|Q \mid \nu z P \\ M &::= 0 \mid \pi.P \end{aligned}$$

where π is a prefix. We denote by Π the set of all (finite) words on this grammar.

P and Q are called *processes*, M is a *summand*. One can notice that this grammar could be given without distinguishing processes and summations (it is equivalent to $P, Q ::= 0 \mid \pi.P \mid P|Q \mid (\nu z)P$). The given definition is nevertheless closer to the definition of the “full” π -calculus and will be easier to extend.

TO DO: Informal account of the interpretation of the processes.

Definition 1.3 (Binding). In each of $x(z).P$ and $\nu z P$, the displayed occurrence of z is *binding* with *scope* P . An occurrence of a name is *bound* if it is, or lies within the scope of, a binding occurrence of the name. An occurrence of a name in a process is *free* if it is not bound.

We write $\text{fn}(P)$ for the set of free names of P and $\text{bn}(P)$ for the set of its bounded names.

Definition 1.4 (Guarded process). In $\pi.P$, we say the process P is *guarded* by the prefix π , and π is *guardian* of P .

TO DO: Purpose of structural congruence.

Definition 1.5 (Structural congruence). *Structural congruence* $\equiv$ is the smallest congruence on processes that satisfies the axioms of Table 1.

Compositional associativity:	$P_1 \mid (P_2 \mid P_3)$	$\equiv$	$(P_1 \mid P_2) \mid P_3$
Compositional commutativity:	$P_1 \mid P_2$	$\equiv$	$P_2 \mid P_1$
Compositional inaction:	$P \mid 0$	$\equiv$	P
Restriction commutativity:	$\nu v \nu w P$	$\equiv$	$\nu w \nu v P$
Restriction inaction:	$\nu z 0$	$\equiv$	0
Restriction/composition:	$\nu z (P_1 \mid P_2)$	$\equiv$	$P_1 \mid \nu z P_2$, if $z \notin \text{fn}(P_1)$

Table 1: The axioms of structural congruence

Each of these axioms express something intrinsic to π -calculus operators. Structural congruence equates all processes we will never want to distinguish for any semantic reason. From now on, we will write $P \mid Q \mid R$ for $(P \mid Q) \mid R$, and $(\nu a_1, \dots, a_n)P$ for $\nu a_1 \dots \nu a_n P$.

Reduction: a process has to... process TODO: Reduction!!!! What for, rules, ...

The main reduction rule states that if, of two processes in parallel, one can send on some channel on which the other can receive, they can perform this action (simultaneously). This rule is written in the language:

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

where $Q\{y/z\}$ is the process Q where the free occurrences of z have been substituted by y . The left process sends the name y via x , and follows on his own from then. The second receives y on the channel x . As a consequence of this, all actions that were possible on the channel z are now made possible on the channel y .

The formal definition of reductions rules is given in Table 2. A reduction $P \rightarrow Q$ is valid if it is possible to build a derivation tree with conclusion $P \rightarrow Q$.

$\frac{}{\bar{x}y.P \mid x(z).Q \mid R \longrightarrow P \mid Q\{y/z\} \mid R} \quad (\pi\text{-red})$	
$\frac{P \longrightarrow Q}{P \mid R \longrightarrow Q \mid R} \quad (\text{Parr.})$	$\frac{P \longrightarrow Q}{(\nu x)P \longrightarrow (\nu x)Q} \quad (\text{Restr.})$
$\frac{P' \equiv P \longrightarrow Q \equiv Q'}{P' \longrightarrow Q'} \quad (\text{Struct.})$	

Table 2: The derivation rules for reductions.

ATTENTION ATTENTION ATTENTION [] The occurrence of R in the first rule might be redundant with the rule Parr (for the existence of a derivation tree).

When are two processes distinguishable? TODO: Strong bisimulation, bisimilarity and congruence.

Definition 1.6 (Strong bisimilarity).

Definition 1.7 (Strong full bisimilarity).

Proposition 1.8 (Simulating τ). *Let's consider the sub-calculus Π' defined as in definition 1.2 where prefixes are restricted to input and output*

$$\pi ::= \bar{x}y \mid (x)y.$$

For all $P \in \Pi$, there exists $[P] \in \Pi'$ s.t. $[P] \sim^c P$.

PROOF: It is sufficient to prove that for any summand of the form $\tau.P$ in Π :

$$\tau.P \sim^c (\nu w, a, b)(\bar{w}a \mid w(b).P) \quad \text{where } w, a, b \notin \text{fn}(P)$$

Let $R = \tau.P$ and $S = (\nu w, a, b)(\bar{w}a \mid w(b).P)$.

Using 1.7: it is clear that any substitution σ will not modify the relation of bisimilarity described above since the two processes only differ by bounded names. It therefore suffices to show that $R \sim S$.

Using 1.6: it is clear that $\sim' = \{(R, S)\} \cup \equiv$ is a strong bisimulation. $\square$

Remark 1.1. This simulation also holds for asynchronous π -calculus since the translation of P belongs to $A\Pi$ if and only if P does.

TO DO: If we ever do summations, we should be careful that the needed bisimilarity for this propositions is:

$$M + \tau.P \sim^c (\gamma w, a, b)(\bar{w}a \mid (M + w(b).P))$$

2 Interaction nets: generalities

Multiwired interaction nets We assume to be given a collection of *symbols*. Each of these symbols s is given with a natural number $\text{ar}(s)$, its *arity*.

An interaction net $\mathcal{P}$ is given by:

- a finite set C of cells, each cell c given with a symbol s_c , a particular port p_c , called *principal port* and a set $\partial(c)$ of *auxiliary ports* s.t. $|\partial(c)| = \text{ar}(s_c)$. The principal and auxiliary ports of a cell are said to be *connected*;
- a set $\mathcal{W}$ of *multiwires* with a function ∂ mapping each multiwire to a set (possibly empty) of free or connected ports.
- a finite set $\mathcal{F}$ of *free ports*;

A cell and a wire are usually pictured as shown in Figure 1 where the branches of the wire can be connected or not to a port of a cell.



TO DO: Picture of a cell and a wire.

Figure 1: Graphical representation of a cell and a wire.

If not specified otherwise, we call *size* of an interaction net the number of its cells.

Interaction The interaction between two cells is made possible when a wire is connected to the principal ports of at least two cells.



TO DO: Picture three-four cells connected; at least two by principal port.

Figure 2: Cells can interact when two principal port belong to a same wire.

Reduction rules are given for every such situation (could be that nothing happens). Reduction rules are usually local, meaning they can be described just by focusing on the two interacting cells, the “rest of the picture” staying exactly as it was before. as shown succinctly in Figure 3.



TO DO: Picture a general reduction rule.

Figure 3: Graphical representation of a generic reduction.

When dealing with nets for π -calculus, we will use a slightly different representation of the cells, because we need some more information about a kind of “wiring” within the cell.

Net morphism A *morphism* from a net M to a net N is a function F s.t.

- each cell c of M is sent on a cell c' of N of same symbol;
- the principal port of c is sent on the principal port of c' ;
- auxiliary ports of c are sent on auxiliary ports of c' ;
- each wire w of M is sent on a wire w' of N and $\partial(w') = F(\partial(w))$;
- each free port of M is sent on a free port of N .

An *isomorphism* on nets is a bijective morphism. When two nets M and N are isomorphic, we write $M \simeq N$.

For our purpose, two isomorphic nets are perfectly equivalent. It is actually one of the main advantages of using geometrical structures rather than formal languages. Morphisms are easier to visualize on graphs than on words.

ω -equivalence Before even considering interaction in nets, we can notice some simple equivalence on nets concerning wires only. For instance, a wire not connected to any port cannot perform any kind of further action and doesn’t represent any future dynamic possibility. We can therefore consider such wires as void. Also, if considering that wires represent channels, having two wires connected together is the same as extending one to connect with the ports of the other.

Formally, we define an ω -reduction on nets (actually on wires of nets). This reduction is proved to be strongly confluent and strongly normalizing.

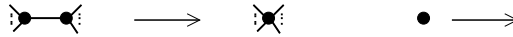


Figure 4: Rules of ω -reduction.

Definition 2.1 (ω -reduction). Let N, N' be two multiwired interaction nets. We write $N \rightarrow_{\omega} N'$ if:

- $\text{wires}(N) = \text{wires}(N') \cup \{w\}$;
- $\partial(w) = \emptyset$;
- for all $v \in \text{wires}(N')$, $\partial_N(v) = \partial_{N'}(v)$.

or if

- $\text{wires}(N) = W \cup \{w_1, w_2\}$ s.t. $\partial(w_1) \cap \partial(w_2) = p$ for a certain port p ;
- $\text{wires}(N') = W' \cup \{w'\}$ s.t. $\partial(w') = \partial(w_1) \cup \partial(w_2) \setminus p$.

The graphical version of these rules are depicted in Figure 4. We write $\rightarrow_\omega^*$ for the transitive and reflexive closure of $\rightarrow_\omega$.

Definition 2.2 (Local network). A local network W is a set $\{w_1, \dots, w_n\}$ of *contiguous* wires:

$$\forall i \leq n, \exists j \leq n \quad \text{such that} \quad \partial(w_i) \cap \partial(w_j) \neq \emptyset.$$

By considering a local network to be an interaction net with no cells, we can extend all definitions of interaction nets to local networks, especially ω -reduction.

Proposition 2.3. *Every local network W ω -reduces in a finite number of steps into a wire w .*

PROOF: Let $\#W$ denote the size of a local network W , i.e. the number of wires in it. If $W \rightarrow_\omega W'$, then

- W' is still a local network;
- $\#W' = \#W - 1$.

So after $\#W$ steps, W will be reduced to a local network of size 1, i.e. a wire. $\square$

We define a *connection* operation on wiring as follows.

Definition 2.4 (Connecting wires). Let v, w be two free wires labeled identically such that their borders don't intersect. Let u be a wire of arity 3. $v \bullet\bullet w$ is the result of plugging u to the free ports of v and w and letting its last port free.

Notice that the result of attaching two wires is not a wire, but a local network. Since a local network ω -reduces to a wire, it is easy to check that the connection operation is associative up to ω -reduction.

Usually, interaction net theory deals only with ω -normal forms. But those abnormal forms have to be taken care of since they can appear during the reduction process. In some way discussed later, it is sometimes even needed to go from a normal form to an abnormal one in order to be able to define correctly some reduction rules. We define such an operation as follows.

Definition 2.5 (ω -expansion). We say a net N' is ω -expanded from a net N and we write $N \rightsquigarrow_\omega N'$ whenever $N' \rightarrow_\omega^* N$.

3 Nets for π -calculus

Let us describe informally the translation of a π -calculus process into interaction nets.

Let $\pi.P$ be a process. The prefix π is represented by a cell, positive in the case of an output prefix, negative in the case of an input one. The name of π is “attached” to the principal port of the cell, it’s co-name is attached to the data port. All other auxiliary ports represent the names of P (even the name and co-name of π if they have occurrences in P).

One can define a cell with no principal port and no data port to represent the τ prefix. But remembering Proposition 1.8 we can avoid complexifying the net syntax by just ignoring τ prefixes in the translation.

Putting two processes in parallel is just done by connecting all ports representing the same names in the two processes. Thus wires really represent channels by which cells can communicate.

Private names are represented by wires not connected to any free port. It means no new process can decide to communicate on it (without it being sent through another channel).

One of the difficulties of the representation is to make sure the sequentiality of prefixes is assured. One way to do this is to associate to each cell of the net a sub-net representing its guarded process. In this case, only prefixes of the same “level” are represented in a net. This way, one can only see the possible communications that can be done right away and not the various ones that can be made later. Such a use of nested nets is usually called *nets with boxes*. It has it’s advantages, so we will present it, but we will insist on another version, which seems to use closer to the spirit of interaction nets.

In that version, cells guard the part of the net above them, therefore assuring no communication can be made inside that subnet. It forces all communicating channels to “enter” through one of its auxiliary ports and “exit” through another in such a way that no two opposite principal ports “above” it share a wire. They will only be able to do so when the given guardian cell disappears in a communication.

3.1 Simple π -nets

Definition 3.1 (Cells of π -nets). We consider two sets of symbols, the *positives* cells $\{\alpha_i, i \in \mathbb{N}\}$ and the *negatives* cells $\{\beta_i, i \in \mathbb{N}\}$.

For all $i \in \mathbb{N}$, $\text{ar}(\alpha_i) = \text{ar}(\beta_i) = 2i + 1$.

The auxiliary ports of a cell are given with a pairing function associating them in pairs. In a pair (p, q) , p is called *entry* and q is called *exit*. The remaining port (since there is always an odd number of auxiliary ports) is called the *data port*.

A graphical representation of a generic π -cell is shown in Figure 5. The auxiliary ports “on top” of the cell are entry ports, the ones “on bottom” are exit auxiliary ports. The principal port is the “angle” of the cell. The last one, on the edge of the angle, is the data port. The dotted lines between entry and exit auxiliary ports can be omitted since the pairing function is obvious.

Definition 3.2 (Ports’ polarity). A port p is said to be:

neutral if it is an auxiliary port of a cell or a free port.

positive if it’s the principal port of a positive cell (i.e. of symbol α_i).

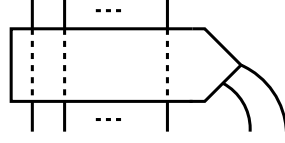


Figure 5: Representation of a π -cell.

negative if it's the principal port of a negative cell (i.e. of symbol β_i).

Definition 3.3 (Wires' polarity). Let w be a wire of a net Π . w is said to be:

neutral if $\partial(w)$ are all neutral ports.

positive if $\partial(w)$ are all positive or neutral ports.

negative if $\partial(w)$ are all negative or neutral ports.

a *cut* if $\partial(w)$ contains at least one positive and one negative port.

In π -nets, the auxiliary ports keep information about connections between them. This is the prpose of the pairing function. It is therefore possible to “follow a signal”, not only in a local network, but also across cells.

Definition 3.4 (Network). We say two wires v, w are *connected* if one of the two following condition holds:

- v and w are contiguous;
- there exist $p \in \partial(v)$, $q \in \partial(w)$ and a cell c of $\mathcal{P}$ such that (p, q) or (q, p) is a pair of auxiliary ports of c .

A *network* is a set $\{w_1, \dots, w_n\}$ of connected wires, i.e. $\forall i \leq n, \exists j \leq n$ such that w_1 and w_2 are connected.

For any wire w in a network N , or any port p of a wire of N , we will say that N is the network of w or the network of p . Since all connected (immediate from the definition) wires are labeled identically, we can talk about the label of a network.

Translating a π -process in π -nets

Definition 3.5 (Pre-net). A *pre-net* $\mathcal{P}$ is an interaction net whose nodes are π -cells labeled by π -calculus prefixes (thus we can refer to the *name* or *co-name* of a cell) and ports are labeled by names, such that:

- positive cells are labeled by output prefixes and negative cells by input prefixes;
- the principal port of a cell is labeled by the name of the cell;
- the data port of a cell is labeled by its co-name;
- the pairing function of a cell associates pairs of ports with the same label;
- all ports of a wire are labeled by the same name;
- if the ports of a wire w are labeled a , w is labeled either a or (a) ;

- a wire is connected to at most one free port.

We don't label ports when the labeling is obvious or unimportant. For instance, if the cell is labeled, we don't need to label its principal or data ports. Also, we don't usually care about the exact symbol of a cell (which gives the number of auxiliary ports). It is usually enough to know if it is positive or negative. In which case we just annotated it by a + or a -.

Definition 3.6 (Free, bounded and final wires). We say a wire w is *final* if $\partial(w)$ contains only principal, data or exit ports (e.g. does not contain any entry ports). A wire w is said to be *free* if $\partial(w)$ contains at least one free port. Otherwise, it is said to be *bounded*.

ATTENTION ATTENTION ATTENTION [If we want to avoid using Eq.2 of Remark 4.1: Final wires labeled by a name in brackets are said to be *restricted* (all final restricted wires are bounded). And all its consequences...

We denote by $\text{fw}(\mathcal{P})$ (resp. $\text{bw}(\mathcal{P})$ and $\text{finw}(\mathcal{P})$) the set of final free wires (resp. of final bounded wires and final wires) of $\mathcal{P}$. Therefore, $\text{finw}(\mathcal{P}) = \text{fw}(\mathcal{P}) \cup \text{bw}(\mathcal{P})$.

We define three operations on final wires that will be necessary to construct nets representing processes. The first will separate a cut wire in two, so it can't act anymore. The two resulting wires will be labeled as the original but one will be positive, the other negative. The second operation will "plumb" together all final wires that are labeled the same. The last one corresponds to plumbing two pre-nets together.

Definition 3.7 (Electrolysis). Let $\mathcal{P}$ be a pre-net, and w a cut wire of $\mathcal{P}$ labeled by x . Let ρ^+ the set of positive ports of w , ρ^- the set of negative ports of w , ρ^0 the set of neutral ports of w and p^f its free ports (if w is free).

Let $\mathcal{P}'$ defined as follows.

- If w is a free wire:
 - the cells of $\mathcal{P}'$ are the cells of $\mathcal{P}$;
 - the free ports of $\mathcal{P}'$ are the free ports of $\mathcal{P}$ plus a new free port p labeled by x ;
 - $\text{wires}(\mathcal{P}') = (\text{wires}(\mathcal{P}) \setminus w) \cup \{w_1, w_2\}$, where w_1, w_2 are labeled x and such that $\partial(w_1) = \rho^+ \cup \rho^0 \cup \{p^f\}$ and $\partial(w_2) = \rho^- \cup \{p\}$.
- If w is a bounded wire:
 - the cells of $\mathcal{P}'$ are the cells of $\mathcal{P}$;
 - the free ports of $\mathcal{P}'$ are the free ports of $\mathcal{P}$;
 - $\text{wires}(\mathcal{P}') = (\text{wires}(\mathcal{P}) \setminus w) \cup \{w_1, w_2\}$, where w_1, w_2 are labeled x and such that $\partial(w_1) = \rho^+ \cup \rho^0$ and $\partial(w_2) = \rho^-$.

w is said to be *split* in $\mathcal{P}'$. A net is said to be *split* when all its final wires are split. Graphically, a splitting is depicted in Figure 6.

We are only interested in splitting final wires, because by construction, we will make sure to never have cuts except as final wires. This is an important feature of π -nets, preventing a guarded process to reduce before its guardian. The fact that all neutral ports are affected to the positive "half" of the split wire is usually irrelevant. In general, we will say a pre-net is split whenever it does not contain any cuts *because they were artificially split*. It is important not to mistake a split net for a normal form.

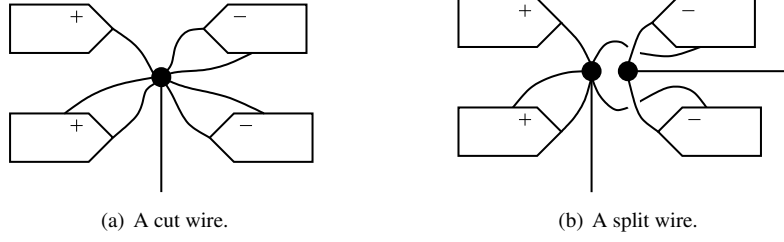


Figure 6: The splitting of a wire (only involved ports were represented).

Definition 3.8 (Plumbing). Let $\mathcal{P}$ be a pre-net.

Let $\{a_1, \dots, a_k\}$ be the set of all labels of final free wires of $\mathcal{P}$ and let $\text{fw}(\mathcal{P}) = \{w_1^1, \dots, w_{n_1}^1, \dots, w_1^k, \dots, w_{n_k}^k\}$ be a partition such that for each $1 \leq i \leq k$, $\{w_1^i, \dots, w_{n_i}^i\}$ are labeled a_i .

Let $\mathcal{P}'$ be the pre-net $\mathcal{P}$ where for each $0 \leq i \leq k$, $\{w_1^i, \dots, w_{n_i}^i\}$ have been connected (as defined in Definition 2.4). $\mathcal{P}'$ is said to be the *plumbing* of $\mathcal{P}$. Graphically, a plumbing is depicted in Figure 7.



Figure 7: Plumbing a pre-net

Definition 3.9 (π -structures). A π -structure is the result of the plumbing of a pre-net.

We will need to “plug” nets together, as it is done in interaction nets.

Definition 3.10 (Gluing). Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be pre-nets. Let $\mathcal{P}$ be the pre-net obtained by disjoint union of $\mathcal{P}_1$ and $\mathcal{P}_2$. We call *gluing* of $\mathcal{P}_1$ and $\mathcal{P}_2$, denoted by $\mathcal{P}_1|\mathcal{P}_2$, the plumbing of $\mathcal{P}$.

Graphically, a gluing ($\mathcal{P}|Q$) is depicted in Figure 8. It is clear that this operation is commutative and associative (up to ω -reduction and graph isomorphism).

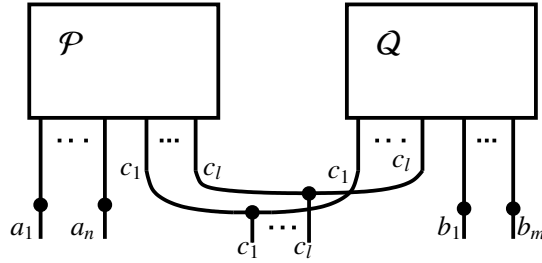


Figure 8: Representation of the gluing of two π -structures.

We need a strange rewriting rule on nets that makes a wire cross a cell in a certain way. An way to imagine it is to see a cell like a wall and a wire like an electrical

wiring. We want to hide the complex wiring behind the wall but make sure there is no short-circuit, and still remember the possible connections. For this, we first separate the wiring in positive and negative parts and only pull out through the wall two wires, that we plug together again.

Definition 3.11 (Pull through). Let $\mathcal{P}$ be a π -structure and c a cell. We can always suppose c to have k pairs of ports not connected to any wire. If not, we replace c by a cell with k more pairs of auxiliary ports, the rest remaining identical. Let w be a wire of $\mathcal{P}$.

- If w is a free cut: let (p_1, q_1) and (p_2, q_2) be two unused pairs of auxiliary ports of c and v be a new wire of arity 3.

Let $\mathcal{P}'$ be the pre-net $\mathcal{P} \cup c \cup v$ where

- w is split in w_+ and w_- ;
- $\text{fp}(w_+)$ is identified to p_1 ;
- $\text{fp}(w_-)$ is identified to p_2 ;
- $q_1, q_2 \in \partial(v)$;
- the third port of v is free.

- If w is a bounded cut: let (p_1, q_1) and (p_2, q_2) be two unused pairs of auxiliary ports of c and v be a new wire of arity 2.

Let $\mathcal{P}'$ be the pre-net $\mathcal{P} \cup c \cup v$ where

- w is split in w_+ and w_- ;
- $\text{fp}(w_+)$ is identified to p_1 ;
- $\text{fp}(w_-)$ is identified to p_2 ;
- $\partial(v) + \{q_1, q_2\}$.

- If w is any free non-cut wire: let (p, q) be an unused pair of auxiliary ports of c and v be a new wire of arity 2.

Let $\mathcal{P}'$ be the pre-net $\mathcal{P} \cup c \cup v$ where

- $\text{fp}(w)$ is identified to p ;
- $q \in \partial(v)$;
- the second port of v is free.

- If w is any bounded non-cut wire: let (p, q) be an unused pair of auxiliary ports of c and v be a new wire of arity 1.

Let $\mathcal{P}'$ be the pre-net $\mathcal{P} \cup c \cup v$ where

- p is added to $\partial(w)$;
- $\partial(v) = \{q\}$;

We then say that w was *pulled through* the cell c .

A graphical representation of the three steps of pulling through is shown in Figure 11(b) to 11(d) (page 13), but with several wires pulled through at once.

We can now define π -nets. A π -net represents a π -process. We therefore define them inductively on the construction of the process.

Definition 3.12 (π -nets). We define π -nets for each rule of the π -calculus grammar.

- The π -structure consisting of
 - one positive prefix cell α_0 (thus of arity 1) labeled $\bar{x}y$,
 - two free ports,
 - a wire of arity 2 connected to the principal port of α_0 and a free port,
 - a wire of arity 2 connected to the data port and a free port,

is a π -net, representing the summand $\bar{x}y.0$. (see Figure 9(a))

- The π -structure consisting of
 - one negative prefix cell β_0 (thus of arity 1) labeled $x(y)$,
 - one free ports,
 - a wire of arity 2 connected to the principal port of α_0 and a free port,
 - a wire of arity 1 connected to the data port,

is a π -net, representing the summand $x(y).0$. (see Figure 9(b)).

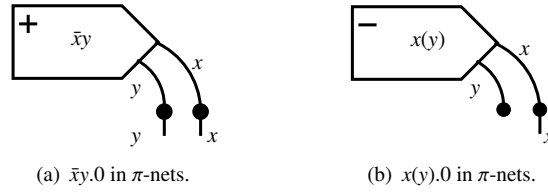


Figure 9: Representation of the processes $\bar{x}y.0$ and $x(y).0$ in π -nets.

- Let $\mathcal{P}$ and $\mathcal{Q}$ be two π -nets representing respectively the processes P and Q . Then the gluing of $(\mathcal{P}|\mathcal{Q})$ is a π -net, representing the process $P|Q$.
- Let $\mathcal{P}$ be a π -net representing the process P and z a free name of P . Let $\mathcal{P}'$ be the π -structure defined as the π -structure $\mathcal{P}$ where the free port labeled z (if it exists) has been removed and the wire connected to it is not anymore. This wire is relabeled (z) . Then $\mathcal{P}'$ is a π -net, representing the process $\nu z P$ (see Figure 10).

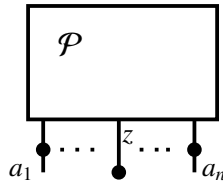


Figure 10: Representation of the process $\nu z P$ where $z \in \text{fn}(P)$.

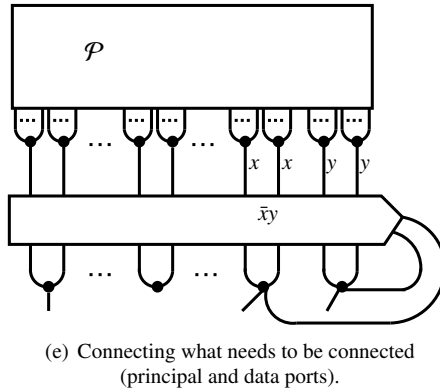
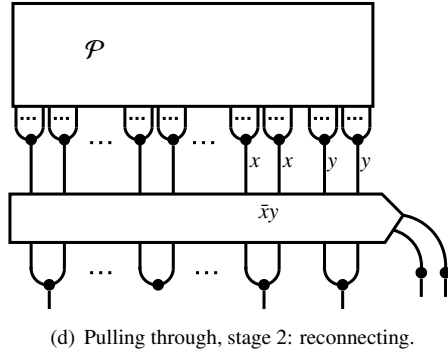
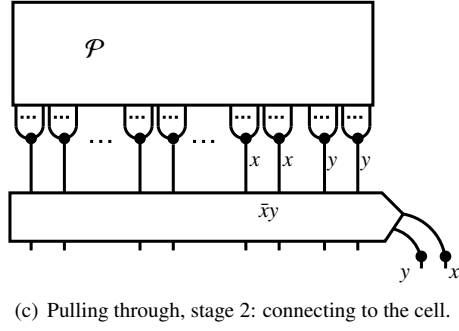
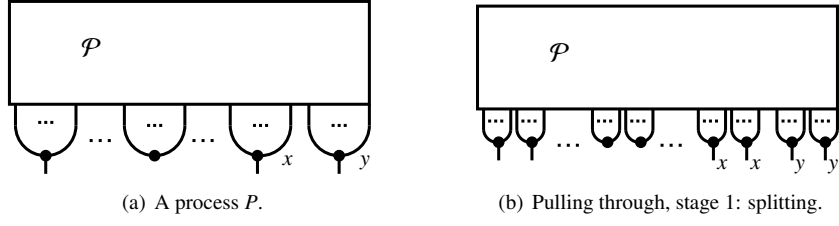


Figure 11: Construction of a representation of a guarded process in π -nets.

- Let $\mathcal{P}$ be a π -net representing the process P . Let n be the number of final wires of $\mathcal{P}$, and c a positive cell of type α_n , labeled $\bar{x}y$; let $\mathcal{P}_1$ be the π -structure $\mathcal{P}$ where all final wires have been pulled through c .

Let $\mathcal{P}_2$ be the pre-net defined as follows:

- $\text{fp}(\mathcal{P}_2) = \text{fp}(\mathcal{P}_1) \cup \{p_y, p_x\}$ where p_y is labeled y and p_x is labeled x ;
- the cells of $\mathcal{P}_2$ are the cells of $\mathcal{P}_1$ (e.g. including c);
- $\text{wires}(\mathcal{P}_2) = \text{wires}(\mathcal{P}_1) \cup \{w_y, w_x\}$ where $\text{ar}(w_y) = \text{ar}(w_x) = 2$, w_x links the principal port of c to p_x , and w_y links the data port of c to p_y .

Let $\mathcal{P}'$ be the plumbing of $\mathcal{P}_2$.

Then $\mathcal{P}'$ is a π -net, corresponding to the process $\bar{x}y.P$. Graphically, the construction of such a π -net is depicted in Figure 11.

- Let $\mathcal{P}$ be a π -net representing the process P . Let n be the number of final wires of $\mathcal{P}$, and c a negative cell of type α_n , labeled $x(z)$; let $\mathcal{P}_1$ be the π -structure $\mathcal{P}$ where all final wires have been pulled through c .

Let $\mathcal{P}_2$ be the pre-net defined as follows:

- $\text{fp}(\mathcal{P}_2) = \text{fp}(\mathcal{P}_1) \cup \{p_x\}$ where p_x is labeled x ;
- the cells of $\mathcal{P}_2$ are the cells of $\mathcal{P}_1$ (e.g. including c);
- $\text{wires}(\mathcal{P}_2) = \text{wires}(\mathcal{P}_1) \cup \{w_z, w_x\}$ where $\text{ar}(w_x) = 2$, w_x links the principal port of c to p_x , and
 - * if there exists a free port p_z of $\mathcal{P}_1$ labeled z , $\text{ar}(w_z) = 2$ and w_z links the data port of c to p_z (therefore, w_z becomes bounded!);
 - * if there is no free port of $\mathcal{P}_1$ labeled z , $\text{ar}(w_z) = 1$ and the only port of w_z is the data port of c (in which case, w_z is also bounded).

Let $\mathcal{P}'$ be the plumbing of $\mathcal{P}_2$.

Then $\mathcal{P}'$ is a π -net, corresponding to the process $x(y).P$.

We have now given a construction for π -nets corresponding to any π -calculus process. Some questions already arise. First of all, are all π -nets translations of π -processes? We can say right away that this is not the case. Is it possible then to characterize among all π -nets the ones that are translations of processes? Finally, given a net that is a translation, is it possible to find the process it is the translation of?

The first two questions we will discuss in part ??? **ATTENTION ATTENTION ATTENTION** [. The last one, we will need to prove the important theorem on correctness of this graphical representation towards π -calculus.

Finding the π -process of a π -net Lets first give some usefull definitions.

Definition 3.13 (Underlying graph). Let $\mathcal{P}$ be a π -net. Let $\mathcal{G}$ be the undirected graph defined as follows:

- the vertices of $\mathcal{G}$ are the cells of $\mathcal{P}$;
- there is a vertex between two vertices c_1, c_2 of $\mathcal{G}$ if and only if there is at least one local network w of $\mathcal{P}$ such that

- $\exists p_1 \in \partial(c_1)$ exit port of c_1 ;
- $\exists p_2 \in \partial(c_2)$ entry port of c_2 ;
- $p_1, p_2 \in \partial(w)$.

Proposition 3.14. 1. if $\mathcal{P} \rightarrow_\omega \mathcal{P}'$, then the underlying tree of $\mathcal{P}$ equals the underlying tree of $\mathcal{P}'$ (up to graph isomorphism).

2. the underlying tree of a π -net $\mathcal{P} = \pi.P$ is a tree, for any prefix π and process P .

PROOF: 1. Let w be a local network and c_1, c_2 be two ports of a net. If $w \rightarrow_\omega w'$, then

$$\partial(w) \cap \{\partial(c_1) \cup \partial(c_2)\} = \partial(w') \cap \{\partial(c_1) \cup \partial(c_2)\}.$$

2. It is proved by induction on the construction of $\mathcal{P}$ that its underlying graph is a forrest. In the case where $\mathcal{P}$ represents a process of the form $\pi.P$, the underlying graph is shown to be a tree of root the cell π in $\mathcal{P}$ and of maximal subforrest $\llbracket P \rrbracket$. $\square$

In this case we call *tree above π* the underlying tree of $\pi.\mathcal{P}$. By extension we can refer to the tree above any cell of a π -net. We call *process tree above a cell c* of $\mathcal{P}$ the maximal subnet (in the sens of pre-nets) of $\mathcal{P}$ containing all and only cells of the tree above c in the underlying graph of $\mathcal{P}$ where all exit, data and principal ports of c are connected by a wire to a free or bounded port respectively to the freeness of the wire they were connected to in $\mathcal{P}$. Finally, we call *process forrest above c* the maximal subnet of the process tree above c not containing c , considering entry ports of the disappeared c as free or bounded depending on the freeness of the network they belonged to. Notice that the process tree and forrest above a cell is usually a pre-net but not a π -net.

We extend in the natural way the notion of gluing on process trees and forrests.

Theorem 3.15. Let P be a π -process and $\mathcal{P}$ it's corresponding π -net. Then, there exist $n \in \mathbb{N}$, $\bar{a} = \{a_1, \dots, a_n\}$ a set of names, $m \in \mathbb{N}$, $\{P_1, \dots, P_m\}$ processes and $\{\mathcal{P}_1, \dots, \mathcal{P}_m\}$ π -nets such that:

- $P \equiv (\nu \bar{a})(P_1 \mid \dots \mid P_m)$;
- $\mathcal{P}$ is the gluing of $\{\mathcal{P}_1, \dots, \mathcal{P}_m\}$;
- for each $i \leq m$, $\exists \bar{b}_i \subseteq \bar{a}$ such that $\mathcal{P}_i = \llbracket (\nu \bar{b}_i)P_i \rrbracket$.

PROOF: By induction on the size of $\mathcal{P}$.

If $\mathcal{P}$ is an output cell c labeled $\bar{x}y$, where:

- the principal and data ports of c are connected to free ports, then $P = \bar{x}y.0$;
- the principal port of c is connected to a unary wire and the data port to a free port, then $P = \nu x \bar{x}y.0$;
- the principal port of c is connected to a free port and the data port to a unary wire, then $P = \nu y \bar{x}y.0$;
- the principal and data ports of c are connected to unary wires, then $P = (\nu x, y) \bar{x}y.0$;

- the principal and data ports of c are connected to the same binary wires, then $x = y$ and $P = \nu x \bar{x}x.0$;

If $\mathcal{P}$ is an input cell c labeled $x(z)$, where:

- the principal port of c is connected to a free port, $P = x(z).0$;
- the principal port of c is connected to a unary wire, $P = \nu x x(z).0$;
- the principal and data port of c are connected to the same binary wire, then $x = z$ and $P = \nu x x(x).0$.

Let $\mathcal{P}$ be a π -net of size at least 2.

Let F be the underlying forrest of $\mathcal{P}$ and $\{c_1, \dots, c_m\}$ the roots of it's trees. Let $\{\mathcal{P}'_1, \dots, \mathcal{P}'_m\}$ be the gluing of the process trees above $\{c_1, \dots, c_m\}$ respectively.

By construction of π -nets, $\mathcal{P}$ is the gluing of $\{\mathcal{P}'_1, \dots, \mathcal{P}'_m\}$ and each $\mathcal{P}'_i$ is the translation of a process P'_i .

By induction, each $\mathcal{P}'_i$ being a smaller π -net, $\forall i \leq m, P'_i \equiv (\nu \bar{a}_i)(P^i_1 \mid \dots \mid P^i_{k_i})$. Therefore

$$P \equiv (\nu \bar{a}_1 (P^1_1 \mid \dots \mid P^1_{k_1})) \mid \dots \mid (\nu \bar{a}_m (P^m_1 \mid \dots \mid P^m_{k_m})).$$

By structural congruence, there exist processes $P_1, \dots, P_m$ such that

$$P \equiv (\nu \bar{a})(P_1 \mid \dots \mid P_m).$$

Let us verify the three conditions are verified:

- $P \equiv (\nu \bar{a})(P_1 \mid \dots \mid P_m)$;
- $\mathcal{P}$ is the gluing of $\{\mathcal{P}_1, \dots, \mathcal{P}_m\}$;
- $\forall i \leq m$, let $\bar{b}_i \subseteq \bar{a}$ such that at least one port of $\mathcal{P}_i$ is labeled by each $b \in \bar{b}_i$. Then $\mathcal{P}_i = \llbracket (\nu \bar{b}_i)P_i \rrbracket$;

□

Remark 3.1. Such a correspondance of processes to a given π -net is made possible by the fact that we suppose each final wire of $\mathcal{P}$ to be labeled by a *different name*, especially restricted ones. If it wasn't the case, we couldn't bring restrictions so far out front of the process, when for example, its the restriction of a name shared by only two processes.

In practice, it is possible to actually give the set of restricted names and all the sub-processes, but not up to structural congruence, rather up to $\sim^c$. We will see later that for this last congruence (thanks to Eq. (2) of Remark 4.1) one can in some cases consider all bounded final names of $\mathcal{P}$ as being restricted.

But one has to be careful doing this, because separating process trees of a net, converting them by α -equivalence and gluing them back together does not necessarily yield the original π -net.

Interaction in π -nets We now give the reduction rules for π -nets. Two cells can interact when their respective principal ports are connected on a same wire. Such a pair of cells is called *critical pair*.

In π -calculus, two processes can interact if they are of “opposite” type: an input and an output. Therefore, we define a rule for reduction in the case two cells of opposite polarities are connected on the same wire, thus a cut wire.

TO DO: Explain that we can forget about the labeling of ports while reducing. We shall only do so if we want to keep track of the remaining names during several steps of reduction.

π -net reduction is defined by an infinite set of rules, that is $\forall m, n \in \mathbb{N}$, there is a rule of interaction between a cell of type α_m and a cell of type β_n . But since the two important ports involved in the reduction are the principal and the data ports, this infinity of rules can be given by a single scheme.

Definition 3.16 (Reduction). Let $\mathcal{P}$ be a π -net. We say that $\mathcal{P}$ π -reduces in $\mathcal{P}'$ and we write $\mathcal{P} \rightarrow_{\pi} \mathcal{P}'$ if $\mathcal{P}$ contains a critical pair (c_1, c_2) of respective types α_m, β_n such that the net rewriting rule given in Figure 12 applied to it yields a π -net $\mathcal{P}'$.

We say that $\mathcal{P}$ *reduces* in $\mathcal{P}'$, and we write $\mathcal{P} \rightarrow \mathcal{P}'$, if there exist two π -nets $\mathcal{P}_1$ and $\mathcal{P}_2$ such that

$$\mathcal{P} \sim_{\omega} \mathcal{P}_1 \rightarrow_{\pi} \mathcal{P}_2 \sim_{\omega} \mathcal{P}'.$$

We denote by $\rightarrow^*$ the transitive and reflexive closure of $\rightarrow$.

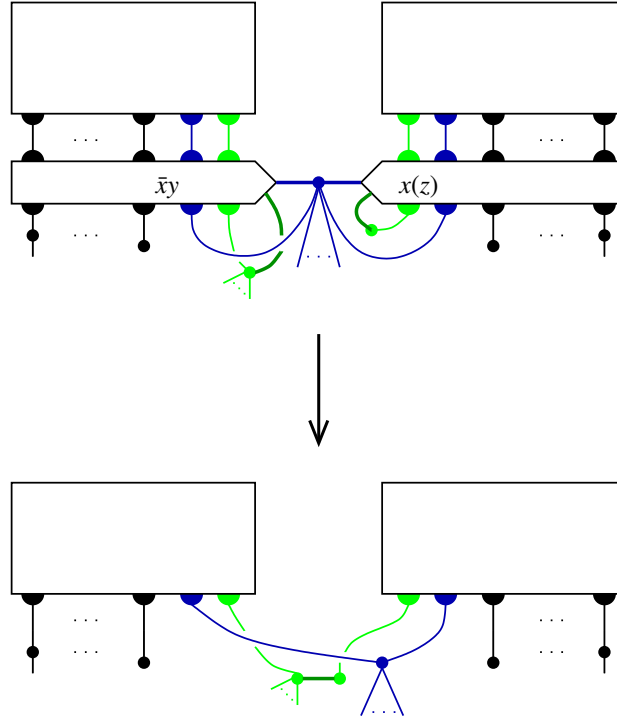


Figure 12: The cut rule of π -net calculus.

Let us describe informally what happens in that rule. But first, note that if y appears free in process on the right, the wire connected to the data port of $\bar{x}y$ is also connected

to some exit ports of $x(z)$. And therefore, will remain connected the the “residue” wire after reduction. It is only to not overload the picture that such a connection has not been shown.

The two cells involved in it disappear, letting freedom to couples of their auxiliary ports to reconnect (or if you prefer, revealing the connections that were hidden “inside” the cell). The principal ports of the two cells also disappear, thus decreasing of two the arity of the cut wire.

Finally, the data ports of the two cells are identified, thus connecting the network corresponding to the name send in output (here the generic y) with the network corresponding to the name waiting for it (here the bounded name z). This identifies the two names. In the case we wish to keep the labeling, all the ports of this “new” network should be labeled y .

We now give a couple of usual results on rewriting systems, applied to π -nets. It is not necessary since we prove later that this representation is correct in relation to π -calculus and these results become immediate. It is anyhow

Theorem 3.17 (Termination). *Every reduction sequence on a π -net $\mathcal{P}$ terminates (i.e. ends with a π -net with no cut wires).*

PROOF: Let $\text{size}(\mathcal{P})$ be the number of cells of $\mathcal{P}$. Then, if $\mathcal{P} \longrightarrow \mathcal{P}'$, $\text{size}(\mathcal{P}') = \text{size}(\mathcal{P}) - 2$. Therefore, the length of any reduction sequence of $\mathcal{P}$ is bounded by $\lfloor \text{size}(\mathcal{P})/2 \rfloor$. $\square$

3.2 Nets with boxes

4 π -nets calculus is the π -calculus

Structural congruence Structural congruence of π -calculus is really the most natural congruence. It defines in some way the operations of the language: commutativity of concurrency, for instance. Two concurrent processes have to be symmetrically considered at least, otherwise the idea of concurrency doesn’t hold.

In this paragraph, we show that the net representation of π -calculus identifies structurally congruent processes, which is exactly the purpose of defining structural congruence as an equality.

Theorem 4.1. *Let P, Q, R be three processes. Then the following isomorphisms hold:*

$$\begin{aligned} \llbracket P \mid (Q \mid R) \rrbracket &\simeq \llbracket (P \mid Q) \mid R \rrbracket \\ \llbracket P \mid Q \rrbracket &\simeq \llbracket Q \mid P \rrbracket \\ \llbracket P \mid 0 \rrbracket &\simeq \llbracket P \rrbracket \\ \llbracket \nu v \nu w P \rrbracket &\simeq \llbracket \nu w \nu v P \rrbracket \\ \llbracket \nu z 0 \rrbracket &\simeq \llbracket 0 \rrbracket \\ \llbracket \nu z (P \mid Q) \rrbracket &\simeq \llbracket P \mid \nu z Q \rrbracket, \text{ if } z \notin \text{fn}(P) \end{aligned}$$

PROOF: It is clear from the rule for net composition and restriction that the isomorphisms hold. Let us see as an example the detail of the last one.

Let $\mathcal{P}$ and $\mathcal{Q}$ be the π -nets of P and Q . Since z doesn’t appear free in P , $\llbracket P \mid Q \rrbracket$ is represented as follows:

Therefore, restricting z yields the following net:

which is obviously the same as restricting z in $\mathcal{Q}$ first, and then composing with $\mathcal{P}$, as shown in Figure 13. $\square$



Figure 13: Representation of $\nu z Q$ and $P \mid \nu z Q$

Remark 4.1. The graphical representation actually identifies a little more than just structural congruence. For instance

$$\llbracket \nu z \pi.P \rrbracket = \llbracket \pi.(\nu z P) \rrbracket$$

when z doesn't appear in π . But this is not a problem, since one can easily show that in this cases

$$\nu z \pi.P \sim^c \pi.(\nu z P) \quad (1)$$

In a certain way, such a congruence is so natural that it could be added to the axioms of structural congruence without betraying its purpose.

Also

$$\llbracket \nu z x(z).P \rrbracket = \llbracket x(z).P \rrbracket.$$

But here again,

$$\nu z x(z).P \sim^c x(z).P \quad (2)$$

TO DO: Proof of this.

ATTENTION ATTENTION ATTENTION [] A problem arises because

$$\llbracket \nu z \pi.P \rrbracket = \llbracket \pi.(\nu z P) \rrbracket$$

even when z is bounded in π (such as in $x(z).P$)!!

Correctness of π -nets

Theorem 4.2 (Correctness). *Let P, Q be two processes. Then*

$$P \rightarrow Q \quad \text{iff} \quad \llbracket P \rrbracket \rightarrow \llbracket Q \rrbracket.$$

Before proving this important theorem, let us prove a usefull lemma.

Lemma 4.3 (Corollary of Theorem 3.15). *Let $\mathcal{P}$ be a π -net containing a critical pair i, o (repectively the output and input cells of the pair). Then there exist $\bar{a}, \bar{r}$ and $\bar{s}$ sets of names, x, y, z names and P, R, S, T processes such that $\mathcal{P}$ is the translation of P and*

$$P \equiv (\nu \bar{a})(\bar{x}y.R \mid x(z).S \mid T).$$

PROOF:

Fact . *By construction of π -nets, a critical pair can only be connected to a final wire.*

Let F be the underlying forrest of $\mathcal{P}$ and $\{c_1, \dots, c_m\}$ the roots of it's trees. By the previous fact, $\{i, o\} \subseteq \{c_1, \dots, c_m\}$. Without loss of generality, we can consider $\{i, o\}$ to be the first two cells of that list.

Let π_i and π_o be the labels of respectively i and o .

Let P' be the process obtained by the first point of Theorem 3.15.

Let $\{\mathcal{R}', \mathcal{S}', \mathcal{P}_3, \dots, \mathcal{P}_m\}$ be the subnets of $\mathcal{P}$ and $\{R', S', P_3, \dots, P_m\}$ be the processes obtained by the same theorem.

Let $T = P_3 \mid \dots \mid P_m$. $\mathcal{R}'$ and $\mathcal{S}'$ are process trees, so they are of the form of a π -net guarded by a cell. Using Theorem 3.15 again and the fact that they are guarded π -nets, we have:

$$\mathcal{R}' = \llbracket (\nu \bar{r})\pi_o.R \rrbracket \quad \text{and} \quad \mathcal{S}' = \llbracket (\nu \bar{s})\pi_i.S \rrbracket.$$

Considering all names are different (since all final wires of $\mathcal{P}$ are labeled differently), by α -equivalence, we can “extract” the restrictions out to the top and conclude the proof. $\square$

Let us now return to the theorem.

PROOF OF THE THEOREM 4.2: Let P and Q be two processes.

[$\Rightarrow$] For each derivation rule, we show that if the premise holds for the π -net translations of P and Q , the conclusion holds for the translations of the processes involved in the conclusion.

π -red. Let P be of the form $\bar{x}y.S \mid x(z).R$, then $Q \equiv S \mid R\{y/z\}$ (since $P \rightarrow Q$). Let $\mathcal{P}, Q$ be the π -nets corresponding to P and Q respectively.

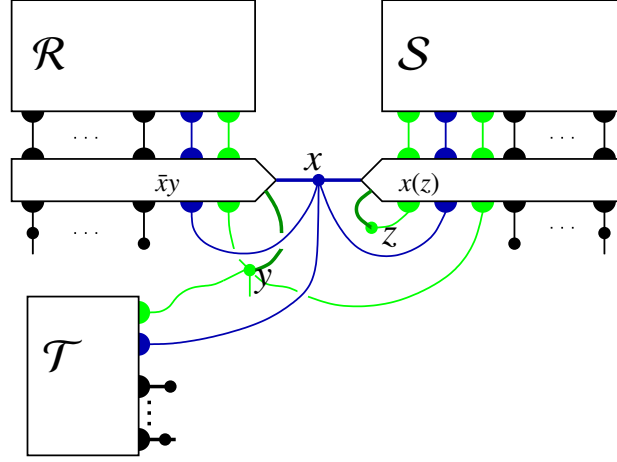
Let o, i be the cells of $\mathcal{P}$ corresponding to the prefixes $\bar{x}y$ and $x(z)$ respectively. Then $\mathcal{P}$ consists of the gluing of the process trees $\mathcal{S}', \mathcal{R}'$ above o and i respectively. The principal ports of i and o are connected to a same (free) wire.

Let S and R be the process forrests above o and i respectively.

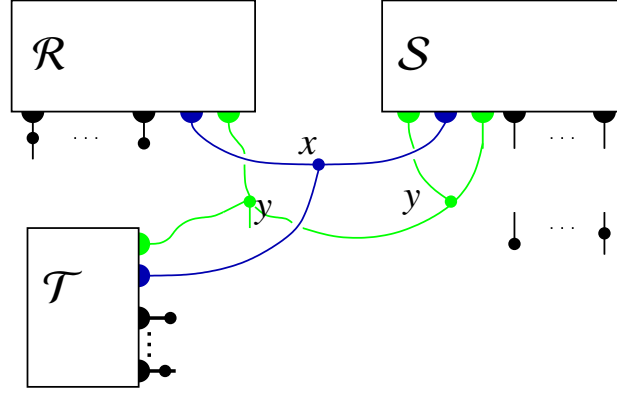
Then Q consists of the gluing of S and $\mathcal{R}''$ where $\mathcal{R}''$ is the net $\mathcal{R}$ where the final wires labeled z has been relabeled y . This means exactly that $\mathcal{R}'' = \llbracket R\{y/z\} \rrbracket$, so $Q = \llbracket S \mid R\{y/z\} \rrbracket$.

Parr. Let P, Q and R be three processes such that $P \rightarrow Q$. Let $\mathcal{P} = \llbracket P \rrbracket$, $\mathcal{Q} = \llbracket Q \rrbracket$ and $\mathcal{R} = \llbracket R \rrbracket$ and lets suppose that $\mathcal{P} \rightarrow \mathcal{Q}$.

Let $\mathcal{P}'$ be the gluing of $\mathcal{P}$ and $\mathcal{R}$. The reduction rule of π -nets being local, the reduction of the critical pair in $\mathcal{P}$ does not affect $\mathcal{R}$ in $\mathcal{P}'$. It is therefore clear that $\mathcal{P}'$ reduces into the gluing of $\mathcal{Q}$ and $\mathcal{R}$, which is the translation of $Q \mid R$.



(a) Representation of P .



(b) Representation of an ω -expanded Q (the final wire labeled y has been expanded into a local network of two wires).

Figure 14: $\mathcal{P}$ reduces in Q in the deduction rule (π -reduct.).

Restr. Let P and Q be two processes such that $P \rightarrow Q$. Let $\mathcal{P} = \llbracket P \rrbracket$ and $Q = \llbracket Q \rrbracket$ and lets suppose that $\mathcal{P} \rightarrow Q$.

- if x is bounded in P , then, if it exists in Q , it is bounded in Q and $\llbracket (\nu x)P \rrbracket = \mathcal{P}$ and $\llbracket (\nu x)Q \rrbracket = Q$, so the result is immediate.
- if x is free in P , then, if it (still) belongs to Q , it is free in Q . Then $\llbracket (\nu x)P \rrbracket$ (respectively $\llbracket (\nu x)Q \rrbracket$) is the net $\mathcal{P}$ (respectively Q) where the final wire labeled x has been “plugged”.

Three possibilities can occur. x can be the label of the principal ports of the critical pair, or it can be the label of the data port of the output cell of the critical pair, or it can be the label of any other free final wire of P . By examining the three cases, one can easily see that bounding x in $\mathcal{P}$ and reducing $\mathcal{P}$ will yield Q with a bounded x , translation of $(\nu x)Q$.

Struct. We have seen already that the π -net translation identifies structurally congruent

processes (up to graph isomorphism). So $\llbracket P' \rrbracket = \llbracket P \rrbracket$ and $\llbracket Q' \rrbracket = \llbracket Q \rrbracket$ and the proof is immediate.

[$\Leftarrow$] Let P and Q be two processes. Let $\mathcal{P} = \llbracket P \rrbracket$ and $\mathcal{Q} = \llbracket Q \rrbracket$ and suppose that $\mathcal{P} \rightarrow \mathcal{Q}$.

This means that there is a critical pair of cells in $\mathcal{P}$. Let i and o be the input and output cells of this pair, and π_i and π_o the prefixes of P that they represent. From Lemma 4.3, we have

$$P \equiv (\nu \bar{a})(\bar{x}y.R \mid x(z).S \mid T).$$

Four possibilities can occur depending on whether the name and co-name of the output cell of the critical pair are restricted or not. This will be seen in the corresponding processes by the fact that $x \in a$ or not and $y \in a$ or not. Let us see only one case, the others being similar. The process P of which $\mathcal{P}$ is the translation in each case is given in Table 3 and we leave to the reader the verification of the other three cases.

Let $\mathcal{R}$ be the process forrest above o , $\mathcal{S}$ the one above i and $\mathcal{T}$ the remaining process forrest. By definition of π -nets, $\mathcal{P}$ is necessarily the gluing of three subnets as shown in Figure 14(a).

By Lemma 4.3, there are three processes R, S and T such that

$$P \equiv (\nu \bar{a}, x)(\bar{x}y.R \mid x(z).S \mid T).$$

This critical pair is eliminated and yields the π -net shown in Figure 14(b). By reduction rule, Q is the gluing of $\llbracket R \rrbracket, \llbracket S \rrbracket$ where the label z has been replaced by the label y and $\llbracket T \rrbracket$. Here again, by construction rules of π -nets, it means that Q is the translation of a process Q such that $Q \equiv (\nu \bar{a}, x)(R' \mid S' \mid T')$. What exactly are these processes?

$T' = T$, since $\mathcal{T} = \mathcal{T}'$. S' is represented by $\mathcal{S}'$ which is $\mathcal{S}$ where the network labeled z is now linked to what was the network of R labeled y , thus all occurrences of S that were z -s are now y -s. So $S' = S\{y/z\}$. Finally, $\mathcal{R}' = \mathcal{R}$, so $R' = R$.

We can easily exhibit a derivation tree for

$$(\nu \bar{a}, x)(\bar{x}y.R \mid x(z).S \mid T) \longrightarrow (\nu \bar{a}, x)(R \mid S' \mid T) :$$

$$\frac{\frac{\frac{\bar{x}y.R \mid x(z).S \longrightarrow R \mid S'}{\bar{x}y.R \mid x(z).S \mid T \longrightarrow R \mid S' \mid T}}{(\nu \bar{a}, x)(\bar{x}y.R \mid x(z).S \mid T) \longrightarrow (\nu \bar{a}, x)(R \mid S' \mid T)}}$$

where the double line represents several occurrences of the Restr. derivation rule. $\square$

5 Extension to promotion

	“ x free”	“ x restricted”
“ y free”	$(\nu \bar{a})(\bar{x}y.P \mid x(z).Q \mid T)$	$(\nu \bar{a}, x)(\bar{x}y.P \mid x(z).Q \mid T)$
“ y restricted”	$(\nu \bar{a}, y)(\bar{x}y.P \mid x(z).Q \mid T)$	$(\nu \bar{a}, x, y)(\bar{x}y.P \mid x(z).Q \mid T)$

Table 3: Four possibilities for a reducible π -net
(where x, y don’t appear in $\bar{a}$).