Representing the Finite π-calculus in Multi-Interaction Nets

Concurrency = Interaction + Non-determinism

Vladimir Alexiev*

615 GSB, Department of Computing Science,
University of Alberta, Edmonton AB T6G 2H1, Canada
phone (403) 492 3854, fax (403) 492 1071, email vladimir@cs.ualberta.ca

Abstract. We extend the Interaction Nets of Lafont (1990) with some non-determinism capabilities, and then show how to implement the finite monadic π-calculus in that system.

1 Introduction

The π-calculus of Milner *et al.* (1992) is one of the most popular theoretical tools for the investigation of concurrent computations. Its popularity is due to its conceptual simplicity, yet great expressive power. However, similar to the λ-calculus, the π-calculus leaves some of the basic low-level components of computation implicit, namely the distribution of values and synchronization, expressed as the global process of substitution. Uncovering this finer computational structure, and dispensing with the important role that syntactic entities like names play in π-calculus, is the goal of this paper.

Interaction Nets (IN) of Lafont (1990) are a novel model of parallel computation, simple and elegant. Their essential properties as relating to parallelism are the locality of interaction and the simplicity of the rewriting process. We introduce a version of IN extended with non-determinism (Multi-Interaction Nets) and translate faithfully the π-calculus to MIN.

Our translation is similar to the π-nets of Milner (1994) and the Interaction Diagrams of Parrow (1995), but we represent blocking (synchronization) in a distributed manner, without using boxes. We also compare our construction to the Concurrent Combinators of Honda and Yoshida (1994a).

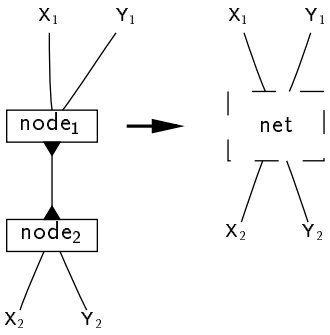
2 Interaction Nets and Non-Determinism

Interaction Nets (IN) were introduced by Lafont (1990) as a simple and elegant model of parallel computation, inspired by Linear Logic's Proof Nets, and designed to be useful both as an (abstract) programming language, and as a useful intermediate representation.

* Supported by a University of Alberta Dissertation Scholarship.

INs are graphs consisting of *nodes* (agents) of certain types and undirected *edges* connecting them. The points of contact of edges and nodes are called *ports* and every node type has a particular signature of ports. In conventional INs every node type has exactly one *principal port* (denoted with a bold triangle), and every port hosts exactly one edge.

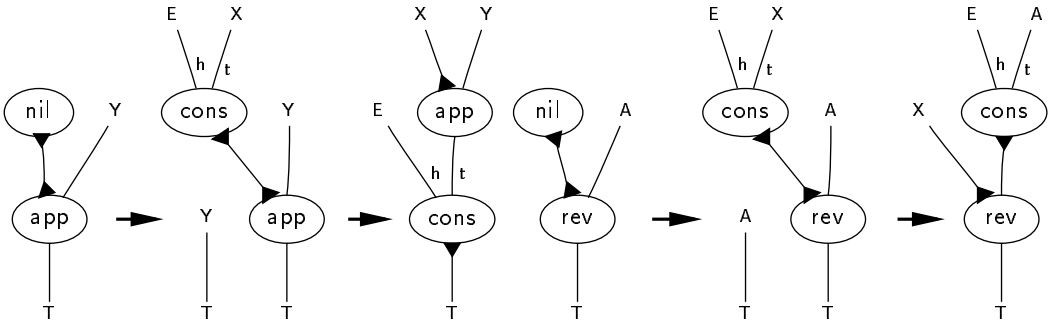
Computation in INs is governed by *interaction rules* of the form



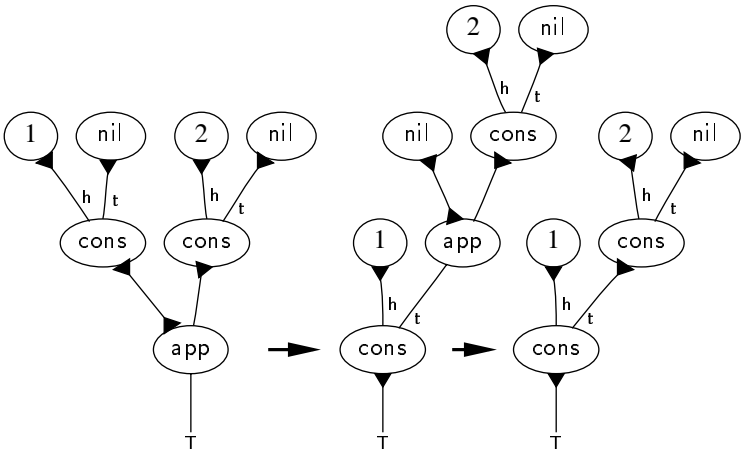
Notation "Generic" node types are indicated with boxes. Whole subnets are indicated with a large dashed boxes. The concrete node types that we will use predominantly in this paper are indicated with ovals.

The left hand side (LHS) of a rule is a *cut*: two nodes facing each other with their principal ports (which we often denote as $\mathsf{node}_1 \bowtie \mathsf{node}_2$). The right hand side (RHS) is a net containing any number of nodes, possibly none. The only requirement is that the RHS should connect all free ends of the LHS, leaving no loose ends.

If the LHS of a rule (a *redex*) is present in a particular net, then the LHS can be removed from the net, and replaced by (rewritten with) the RHS. For example, here are some simple list processing rules (append, and reverse using an accumulator):



and the reduction corresponding to appending the lists $(1, \mathsf{nil})$ and $(2, \mathsf{nil})$ to obtain the list $(1, 2, \mathsf{nil})$:



Interaction Nets are a restricted form of graph rewriting enjoying some nice properties:

Binary Interaction Only two nodes participate in a single rewriting step.
Simplicity It is easy to apply a rule, because the application is uniform.
Locality The applicability of a rule is determined by local inspection, and the effect of a rule is a local modification of the net.

Furthermore, conventional INs are (trivially) **confluent** because of the following properties:

- A redex consists of only two nodes and their connecting edge.
- A node has only one principal port, therefore can participate in at most one redex.
- Reduction is completely local in that it only changes a redex, but leaves the rest of the net untouched.
- There is only one rule matching a given redex.[1]

Therefore every two redexes are necessarily disjoint, and no critical pairs are possible. This, together with the locality of interaction, means that the reduction of a given redex can never preclude the reduction of another redex, therefore the two reductions can be done in either order, or in parallel. This implies a strong form of confluence, the diamond property in single-step reductions:

$$\begin{array}{ccc} \forall N & \xrightarrow{R_1} & \forall N_1 \\ {\scriptstyle R_2}\downarrow & & \downarrow{\scriptstyle R_2} \\ \forall N_2 & \xrightarrow{R_1} & \exists Z \end{array}$$

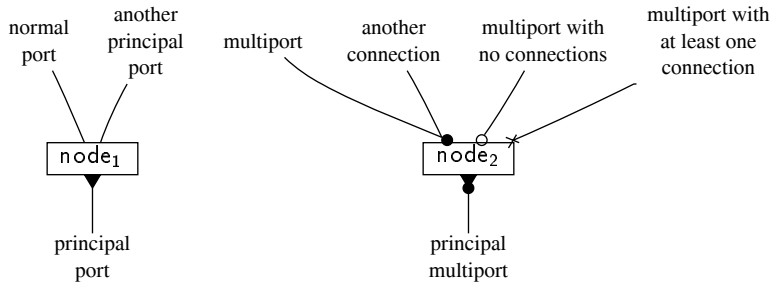
[1] If the redex is symmetric (consists of two nodes of the same type) then the RHS of the rule must also be symmetric, so that no matter how the rule is instantiated, the outcome is the same.

Not only all normalizing reduction sequences lead to the same result, but they even all have the same length. Every net can evolve in essentially only one way.

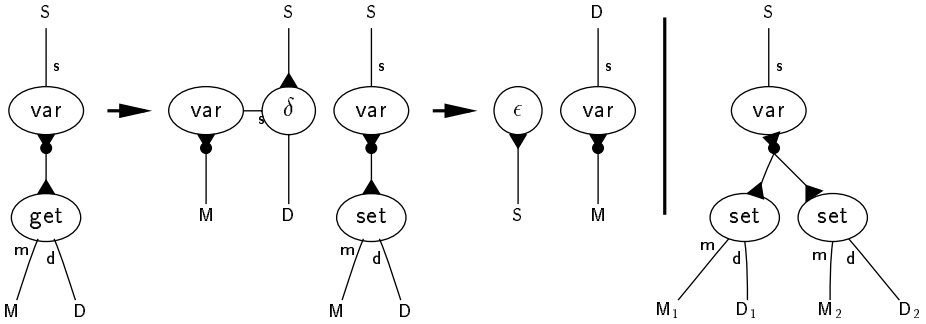
2.1 Non-Determinism

The strong confluence of conventional INs account to a large extent for their simplicity and attractiveness, and for some of their deeper theoretical aspects such as deadlock-freeness of certain syntactically-identifiable fragments. However, it also means that IN cannot be used to model and implement **non-deterministic** systems such as concurrent object-oriented systems or the π-calculus. In such systems an object/agent/process typically can be connected to more than one other agent, and the interaction with one of them may well preclude interaction with others. For example, consider the process $c!a, c!b, c?x.P$.[2] It can reduce to either $c!b, P[a/x]$ or $c!a, P[b/x]$ which are not equivalent if a and b are different names.

We therefore introduce an extended version of IN that we call *Multi-Interaction Nets* (MIN), in which a node may have more than one principal port, and a port may have zero or more edges, if it has been designated as a *multiport*.



We can represent typical object-oriented notions such as state change like this:

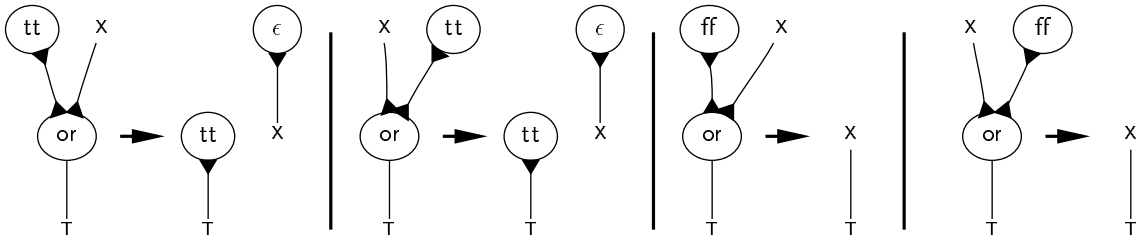


Here var accepts get/set requests from its principal (message) port, and stores a private state at its s port. The request nodes get and set have a port d for the data item and another port m for the next message to the same "object". (The eraser ϵ and duplicator δ are standard IN components.) The evolution of

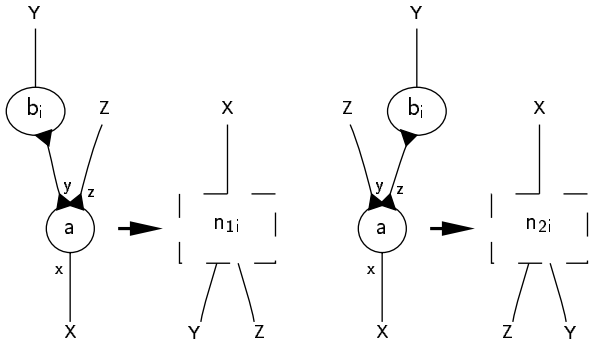
[2] See §3 for the π-calculus notation that we use.

a var that has two competing requests (as shown in the RHS of the figure) may depend on the order in which these requests are served.

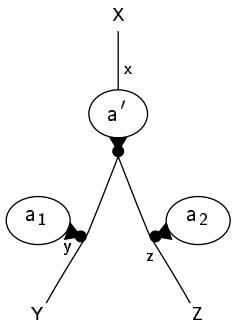
There are several ways of introducing non-determinism in IN, which I explore in my forthcoming PhD dissertation. For this paper we will use INs extended with multiports as introduced above (INMP), and will also allow multiple principal ports per node (INMPP). Here is an example of an INMPP:



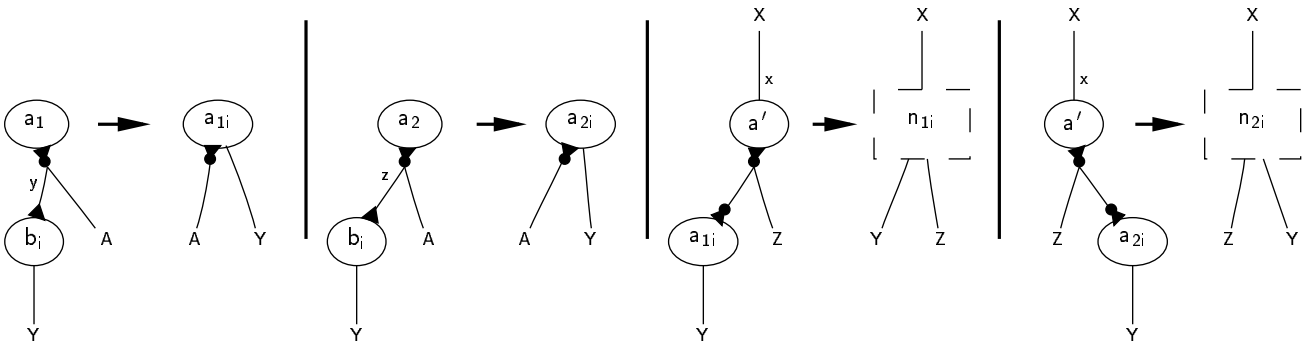
Capturing INMPP in INMP As will be seen later, we find it very convenient to use multiple principal ports in our translation of the π-calculus, because of the ability of INMPP have a node be attentive towards several ports simultaneously. We note briefly that it is in fact possible to reduce INMPP to INMP, by splitting a node with more than one principal port in parts. For simplicity, we show here the case of two principal ports. Let a be a node type with two principal ports y and z and let its reductions with different node types b_i be given by the rules



Then we translate $\mathsf{a}(X, Y, Z)$ to the following configuration



where the auxiliary markers a_1 and a_2 are governed by the rules



By the above construction, we could limit our version of IN to INMP only (not INMPP). But for simplicity, we will use INMPP too, and we call the combination MIN=INMP+INMPP.

2.2 Special Rule Selection

Rule selection in conventional IN is quite simple: only one rule may apply to a redex, and a node may be involved in only one redex. Rule selection in MIN is more complicated. First a principal port of a node is chosen non-deterministically. If it is a multiport then one of its edges is chosen non-deterministically. If the other end of the edge is a principal port, then there may be several rules matching the redex.

In order to accommodate the need for special clean-up rules, we allow to further *qualify* the rules by constraints on the current arities of ports.[3] We consider the following constraints (see the figure in the beginning of §2.1):

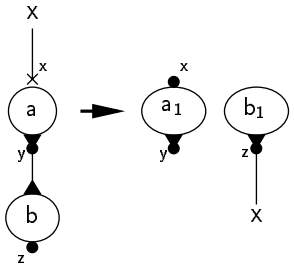
$\bullet$ no constraint $\quad \circ$ zero edges[4] $\quad \times$ at least one edge.

The rule satisfying the largest set of most-specific constraints is chosen (the specificity ordering being $\bullet < \circ, \bullet < \times\}$). We will use only rule sets that are well-ordered by specificity. For an example of the use of such special rules, see §4.8.

A MIN rule can only specify the evolution of a finite number of edges, but the total number of edges on the LHS is not bound a-priori, if the LHS contains multiports. Therefore, it is implied that any edges not specifically mentioned in the LHS are transferred en-masse to ports of the same name on the RHS. For example, here the edge X—a.x of the LHS is explicitly transferred to X—b_1.z of the RHS. In addition, all other potential edges connected to a.x are transferred to a_1.x, those of a.y to a_1.y, and those of b.z to b_1.z.

[3] Not necessarily the principal ports involved in the cut.

[4] Obviously the principal ports involved in the cut have at least one edge, so the constraint means "one edge" if it is applied to such a port.



3 The π-calculus

In this paper we consider the finite part of the monadic π-calculus (*i.e.* no choice and replication).[5] Let $a, b, \ldots, x, y, \ldots$ stand for *channel names*. Then *processes* $P, Q, \ldots$ are defined inductively as follows

Zero 0 is the empty (do-nothing) process.

Parallel Composition P, Q behaves as both P and Q, possibly interacting with each other.

Output Prefix $c!v.P$[6] sends the value v along the channel c, then behaves as the process P.

Input Prefix $c?x.P$ receives a value from channel c, then behaves as the process P, wherein the value is substituted for x.

Hiding/Restriction $(c)P$ is just like P, but the channel c is hidden: no values can be sent/received over that channel by other processes.

In $c!v$ and $c?v$, c is in *subject* position, while v is in *object* position. We will sometimes denote a prefix as π. We abbreviate $\pi.0$ to simply π. Free and bound names are defined as usual (x becomes bound in $c?x.P$ and $(x)P$). $a!(x).P$ is used as an abbreviation for $(x)a!x.P$.

3.1 Structural Congruence

A natural way to simplify the presentation of the reduction system of the π-calculus (see below) is to introduce structural rules which describe equivalence classes of process terms that are essentially the same, and only differ in their syntactical presentation. $\equiv$ is the smallest congruence satisfying the rules below:

1. $P \equiv P[y/x]$[7] if x is not free in P and y is fresh (α renaming). We will tacitly assume for the rest of this paper that every bound name is different (renamed apart) from every free name and from all other bound names.

[5] Please note that *e.g.* Honda and Yoshida (1994a) take a similar approach, and later develop their combinatory system to accommodate these elements in (Honda and Yoshida, 1994b).

[6] We use this Occam-like notation instead of the conventional notation $\bar{c}\langle x\rangle.P$, mainly because it is more clear in the typescript.

[7] Here $[y/x]$ denotes the substitution of y for x.

2. $0, P \equiv P; P, Q \equiv Q, P; (P, Q), R \equiv P, (Q, R)$: the parallel constructor makes a commutative monoid.
3. $0 \equiv (x)0$: there is nothing to hide in 0.
4. $(x)P, Q \equiv (x)(P, Q)$ if x does not occur in Q: the scope of a restriction can be extended if there's nothing to hide. This also implies that $Q \equiv (x)Q$ given the same proviso.
5. $(x)(y)P \equiv (y)(x)P$.

By the last three rules, we can assume w.l.o.g. that all restrictions are pushed out to the top level.

3.2 Reduction Rules

We use an (unlabeled) reduction system for the π-calculus, because we find it more natural for our correctness proofs below than a labeled transition system. It is the smallest relation $\rightarrow$ closed under the rules

Comm $a?x.P, a!c.Q \rightarrow P[c/x], Q$: this rule, analogous to β-reduction, is the essence of the π-calculus. It allows a process to communicate a name to another process.

Struc If $P \rightarrow Q$ and $P \equiv P'$, $Q \equiv Q'$ then $P' \rightarrow Q'$. This incorporates the structural congruence into the reduction relation.

Par If $P \rightarrow Q$ then $P, R \rightarrow Q, R$: adding a parallel component does not decrease the possibilities for reduction.

Res If $P \rightarrow Q$ then $(x)P \rightarrow (x)Q$: restriction does not decrease the possibilities for reduction of the internal process.

Multi-step reduction is defined as $\Rightarrow \stackrel{\text{def}}{=} \stackrel{*}{\rightarrow} \cup \equiv$.

3.3 The Many Roles of Prefix

The **Comm** rule plays several important roles that we would like to implement in IN as independently as possible:

Value passing The value c is delivered from the sender to the receiver process.

Value distribution If the input variable x appears in several places in the receiver P, the value c should be delivered to all these points.

Synchronization Reduction is not a congruence over prefix, so the prefixed process(es) cannot perform any other action before the outer communication is complete.

The prefix blocks any possible interactions inside the prefixed process, as well as interactions between it and other processes. Synchronization in the π-calculus is *global* with respect to the prefixed process. We will see below that our translation into IN implements synchronization in a distributed manner.

4 Representing the π-calculus in MIN

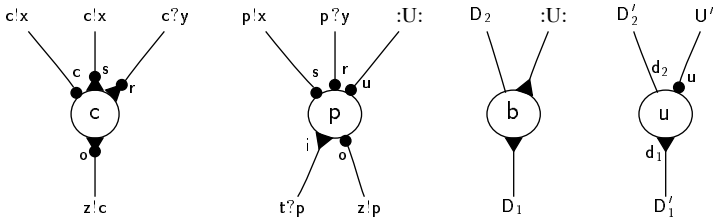
Our motivation for attempting to represent the π-calculus in MIN is threefold:

- To test the expressive power of an extension of IN that we found interesting for other purposes as well (*e.g.* concurrent object-oriented programming).
- To represent explicitly some aspects of the dynamics of computation of the π-calculus that are left implicit by the standard formulations (substitution).
- To capture the mobility features of the π-calculus in a finitary (name-free, combinatory) framework.

We first introduce some basic building blocks for representing the π-calculus (we call the particular MIN instance MIN_π), then give a translation $[\![\cdot]\!]$ from Π to MIN_π, and finally sketch some correctness results about the translation.

4.1 MIN_π Nodes

The node types of MIN_π are classified as *primary* (those that can appear in the translation $[\![P]\!]$ of a process) and *auxiliary* (used only during intermediate computation steps). The main nodes are:



A brief description of the main nodes follows:

channel c corresponds to a π-calculus channel name. Its output multiport o connects it to all channels z for which c is an output object. The send multiport s connects it to all nodes x that are output objects of c. The committed multiport c is similar to s, but it holds only output objects x for which we are certain that they are ready to be sent. The receive multiport r connects c to all nodes y that are input objects of c.

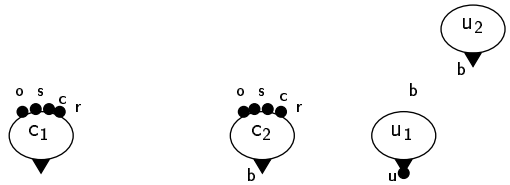
placeholder p [8] corresponds to the bound variable in a π-calculus input prefix, *i.e.* it is the input object of some node. Its ports are similar to the ports of c, but you will notice that its only principal port is input i, because p cannot interact until it has been instantiated with a channel in a communication. It also has an unblock port u that sends a signal to certain blocking (synchronization) machinery that is dependent on the prefix.

[8] This term comes from (Parrow, 1995).

blocker b is part of the synchronization machinery. It will shortcut its data ports when it receives an unblocking signal on its principal port.

unblocker u will also shortcut its data ports d_1 and d_2 when the time is right, and will unblock all blockers that are attached to its u port.

We also use auxiliary nodes of the following types



They are mostly used as intermediate states in the evolution of main nodes (and thus serve to sequentialize this evolution).

4.2 Labels

Instead of denoting channel and placeholder nodes with their node types c and p, we *label* the nodes with the π-calculus names that they correspond to, *e.g.* we draw a c node labeled a as an oval containing a (the port types will always allow us to determine the node type unambiguously), and denote it in text as c[a]. If a name is hidden in a π-calculus process, we place the label in parentheses, *e.g.* p[(x)] or no label at all. (For brevity, we don't always parenthesize placeholder labels, which should always be considered hidden). These labels have several uses:

- As a convenience for the reader.
- The incremental building of a net during the translation process uses the labels to decide which nodes to merge.
- They make finer distinctions between nets, so that *e.g.* the nets corresponding to $a!b$ and $a!c$ are considered different, even though they are graphically the same.

However, the labels play *no role* in the MIN_π reduction process.

Definition 1 (Labeled MIN_π isomorphism). *Two MIN_π M and N are called isomorphic, $M \approx N$, if there is a graph isomorphism between them that also respects the node labels (hidden/placeholder labels are excluded from this distinction).*

4.3 The Translation

We define a translation function $[\![\cdot]\!]: \Pi \mapsto \text{MIN}_\pi$ from π-calculus processes P to MIN_π nets N (*i.e.* $N = [\![P]\!]$). Simultaneously with it we define a set of blocking points $\mathcal{B}[\![P]\!]$, which are ("the middles of") edges of N. As explained in §4.2, c

nodes of $[\![P]\!]$ bear labels throughout the translation process, but these labels do not influence the MIN_π reduction process. Let $n(N)$ and $e(N)$ denote the nodes and edges of N, and $n(M) + n(N)$ denotes amalgamated sum, where nodes in $n(M)$ and $n(N)$ with the same label are identified. We also write $N = (n, e)$ for $n = n(N)$, $e = e(N)$.

We define the translation by induction on the structure of P (§3), but make a distinction between atoms $\pi.0$ and proper prefixes $\pi.P$.

Zero $[\![0]\!] = (\emptyset, \emptyset)$ and $\mathcal{B}[\![0]\!] = \emptyset$.

Parallel Composition $[\![P, Q]\!] = (n[\![P]\!] + n[\![Q]\!], e[\![P]\!] \cup e[\![Q]\!])$ and $\mathcal{B}[\![P, Q]\!] = \mathcal{B}[\![P]\!] \cup \mathcal{B}[\![Q]\!]$. Nodes of the same label are identified, and their edge sets are merged, but no edges are identified.

Hiding/restriction $[\![(c)P]\!] = [\![P]\!]\backslash\mathsf{c}$ ($[\![P]\!]\backslash\mathsf{c}$ is the same as $[\![P]\!]$, but the label of node c (if any) is erased/parenthesized). $\mathcal{B}[\![(c)P]\!] = \mathcal{B}[\![P]\!]$.

Input Atom $[\![a?x]\!]$[9]$= (\{\mathsf{c[a]}, \mathsf{p[(x)]}\}, \{\mathsf{a.r{-}x.i}\})$ (channel, placeholder, and an edge connecting their r and i ports respectively). $\mathcal{B}[\![a?x]\!] = \{\mathsf{a.r{-}x.i}\}$.

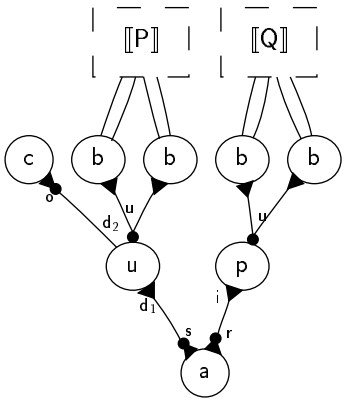
Output Atom $[\![a!x]\!]$[10]$= (\{\mathsf{c[a]}, \mathsf{c[x]}\}, \{\mathsf{a.s{-}x.o}\})$ (two channels and an edge connecting their s and o ports). $\mathcal{B}[\![a!x]\!] = \{\mathsf{a.s{-}x.o}\}$.

Input Prefix $[\![a?p.Q]\!]$ is defined from $[\![Q]\!]$ thus (see the right half of the figure below):[11]

1. If a channel node labeled a is not present in $[\![Q]\!]$, a new channel a is added.
2. If a channel node labeled p is not present in $[\![Q]\!]$, a new placeholder p is added. Otherwise, p's type is demoted from channel to placeholder. (In this latter case, effectively the placeholder p is merged to a channel in $[\![Q]\!]$, in contrast to the Parallel Composition case above.)
3. A new edge a.r−p.i is added.
4. Blockers b are inserted in every edge of $\mathcal{B}[\![Q]\!]$ and their principal ports are connected to p.u.

More formally, $n[\![a?p.Q]\!] = (n[\![Q]\!] + \mathsf{c[a]} + \mathsf{p})\backslash\backslash\mathsf{p} \cup \bigcup_{e \in \mathcal{B}[\![Q]\!]} \mathsf{b}_e$ where $n\backslash\backslash\mathsf{p}$ is like n, but with the type of node p demoted from channel to placeholder and its label erased. $e[\![a?p.Q]\!] = e[\![Q]\!] \cup \{\mathsf{a.r{-}p.i}\} \cup \bigcup_{e \bowtie (n_1, n_2) \in \mathcal{B}[\![Q]\!]} \{\mathsf{p.u{-}b}_e, \mathsf{b}_e\mathsf{.d_1{-}n_1}, \mathsf{b}_e\mathsf{.d_2{-}n_2}\}$. Finally, $\mathcal{B}[\![a?p.P]\!] = \{\mathsf{a.r{-}p.i}\}$, and the fact that it is a singleton set is the key to the polynomial nature of our translation.

[9] More precisely, $[\![a?x.P]\!]$ where $P \equiv 0$.
[10] More precisely, $[\![a!x.P]\!]$ where $P \equiv 0$.
[11] Please imagine that a and c can potentially be immersed in $[\![P]\!]$, and a and p can be immersed in $[\![Q]\!]$.



Output Prefix $[\![a!c.P]\!]$ is defined similarly, but an extra node u is added, because c may be the output subject of more than one prefix (see the left half of the figure):

1. New channels a and c are added if not present in $[\![P]\!]$.
2. An unblocker u is added.
3. New edges $\mathsf{a.s}{-}\mathsf{u.d_1}$ and $\mathsf{u.d_2}{-}\mathsf{c.o}$ are added.
4. Blockers b are inserted in every edge of $\mathcal{B}[\![P]\!]$ and their principal ports are connected to u.

More formally, $n[\![a!c.P]\!] = (n[\![P]\!] + \mathsf{c[a]} + \mathsf{c} + \mathsf{u}) \cup \bigcup_{e \in \mathcal{B}[\![P]\!]} \mathsf{b}_e$. $e[\![a!c.P]\!] = e[\![P]\!] \cup \{\mathsf{a.s}{-}\mathsf{u.d_1}, \mathsf{c.o}{-}\mathsf{u.d_2}\} \cup \bigcup_{e \bowtie (n_1,n_2) \in \mathcal{B}[P]} \{\mathsf{u.u}{-}\mathsf{b}_e, \mathsf{b}_e\mathsf{.d_1}{-}\mathsf{n_1}, \mathsf{b}_e\mathsf{.d_2}{-}\mathsf{n_2}\}$. $\mathcal{B}[\![a!c.P]\!] = \{\mathsf{a.s}{-}\mathsf{u.d_1}\}$

Proposition 2 (Complexity of the Translation). *The size of $[\![P]\!]$ (number of nodes plus number of edges) is linear in the size of P (number of prefixes).*

Proof. A simple structural induction, based on the fact that $|\mathcal{B}[\![P]\!]|$ is equal to the number of top-level parallel components of P. □

4.4 Structural Correspondence

Notation We call nets $N = [\![P]\!]$ *primary*, and nets that don't correspond to a process *auxiliary*. (We only consider auxiliary nets that are the reducts of primary nets in this paper.)

We now check statically that our translation neither identifies processes that are not structurally equivalent, nor fails to identify equivalent processes.

Theorem 3. $[\![P]\!] \approx [\![Q]\!]$ *iff* $P \equiv Q$ *($\approx$ as per Definition 1).*

Proof. **If** Case analysis on the definition of structural equivalence in §3.1.

1. α-renaming. Since the bound labels of P are removed in $[\![P]\!]$, we have $[\![P]\!] \approx [\![P[y/x]]\!]$ for any name x not free in P.
2. Monoidal structure of parallel composition. Easy, due to $[\![0]\!] = (\emptyset, \emptyset)$ and the associativity and commutativity of amalgamated sum $n_1 + n_2$ and union $e_1 \cup e_2$.

3. – 5. are obvious.

Only if We build an inverse translation $]\!]N[\!]$ and we prove that it preserves structural congruence (see below). □

4.5 Inverse Translation

We define an inverse translation $]\!] \cdot [\!] : \mathrm{MIN}_\pi \mapsto \Pi/_\equiv$ from primary nets to a representative of the structural equivalence class of the original process, $]\!]N[\!] = P$. It breaks a composite net into parallel components, strips prefixes and associated blocking machinery, and works recursively.

As a preliminary step, we take care of hidden nodes: $]\!]N[\!] = (c_1 \ldots c_n)]\!]N'[\!]$ where $c_1 \ldots c_n$ are all the c nodes of N with hidden labels, and N' is like N but with these labels un-hidden.

We are careful to define $]\!]N[\!]$ to be deterministic, so we first find exhaustively all parallel components. A *blocking region* b of a main net N is a minimal non-empty set of directed edges $e \subset e(N)$ closed under a "principal domination" condition: if $\mathsf{y.q{-}x.p} \in e$ is an edge of N that comes into the single principal port p of node x[12] then $\mathsf{x.p_i{-}y_i.q_i} \in e$ for all other edges of x. By minimality, a blocking region b either consists of one edge both ends of which are not single-principal ports, or is a tree of edges related by principal domination. In other words, b cannot contain unrelated edges.

Lemma 4. *Two blocking regions b_1 and b_2 are either disjoint or one is a subset of the other.*

Proof. Assume the opposite. Since b_1 and b_2 are trees, by following non-shared edges we can reach a shared edge. The source node x of that shared edge must have two incoming non-shared edges, one in b_1 and the other in b_2. But this is impossible, because x cannot have multiple principal ports, nor principal multi-ports. □

Lemma 5. *Let M be the subgraph of N generated by a blocking region b of N (M contains all edges in b and their adjacent nodes). Then M is a main net.*

Proof. Follows from the principal domination closeness of b, after observing that all edges of $[\![P]\!]$ in $[\![a!c.P]\!]$ and $[\![a?p.P]\!]$ can be reached from $\mathsf{a.s{-}u.d_1}$ and $\mathsf{a.r{-}p.i}$ respectively (see the figure in §4.3). This observation can be proved easily by induction on the structure of $[\![P]\!]$. □

Let $B = \{b_1, \ldots, b_n\}$ be the set of all maximal blocking regions of N. By lemma 4 and maximality, they partition the edges of N: $e(N) = b_1 + \cdots + b_n$. By lemma 5, they all are main nets: $b_i = [\![N_i]\!]$. So we define the first step of our inverse translation (breaking into parallel components) thus: if N has more than one maximal blocking regions $\{b_1, \ldots, b_n\}$, then

$$]\!]N[\!] \stackrel{\text{def}}{=}]\!]N_1[\!], \ldots,]\!]N_n[\!]$$

[12] This means that x is one of $\mathsf{p}, \mathsf{b}, \mathsf{u}$ but not c, since c has several principal ports.

where N_i is the subgraph of N generated by b_i.

To define the inverse translation of a maximal ("main") blocking region b, we have to consider two cases:

1. b consists of a single edge. The edge can be $\mathsf{c[a].s{-}c.o}$ or $\mathsf{c[a].r{-}p.i}$. Then $]\!]b[\!]\stackrel{\text{def}}{=} a!c$ or $]\!]b[\!]\stackrel{\text{def}}{=} a?p$ respectively.
2. b is a tree of several edges. Then it has a single root edge, for otherwise the root edges would be unrelated. Furthermore, b must correspond to the left or the right half of the figure in §4.3, because it is main (subgraph of a main net $N = [\![R]\!]$).

   (a) The root edge is $\mathsf{c[a].s{-}u.d_1}$. Then $]\!]b[\!]\stackrel{\text{def}}{=} a!c.]\!]P_1[\!]$ where P_1 is the subnet connected to the b nodes (labeled $[\![P]\!]$ in the figure).

   (b) The root edge is $\mathsf{c[a].r{-}p.i}$. Then $]\!]b[\!]\stackrel{\text{def}}{=} a?p.]\!]Q_1', Q_2'[\!]$ where Q_1 is the subnet connected to the b nodes (labeled $[\![Q]\!]$ in the figure), Q_2 are optional subtrees of b attached to the s and r ports of p, and Q' is like $]\!]Q[\!]$, but with the type of p promoted to c.

The proof of Theorem 3 is completed by the following lemma.

Lemma 6. *If two main nets are isomorphic, $M \approx N$, then $]\!]M[\!] \equiv]\!]N[\!]$.*

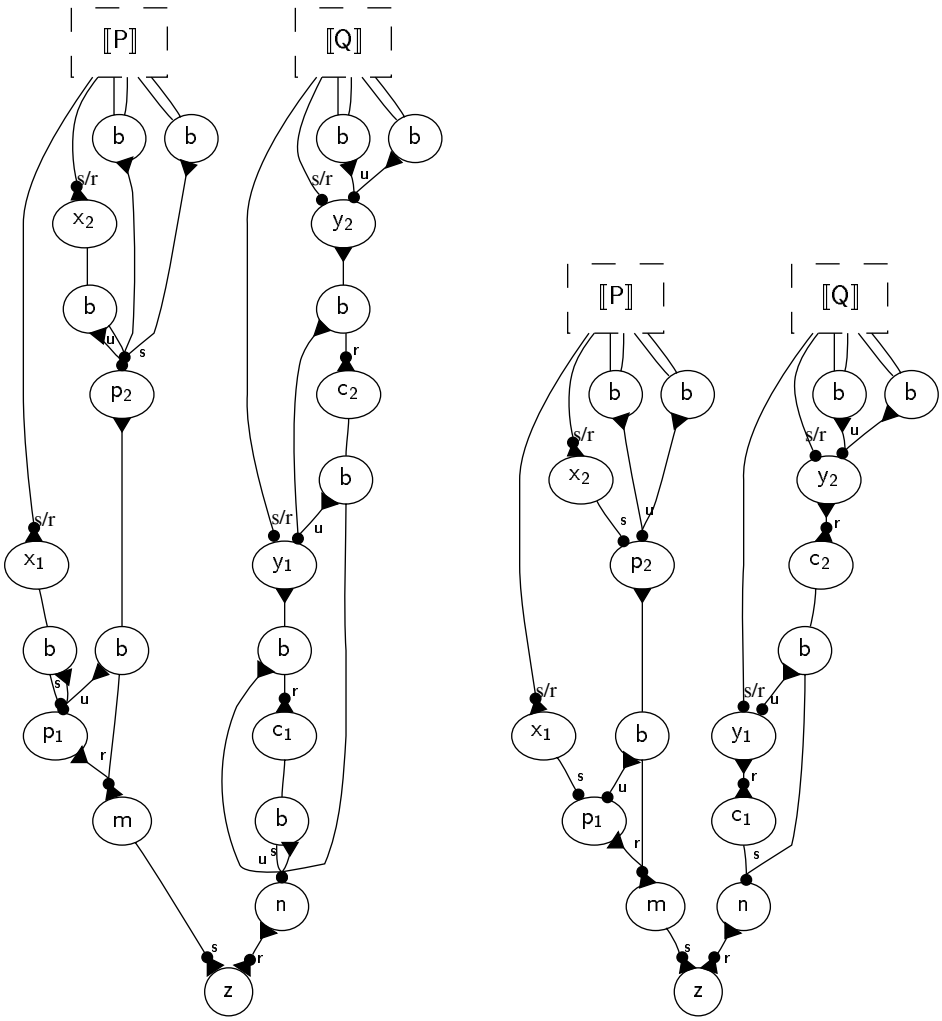
Proof. By isomorphism, the blocking regions of M and N are also isomorphic. Then by the determinsm of $]\!]\cdot[\!]$, it follows that $]\!]M[\!]$ and $]\!]N[\!]$ are essentially the same, up to α-renaming of hidden labels. □

4.6 Example

In order to clarify our translation, we give a substantial example, the Honda-Tokoro construction. It implements polyadic (n-ary) blocking prefix in terms of monadic (1-ary) output atoms and monadic input blocking prefix. It is also known as the *zipper construction*. For $n = 2$, the construction is as follows:

$$z![x_1x_2].P = (m)z!m, m?p_1.(p_1!x_1, m?p_2.(p_2!x_2, P))$$
$$z?[y_1y_2].Q = (c_1c_2)z?n.(n!c_1, c_1?y_1.(n!c_2, c_2?y_2.Q))$$

The translation of $[\![z![x_1x_2].P, z?[y_1y_2].Q]\!]$ is on the left side of the figure below. Note that we don't need any u nodes since there are no output prefixes.



Apart from the blocking machinery needed to sequentialize the interactions $\mathsf{p_i} \bowtie \mathsf{c_i}$ $(i = 1, 2)$ on the shared channel $\mathsf{m} \bowtie \mathsf{n}$, the construction is symmetric.

Unnecessary Blocking Due to the linear nature of π-calculus terms, one is forced to introduce more blocks than are really necessary. Let's consider the *dependency relation* $\prec$ between prefixes in a process P, which is the transitive closure of the following rules:

1. If $\pi.Q$ is a subterm of P then $\pi \prec \pi'$ for every π' occurring in Q (blocking; explicit synchronization).
2. $a?x \prec \{b!x, x!b, x?b\}$[13] for those of the prefixes that occur in P (provision; data-dependency-based synchronization).

[13] Here $\pi \prec S$ means $\pi \prec \pi'$ for every $\pi' \in S$.

Since the π-calculus only allows disjoint or properly nested scopes, but not overlapping scopes, the structure of $\prec$ is a directed forest. For example, it is impossible to have $\pi_1 \prec \pi$ and $\pi_2 \prec \pi$ without also having $\pi_1 \prec \pi_2$ or $\pi_2 \prec \pi_1$. Furthermore, since the scopes of input prefix and synchronization are confounded, 1 subsumes 2.

There are no such limitations in our MIN_π representation, which allows us to minimize the use of blocks. For example, the only blocks that are necessary in the Honda-Tokoro construction are blocks that enforce $m?p_1 \prec m?p_2$, $n!c_1 \prec n!c_2$, $p_2!x_2 \prec P$ and $c_2?y_2 \prec Q$, therefore we can easily remove 5 blocks. (See the right part of the figure above.)

4.7 Send/Receive

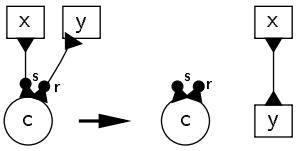
We now describe the interaction rules that govern MIN_π. Communication is implemented in three discrete steps, corresponding to the different roles of prefix that we mentioned in §3.3:

Send/Receive A pair of s/r links of a channel c is chosen non-deterministically, and the objects at the ends of these links (say x in $a!x$ and y in $a?y$) are put in contact.

Input/Output (Link Migration) All links of the placeholder p are transferred to the channel c that is instantiating it.

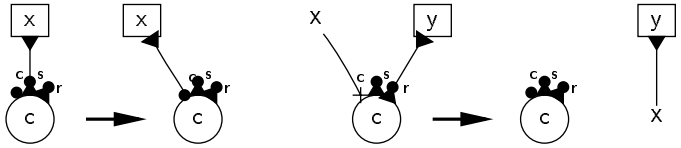
Unblocking The processes that were blocked by the two prefixes (*e.g.* P and Q in $a?p.P$ and $a!c.Q$) are unblocked and can interact further.

In the first step, a channel c that has *active*[14] nodes attached to both its s and r ports, shortcuts them in order to make it possible for them to interact. MIN_π does not give us easy means to secure two active nodes atomically. The easiest would be to employ a ternary rule[15]



However, IN and MIN_π allow only binary rules.

Therefore we have to do it in two separate sub-steps, which may be distant in time. In the first sub-step channel c secures an active node from port s and stores it in the set of committed output objects on port c. In the second sub-step it puts an active node from port r in contact with an edge from port c. Both edges are selected non-deterministically.



[14] Linked to c through their principal port.

[15] Remember that boxes denote generic node types.

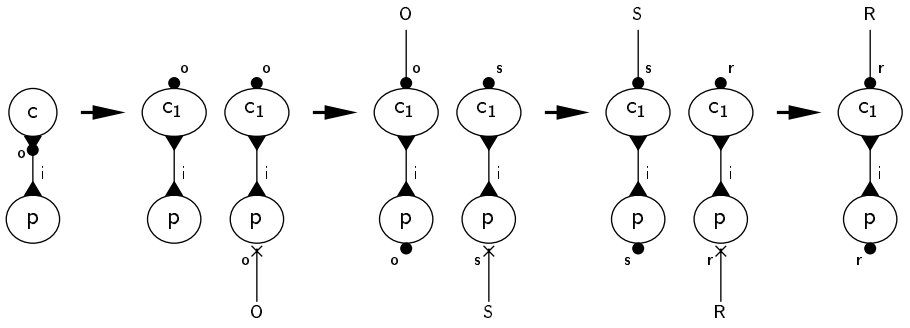
The correctness of this switcheroo depends on the following

Lemma 7 (Monotonicity of Activation). *Once a node on* c.s *becomes active, it cannot become inactive until it is transferred to* c.c*, and later detached from* c *by the above rule.*

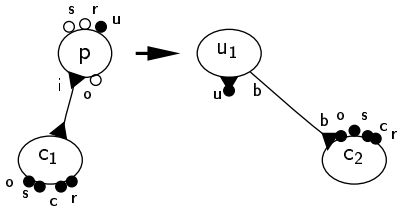
Proof. Examination of the possible node types and applicable rules later in this section. The possible nodes on c.s in a translation $[\![P]\!]$ are u.d_1 (active), c.o (active) and b (inactive). The rules that affect c.s are immigration of links from p.s (see §4.8) which are again nodes of the same types, and the removal of b during unblocking (see §4.9) which exposes an active u.d_1 or c.o. Once a link is active, it can only be transferred to c.c where it will remain active until it is detached by the second rule above. □

4.8 Input/Output (Link Migration)

The second step of a communication $a!c.P, a?p.Q$ is the merging of the output object node c and the input object node p, corresponding to the π-calculus substitution $Q[c/p]$ (c is instantiated for p). In MIN$_\pi$, this entails a migration of all edges of p to c. Since c may be output object of more than one output prefix, if it keeps its type c during the immigration process, it may be subjected to another send-receive interaction while the first one is in progress. Then c may immigrate links from another placeholder p', and therefore will interlace link immigration from two different placeholders. Such interlaced immigration increases the parallelism of the implementation, and should therefore be seen as a positive aspect. However, in this paper we prefer to disallow it, in order to simplify the proof of correctness of our translation. To this end, c changes its type to c_1 throughout the immigration process. c_1 is committed to that process and cannot perform any other interaction until it is completed.



Here we use arity constraints (see §2.2) to guide the migration process. After migration is finished, the placeholder p becomes an unblocker u_1 and c becomesc_2, waiting for a signal from u_1 that unblocking is completed (described in §4.9).

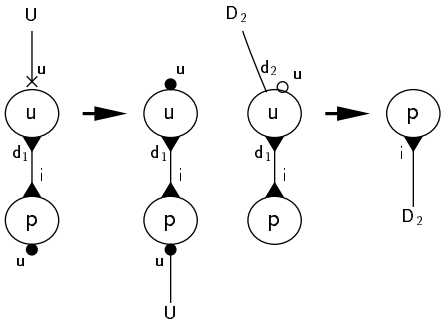


Lemma 8 (Confluence of Migration). *1. The rules of this subsection are confluent (as formulated at the end of §2). 2. There is no outside interference, i.e. after the initial interaction* $\mathsf{c} \bowtie \mathsf{p}$ *and until the final interaction* $\mathsf{c_1} \bowtie \mathsf{p}$*, no other nodes but the ones above can participate.*

Proof. Simple. Both $\mathsf{c_1}$ and p have a single principal port, and any two link migration rules commute. □

4.9 Blocking and Unblocking

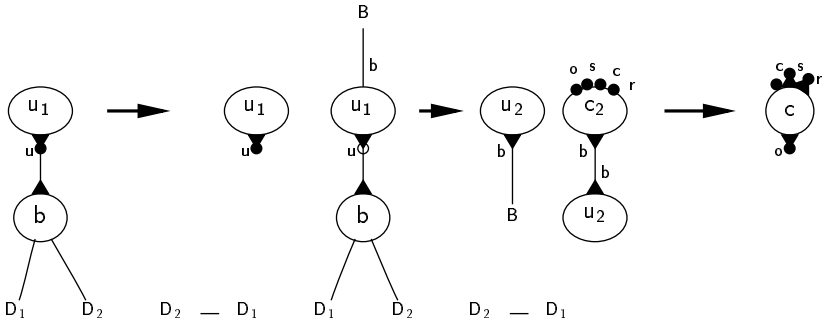
The send/receive rules of §4.7 are insensitive wrt the type of the object, therefore they apply equally well to $\mathsf{c} \bowtie \mathsf{p}$ (corresponding to an output atom $a!c$ interacting with an input $a?p.Q$), and to $\mathsf{u} \bowtie \mathsf{p}$ (corresponding to an output prefix $a!c.P$ interacting with an input $a?p.Q$).[16] The case $\mathsf{c} \bowtie \mathsf{p}$ was described above in §4.8. In the case $\mathsf{u} \bowtie \mathsf{p}$, u migrates all its blocking links to p and then disappears:



Once this migration is completed, a $\mathsf{c} \bowtie \mathsf{p}$ redex results, and the rules of §4.8 apply.

To complete the communication we must unblock all blockers b controlled by the prefix. More specifically, the node $\mathsf{u_1}$ that was borne by the placeholder p in §4.8 should dismiss all the blockers that it controls.

[16] Note that if we limit our consideration to the ν (asynchronous π) calculus where output prefix is not allowed, then we don't need u.



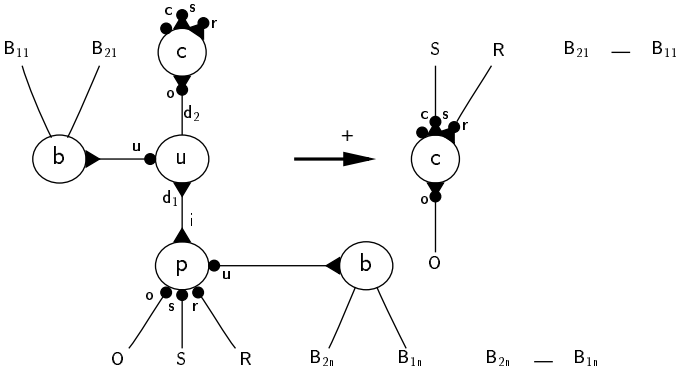
At the end of the unblocking process (directed by the arity constraint on the principal port), u_1 turns into u_2 and signals c_2 that the communication has been completed.

Lemma 9 (Confluence of Unblocking). *1. The rules of this subsection are confluent. 2. There is no outside interference.*

Proof. Simple examination. Only u_1 has a principal multiport, but the order in which it dismisses b nodes is irrelevant. □

The lemmas in this and the previous subsection mean that after the non-deterministic selection of a $\mathsf{c.o}$ link, computation can proceed in essentially only one way, giving us the following aggregate result.

Lemma 10. *In the figure below (with any number of* b *nodes attached to* u *and* p*, and any number of* $\mathsf{O}, \mathsf{S}, \mathsf{R}$ *links), the* MIN_π *on the LHS can reduce in only one way, to the* MIN_π *on the RHS.* □

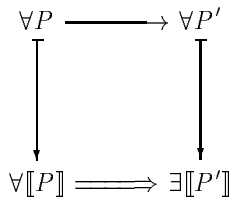


This completes the description of the MIN_π interaction rules.

4.10 Operational Correspondence

We now set out to prove that our translation serves its purpose of faithfully modeling the π-calculus. Completeness is easy, as is to be expected of any decent implementation encoding.

Theorem 11 (Completeness). *For every single-step process reduction $P \to P'$ there exists a corresponding multi-step net reduction $[\![P]\!] \Rightarrow [\![P']\!]$.*

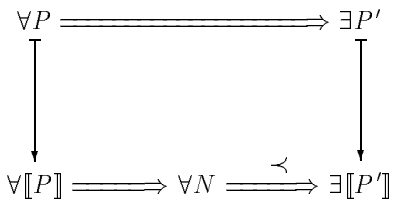


Proof. Case analysis on the reduction rules in §3.2. For Comm, we use Lemma 10: starting from the net in §4.3, after the s and r links on a are committed, there is a (unique) reduction which removes all b and migrates the links of p to c, which is the same as $[\![P, Q[c/p]]\!]$. For Struc, we need Theorem 3. Par and Res (top-level name restriction) do not decrease the possibilities for MIN_π reduction. □

As is usual with implementation encodings, Soundness presents a greater challenge. We have gone to great lengths to make the reduction process as deterministic as possible. For example in §4.8 we could increase the degree of parallelism by leaving c with the same type throughout the process, instead of sequentializing it using c_1 and c_2. However, until some general results about partial confluence of MINs are available (or coinductive techniques for IN are better understood, (Fernández and Mackie, 1998)), we prefer to simplify the reduction process in order to obtain more easily our correspondence result.

We state the following

Theorem 12 (Soundness). *(a) For every multi-step reduction $[\![P]\!] \Rightarrow N$ there exists an extension $N \Rightarrow [\![P']\!]$ such that $P \Rightarrow P'$.*



(b) For every $[\![P]\!] \overset{\prec}{\Rightarrow} [\![P']\!]$ holds $P \Rightarrow P'$.

We will explain the $\overset{\prec}{\Rightarrow}$ symbol below.

Note that 12.(b) alone is not sufficient to guarantee well-behavedness, because it leaves the possibility of bad reductions from $[\![P]\!]$ that never lead to a primary net.

Unless we impose some restrictive reduction strategies on net reduction, we cannot prove a similar result for single-step process reduction. The reason is that we are implementing an "atomic" reduction relation with a completely distributed system. $[\![P]\!]$ can start reducing several independent reductions of P simultaneously (or interleaving them), and none of the intermediate nets will be

primary. The theorem states however that no matter how long this net reduction is, it can always be completed to a primary net $[\![P']\!]$ such that $P \Rightarrow P'$.

In order to prove the theorem, we explore the space of reducts of $[\![P]\!]$ and its confluence properties. According to §4.3, the possible redexes of $[\![P]\!]$ are of the form $\mathsf{c.s} \bowtie \mathsf{c.o}$, $\mathsf{c.s} \bowtie \mathsf{u.d_1}$ and $\mathsf{c.r} \bowtie \mathsf{p.i}$. According to §4.7, the former two cause $\mathsf{c.o/u.d_1}$ to be transferred (committed) from $\mathsf{c.s}$ to $\mathsf{c.c}$, at which point the latter redex becomes applicable. We will call c nodes with something on $\mathsf{c.c}$ *committed channels*.

A committed channel with a $\mathsf{p.i}$ on its $\mathsf{c.r}$ port may choose to shortcut one of its commitments with $\mathsf{p.i}$, and become non-committed if it has no more commitments, or stay committed. Lemma 10^{17} applies to the redex formed of the commitment and $\mathsf{p.i}$, so we may reduce it immediately to the RHS. This will unblock the subordinate processes of c and p (if any) as well, and leave us with a an *almost primary* net, one that may eventually contain committed channels.

Thus, it turns out that the structure of the reduct space is quite simple:

- It may contain channels in various stages of commitment.
- It may contain parts that are in the process of link migration and unblocking, but by confluence we can complete the process in essentially only one way.

Notation Given a net M, its *completion* N is obtained by firing all possible rules of §4.8 and §4.9. We denote this as $M \rhd N$. Lemma 10 guarantees that the completion is unique.

There is a little technical problem with commitment. Take the simple net corresponding to $\mathsf{c!v}$. In that net v is attached to $\mathsf{c.s}$. The commitment rule may move v to $\mathsf{c.c}$, but this is not a primary net anymore. And since there is no possibility for communication, it cannot evolve at all, and will remain non-primary. We therefore consider backing up commitments.

Notation Given a net M, its *uncommitment* N is obtained by moving any links from $\mathsf{c.c}$ to $\mathsf{c.s}$, for all channels c. We denote this $M \prec N$. We define $\stackrel{\prec}{\Rightarrow} = \Rightarrow\prec$. This is an operation external to MIN_π; a net itself can never back up a commitment.

Neither $\rhd$ nor $\prec$ make any decisions. The former forges ahead completing all communications that have been committed, the latter removes commitments to allow us to evaluate what we have.

As for 12.(b), we can prove it by tracing where commitments are consumed, completing at these checkpoints, and relating the result to single-step process reductions.

5 Related Work

The only work which deals with an application of MIN to a process calculus that we are aware of is the unpublished paper (Gay, 1991). It translates a confluent fragment of CCS into INs. The restriction to confluence is not arbitrary, it is due to the essential confluence of conventional IN (see §2.1).

[17] Or an analogous lemma without u, and a direct $\mathsf{c} \bowtie \mathsf{p}$ cut instead.

5.1 π-nets and Interaction Diagrams

Since the very inception of the π-calculus, Flow Graphs were sometimes used to represent it graphically (Milner *et al.*, 1992). More recently, two other graphical formalisms were introduced, *π-nets* (Milner, 1994, 1993) and *Interaction Diagrams* (Parrow, 1995). Our translation is quite similar to these constructions, but we also implement synchronization in a distributed manner, only using local interaction. Our construction is more similar to INs because nodes have separate ports and we use only dyadic interaction.

These formalisms give additional insights into the finer computational structure of the π-calculus, but still leave some of the computational structure implicit and use names essentially. Namely, they represent prefix through the use of boxes (non-local computation elements). Nodes bear the names/numbers of the π-calculus names they represent, and communication manipulates these names/numbers "at runtime".

(Parrow, 1995, sec. 10) suggests that synchronization may be implemented locally by increasing the arities of every channel, but unfortunately the sketch given appears to be incorrect (or correct for only a limited setting). The author proposes to implement $a?\boldsymbol{x}.P$ by adding a control channel b to every output prefix $d!\boldsymbol{y}$ appearing in P, and making $a?\boldsymbol{x}$ instantiate these channels when it reduces. Input prefixes $d?\boldsymbol{y}$ also get an auxiliary channel, but it is not "hooked up" to $a?\boldsymbol{x}$, its purpose only being to match the control channel of $d!\boldsymbol{y}$. This indeed stops internal communications of $d?\boldsymbol{y}$, but it does not stop communications with the outside of P. Therefore the construction only works if all subjects in P are private channels. Nor will it be correct to block *all* interactions on d, because in $a?x.(d!y, d?y), d!z, d?z$, the prefixes $d!?y$ must wait, but $d!?z$ may proceed.

5.2 Concurrent Combinators

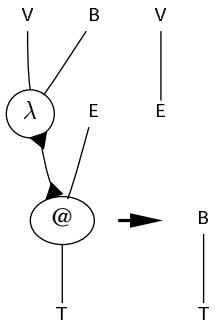
More recent is the work on Concurrent Combinators (*cc*) (Honda and Yoshida, 1994a,b), which "analyzes away" the prefix constructions of π-calculus (input, output and replication prefix) and represents all of their expressive power in a finite system of atomic combinators. This work does the same for the foundations of concurrent computation as the work of Curry *et al.* does for the foundations of sequential functional computation. A graph representation of the same system is developed in (Yoshida, 1994) and is named *Process Graphs* (PG).

cc gave us a big inspiration for the present work. They can be seen as a dual graph representation of our construction. For example, where MIN_π represents $a!b$ as two nodes a, b and a connecting edge, *cc* represents the same process as one *message* node $M(a, b)$ with two links a, b. However, we perceive the following shortcomings of *cc* and try to address them in this work:

- *cc* are more complicated since they use "aggregate" nodes, *i.e.* nodes corresponding to a configuration of π-calculus channels. For example, they have a node $S(u, v, w)$ corresponding to $u?x.v?y.w!y$. The genesis of the combinators and the corresponding rules is not obvious, and the proof that the

set of combinators is closed wrt process constructors is highly non-trivial. In contrast, our (main) node types correspond directly to π-calculus channels and some blocking machinery, and our rules come naturally.

- *cc* are inspired by combinatory term rewriting rather than graph rewriting. As a consequence, shortcut edges are not allowed in the RHS of rules. For example, the typical IN rule corresponding to a first-order rewriting $@(\lambda v.b, e) \to b[a/v]$ is not allowed in *cc*.



- Therefore one is forced to use *forwarders* whose role is purely bureaucratic: to disappear when a message arrives at their primary port, connecting it to their auxiliary port. Notwithstanding their trivial nature, effort has to be expended for their bookkeeping. Furthermore, one doesn't always have a term reduce to the "expected" term, but has to be content with a reduction up to a certain simulation relation $\succ$ that effaces forwarders (see *e.g.* (Honda and Yoshida, 1994a, Theorem 4.4)).
- The translation of π-calculus into *cc* is asymmetric, using only one constructor (the *message*) and several destructors that come from the input prefix. The use of forwarders only makes this asymmetry stronger. We believe that a large part of the complexity of the translation can be attributed to this property, and that a more symmetric translation results in a simpler, or at least more natural, translation.
- The translation from π-calculus to *cc* is not *uniform* in that $[\![P, Q]\!]$ introduces some extra machinery (a *duplicator* D whose role is to multiplex an unblocking signal to the two components), and $[\![0]\!] \neq \emptyset$.
- The translation can be exponential. For example, the translation of

$$c?x_1.x_1?x_2.\ldots.x_{n-1}?x_n.x_n!v$$

is a PG with $\frac{3^{n+1}-1}{2}$ nodes. The reason for such an explosion is that (almost) every node X of G in the translation of $x_i?x_{i+1}.G$ is surrounded by 3 new nodes —a duplicator D, a binder B_l and a synchronizer S— which when given a message, turn to two forwarders. This intractability can probably be eliminated through a more refined translation, one that does not deliver an activation signal from a reception to every node of the receiving graph, but only to some *controlling* nodes (like our $\mathcal{B}[\![P]\!]$). But we trace at least part of the reason to the use of forwarders.

On the other hand, *cc* have the advantage over our construction of being a combinatory system for the π-calculus, in the sense that they can be expressed in π and they form a closed system. Our MIN_π is a system external to π, and furthermore one that is not yet studied in depth.

6 Future Work

The work reported here is quite exploratory in nature: we extend a well-known elegant system (IN) and explore the expressive power of the resulting system. However, it would be necessary to also study the nature of the system itself. How far have we strayed from INs? Which of the theoretical results for IN hold in our system? For example, can one identify simply deadlock-free fragments? Is there a universal combinatory system for MIN? What are the appropriate behavioral equalities for MIN?

6.1 Replication and Choice

Two important π-calculus constructors are missing form our consideration, *replication/recursion* and *choice (sum)*. These are the constructions that give the π-calculus full computing power (loops and conditionals). We hope to address these aspects of the π-calculus in future work.

Replication (the duplication of a process) and choice (the discarding of alternative processes) are similar to Linear Logic (LL) contraction and weakening respectively, which in the traditional Proof Net theory of LL are implemented as non-local operations through the duplication and erasure of "boxes". It would not be hard to postulate replication and choice in a similar way in this work, however we would like to preserve the local character of MIN at any price. Recent developments in the area of optimal lambda reduction (Gonthier *et al.*, 1992) and sharing graphs (Guerrini *et al.*, 1997) demonstrate how one can present Proof Nets in a completely local fashion. The interaction of these "structural boxes" and the blocking "layers" that we use is quite subtle.

Bibliography

M. Fernández and I. Mackie. Coinductive techniques for operational equivalence of interaction nets. In *Symp. on Logic in Computer Science (LICS'98)*. 1998.

S. J. Gay. Translating confluent CCS into interaction nets. 1991.

G. Gonthier, M. Abadi, and J.-J. Lévy. Linear logic without boxes. In *Logic in Computer Science (LICS'92)*, pages 223–234. IEEE Computer Society Press, Santa Cruz, CA, 1992.

S. Guerrini, S. Martini, and A. Masini. Coherence for sharing proof-nets. Technical Report IRCS-97-03, University of Pennsylvania, 1997.

K. Honda and N. Yoshida. Combinatory representation of mobile processes. In *Principles of Programming Languages (POPL'94)*, pages 348–360. Portland, Oregon, 1994a. ISBN 0-89791-636-0.

K. Honda and N. Yoshida. Replication in concurrent combinators. In M. Hagiya and J. C. Mitchell, editors, *Theoretical Aspects of Computing Science (TACS'94)*, number 789 in LNCS, pages 786–805. Sendai, Japan, 1994b.

Y. Lafont. Interaction nets. In *Principles of Programming Languages (POPL'90)*, pages 95–108. ACM, San Francisco, CA, 1990.

R. Milner. An action structure for the synchronous π-calculus. In Z. Esik, editor, *Fundamentals of Computation Theory (FCT'93)*, number 710 in LNCS, pages 87–105. Szeged, Hungary, 1993.

R. Milner. Pi-nets: a graphical form of π-calculus. In *European Symposium on Programming (ESOP'94)*, number 788 in LNCS, pages 26–42. 1994.

R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, parts I and II. *Information and Computation*, 100(1):1–77, 1992.

J. Parrow. Interaction diagrams. *Nordic Journal of Computing*, (2):407–443, 1995. Earlier version appeared in *A Decade of Concurrency: Reflections and Perspectives. REX School and Symposium*, J.W. de Bakker, W.-P. de Roever and G. Rozenberg (ed), June 1993, LNCS 803; and as SICS Research report R93:06.

N. Yoshida. Graph notation for concurrent combinators. In T. Ito and A. Yonezawa, editors, *Theory and Practice of Parallel Programming (TPPP'94)*, number 907 in LNCS, pages 393–412. Sendai, Japan, 1994.