Accepted for publication in Math. Struct. in Comp. Science

Comparing the Expressive Power of the Synchronous and the Asynchronous π-calculi[†]

Catuscia Palamidessi

INRIA Futurs

LIX, École Polytechnique, 91128 Palaiseau Cedex, France

catuscia@lix.polytechnique.fr

The Asynchronous π-calculus, proposed by Honda and Tokoro (1991) and, independently, by Boudol (1992), is a subset of the π-calculus (Milner et al., 1992) which contains no explicit operators for choice and output-prefixing. The communication mechanism of this calculus, however, is powerful enough to simulate output-prefixing, as shown by Honda and Tokoro (1991) and by Boudol (1992), and input-guarded choice, as shown by Nestmann and Pierce (2000). A natural question arises, then, whether or not it is as expressive as the full π-calculus. We show that this is not the case. More precisely, we show that there does not exist any uniform, fully distributed translation from the π-calculus into the asynchronous π-calculus, up to any "reasonable" notion of equivalence. This result is based on the incapability of the asynchronous π-calculus to break certain symmetries possibly present in the initial communication graph. By similar arguments, we prove a separation result between the π-calculus and CCS, and between the π-calculus and the π-calculus with internal mobility, a subset of the π-calculus proposed by Sangiorgi where the output actions can only transmit private names.

1. Introduction

Communication is one of the fundamental concepts in concurrent and distributed computation, and can be of many kinds: synchronous, asynchronous, one-to-one, one-to-many, etc. In this paper we focus on the distinction between synchronous and asynchronous. Synchronous communication is usually understood as *simultaneous* exchange of information between the partners; a "real life" example is the telephone[1]. In contrast, in asynchronous communication the action of sending a message and the action of reading it

[†] Work supported by the NSF-POWRE grant EIA-0074909.

[1] Of course, communication by telephone can be thought as simultaneous only when the transmission time is negligible wrt the "clock" of the partners, which is a reasonable assumption when the partners are humans.

usually take place at different times. An example is email. The advantages and disadvantages of the two methods are well known: the first is more costly, because it requires the partners to synchronize to establish the communication, but then, once established, it is more effective.

1.1. *Motivations*

A question which arises naturally is whether these two mechanisms are equivalent; i.e., whether the one can implement the other. One direction seems simple, at least in principle: asynchronous communication can be simulated by inserting between each pair of communicating agents a "buffer" process, see for instance (Milner, 1989) and (He et al., 1990)[2]. The other direction, on the contrary, is not clear and researchers in the field seem to have radically different opinions about it.

The motivation for this work arises from the attempt of solving, or at least clarifying, this question. In the author's opinion, the crucial point is which other mechanisms are available in combination with synchronous communication: If the processes can make choices together, based on the information that they exchange simultaneously, then synchronous communication is intuitively more powerful. This intuition is supported by the example of two people who try to take a common decision by using email instead of the telephone: If they act always in the same way, i.e. they send at the same time identical mails and react in the same way to what they read, then an agreement may be never reached.

The π-calculus (Milner et al., 1992) is a convenient framework to study this problem. In fact, the π-calculus is a synchronous paradigm which contains an "asynchronous" fragment (Boudol, 1992; Honda and Tokoro, 1991). We can thus work in a uniform context. But, more important, the asynchronous π-calculus is one of the richest paradigm for asynchronous-communication concurrency introduced so far, hence a negative result regarding this language is more significant.

1.2. *Background*

The asynchronous π-calculus differs from the π-calculus for the lack of the choice and the output prefix operators. The underlying model of interaction among processes, however, is the same as in the π-calculus (the communication rule is based on handshaking, i.e. on the simultaneous execution of complementary actions). The reason why it is considered asynchronous is that, due to the lack of output prefix, an output action can only be written "in parallel" with other activities. More precisely, in the π-calculus we can write $P = \bar{x}.P'$ to represent a process P that performs an output on channel x, and continues as P' afterward, and we can write $Q = x.Q'$ to represent a process Q that performs an

[2] Depending on the particular kind of asynchronous communication this implementation can be more or less complicated; for instance if the order in which messages are sent is to be maintained, then we need a FIFO buffer, whose definition as a process requires guarded nondeterminism. For unordered communication we just need a bag, which can be defined by using input, output, and replication only.

input on x and continues as Q' afterward. (In the π-calculi input and output actions have parameters (names), but here for simplicity we omit them.) Furthermore, by using the restriction operator we can enforce the synchronization of P and Q on x, so that the processes can proceed only when the communication along x takes place. From the point of view of the sender, the attempt to perform the output provokes the suspension of P, and the execution of the matching input action (reception) resumes P at its continuation point P'. In the asynchronous π-calculus, on the contrary, we can write a sender process only in the form $P = \bar{x} \,|\, P'$, where $|$ is the parallel operator. Since it is performed in parallel, the output action $\bar{x}$ does not automatically suspend the sender, and there is no primitive notion of continuation point. One can think of $\bar{x} \,|\, P'$ as a process which performs an output on x at some unspecified moment and of the handshaking between $\bar{x}$ and $x.Q'$ as the moment in which the message is received. The reception enables the continuation point Q' in the receiver, but it does not cause (directly) any resumption of activity in the sender[3].

Of course, the effect of the output prefix can be simulated by implementing a *rendez-vous* mechanism, in which the receiver sends, upon reception of the message, an acknowledgment to the sender, and the sender waits until it receives such acknowledgment. This kind of technique was in fact used by Boudol (1992) to define an encoding of the output prefix in the asynchronous π-calculus. Although the technique may be not surprising, the appeal and the novelty of this encoding consists in an elegant use of the primitives for link mobility, thanks to which the translation can be defined in a compact and fully compositional way. Independently, Honda and Tokoro (1991) proposed an encoding even more compact, and also fully compositional, in which it is the receiver which takes the initiative of synchronizing with the sender. It is probably fair to say that part of the success of the asynchronous π-calculus, at least in the early days, was due to the encoding of Honda and Tokoro. Another important factor, of course, was that it could be implemented in a relatively simple and natural way. The first implementation of the asynchronous π-calculus (actually a version of it, called PICT) was developed by Pierce and Turner (1998).

Both the encodings of Boudol and of Honda and Tokoro work only when the output prefix is not used in combination with the choice operator. In a subsequent paper Honda and Tokoro (1992) showed an encoding for the choice operator, but only for the simple case of the local (aka internal, or blind) choice. More recently, however, Nestmann and Pierce (2000) showed that input-guarded choice can also be encoded compositionally in the asynchronous π-calculus[4]. This result was another fundamental contribution towards the affirmation of the asynchronous π-calculus as a practically useful paradigm: input-

[3] Note that this kind of communication is unordered and that $(\bar{x} \,|\, P') \,|\, x.Q'$ is equivalent (modulo silent actions) to $\nu y(\bar{y}.P' \,|\, B \,|\, x.Q')$ where B is the "bag" process $!\, y.\bar{x}$ and y is a fresh name.

[4] Nestmann and Pierce actually were interested in obtaining a so-called *fully abstract* translation, and proposed two kinds of encoding. The first one is fully abstract wrt *weak bisimulation* (see, for instance, (Milner, 1989)), but it introduces divergences. The second one is fully abstract only wrt a weaker relation called *coupled simulation* (Parrow and Sjödin, 1992), but it is divergence-free. Here we refer to the second encoding, as we take the point of view that a notion of encoding used to relate the expressive power of two languages should not introduce divergences.

guarded choice is considered a very convenient mechanism for programming concurrent systems, as it allows a process to suspend on two (or more) alternative input channels, and to resume as soon as one of them receives a datum. A language without this feature will have to use busy waiting or risk that a process is stuck forever on the "wrong" channel. Most implementations of languages based on CSP (Hoare, 1978; Hoare, 1985), for instance, provide this construct as a primitive.

After Nestmann and Pierce showed that the input-guarded choice does not represent a gap in the expressive power, several authors have used a presentation of the asynchronous π-calculus which includes this construct as an operator of the language (see for instance Boreale and Sangiorgi (1998a), and Amadio et al. (1998)).

With a slight modification, the translation of Nestmann and Pierce can be combined with the translation of Boudol, so to provide an encoding of both the output prefix and the input-guarded choice. Still, the π-calculus offers something more, namely the possibility for the two partners of the communication to make choices together. Consider for instance a process P ready to send data (alternatively) to channels x, y and z, and assume that process Q is ready to receive data (alternatively) from x, y and w. In the π-calculus, P and Q can be specified in such a way that they will choose x or y (arbitrarily), and neither of them will select the "wrong channel" (that is z for P and w for Q). More precisely, what we need for this specification is the so-called *separate choice* construct: an output-guarded choice for P ($P = \bar{x}.P_1 + \bar{y}.P_2 + \bar{z}.P_3$) and an input-guarded choice for Q ($Q = x.Q_1 + y.Q_2 + w.Q_3$). Can such a construct be implemented in the asynchronous π-calculus? Clearly, one could implement it by backtracking from the wrong attempts, or by using a third process to coordinate the activities of P and Q. However, Nestmann (2000) showed the surprising result that it is possible to encode such mechanism even without introducing divergences (such as those which would arise from backtracking loops) and in a fully distributed way, i.e. without introducing coordinator processes.

1.3. *The contribution of this paper*

At this point one may doubt that there be any interesting cases of inter-process coordination, expressible in the π-calculus, that cannot be expressed in the asynchronous π-calculus as well. But in fact there are. Consider the following modification of previous example: A process P' ready to output on x and to input from y (alternatively), and a process Q' ready to output on y and to input from x (alternatively). In the π-calculus we can simply use the so-called *mixed-choice* construct to define $P' = \bar{x}.P_1 + y.P_2$ and $Q' = x.Q_1 + \bar{y}.Q_2$, and then enforce communication on x and y. Apparently this example is similar to previous one, but there is a fundamental difference: P' and Q' are symmetric here, at least in their initial action, whereas in previous example P initially can only send and Q can only receive. The encoding of Nestmann uses a protocol in which send and receive play completely different roles. As we will show in this paper, it is in general not possible to simulate the behavior of P' and Q' in the asynchronous π-calculus, and not even in the π-calculus with separate choice. Intuitively, the reason is that the agreement on the communication channel (x or y) represents a situation in which the initial symmetry of P' and Q' is broken, and this cannot be achieved in a language which supports separate choice only. For proving this result, we use techniques from the field of Distributed Computing. In particular we show that in certain symmetric networks it is not possible, with the separate-choice π-calculus, to solve the leader election problem, i.e. to guarantee that all processes will reach a common agreement (elect the leader) in a finite amount of time. It is possible, on the contrary, to solve this problem with the mixed-choice π-calculus.

The use of this technique has been inspired by the work of Bougé (1988), who showed a similar separation result concerning the CSP (Hoare, 1978; Hoare, 1985) and the fragment of CSP with input-guards only, CSP_{in}. However the mixed-choice π-calculus is a much richer language than CSP_{in}, and our result could not be derived from the result of Bougé. Interestingly, Bougé (1988) proved also that CSP_{in} cannot be encoded into its choice-free fragment, by using similar techniques. This result does not hold in the context of the π-calculus, as shown by the above mentioned result of Nestmann and Pierce. In the last section of this paper we will go into further details about the relation between (Bougé, 1988) and our work.

Another question that we investigate in this paper is to what extent the π-calculus is more powerful than its "ancestor" CCS (Milner, 1989). For the sake of homogeneity, we consider the value-passing version of CCS, in which the input and the output actions carry value parameters (messages). This language, that we will call here CCS_{vp}, can be seen as a subset of the π-calculus (except for the relabeling operator, see Section 1.5). The main difference is that in the π-calculus the messages are names which can later be used as communication channels, thus allowing to change dynamically the structure of the communication graph (link mobility). In combination with the mixed choice, link mobility is a very powerful feature for coordination of distributed activities, since it allows two remote processes, originally not directly connected, to establish a direct communication link x and to take decision together on the basis of the synchronous exchange of information along x. By using a technique similar to the above one (existence/non-existence of a certain symmetric electoral system) we show that this capability makes the mixed-choice π-calculus strictly more expressive than CCS_{vp}.

Finally, we consider the expressiveness of the language π_I, the π-calculus with internal mobility, proposed by Sangiorgi (1996). This is a subset of the (mixed-choice) π-calculus in which the parameters of output actions can be private names only. Because of this restriction, π_I enjoys pleasant properties such as symmetric rules for input and output, and a much simpler theory for bisimulation equivalence. Boreale (1998) has shown that the asynchronous version of π_I is essentially as expressive as the asynchronous π-calculus. His encoding is based on the following idea: the main use of sending a non-private name in the π-calculus is when a process P send a link x to Q via an intermediate process R. P and Q can then communicate directly on x. In π_I, R could not send x to Q, because it can send only his private names. However, the above mechanism can be simulated by installing in R a repeater for x which receives messages from P and send them to Q, and viceversa. Intuitively, however, this idea does not work in the presence of (mixed) choice, because the choice tests the possibility of communication (guard), and selects the corresponding branch, in one single atomic step. In the encoding of Boreale the communication along x is not atomic anymore, and therefore the atomicity of the guarded-choice mechanism cannot be trivially encoded, unless the target language supports a much richer notion of choice, such as a choice depending on the possibility of performing a sequence of actions instead than a single action. By using an argument similar to the one illustrated above for CCS_{vp}, we will show that in fact the mixed-choice π-calculus cannot be encoded in π_I.

1.4. *The notion of encoding*

In the whole discussion above we have used the existence or non-existence of an encoding as the criterion to compare the expressive power of two languages. The various encodings presented in literature, however, satisfy different structural and semantic requirements. What should be the properties of a good notion of encoding, to use as a basis for defining the concept of expressive power? We do not have a definitive answer. In the context of this paper, since we are interested in proving negative results (i.e. non-existence of an encoding) we consider a *minimal* set of requirements. More specifically, we require an encoding $[\![\cdot]\!]$ to be

— *uniform*, i.e.
  - homomorphic wrt the parallel operator, namely $[\![P \mid Q]\!] = [\![P]\!] \mid [\![Q]\!]$, and
  - renaming preserving, in the sense that for any permutation of names σ in the domain of the source language there exists a permutation of names θ in the domain of the target language such that $[\![\sigma(P)]\!] = \theta([\![P]\!])$[5].

— *semantically reasonable*, i.e. preserving the relevant observables and the termination properties.

These conditions will be defined more precisely in Section 7.

The requirement of homomorphism wrt the parallel operator ensures that two parallel processes are translated into two parallel processes, i.e. no coordinator is added by the translation. Therefore we can interpret this requirement as the condition that the degree of distribution of the processes be maintained by the translation. This condition makes the notion of encoding suitable to compare expressiveness of languages for distributed systems, where processes are expected to coordinate without the help of a centralized control.

The requirement of renaming preserving ensures that the translation does not depend on channel names. This condition seems natural if we want the encoding to preserve the portability of processes across the nodes of a distributed network.

All the encodings discussed above satisfy the two criteria of uniformity and reasonableness, with the possible exception of the encoding of Boreale, for which the preservation of a reasonable semantics is an open question. Boreale in fact has shown the correctness of his encoding only wrt barbed bisimulation, which is not sensitive to internal loops.

[5] Note that in (Palamidessi, 1997) we had a stronger condition, namely $[\![\sigma(P)]\!] = \sigma([\![P]\!])$. We realized however that the latter condition would be too strong, for instance it would make problematic the introduction of new names in the translation.

1.5. *The π-calculus hierarchy*

Figure 1 summarizes the results discussed above and introduces some terminology:

— π is the full π-calculus, as proposed in (Milner et al., 1992). In the π-calculus the choice operator (+) is *free*, in the sense that we can apply it to two arbitrary processes. For instance, we can write $(P|Q) + R$.

— π_m stands for mixed-choice π, the subset of π where the + can occur only among prefixed processes (e.g. $\Sigma_i \alpha_i.P_i$): the so called guarded choice operator. Here we say "mixed choice" to emphasize the fact that we can have both input and output (and silent) prefixes in the same guarded choice. We also omit from π_m the match and the mismatch operators. This is only because the languages in the lower part of the diagram are traditionally presented without these operators, and the result of non encoding between π_m and π_s would be weaker if π_m had them. In general, the results obtained in this paper are independent from their presence or absence.

— π_I is the internal-mobility π-calculus introduced by Sangiorgi (1996): in this language, an output parameter can only be written in the context of a restriction operator, e.g. $\nu x\, \bar{y}x.P$, also denoted as $\bar{y}(x).P$. Another characteristics is that π_I uses recursion instead than iteration. This is not accidental: in the context of π_I iteration is strictly less expressive then recursion. In π, on the contrary, recursion can be encoded by iteration (Milner, 1993). For the rest π_I is a subset of π, hence Milner's encoding of recursion extends naturally to an encoding of π_I into π.

— CCS_{vp} represents value-passing CCS without the relabeling operator. Value-passing means that actions have parameters, but unlike π these parameters cannot be used as channels in other actions. If we consider values simply as symbols (i.e. we do not consider operators on values as part of the language), then value-passing CCS is a subset of π except for the relabeling operator: such operator which does not exist in π and according to Pugliese (1997) it cannot be encoded either. For this reason we do not consider the relabeling operator here. It is worth noting, however, that the non-encoding of π_m into CCS_{vp} does not depend on the absence of the the relabeling operator (See Section 5).

— π_s stands for separate-choice π, the subset of π_m where the prefixes in a choice must be of the same kind plus, possibly, τ. Namely, in a guarded choice $\Sigma_i \alpha_i.P_i$ the α_i's which are not τ must all be either input or output action.

— π_i represents input-choice π, the subset of π_s where there is no output prefix and only input actions can be used in a choice. Namely, in $\Sigma_i \alpha_i.P_i$ all the α_i's must be input actions.

— π_{nc} stands for choiceless π, the subset of π_s without choice, but with the output prefix.

— π_a is the asynchronous π-calculus, namely the subset of π_{nc} without output prefix.

In Figure 1 some encodings are the obvious identity encodings holding between a language and a superset of it. As for the non-trivial encodings, (1) has been proposed by Nestmann and Pierce (2000), (2) has been proposed by Nestmann (2000), (3) represents the two encodings proposed by Honda and Tokoro (1991) and by Boudol (1992), and (4) is based on Milner's encoding of recursion into iteration. The three non-encodings are

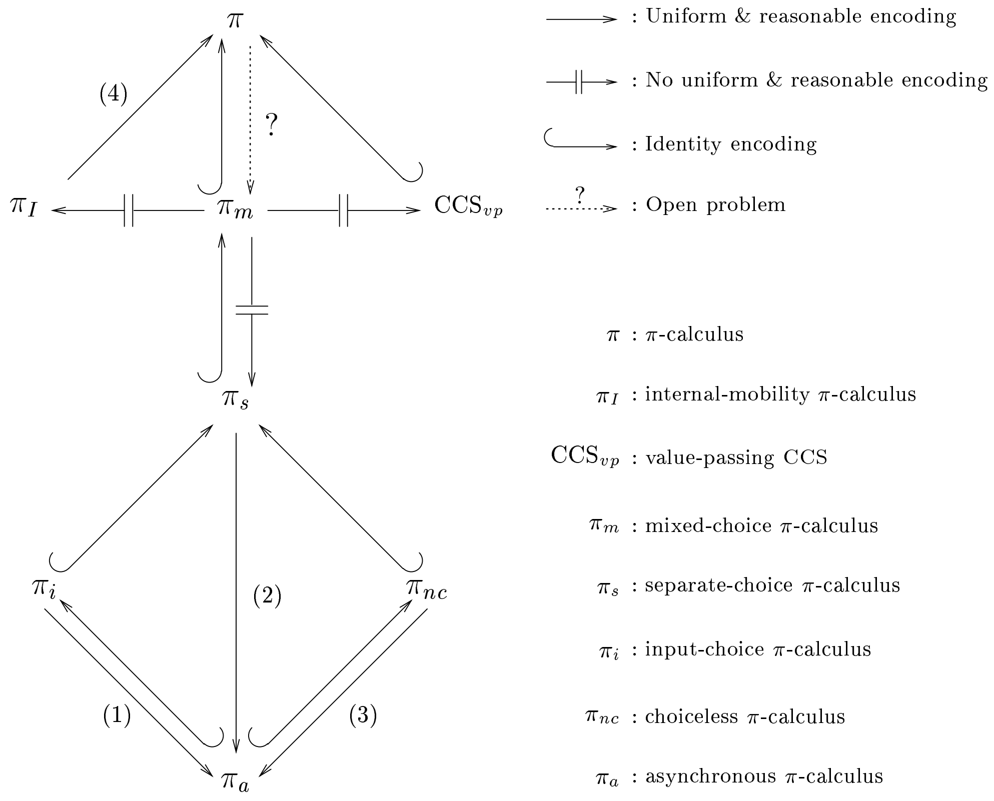

Fig. 1. The π-calculus hierarchy.

presented in this paper. The encoding of π into π_m is an open problem. It is likely that the free choice construct and the match and mismatch operators add expressive power. However, this result may not be obtainable with the weak requirements that we have in this paper for the notion of encoding.

Figure 1 shows only a few of the many variants of the π-calculus and of the many encodings and separation results which have been investigated in literature. We have considered here only the ones which to our opinion are the most relevant to the issues investigated in this paper. For a much more exhaustive overview of the variants of π and of their expressive power we recommend the excellent book of Sangiorgi and Walker (2001).

1.6. *Organization of the paper*

The rest of the paper is organized as follows: next section recalls basic definitions. Section 3 reformulates in the setting of the π-calculus the notions of symmetric and electoral system. Section 4 shows the main result of the paper, i.e. the non-existence of symmetric electoral systems in the asynchronous π-calculus. Section 5 discusses existence of symmetric electoral systems for the synchronous case, i.e. the π-calculus, CCS_{vp}, and π_I. Section

6 interprets previous results as non-encodability results. Section 7 discusses related work and concludes.

1.7. *Relation with the previous version of this work*

A preliminary version of this paper appeared in (Palamidessi, 1997). The principal differences in the present version are: (a) The main separation result was previously shown between the mixed-choice π-calculus and the asynchronous π-calculus. Here we show that the separation lies exactly between the mixed-choice and the separate-choice (this could also be proved indirectly by combining the result in (Palamidessi, 1997) and the result in (Nestmann, 2000)). (b) We give the proof of the separation between the π-calculus and CCS_{vp}. (In (Palamidessi, 1997) this proof was only sketched.) (c) We show an additional separation result, between the π-calculus and π_I, similar to the one between the π-calculus and CCS_{vp}. (d) We consider a weaker condition on the notion of encoding (and hence we strengthen the separation results).

2. Preliminaries

In this section we give the definitions of π_m, of π_s, and of the notion of hypergraph, which will be used to represent the communication structure of a network of processes.

2.1. *The mixed-choice π-calculus*

We present here π_m, the mixed-choice π-calculus. This is a subset of the π-calculus which does not have the match and mismatch operators, and in which the free choice is restricted to be guarded choice. Note that several recent papers adopt a presentation of the π-calculus that actually coincides with π_m, see for instance (Boreale and Sangiorgi, 1998; Sangiorgi, 1996).

Let $\mathcal{N}$ be a countable set of *names*, $x, y, \ldots$. The set of prefixes, $\alpha, \beta, \ldots$, and the set of π-calculus processes, $P, Q, \ldots$, are defined by the following abstract syntax:

$$\begin{array}{lcll} \textit{Prefixes} & \alpha & ::= & x(y) \mid \bar{x}y \mid \tau \\ \textit{Processes} & P & ::= & \sum_i \alpha_i.P_i \mid \nu x P \mid P|P \mid\ !P \end{array}$$

Prefixes represent the basic actions of processes: $x(y)$ is the *input* of the (formal) name y from channel x; $\bar{x}y$ is the *output* of the name y on channel x; τ stands for any silent (non-communication) action.

The process $\sum_i \alpha_i.P_i$ represents guarded (global) choice and it is usually assumed to be finite. We will use the abbreviations $\mathbf{0}$ (*inaction*) to represent the empty sum, $\alpha.P$ (*prefix*) to represent sum on one element only, and $P+Q$ for the binary sum. The symbols νx, $|$, and $!$ are the *restriction*, the *parallel*, and the *replication* operator, respectively.

To indicate the structure of a process expression we will use the following conventions: $P_0 \,|\, P_1 \,|\, P_2 \,|\, \ldots \,|\, P_{k-1}$ stands for $(\ldots((P_0 \,|\, P_1) \,|\, P_2) \,|\, \ldots \,|\, P_{k-1})$, i.e. the parallel operator is left associative, and $\alpha_1.P_1 \,|\, \alpha_2.P_2$ stands for $(\alpha_1.P_1)|(\alpha_2.P_2)$, i.e. the prefix operator has precedence over $|$. In all other cases of ambiguity we will use parentheses.

The operators νx and $y(x)$ are *x-binders*, i.e. in the processes νxP and $y(x).P$ the occurrences of x in P are considered *bound*, with the usual rules of scoping. The set of the *free names* of P, i.e. those names which do not occur in the scope of any binder, is denoted by $fn(P)$. The *alpha-conversion* of bound names is defined as usual, and the renaming (or substitution) $P\{y/x\}$ is defined as the result of replacing all occurrences of x in P by y, possibly applying alpha-conversion to avoid capture.

The operational semantics is specified via a transition system labeled by *actions* $\mu, \mu' \ldots$. These are given by the following grammar:

$$\textit{Actions} \quad \mu \quad ::= \quad xy \mid \bar{x}y \mid \bar{x}(y) \mid \tau$$

Action xy corresponds to the input prefix $x(z)$, where the formal parameter z is instantiated to the actual parameter y (see Rule I-SUM in Table 1). Action $\bar{x}y$ correspond to the output of a free name. The *bound output* $\bar{x}(y)$ is introduced to model *scope extrusion*, i.e. the result of sending to another process a private (ν-bound) name. The bound names of an action μ, $bn(\mu)$, are defined as follows: $bn(\bar{x}(y)) = \{y\}$; $bn(xy) = bn(\bar{x}y) = bn(\tau) = \emptyset$. Furthermore, we will indicate by $n(\mu)$ all the *names* which occur in μ.

In literature there are two definitions for the transition system of the π-calculus which induce the so-called *early* and *late* bisimulation semantics respectively. Here we choose to present the first one because the early strong bisimulation semantics is coarser than the late one. Therefore, since our notion of reasonable semantics is coarser than strong bisimulation, a separation result with the early transition system is more significant.

The rules for the early semantics are given in Table 1. We use a congruence $\equiv$ and Rule CONG to simplify the presentation. We define this congruence as follows:

(i) $P \equiv Q$ if Q can be obtained from P by alpha-conversion, notation $P \equiv_\alpha Q$,

(ii) $(\nu xP) \mid Q \equiv \nu x(P \mid Q)$ if $x \notin fv(Q)$ (scope expansion).

Some presentation of the labeled transition system of the π-calculus use a coarser definition of $\equiv$ obtained by adding other structural axioms like the commutativity of $\mid$ (see for instance (Milner et al., 1993)). Other presentations, like (Sangiorgi, 1996), define $\equiv$ as alpha conversion only, and use a congruence rule of the form

$$\frac{P' \equiv P \qquad P \xrightarrow{\mu} Q}{P' \xrightarrow{\mu} Q}$$

The reasons why we choose the above intermediate definition of $\equiv$ is because it seems to be the most suitable to prove the main theorem in Section 4. Given the way the systems we consider are structured, we do not need the symmetric axiom for scope expansion.

2.2. *The separate-choice π-calculus*

The separate-choice π-calculus, π_s, is the subset of π_m in which output and input prefixes cannot be present in the same guarded choice. This restriction can be specified by the

I-Sum	$\sum_i \alpha_i.P_i \xrightarrow{xy} P_j\{y/z\}$	$\alpha_j = x(z)$
O/τ-Sum	$\sum_i \alpha_i.P_i \xrightarrow{\alpha_j} P_j$	$\alpha_j = \bar{x}y$ or $\alpha_j = \tau$
Open	$\dfrac{P \xrightarrow{\bar{x}y} P'}{\nu y P \xrightarrow{\bar{x}(y)} P'}$	$x \neq y$
Res	$\dfrac{P \xrightarrow{\mu} P'}{\nu y P \xrightarrow{\mu} \nu y P'}$	$y \notin n(\mu)$
Par	$\dfrac{P \xrightarrow{\mu} P'}{P\|Q \xrightarrow{\mu} P'\|Q}$	$bn(\mu) \cap fn(Q) = \emptyset$
Com	$\dfrac{P \xrightarrow{xy} P' \quad Q \xrightarrow{\bar{x}y} Q'}{P\|Q \xrightarrow{\tau} P'\|Q'}$	
Close	$\dfrac{P \xrightarrow{xy} P' \quad Q \xrightarrow{\bar{x}(y)} Q'}{P\|Q \xrightarrow{\tau} \nu y(P'\|Q')}$	$y \notin fn(P)$
Rep	$\dfrac{P \mid !P \xrightarrow{\mu} P'}{!P \xrightarrow{\mu} P'}$	
Cong	$\dfrac{P' \equiv P \quad P \xrightarrow{\mu} Q \quad Q \equiv Q'}{P' \xrightarrow{\mu} Q'}$	

Table 1. *The early-instantiation transition system for π_m. The symmetric versions of* Par, Com *and* Close *are omitted.*

following modification in the grammar:

$$
\begin{array}{rlcl}
\textit{InputPrefixes} & \alpha^I & ::= & x(y) \mid \tau \\
\textit{OutputPrefixes} & \alpha^O & ::= & \bar{x}y \mid \tau \\
\textit{Processes} & P & ::= & \sum_i \alpha_i^I.P_i \mid \sum_i \alpha_i^O.P_i \mid \nu x P \mid P|P \mid !P
\end{array}
$$

The operational semantics of π_s is the same as that of π_m, and it is described by the rules of Table 1.

2.3. *Hypergraphs and automorphisms*

In this section we recall the definition of *hypergraph*, which generalizes the concept of graph essentially by allowing an edge to connect more than two nodes.

A hypergraph is a tuple $H = \langle N, X, t\rangle$ where N, X are finite sets whose elements are called *nodes* and *edges* (or *hyperedges*) respectively, and t (*type*) is a function which assigns to each $x \in X$ a set of nodes, representing the nodes *connected* by x. We will also use the notation $x : n_1, \ldots, n_k$ to indicate $t(x) = \{n_1, \ldots, n_k\}$.

The concept of graph automorphism extends naturally to hypergraphs: Given a hypergraph $H = \langle N, X, t\rangle$, an *automorphism* on H is a pair $\sigma = \langle \sigma_N, \sigma_X\rangle$ such that $\sigma_N : N \to N$ and $\sigma_X : X \to X$ are permutations which preserve the type of edges, namely for each $x \in X$, if $x : n_1, \ldots, n_k$, then $\sigma_X(x) : \sigma_N(n_1), \ldots, \sigma_N(n_k)$.

It is easy to see that the *composition* of automorphisms, defined componentwise as $\sigma \circ \sigma' = \langle \sigma_N \circ \sigma'_N, \sigma_X \circ \sigma'_X\rangle$, is still an automorphism. Its identity is the pair of identity functions on N and X, i.e. $id = \langle id_N, id_X\rangle$. It is easy to show that the set of automorphisms on H with the composition forms a group.

Given H and σ as above, the *orbit* of $n \in N$ generated by σ is defined as the set of nodes in which the various iterations of σ map n, namely:

$$O_\sigma(n) = \{n, \sigma(n), \sigma^2(n), \ldots, \sigma^{h-1}(n)\}$$

where σ^i represents the composition of σ with itself i times, and h is the least such that $\sigma^h = id$. It is possible to show that the orbits generated by σ constitute a partition of N.

We say that an automorphism σ is *well-balanced* if all its orbits have the same cardinality.

Example 2.1. Figure 2 illustrates various hypergraphs. Hypergraphs 1 and 2 correspond to standard graphs, in the sense that each of their edges connects only two nodes. In both of them we can define well-balanced automorphisms with

— one single orbit with six nodes, or
— two orbits with three nodes each, or
— three orbits with two nodes each

Of course, also the identity is a well-balanced automorphism (as in any hypergraph) and in this case it would have six orbits of cardinality one.

Hypergraph 3 has six nodes and three edges, each of which connecting three nodes. This hypergraph has two well balanced automorphisms (apart from the identity), each with two orbits of cardinality three.

Finally, Hypergraph 4 has seven nodes and three edges, each of which connecting four nodes. This hypergraph does not have any well-balanced automorphism except for the identity, because the central node has three incident edges while every other node has at most two incident edges.

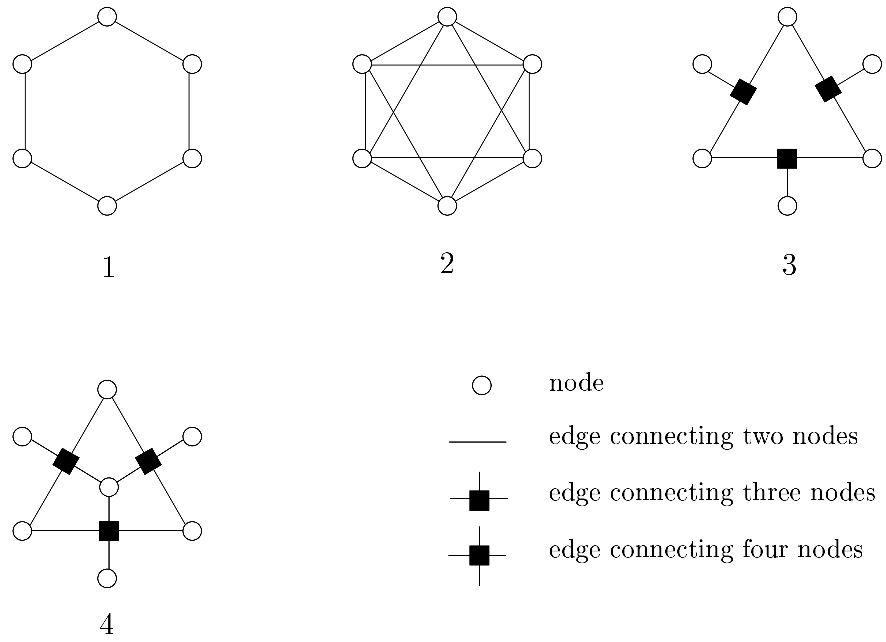

Fig. 2. Examples of hypergraphs.

3. Electoral and Symmetric systems

In this section we adapt to the π-calculi (a simplified version of) the notions of electoral system and symmetric network as given by Bougé (1988).

3.1. *Election of a leader in a network*

We first need to introduce the concepts of *network*, *network computation* and the *projection* of a computation over a component of the network.

A network represents a system of parallel processes with possibly some top-level applications of the restriction operators. The only difference between the notion of network and that of process is that in a network we want to represent explicitly the intended distribution. For instance, the process $((P_0 \mid P_1) \mid P_2)$ (which, because of our associativity convention, we can write as $P_0 \mid P_1 \mid P_2$), may be interpreted as a network of three parallel processes, P_0, P_1 and P_2, or as a network of two parallel processes $P_0|P_1$ and P_2. Formally, a network is defined as a tuple of the form

$$\langle\langle x_0, x_1, \ldots, x_{n-1}\rangle, \langle P_0, P_1, \ldots, P_{k-1}\rangle\rangle \tag{1}$$

where the P_i's are processes meant to run in parallel and the x_i's are names which are meant to be globally bound (i.e. bound at the top level in the whole system). More precisely, the network (1) is meant to represent the process

$$P = \nu x_0 \ \nu x_1 \ldots \nu x_{n-1}(P_0 \mid P_1 \mid \ldots \mid P_{k-1}) \tag{2}$$

It will be convenient to assume that the bound names in $P_0, P_1, \ldots, P_{k-1}$ are different from each other, from $x_0, x_1, \ldots, x_{n-1}$, and from all the free names (bound-names convention).

From now on we will use this process notation to represent the network, with the convention that whenever we write an expression like that in (2) we mean that the network is constituted exactly by the k processes $P_0, P_1, \ldots, P_{k-1}$. We will also use $[Q]$ to denote a process of the form $\nu x_0\ \nu x_1 \ldots \nu x_{n-1}\ Q$. Thus whenever the specific bound names are not relevant we will simply represent the network above as

$$[P_0 \,|\, P_1 \,|\, \ldots \,|\, P_{k-1}].$$

A computation C for a network is a (possibly ω-infinite) sequence of transitions

$$\begin{array}{rcl}
[P_0 \,|\, P_1 \,|\, \ldots \,|\, P_{k-1}] & \xrightarrow{\mu^0} & [P_0^1 \,|\, P_1^1 \,|\, \ldots \,|\, P_{k-1}^1] \\
 & \xrightarrow{\mu^1} & [P_0^2 \,|\, P_1^2 \,|\, \ldots \,|\, P_{k-1}^2] \\
 & \vdots & \\
 & \xrightarrow{\mu^{n-1}} & [P_0^n \,|\, P_1^n \,|\, \ldots \,|\, P_{k-1}^n] \\
 & (\xrightarrow{\mu^n} & \ldots)
\end{array}$$

with $n \geq 0$. Note that at each computation step we may need to apply the Cong rule on the right side of the transition in order:

— to maintain the bound-names convention, i.e. to keep the bound names different from each other and from the free names, and

— to maintain the parallel structure of the network. In fact a transition generated by the Close rule would group a set of processes in the scope of a restriction operator, thus we need the Cong rule with the axioms for scope expansion to bring the restriction operator at the top level and re-establish the number of components to k.

We will represent a computation like the above also by $C : P \stackrel{\tilde{\mu}}{\Longrightarrow} P^n$ (by $C : P \stackrel{\tilde{\mu}}{\Longrightarrow}$ if it is infinite), $\tilde{\mu}$ being the sequence $\mu^0\mu^1 \ldots \mu^{n-1}(\mu^n \ldots)$, and P^n being the process $[P_1^n \,|\, P_2^n \,|\, \ldots \,|\, P_k^n]$. The relation $C \prec C'$ (C' *extends* C) is defined as usual: let $C : P \stackrel{\tilde{\mu}}{\Longrightarrow} P^n$. Then $C \prec C'$ iff there exists $C'' : P^n \stackrel{\tilde{\mu}'}{\Longrightarrow} P^{n+n'}$ with $n' \geq 1$, or $C'' : P^n \stackrel{\tilde{\mu}'}{\Longrightarrow}$, and $C' = CC''$ (identifying the two occurrences of P^n). We will denote by $C' \setminus C$ the continuation C''. Note that according to this definition infinite computations cannot be extended. This is consistent with the fact that we admit only ω-infinite (i.e. not transfinite) computations.

Given P and C as above, the projection of C over P_i, $\mathit{Proj}(C, i)$, is defined as the "contribution" of P_i to the computation. More formally, $\mathit{Proj}(C, i)$ is the sequence of steps

$$\begin{array}{ccc} P_i & \stackrel{\tilde{\mu}^0}{\Longrightarrow} & Q_i \\ P_i^1 & \stackrel{\tilde{\mu}^1}{\Longrightarrow} & Q_i^1 \\ P_i^2 & \stackrel{\tilde{\mu}^2}{\Longrightarrow} & Q_i^2 \\ & \vdots & \\ P_i^{n-1} & \stackrel{\tilde{\mu}^{n-1}}{\Longrightarrow} & Q_i^{n-1} \end{array}$$
$$\left(\begin{array}{ccc} P_i^n & \stackrel{\tilde{\mu}^n}{\Longrightarrow} & Q_i^n \\ & \vdots & \end{array} \right)$$

where the definition of $P_i^m \stackrel{\tilde{\mu}^m}{\Longrightarrow} Q_i^m$ depends on the proof tree T which generates the transition step $[P_0^m \mid P_1^m \mid \ldots \mid P_{k-1}^m] \stackrel{\mu^m}{\longrightarrow} [P_0^{m+1} \mid P_1^{m+1} \mid \ldots \mid P_{k-1}^{m+1}]$:

— If P_i^m is *active* during the transition, namely T contains a node of the form $P_i^m \stackrel{\mu}{\longrightarrow} R$, then

$$(P_i^m \stackrel{\tilde{\mu}^m}{\Longrightarrow} Q_i^m) \quad = \quad (P_i^m \stackrel{\mu}{\longrightarrow} R)$$

Note that either P_i^m is the only process active during this transition, and in this case $\mu^m = \mu$, or P_i^m is involved in a communication step (i.e. it is in the premise of a rule COM or CLOSE) and in this case $\mu^m = \tau$ and μ is a communication action. Note also that Q_i^m and P_i^{m+1} may be different because T may contain at some lower level an application of the CONG rule. However, Q_i^m can only differ from P_i^{m+1} for the presence of restriction operators and/or some renaming.

— If P_i^m is *idle* during the transition, namely T does not contain any node of the form $P_i^m \stackrel{\mu}{\longrightarrow} R$, then $P_i^m \stackrel{\tilde{\mu}^m}{\Longrightarrow} Q_i^m$ is empty, namely $Q_i^m = P_i^m = P_i^{m+1}$ and $\tilde{\mu}^m$ is empty.

Note that the notation $\mathit{Proj}(C, i)$ is not accurate: the projection is not a function of C, but rather of the sequence of proof trees which generate C. However this distinction is inessential here.

In order to define the notion of electoral system we assume the existence of a special channel *out* to be used for communicating with the "external world", and therefore free (unbound). Furthermore we assume that the set of names $\mathcal{N}$ contains a special subset equipped with a one-to-one mapping with the natural numbers, which we will use to identify the individual processes in a network. For the sake of simplicity we shall denote these names directly by natural numbers, but one should keep in mind that this is just a notation, i.e. we are not adding any arithmetical capability to the calculus. We will assume, without loss of generality, that these names are not used as bound names.

Intuitively an electoral system has the property that at each possible run the processes will agree sooner or later on "which of them has to be the leader", and will communicate this decision to the "external world" by using the special channel *out*.

Definition 3.1. (Electoral system) A network $P = [P_0 \mid P_1 \mid \ldots \mid P_{k-1}]$ is an electoral system if for every computation C for P there exists an extension C' of C and there exists $n \in \{0, 1, \ldots, k-1\}$ (the "leader") such that for each $i \in \{0, 1, \ldots, k-1\}$ the projection $\mathit{Proj}(C', i)$ contains one output action of the form $\overline{out}\, n$, and no extension of C' contains any other action of the form $\overline{out}\, m$, with $m \neq n$.

Note that for such a system an infinite computation C must contain already all the output actions of each process because C cannot be extended.

3.2. *Symmetric networks*

In order to define the notion of *symmetric network*, we have to consider its communication structure, which we will represent as an hypergraph. Intuitively the nodes represent the processes, and the edges represent the communication channels connecting the processes. We exclude the special channel *out* since processes cannot use it to communicate with each other.

Definition 3.2. (Hypergraph associated to a network) Given a network $P = [P_0 \,|\, P_1 \,|\, \ldots \,|\, P_{k-1}]$, the hypergraph associated to P is $H(P) = \langle N, X, t\rangle$ with $N = \{0, 1, \ldots, k-1\}$, $X = \mathit{fn}(P_0 \,|\, P_1 \,|\, \ldots \,|\, P_{k-1}) \setminus \{\mathit{out}\}$, and for each $x \in X$, $t(x) = \{n \,|\, x \in \mathit{fn}(P_n)\}$.

We extend now the notion of automorphism to networks so to take into account the use of the special names as process identifiers, and the role of the binders at the top level.

Definition 3.3. (Network automorphism) Given a network

$$P = \nu x_0 \ \nu x_1 \ldots \nu x_{n-1}(P_0 \,|\, P_1 \,|\, \ldots \,|\, P_{k-1})$$

let $H(P) = \langle N, X, t\rangle$ be the hypergraph associated to P. An automorphism on P is any automorphism $\sigma = \langle \sigma_N, \sigma_X\rangle$ on $H(P)$ which satisfies the following additional conditions:

— σ_X coincides with σ_N on $N \cap X$, i.e. for every $n \in N \cap X$ we have $\sigma_X(n) = \sigma_N(n)$ (remember that N is a set of natural numbers and that the natural numbers are assumed to represent also a special subset of names).

— σ_X must preserve the distinction between free and bound names, i.e.

$$x \in \{x_0, x_1, \ldots, x_{n-1}\} \ \text{ if and only if } \ \sigma(x) \in \{x_0, x_1, \ldots, x_{n-1}\}, \ \text{ for each } \ x \in X$$

Thanks to the fact that $\sigma_N(\cdot)$ and $\sigma_X(\cdot)$ coincide on the intersection of their domains, we can simplify the notation and use $\sigma(\cdot)$ to represent both $\sigma_N(\cdot)$ and $\sigma_X(\cdot)$. We will also, with a slight abuse of notation, use the hypergraph H to denote the domain of $\sigma_X(\cdot)$, i.e. the set $N \cup X$.

Intuitively, a network P is symmetric with respect to an automorphism σ iff for each i the process associated to the node $\sigma(i)$ is identical (modulo alpha-conversion) to the process obtained by σ-renaming the process associated to the node i.

The notion of σ-renaming is the obvious extension of the standard notion of renaming (see the preliminaries). More formally, given a process Q, first apply alpha-conversion so to rename all bound names into fresh ones, extend σ to be the identity on these new names, and define $\sigma(Q)$ by structural induction as indicated below.

$$\begin{array}{rcl}
\sigma(\tau) & = & \tau \\
\sigma(x(y)) & = & \sigma(x)(y) \\
\sigma(\bar{x}y) & = & \overline{\sigma(x)}\sigma(y) \\
\sigma(\sum_i \alpha_i.P_i) & = & \sum_i \sigma(\alpha_i).\sigma(P_i) \\
\sigma(\nu x P) & = & \nu x \ \sigma(P) \\
\sigma(P \mid Q) & = & \sigma(P) \mid \sigma(Q) \\
\sigma(!P) & = & !\sigma(P)
\end{array}$$

Furthermore we need to define the application of σ on the actions. For τ and $\bar{x}y$ the definition is the same as above. For the other actions we have:

$$\begin{array}{rcl}
\sigma(xy) & = & \sigma(x)\sigma(y) \\
\sigma(\bar{x}(y)) & = & \overline{\sigma(x)}(y)
\end{array}$$

We are now ready to give the formal definition of symmetric network:

Definition 3.4. (Symmetric network) Consider a network $P = [P_1 \mid P_2 \mid \ldots \mid P_k]$, let $H(P) = \langle N, X, t\rangle$ be its associated hypergraph, and let σ be an automorphism on P. We say that P is *symmetric wrt* σ iff for each node $i \in N$, $P_{\sigma(i)} \equiv_\alpha \sigma(P_i)$ holds. We also say that P is *symmetric* if it is symmetric wrt all the automorphisms on $H(P)$.

Note that if P is symmetric wrt σ then P is symmetric wrt all the powers of σ.

4. Non existence of symmetric electoral systems in π_s

In this section we present our first result, which says that for certain communication graphs it is not possible to write in π_s a symmetric network solving the election problem.

We first need to show that the π_s enjoys a certain kind of *confluence* property:

Lemma 4.1. Let P be a process in π_s. Assume that P can make two transitions $P \xrightarrow{\bar{x}[y]} Q$ and $P \xrightarrow{zw} R$, where $\bar{x}[y]$ stands for an output action either bound ($\bar{x}(y)$) or unbound ($\bar{x}y$). Then there exists S such that $Q \xrightarrow{zw} S$ and $R \xrightarrow{\bar{x}[y]} S$ (see Figure 3).

Proof Observe that x and z must be free names in P. The rule which has produced the $\bar{x}[y]$ transition can be only O/τ-Sum, Open, Res, Par, Rep, or Cong. In the last five cases the assumption is again a $\bar{x}[y]$ transition. By repeating this reasoning (descending the tree), we must arrive to a leaf transition of the form

$$P^O = \sum_i \alpha_i^O.P_i \xrightarrow{\alpha_j^O} P_j \quad \text{where } \alpha_j^O = \bar{x}y$$

Analogously, the rule which has produced the zw transition can be only I-Sum, Res,

$$\begin{tikzcd}
& P \arrow[dl, "\bar{x}[y]"'] \arrow[dr, "zw"] & \\
Q \arrow[dr, dashed, "zw"'] & & R \arrow[dl, dashed, "\bar{x}[y]"] \\
& S &
\end{tikzcd}$$

Fig. 3. An illustration of Lemma 4.1.

Par, Rep, or Cong. In the last four cases the assumption is again a zw transition. By repeating this reasoning (descending the tree), we must arrive to a leaf transition of the form

$$P^I = \sum_i \alpha_i^I.Q_i \xrightarrow{\alpha_j^I} Q_j\{w/v\} \quad \text{where } \alpha_j^I = z(v)$$

Now, P^O and P^I must be two parallel processes in P, i.e. there must be a subprocess in P of the form $T[P^I] \,|\, U[P^O]$ (modulo $\equiv$), i.e. $P \equiv V[T[P^O] \,|\, U[P^I]]$ (here $T[\,]$, $U[\,]$ and $V[\,]$ represent contexts, with the usual definition). Furthermore, the $\bar{x}[y]$ transition and the zw transition must have been obtained by the application of the rule Par to this subprocess, i.e. $Q \equiv V[T[P_j] \,|\, U[P^I]]$ and $R \equiv V[T[P^O] \,|\, U[Q_j\{w/v\}]]$. By applying again the rule Par (plus all the other rules in the trees for the $\bar{x}[y]$ and the zw transitions) we obtain the transitions $Q \xrightarrow{zw} S$ and $Q' \xrightarrow{\bar{x}[y]} S$ where $S = V[T[P_j] \,|\, U[Q_j\{w/v\}]]$. $\square$

We are now ready to prove the non-existence result. The intuition is the following: In the attempt to reach an agreement about the leader, the processes of a symmetric network have to “break the initial symmetry”, and therefore have to communicate. The first such communication, however, can be repeated, by the above lemma and by symmetry, by all the pair of processes of the network. The result of all these transitions will still lead to a symmetric situation. Thus there is a (infinite) computation in which the processes never succeed to break the symmetry, which means no leader is elected.

Theorem 4.2. Consider a network $P = [P_0 \,|\, P_1 \,|\, \ldots \,|\, P_{k-1}]$ in π_s, with $k \geq 2$. Assume that P has an automorphism σ with only one orbit, and that P is symmetric wrt σ. Then P cannot be an electoral system.

Proof Assume by contradiction that P is an electoral system. We will show that we can then construct an infinite increasing sequence of computations for P, $C_0 \prec C_1 \prec \ldots \prec C_h \ldots$, such that for each j, $C_j : P \stackrel{\tilde{\mu}^j}{\Longrightarrow} P^j$ does not contain any action of the form $\overline{out}\, n$, and P^j is still symmetric wrt σ_j, where σ_j is an automorphism with only one orbit obtained from σ by adding associations on the new names introduced during the computation, and by eliminating the associations on the names that have disappeared.

This gives a contradiction, because the limit of this sequence is an infinite computation for P which does not contain any action of the form $\overline{out}\,n$.

We will prove the above statement by induction. In order to understand the proof, it is important to notice that if σ has only one orbit then, for each $i \in \{0, 1, \ldots, k-1\}$,

$$O_\sigma(i) = \{i, \sigma(i), \ldots, \sigma^{k-1}(i)\} = \{0, 1, \ldots, k-1\}$$

$h = 0)$ Define C_0 to be the empty computation.

$h+1)$ Let $C_h : P \stackrel{\tilde{\mu}^h}{\Longrightarrow} P^h$ where P^h contains k processes and is symmetric wrt a one-orbit automorphism σ_h. We show how to construct $C_{h+1} : P \stackrel{\tilde{\mu}^{h+1}}{\Longrightarrow} P^{h+1}$, and a new one-orbit automorphism σ_{h+1} for P^{h+1}, such that P^{h+1} contains k processes and is symmetric wrt σ_{h+1}.

Since P is an electoral system, it must be possible to extend C_h to a computation C which contains (k) actions $\overline{out}\,n$, for a particular $n \in \{0, 1, \ldots, k-1\}$. Observe that the first action μ of $C \setminus C_h$ cannot be $\overline{out}\,n$. Otherwise, let P_i^h be the component which performs this action. Then P_i^h must contain the subprocess $\overline{out}\,n$. By symmetry, $P^h_{\sigma_h(i)} \equiv_\alpha \sigma_h(P_i^h)$ and therefore $P^h_{\sigma_h(i)}$ must contain the subprocess $\overline{out}\,\sigma(n)$. Furthermore, $\sigma_h(n)$ is free, because we have assumed that numbers cannot be used as bound names. Hence there must be an extension of C where the action $\overline{out}\,\sigma_h(n)$ occurs. This implies (for the hypothesis that P is an electoral system), that $\sigma_h(n) = n$. Given that $k \geq 2$, σ_h must generate more than one orbit. Contradiction.

In conclusion, μ must be an action different from $\overline{out}\,n$. We have two different situations depending on whether the transition is generated by the move of one process only, or by a communication between two processes (i.e. it involves the COM or the CLOSE rule).

The transition is the result of the move of one process only) In this part of the proof, in order to simplify the notation we will assume, without loss of generality, that

$$\sigma(0) = 1,\ \sigma^2(0) = 2,\ \ldots\ \sigma^{k-1}(0) = k-1,\ \sigma^k(0) = 0$$

Therefore

$$P = [P_0 \,|\, P_{\sigma(0)} \,|\, \ldots \,|\, P_{\sigma^{k-1}(0)}]$$

and, by symmetry,

$$P \equiv_\alpha [P_0 \,|\, \sigma(P_0) \,|\, \ldots \,|\, \sigma^{k-1}(P_0)].$$

We also assume, without loss of generality, that P_0^h is the component that performs the step, and let this step be

$$P_0^h \stackrel{\mu_0}{\longrightarrow} P_0^{h+1}$$

Using the symmetry of P^h, i.e. the fact that $P_i^h = P^h_{\sigma_h^i(0)} \equiv_\alpha \sigma_h^i(P_0^h)$ for every $i \in \{0, 1, \ldots, k-1\}$, we can mimic the step of P_0^h with every P_i^h and derive the transitions

$$\begin{array}{ccc} P_1^h & \xrightarrow{\mu_1} & P_1^{h+1} \\ P_2^h & \xrightarrow{\mu_2} & P_2^{h+1} \\ & \vdots & \\ P_{k-1}^h & \xrightarrow{\mu_{k-1}} & P_{k-1}^{h+1} \end{array}$$

such that

— μ_i is not of the form $\overline{out}\, n$ for any $i, n \in \{0, 1, \ldots, k-1\}$,
— the bound-names convention is respected in the P_i^{h+1}'s.
— the bound names of μ_i (if any) are different from the free names of all the other processes (and because of the bound-names convention in P^h, we do not even need to use α conversion here).

Thanks to the latter property, we can compose the displayed transitions into a computation

$$P^h \stackrel{\tilde{\mu}}{\Longrightarrow} P^{h+1}$$

where:

— $P^{h+1} = [P_0^{h+1} \mid P_1^{h+1} \mid \ldots \mid P_{k-1}^{h+1}]$,
— $\tilde{\mu} = \mu'_0 \mu'_1 \ldots \mu'_{k-1}$, where each μ'_i is equal to μ_i except if μ_i is a free output action, in which case the argument may become bound in μ'_i due to the restrictions at the top level in P^h.

It remains to show that we can construct a one-orbit automorphism σ_{h+1} for P^{h+1} such that P^{h+1} is symmetric wrt it. To this purpose we need to distinguish various cases depending on μ_0.

$\mu_0 = \tau$) In this case we have that, for every $i \in \{0, 1, \ldots, k-1\}$, $\mu_i = \tau$ and $P_i^{h+1} \equiv_\alpha \sigma_h^i(P_0^{h+1})$. Hence we can simply define σ_{h+1} as σ_h restricted to (the edges and the nodes of) $H(P^{h+1})$.

$\mu_0 = \bar{x}_0 y_0$) We have that, for every $i \in \{0, 1, \ldots, k-1\}$, $\mu_i = \overline{\sigma_h^i(x_0)} \sigma_h^i(y_0)$ and $P_i^{h+1} \equiv_\alpha \sigma_h^i(P_0^{h+1})$. Hence also in this case we can define σ_{h+1} as σ_h restricted to $H(P^{h+1})$.

$\mu_0 = \bar{x}_0(y_0)$) We have that, for every $i \in \{0, 1, \ldots, k-1\}$, $\mu_i = \overline{\sigma_h^i(x_0)}(y_i)$ for some y_i that is different from every other bound and free name in the other components of P^h.

Define σ_{h+1} as follows:

$$\sigma_{h+1}(z) = \begin{cases} y_{\sigma_h(i)} & \text{if } z = y_i \text{ and } z \in H(P^{h+1}) \\ \sigma_h(z) & \text{if } z \neq y_i \text{ and } z \in H(P^{h+1}) \\ \text{undefined} & \text{otherwise} \end{cases} \qquad (3)$$

Note that σ_{h+1} is well-defined, because for all $j \in \{0, 1, \ldots, k-1\}$ $\sigma_{h+1}(j) = \sigma_h(j)$

since j does not occur bound, and therefore j cannot be one of the y_i's. Hence

$$P_j^{h+1} = P^{h+1}_{\sigma^j_{h+1}(0)}$$

Finally, observe that σ_{h+1} generates only one orbit and that

$$P_j^{h+1} \equiv_\alpha \sigma^j_{h+1}(P_0^{h+1})$$

Therefore P^{h+1} is symmetric wrt σ_{h+1}.

$\mu_0 = x_0 y_0$) If y_0 is not new, i.e. it occurred in P^h, or if $y_0 \in \{0, 1, \dots, k-1\}$, then we can choose the transitions above so that for every $i \in \{0, 1, \dots, k-1\}$, $\mu_i = \sigma_h^i(x_0)\sigma_h^i(y_0)$ and $P_i^{h+1} \equiv_\alpha \sigma_h^i(P_0^{h+1})$. Hence also in this case we can define σ_{h+1} as σ_h restricted to $H(P^{h+1})$.
If, on the contrary, y_0 is new, then we can choose the transitions above so that for every $i \in \{0, 1, \dots, k-1\}$, $\mu_i = \sigma_h^i(x_0)y_i$ for some y_i that is also new. (this is possible because in the early semantics we can instantiate an input action with an arbitrary name). Then, proceed as in the case ($\mu_0 = \bar{x}_0(y_0)$).

The transition results from the communication of two processes) This is the part of the proof where we use the specific property of π_s illustrated by Lemma 4.1.
The interesting case is when the two agents are in *different nodes* of the communication graph. (If the agents are inside the same node, say P_i^h, then we have a transition $P_i^h \xrightarrow{\tau} P_i^{h+1}$ and we proceed like in previous case.) Let P_i^h and P_j^h be the two processes, with $i \neq j$. We have two transitions $P_i^h \xrightarrow{\mu_i} Q_i$ and $P_j^h \xrightarrow{\mu_j} R_j$, where μ_i and μ_j are complementary. Assume without loss of generality that μ_i is the input action, and μ_j is the output action. Since σ_h generates only one orbit, there exists $r \in \{1, \dots, k-1\}$ such that $j = \sigma_h^r(i)$. Assume for simplicity that r and k are relatively prime[6], and let $\theta = \sigma_h^r$. Then $P_j^h = P_{\theta(i)}^h$ and we can write $R_{\theta(i)}$ for R_j. Let us first consider the case in which the first step of $C \setminus C_h$ has been produced by an application of the COM rule. Then we have a transition

$$P_i^h \mid P_{\theta(i)}^h \xrightarrow{\tau} Q_i \mid R_{\theta(i)}$$

By symmetry, we have that $P_{\theta(i)}^h \xrightarrow{\theta(\mu_i)} \theta(Q_i)$. By Lemma 4.1 we then have the transitions $R_{\theta(i)} \xrightarrow{\theta(\mu_i)} R'$ and $\theta(Q_i) \xrightarrow{\mu_j} R'$ for some R'. Let us define $P_{\theta(i)}^{h+1} = R'$. By symmetry, we also have $P_{\theta^2(i)}^h \equiv P_{\theta(j)}^h \xrightarrow{\theta(\mu_j)} \theta(R_j)$, and $\theta(\mu_i)$, $\theta(\mu_j)$ are complementary, hence we can combine them into a transition

$$R_{\theta(i)} \mid P_{\theta^2(i)}^h \xrightarrow{\tau} P_{\theta(i)}^{h+1} \mid R_{\theta^2(i)}$$

[6] If they are not, then in the rest of the proof k has to be replaced by the least p such that $pk = rq$, for some q.

with $R_{\theta^2(i)} = \theta(R_j)$. By repeatedly applying this reasoning, we obtain

$$\begin{array}{rcl} R_{\theta^2(i)} \mid P^h_{\theta^3(i)} & \xrightarrow{\tau} & P^{h+1}_{\theta^2(i)} \mid R_{\theta^3(i)} \\ & \vdots & \\ R_{\theta^{k-2}(i)} \mid P^h_{\theta^{k-1}(i)} & \xrightarrow{\tau} & P^{h+1}_{\theta^{k-2}(i)} \mid R_{\theta^{k-1}(i)} \end{array}$$

and $R_{\theta^{k-1}(i)} \xrightarrow{\theta^{k-1}(\mu_i)} P^{h+1}_{\theta^{k-1}(i)}$. Finally, observe that from the transition $\theta(Q_i) \xrightarrow{\mu_j} R'$ above we can derive $\theta^k(Q_i) \xrightarrow{\theta^{k-1}(\mu_j)} \theta^{k-1}(R')$. But $\theta^k = \sigma_h^{kr} = id$, hence we have $Q_i \xrightarrow{\theta^k(\mu_j)} P_i^{h+1}$, where we have defined P_i^{h+1} to be $\theta^{k-1}(R')$. Therefore we can compose also these transitions, thus "closing the circle", and we obtain

$$R_{\theta^{k-1}(i)} \mid Q_i \xrightarrow{\tau} P^{h+1}_{\theta^{k-1}(i)} \mid P_i^{h+1}$$

The composition of the displayed transitions gives us the intended continuation[7]:

$$P^h = [P_i^h \mid P^h_{\theta(i)} \mid \ldots \mid P^h_{\theta^{k-1}(i)}] \stackrel{\tilde{\tau}}{\Longrightarrow} [P_i^{h+1} \mid P^{h+1}_{\theta(i)} \mid \ldots \mid P^{h+1}_{\theta^{k-1}(i)}]^8$$

Finally define $P^{h+1} = [P_i^{h+1} \mid P^{h+1}_{\theta(i)} \mid \ldots \mid P^{h+1}_{\theta^{k-1}(i)}]$ and observe that σ_h (restricted to $H(P^{h+1})$) is still an automorphism for P^{h+1} and that P^{h+1} is still symmetric with respect to it. Note that $H(P^{h+1})$ may differ from $H(P^h)$ because some edges may have disappeared and because the COM rule may have extended the set of nodes that share a certain edge. However, $H(P^{h+1})$ does not contain any new edges because COM only transmits free names, corresponding to existing edges. Hence we can define σ_{h+1} as the restriction of σ_h to $H(P^{h+1})$.
Consider now the case in which the first step of $C \setminus C_h$ is obtained by an application of the CLOSE rule. Then the transition is of the form

$$P_i^h \mid P^h_{\theta(i)} \xrightarrow{\tau} \nu y_i(Q_i \mid R_{\theta(i)})$$

where y_i is the name transmitted in the communication. By following the same rea-

[7] Under the assumption that r and k are relatively prime, also θ has only one orbit. If we drop this assumption, and hence we replace k by the smallest p such that $pk = rq$ for some q, then the computation we have constructed involves only the processes of the nodes in $O_\theta(i) = \{i, \theta(i), \ldots, \theta^{p-1}(i)\}$. To complete computation we have to repeat the reasoning for the other orbits of θ: $O_\theta(\sigma_h(i))$, $O_\theta(\sigma_h^2(i)) \ldots O_\theta(\sigma_h^{q-1}(i))$.

[8] We are using a sloppy notation here: the processes should be permuted so to have their indexes in increasing order.

soning as before, we obtain the transitions

$$\begin{array}{rcl}
R_{\theta(i)} \mid P^h_{\theta^2(i)} & \xrightarrow{\tau} & \nu y_{\theta(i)}(P^{h+1}_{\theta(i)} \mid R_{\theta^2(i)}) \\
R_{\theta^2(i)} \mid P^h_{\theta^3(i)} & \xrightarrow{\tau} & \nu y_{\theta^2(i)}(P^{h+1}_{\theta^2(i)} \mid R_{\theta^3(i)}) \\
& \vdots & \\
R_{\theta^{k-2}(i)} \mid P^h_{\theta^{k-1}(i)} & \xrightarrow{\tau} & \nu y_{\theta^{k-2}(i)}(P^{h+1}_{\theta^{k-2}(i)} \mid R_{\theta^{k-1}(i)}) \\
R_{\theta^{k-1}(i)} \mid Q_i & \xrightarrow{\tau} & \nu y_{\theta^{k-1}}(P^{h+1}_{\theta^{k-1}(i)} \mid P^{h+1}_i)
\end{array}$$

Note that, thanks to the bound-names convention for P^h, all the y_j's are different from each other and from the free variables. We can then combine the above transitions, and use the *Cong* rule with scope expansion to push the restriction operators at the top-level of the network, thus obtaining the derivation

$$P^h = [P^h_i \mid P^h_{\theta(i)} \mid \ldots \mid P^h_{\theta^{k-1}(i)}] \stackrel{\tilde{\tau}}{\Longrightarrow} [P^{h+1}_i \mid P^{h+1}_{\theta(i)} \mid \ldots \mid P^{h+1}_{\theta^{k-1}(i)}]$$

Note that $H(P^{h+1})$ may contain some of the y_j's as additional edges, because if these names occur in the P^{h+1}_j's, they occur free. We need therefore to expand the automorphism accordingly. This can be done by defining σ_{h+1} exactly as in (3). It is easy to see that σ_{h+1} has one orbit and that P^{h+1} is symmetric wrt it. □

In (Bougé, 1988) a less restrictive notion of symmetry is considered for proving negative results. Namely, the automorphism σ can have more orbits, provided that they all have the same cardinality (i.e. σ can be well-balanced). In the framework of (Bougé, 1988) this is a significant generalization, because the language considered there, CSP_{in}, can have the parallel operator only at the top level. Hence the condition of a single orbit, there, would impose that *all* the parallel processes present in the network have the same code (modulo renaming).

In our framework, on the contrary, we do not have this restriction, and the above mentioned generalization is not essential. In fact, we can easily extend Theorem 4.2 to well-balanced automorphisms:

Corollary 4.3. Consider a network $P = [P_0 \mid P_1 \mid \ldots \mid P_{k-1}]$ in π_s, and assume that the associated hypergraph $H(P)$ admits a well-balanced automorphism $\sigma \neq id$, and that P is symmetric wrt σ. Then P cannot be an electoral system.

Some examples of hypergraphs with well-balanced automorphisms are the hypergraphs 1 and 2 in Figure 4. The nodes with the same filling represent nodes in the same orbit.

Proof of Corollary 4.3 The idea is to transform a network P with a well-balanced automorphism into a network Q with a one-orbit automorphism by grouping together the nodes in the same orbit in $H(P)$ into one single node in $H(Q)$. For example, Hypergraphs 1 and 2 in Figure 4 are transformed into Hypergraphs 3 and 4, respectively.

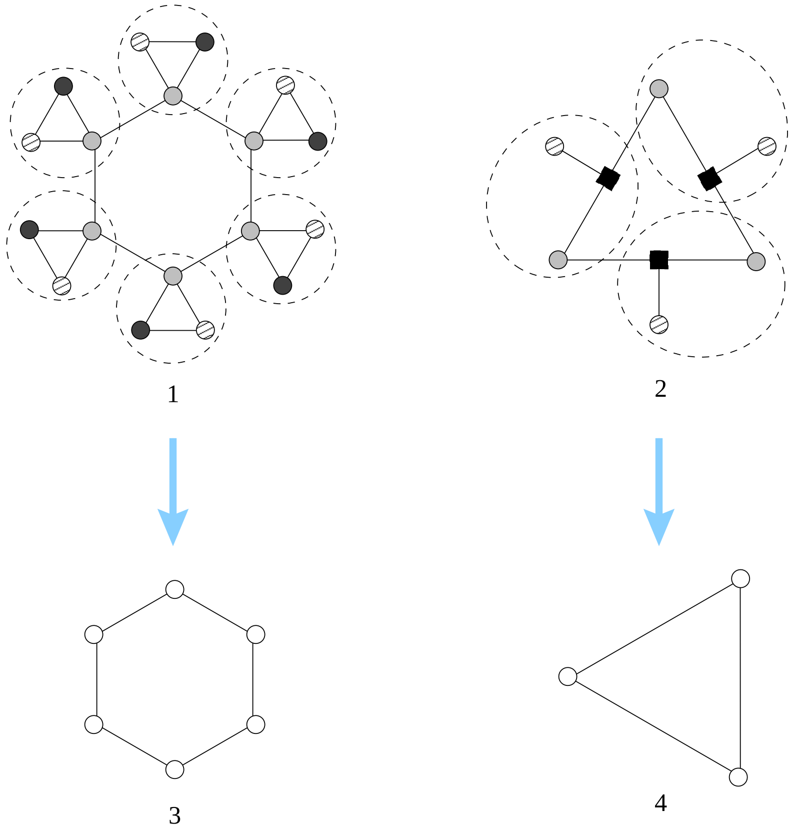

Fig. 4. Examples of hypergraphs with well-balanced automorphisms and their transformation into hypergraphs with one-orbit automorphisms.

Assume that σ generates p orbits of cardinality q, and assume, without loss of generality, that $0, 1, \ldots, p-1$ belong to different orbits, and

$$\begin{array}{cccc}
\sigma(0) = p & \sigma(1) = p+1 & \ldots & \sigma(p-1) = 2p-1 \\
\sigma^2(0) = 2p & \sigma^2(1) = 2p+1 & \ldots & \sigma^2(p-1) = 3p-1 \\
\vdots & \vdots & \vdots & \vdots \\
\sigma^{q-1}(0) = (q-1)p & \sigma^{q-1}(1) = (q-1)p+1 & \ldots & \sigma^{q-1}(p-1) = qp-1
\end{array}$$

Define the processes

$$\begin{array}{rcl}
Q_0 & = & P_0 \,|\, P_1 \,|\, \ldots \,|\, P_{p-1} \\
Q_1 & = & P_p \,|\, P_{p+1} \,|\, \ldots \,|\, P_{2p-1} \\
 & \vdots & \\
Q_{q-1} & = & P_{(q-1)p} \,|\, P_{(q-1)p+1} \,|\, \ldots \,|\, P_{qp-1}
\end{array}$$

Consider now the network $Q = [Q_0 \,|\, Q_1 \,|\, \ldots \,|\, Q_{q-1}]$. Clearly Q and P generate the same computations (they are strongly bisimilar), but the associated hypergraph, $H(Q)$, is different: $H(Q)$ is "an abstraction" of $H(P)$ in the sense that certain nodes of $H(P)$ are "grouped together" in the same node of $H(Q)$, as explained before.

Note that Q may contain names corresponding to non-existing nodes. To eliminate them, consider the renaming $\rho : \{0, 1, \ldots, qp-1\} \to \{0, 1, \ldots, q-1\}$ defined by

$$\rho(n) = n \ \mathbf{div} \ p$$

i.e. $\rho(n)$ is the result of the integer division of n by p, and define $Q' = \rho(Q)$. It is easy to see that the traces of the projections of Q' are the same as those in Q modulo the renaming ρ.

In the hypergraph $H(Q')$ the nodes N are, of course, $0, 1, \ldots, p-1$. The edges X are the same as the edges of $H(P)$ minus $\{p, p+1, \ldots, qp-1\}$. (Note that in Figure 4 the edges internal to the nodes of the transformed graph are not represented.)

Now, consider the pair $\theta = \langle \theta_N, \theta_X \rangle$ with $\theta_N(0) = 1$, $\theta_N(1) = 2, \ldots, \theta_N(q-1) = 0$, and $\theta_X = \sigma_X$ restricted to $H(Q')$. It is easy to see that θ is an automorphism on $H(Q')$ with only one orbit, and that Q' is symmetric wrt θ.

Finally, observe that if P is an electoral system then also Q' is an electoral system, and apply Theorem 4.2. ☐

5. Existence of symmetric electoral systems in π_m

The negative result of previous section does not apply to π_m: its mixed-choice construct, in fact, makes it possible to establish a simultaneous agreement among two processes, thus breaking the symmetry. Note that the presence of mixed choice invalidates the confluence property of Lemma 4.1.

Consider for example the election problem in a symmetric network consisting of two nodes P_0 and P_1 only, and two private edges, x_0 and x_1, connecting them. A π-calculus specification which solves the problem is $P = \nu x_0 \, \nu x_1 \; (P_0 \,|\, P_1)$, where:

$$\begin{array}{rcl} P_i & = & \overline{x}_i y.\overline{out}\ i \\ & & + \\ & & x_{i\oplus 1}(y).\overline{out}\ \langle i \oplus 1 \rangle \end{array} \tag{4}$$

with $i \in \{0, 1\}$ and $\oplus$ being the sum modulo 2. The argument y is not relevant, it is present only because in π_m actions must have an argument.

Note that the above electoral system, although very simple, has an automorphism with only one orbit, hence by Theorem 4.2 it cannot be expressed in π_s.

What happens when the hypergraph is more complicated? We claim that in π_m the existence of symmetric electoral systems is guaranteed in a large number of cases:

Claim 1. Let H be a *connected* hypergraph (i.e. each pair of nodes are connected by a sequence of edges). Then there exists a symmetric electoral system P in π_m such that $H(P) = H$. (Remember that "symmetric" means symmetric wrt every possible automorphism on the hypergraph.)

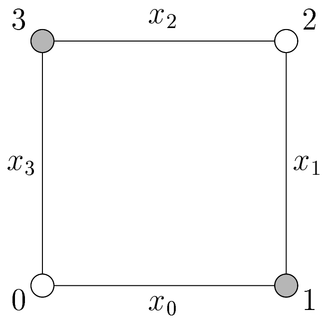

Fig. 5. A hypergraph which is connected, but not fully connected since the two pairs of nodes with the same fillings are not directly connected by any edge.

Our claim is substantiated by the following idea for an electoral algorithm:
Let k be the number of nodes. The generic process P_i executes the following:

1 Broadcast a private name y_i to all the other processes (possible thanks to the connectivity hypothesis) and, meanwhile, receive the private name y_j of each other process P_j. In this way the hypergraph becomes fully connected.
2 Repeat (at most k times) a choice where one guard is an output action on y_i, while the others are input actions on the y_j's. If at a certain point an input is selected, then go to 4.
3 If this point is reached, then P_i is the leader. Broadcast this information to all the other processes, output $\overline{out}\ i$ and terminate.
4 Wait to receive the name of the leader. Then send it on out and terminate. □

Note that the above algorithm works under the assumption that each process knows what is the total number of processes in the network.

It is difficult to make the above argument more formal while keeping it general, since the details of the algorithm (like how to broadcast the private name) depend on the structure of the hypergraph.

For proving separation results between π_m and the other languages, however, it is sufficient to show that π_m can solve the symmetric electoral problem in one hypergraph, suitably chosen. We will consider the simple hypergraph in Figure 5.

Proposition 5.1. Let H be the hypergraph illustrated in Figure 5. Then, there exists a symmetric electoral system $P = \nu x_0\, \nu x_1\, \nu x_2\, \nu x_3\, (P_0 \,|\, P_1 \,|\, P_2 \,|\, P_3)$, in π_m, such that $H(P) = H$.

Proof For every $i \in \{0, 1, 2, 3\}$ we define P_i following the idea illustrated above. First P_i broadcasts its private name y_i to all the other nodes, and receives the private names of all the other nodes (first phase). Then each process uses y_i to compete for the election (second phase).

For the first phase of the algorithm, we define

$$P_i = \nu y_i\, P_i^3 \tag{5}$$

where, intuitively, P_i^3 represents a process that receives three names from its left neighbor

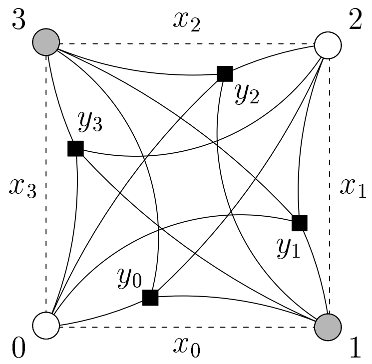

Fig. 6. The evolution of the hypergraph in Figure 5 after running the first phase of the algorithm, i.e.the code described in (5) and (6) up to the Q_i^3's.

via the channel x_{i-i} and sends three names to its right neighbor via the channel x_i, in such a way that the first name sent is y_i, and the other two are the first two names received from the left neighbor. In this way each name is broadcasted to all processes.

At this point it would seem natural to define $P_i^k = \overline{x}_i y_{i+k+1} \,|\, x_{i-1}(y_{i+k}) \,.\, P_i^{k-1}$ for $1 \leq k \leq 3$, but there is a problem: the output $\overline{x}_i y_{i+k}$ in P_i^{k-1} may become available before $\overline{x}_i y_{i+k+1}$ is consumed, so the right neighbor may receive y_{i+k} before y_{i+k+1}. This would be incorrect because the order is relevant: the first name received (intended to be y_{i+k+1}) will be retransmitted $k-1$ times while the second (intended to be y_{i+k}) will be retransmitted only $k-2$ times.

In order to solve the above problem we need to sequentialize the output actions. We do this by using an idea similar the one used Honda and Tokoro (1991) for the encoding of the output prefix. Namely, we replace each action by a pair of actions: $\overline{x}y$ becomes $x(w)$ followed by $\overline{w}y$, and $x(y)$ becomes $\overline{x}w$, where w is a fresh name, and $w(y)$.

The definition of P_i^k then is as follows (the symbols $+$ and $-$ here represent the sum and difference modulo 4, respectively):

$$
\begin{array}{rcll}
P_i^k & = & \nu w \ (\overline{x}_{i-1} w \mid x_i(w') \,.\, (\overline{w}' y_{i+k+1} \mid w(y_{i+k}) \,.\, P_i^{k-1})\ ) & \text{for } 1 \leq k \leq 3 \\
P_i^0 & = & Q_i^3 &
\end{array}
\tag{6}
$$

The purpose of Q_i^3 is to perform the second phase of the algorithm, i.e. to compete for the election. The meaning of the superscript 3 will be clear later. By the time each P_i reaches the point Q_i^3, the original hypergraph has evolved into the fully connected hypergraph illustrated in Figure 6. Note that the x_i's are no longer there because they do not occur in the Q_i^3's.

Interestingly, this first phase of the algorithm is within the syntax of π_s and π_a, but not of π_I nor of the π-calculus version of CCS$_{vp}$. By "π-calculus version of CCS$_{vp}$" we mean the language obtained from CCS$_{vp}$ by replacing the restriction operator of CCS$_{vp}$ with the one of the π-calculus, and other obvious changes of this kind.

We now describe the Q_i^k's, for $0 \leq k \leq 3$. As stated above, their purpose is to compete for the election. Each node i tries to "dominate" the other nodes by sending messages on its own channel y_i. Each time i sends a message on y_i to another node, say j, the nodes dominated by j, and j itself, become dominated by i. The superscript k, initially 3, in Q_i^k indicates how many additional nodes i needs to dominate before becoming a winner. Obviously i becomes the winner whenever it succeed to dominate all the other nodes, i.e. whenever $k = 0$. At this point i executes Q_i^0, namely it tells the name of the winner (its own name i) to each other node j by using j's channel y_j, and then it outputs its own name i on *out*. Note that in order for i to know that it is the winner it is sufficient to keep track of the *number* of nodes that it dominates[9]. In order to do this, whenever i sends a message on y_i to a node j, the message is a new name z. Then j sends back to i a new name s. These names z and s are used by j to communicate to i the number d of nodes that it dominates: j does this by outputting d times on s and then one time on z. Correspondingly, i executes the part R_i^k, namely it receives all the (d) inputs on s, then one input on z, and finally it becomes Q_i^{k-d-1}. Note that R^0 is never reached, so it does not need to be defined.

Of course, by symmetry, we need to make it possible for i to lose the competition. Thus in alternative to the output on y_i the node i must also try to receive input from all the other nodes, on their own channels. If one of these input actions succeeds, then i becomes dominated and it exits the competition. At this point i executes S_i^d: first it comunicates to the dominator a new name s (whose purpose is described above) and the number $(3 - k)$ of nodes that i currently dominates, and then (last two instructions of S_i^d) it waits to receive the name of the winner (from the winner), on its own channel, and then it sends such name on *out*.

[9] In a preliminary version of this paper there was an error: the number of nodes currently dominated by i was calculated by adding 1 each time i was doing an output on y_i. The error was pointed out by Peng Wu, and corrected by the author in the way described here. Note that the corresponding code in (7) is tuned for the case of 4 nodes. Later Peng Wu has coded the algorithm for the case of arbitrary rings, by using counters, and he has verified his program under the Mobility Workbench model checker.

The definition of the Q_i^k's is as follows:

$$
\begin{array}{rcll}
Q_i^k & = & \nu z\ (\ \ \overline{y}_i z\ .\ z(s)\ .\ R_i^k & \text{for } 1 \leq k \leq 3 \\
& & \quad + & \\
& & \quad y_{i+1}(z)\ .\ S_i^{3-k} & \\
& & \quad + & \\
& & \quad y_{i+2}(z)\ .\ S_i^{3-k} & \\
& & \quad + & \\
& & \quad y_{i+3}(z)\ .\ S_i^{3-k} & \\
& & \ ) & \\
& & & (7) \\
Q_i^0 & = & \overline{y}_{i+1} i\ .\ \overline{y}_{i+2} i\ .\ \overline{y}_{i+3} i\ .\ \overline{out}\ i & \\
& & & \\
R_i^k & = & \ s(w)\ .\ R_i^{k-1} & \\
& & \ + & \\
& & \ z(w)\ .\ Q_i^{k-1} & \\
& & & \\
S_i^d & = & \nu s\ (\ \overline{z}s\ .\ \underbrace{\overline{s}s\ .\ \ldots\ .\ \overline{s}s}_{d \text{ times}}\ .\ \overline{z}s\ .\ y_i(n)\ .\ \overline{out}\ n\ ) &
\end{array}
$$

Note that the P_i's have all the same code except for a renaming which corresponds to the structure of the hypergraph, hence P is symmetric wrt every automorphism on the hypergraph. □

In contrast to the first phase of the algorithm, the second phase, displayed in (7), is not within the syntax of π_a or π_s, but it is within the syntax of π_I, and an analogous process can also be written in CCS$_{vp}$. Hence for an hypergraph like the one in Figure 6 the symmetric electoral problem can be solved also in these two languages. More in general, we believe that the problem can be solved in CCS$_{vp}$ or in π_I for *any fully connected graph*.

Claim 2. Let H be a *fully connected* hypergraph (i.e. each pair of nodes are connected directly by an edge). Then there exists a symmetric electoral system P, in CCS$_{vp}$ or in π_I, such that $H(P) = H$.

Again, it is difficult to prove this claim in general (wrt any fully connected hypergraph) because the precise steps of the algorithm depend on the structure of the hypergraph.

In the next section, we investigate the limitations of CCS$_{vp}$ and π_I by showing a class of hypergraphs for which the symmetric electoral problem cannot be solved with these two languages.

6. Non existence of symmetric electoral systems in CCS$_{vp}$ and in π_I

The mechanisms of name-passing and scope extrusion, which makes it possible in the π-calculus (and in π_m) to extend dynamically the communication structure of the network, are essential for the solution to the electoral problem illustrated in Proposition 5.1. In fact, as we shall see, the part of the solution which completes the hypergraph cannot be expressed in CCS_{vp} or in π_I.

More in general, CCS_{vp} and π_I cannot express any solution to the symmetric electoral problem in a hypergraph like the one of Figure 5. Intuitively, the problem is that the symmetry cannot be broken as long as there is no direct connection (channel) between symmetric nodes (i.e. the nodes in the same orbit). And in CCS_{vp}, as well as in π_I, there is no way to create a new direct connection between two nodes, unless, in the case of π_I, they are already sharing a channel.

Theorem 6.1. Let $P = [P_0 \,|\, P_1 \,|\, \ldots \,|\, P_{k-1}]$ be a network in CCS_{vp} or in π_I, and let the associated hypergraph $H(P) = \langle N, X, t\rangle$ admit a well-balanced automorphism $\sigma \neq id$ such that P is symmetric wrt σ and, for each $n \in N$, there exist no h such that $\sigma^h(n) \neq n$ and $\{n, \sigma^h(n)\} \subseteq t(x)$ for some $x \in X$. Then P cannot be an electoral system.

An example of such network is represented in Figure 5: let σ be the automorphism defined as $\sigma(0) = 2$, $\sigma(2) = 0$, $\sigma(1) = 3$, and $\sigma(3) = 1$. Clearly σ is well balanced (it has two orbits of cardinality two). Note that the hypotheses of the above theorem are satisfied, because there is no edge between 0 and 2, and between 1 and 3.

Proof of Theorem 6.1 The proof is analogous to that of Theorem 4.2, and it is based on the construction of an infinite computation from P where no leader is elected.

Suppose that at a certain step of the computation P has evolved into P^h, no leader has been elected yet, and P^h is symmetric wrt a well balanced automorphism σ_h which satisfies the hypotheses of the theorem. Consider the first step from P^h. For the same reasons explained in the proof of Theorem 4.2, this step cannot be of the form $\overline{out}\, n$. We have two cases:

The transition is the result of the move of one process only) Assume, without loss of generality, that P_0^h is the process which makes the move. Let this move be

$$P_0^h \xrightarrow{\mu_0} P_0^{h+1}$$

Since $P^h_{\sigma_h^i(0)} \equiv_\alpha \sigma_h^i(P_0^h)$, By symmetry, we can construct derivations

$$\begin{array}{ccc}
P^h_{\sigma_h(0)} & \xrightarrow{\mu_1} & P^{h+1}_{\sigma_h(0)} \\
P^h_{\sigma_h^2(0)} & \xrightarrow{\mu_2} & P^{h+1}_{\sigma_h^2(0)} \\
 & \vdots & \\
P^h_{\sigma_h^{q-1}(0)} & \xrightarrow{\mu_{q-1}} & P^{h+1}_{\sigma_h^{q-1}(0)}
\end{array}$$

(where q is the cardinality of the orbits of σ) such that

$$P^h_{\sigma_{h+1}^i(0)} \equiv_\alpha \sigma_{h+1}^i(P_0^h)$$

where σ_{h+1} is constructed by adding to σ_h the associations on the new names introduced by the above transitions, if any, as in the proof of Theorem 4.2. Note that σ_{h+1} is well balanced and coincides with σ_h on all the processes P^h_j such that j not in the orbit of 0. For such processes, define $P^{h+1}_j = P^h_j$. By composing the transitions above, we get a non-empty sequence of transitions from P^h to $P^{h+1} = [P^{h+1}_0 \,|\, P^{h+1}_1 \,|\, \ldots |P^{h+1}_{k-1}]$ which does not contain $\overline{out}\, n$. Finally, observe that P^{h+1} is symmetric wrt σ_{h+1}.

The transition results from the communication of two processes) This is the crucial part of the proof, which distinguishes CCS_{vp} and π_I from π_m. Again, the interesting case is when the two communicating processes are in different nodes. Assume, without loss of generality, that P^h_0 and P^h_i are the partners in the communication. Note that, by the hypothesis that processes in the same orbit are not connected, 0 and i must be in different orbits. Let us consider the case of π_I first. We have that the communication can only derive from the rule Close, because output actions in π_I can only be of the form $\bar{x}(y)$. Assume without loss of generality that P^h_0 is the sender and that the communication action is $\bar{x}_0(y_0)$. we have:

$$P^h_0 | P^h_i \quad \stackrel{\tau}{\longrightarrow} \quad \nu y_0 \; (P^{h+1}_0 | P^{h+1}_i)$$

By symmetry, i.e. since $P^h_{\sigma^j_h(0)} \equiv_\alpha \sigma^j_h(P^h_0)$ and $P^h_{\sigma^j_h(i)} \equiv_\alpha \sigma^j_h(P^h_i)$, we also have

$$\begin{array}{rcl}
P^h_{\sigma_h(0)} | P^h_{\sigma_h(i)} & \stackrel{\tau}{\longrightarrow} & \nu y_{\sigma_{h(0)}} \; (P^{h+1}_{\sigma_h(0)} | P^{h+1}_{\sigma_h(i)}) \\
P^h_{\sigma^2_h(0)} | P^h_{\sigma^2_h(i)} & \stackrel{\tau}{\longrightarrow} & \nu y_{\sigma^2_h(0)} \; (P^{h+1}_{\sigma^2_h(0)} | P^{h+1}_{\sigma^2_h(i)}) \\
 & \vdots & \\
P^h_{\sigma^{q-1}_h(0)} | P^h_{\sigma^{q-1}_h(i)} & \stackrel{\tau}{\longrightarrow} & \nu y_{\sigma^{q-1}_h(0)} (P^{h+1}_{\sigma^{q-1}_h(0)} | P^{h+1}_{\sigma^{q-1}_h(i)})
\end{array}$$

where all y_l's are distinct, and distinct from the free names (by the bound-names convention on P^h), and

$$P^{h+1}_{\sigma^j_{h+1}(0)} \equiv_\alpha \sigma^j_{h+1}(P^{h+1}_0) \;\text{ and }\; P^{h+1}_{\sigma^j_{h+1}(i)} \equiv_\alpha \sigma^j_{h+1}(P^{h+1}_i)$$

where σ_{h+1} is defined as in (3). For any j which is neither in the orbit of 0, nor in the orbit of i, define $P^{h+1}_j = P^h_j$, and let

$$P^{h+1} = [\nu y_0 \; \nu y_1 \ldots \nu y_{q-1} (P^{h+1}_0 \,|\, P^{h+1}_1 \,|\, \ldots |P^{h+1}_{k-1})]$$

By using the *Cong* rule with scope expansion, we can combine the above transitions into a computation

$$P^h \quad \stackrel{\tilde{\tau}}{\Longrightarrow} \quad P^{k+1}$$

Finally, note that:

— P^{k+1} is symmetric wrt to σ_{h+1}, and

— $H(P^{k+1})$ differs from $H(P^k)$ only for the presence of the new edges y_j between the nodes $P^{h+1}_{\sigma^j_h(0)}$ and $P^{h+1}_{\sigma^j_h(i)}$ (which were already connected by the edge $x_{\sigma^j_h(0)}$). None of the existing edges have changed their type, i.e. two nodes that were not connected by any edge in $H(P^h)$ are still not connected by any edge in $H(P^{h+1})$.

In the case of CCS_{vp}, the proof is analogous. The crucial point here is that the objects of the communications, i.e. the y_j's, can only be values. Therefore they cannot be used as communication channels in later steps of the computation[10]. □

7. Uniform encoding

In this section we use the above results to show the non-encodability of the π_m into its asynchronous subsets, into CCS_{vp}, and into π_I, under certain requirements on the notion of encoding $[\![\cdot]\!]$.

There is no agreement on what should be a good notion of encoding, and perhaps indeed there should not be a unique notion, but several, depending on the purpose. However, it seems reasonable to require at least the two following properties:

1 compositionality,
2 preservation of some intended semantics.

For a distributed system, however, it seems reasonable to strengthen the notion of compositionality on the parallel operator by requiring that it is translated homomorphically, namely

$$[\![P \mid Q]\!] \quad = \quad [\![P]\!] \mid [\![Q]\!] \tag{8}$$

In this way we can ensure that the translation maintains the degree of distribution of the system, without introducing additional processes with coordination functions.

Likewise, it seems reasonable to require that the encoding "behaves well" with respect to channel renamings, i.e. for any permutation of names σ in the domain of the source language there exists a permutation of names θ in the domain of the target language such that $\forall i \in N \; \sigma(i) = \theta(i)$ and

$$[\![\sigma(P)]\!] = \theta([\![P]\!]) \tag{9}$$

We will say that an encoding that satisfies (8) and (9) is *uniform*[11].

Concerning the notion of semantics, we call "reasonable" a semantics which distinguishes two processes P and Q whenever there exists a maximal (finite or infinite) computation of P in which the intended observables (some visible actions) are different from the observables in any (maximal) computation of Q. In the following, our intended observables are the actions performed on channel *out*. Note that the above condition cannot be satisfied by a semantics which is insensitive to infinite τ loops, such as weak bisimulation or coupled bisimulation.

[10] In the case of CCS_{vp} the notion of hypergraph associated to a network should be slightly different, i.e. we should distinguish between edges that represent channels and edges that represent only values and therefore cannot be used as communication channels. Also, we should use scope extrusion as a bisimilarity law.

[11] The definition of uniformity has emerged from discussion with Iain Phillips and Maria Grazia Vigliotti. They pointed out the necessity of the condition $\forall i \in N \; \sigma(i) = \theta(i)$ in order for the encoding to preserve symmetry, and hence for Corollary 7.1 to hold. They also pointed out the necessity of the condition about connectivity in Corollary 7.2.

Corollary 7.1. There exist no uniform encoding of π_m into π_s preserving a reasonable semantics.

Proof Uniformity preserves symmetry, and a reasonable semantics distinguishes an electoral system from a non-electoral one. Hence apply Theorem 4.2 and Proposition 5.1. Note that the hypergraph in Proposition 5.1 has one-orbit automorphisms. □

For the next result we need a further condition: We say that an encoding $[\![\cdot]\!]$ *does not increase the level of connectivity of a network* if for all processes P and Q, if $fn(P) \cap fn(Q) = \emptyset$, then $fn([\![P]\!]) \cap fn([\![Q]\!]) = \emptyset$.

Corollary 7.2. There exist no uniform encoding of π_m into CCS_{vp} or into π_I which does not increase the level of connectivity and which preserves a reasonable semantics.

Proof Analogous, by Theorem 6.1 and Proposition 5.1. Note that the hypergraph in Proposition 5.1 has well-balanced automorphisms satisfying the conditions of Theorem 6.1. □

Note that if we relax condition (8), imposing just generic compositionality instead, i.e.

$$[\![P \mid Q]\!] \quad = \quad C[\, [\![P]\!], [\![Q]\!] \,] \tag{10}$$

with $C[\cdot,\cdot]$ generic context, then these non-encodability results do not hold anymore. In fact, we could give an encoding of the form

$$[\![P \mid Q]\!] \quad = \quad \nu y_1 \nu y_2 \ldots \nu y_n ([\![P]\!] \mid M \mid [\![Q]\!])$$

where M is a "monitor" process which coordinates the activities of P and Q, interacting with them via the fresh channels $y_1, y_2, \ldots, y_n$. The translation of a network $P_1 \mid P_2 \mid \ldots \mid P_n$ would then be a tree with the P_i's as leaves, and the monitors as the other nodes. The disadvantage of this solution is that it is not a distributed implementation; on the contrary, it is a very centralized one.

8. Discussion and future work

The non-existence results of this work hold even if we disregard *unfair* computations. The proof of Theorem 4.2 in fact can be slightly modified so that for the construction of C_{h+1} from C_h we consider each time a different process in the network. In this way, the limit of the sequence is a fair computation.

Our Theorems 4.2 and 6.1 correspond to Theorems 3.2.1 and 4.2.1 in (Bougé, 1988), for CSP_{in} and CSP respectively. The main difference with those results is that here we are dealing with much richer languages. In particular, both the π-calculi and CCS_{vp} admit the parallel operator inside every process, and not just at the top-level as it is the case for CSP_{in} and CSP (at least, for the versions considered in (Bougé, 1988): all processes in a network are strictly sequential). This leads to an essential difference. Namely, the proof of Bougé shows that the network can get stuck in the attempt to elect a leader: since an output action in CSP_{in} can be only sequential, the prefix of a computation which leads to the first output action, repeated by all processes, brings to a global deadlock. Our proof, on the contrary, shows that the system can run forever without reaching an agreement: whenever a first output action occurs, all the other processes can execute their corresponding output action as well, and so on, thus generating an infinite computation which never breaks the symmetry. Another difference is that in the π-calculus the network can evolve dynamically. This is the reason why Theorem 4.2.1 in (Bougé, 1988) does not hold for π_m (as shown by our Theorem 1). This feature complicates the proof of Theorems 4.2 since we have to take into account the evolution of the automorphism.

The use of the parallel operator as a free constructor usually enhances significantly the expressive power of a language. It is for instance essential for implementing choice (at least in a restricted form). In fact, Bougé (1988) has shown that it is not possible to encode CSP_{in} into CSP_{no} (the sublanguage of CSP with neither input nor output guards in the choice), while Nestmann and Pierce (2000) have shown that the π-calculus with input-guarded choice (π_i) can be embedded into the π-calculus without choice (π_a). The crucial point is that the parallel operator allows to represent the main characteristic of the choice, namely the simultaneous availability of its guards.

Sangiorgi and Walker (2001) cite our separation result between π_m and π_s wrt a slightly different semantic condition, which is the following:

> *For any P and any $N \subseteq fn(P)$, if every maximal computation of P contains exactly one action whose subject is in N, then every maximal computation of $[\![P]\!]$ contains exactly one action whose subject is in N.*

Strictly speaking, with this condition the separation result does not follow from Theorem 4.2. The problem is that our notion of electoral systems requires all processes to execute the action $\overline{out}\,n$ after the leader is elected. This requirement corresponds to imposing that all processes will eventually know whom the winner is, which is a standard condition in the notion of electoral system found in literature. However, we could have considered a simpler (more permissive) notion of electoral system, obtained by requiring, in Definition 3.1, that C' contains only one action of the form $\overline{out}\ n$ (performed, presumably, by the winner). All results presented in this paper would remain valid under this new notion of electoral system, and in this way the separation result could be proved also wrt the above variant of the semantic condition.

One way to interpret the results presented in this paper is that mixed choice is a really difficult mechanism to implement. The only possibility to achieve a fully distributed and symmetry-preserving implementation probably is to use randomized techniques. Francez and Rodeh (1980) have proposed a randomized implementation of CSP, however, their solution it is not fully satisfactory because it is not robust wrt adverse scheduling strategies. We are currently investigating a probabilistic extension of π_i for this purpose, called π_{pa} (Herescu and Palamidessi, 2000). We have shown that in π_{pa} it is possible to express the solution to some of the leader election problems that would not be solvable in π_i (or π_s), so this seems encouraging.

Acknowledgements

I would like to thank the following people for stimulating and insightful discussions: Ilaria Castellani, Pat Lincoln, Dale Miller, Uwe Nestmann, Prakash Panangaden, Benjamin

Pierce, Rosario Pugliese, Julian Rathke, Davide Sangiorgi, Scott Smolka and Eugene Stark. Special thanks to Wu Peng for pointing out errors (now corrected) in Definitions (6) and (7). I gratefully acknowledge also Iain Phillips and Maria Grazia Vigliotti for pointing out that a previous notion of uniform embedding was problematic, and for helping to develop the present formulation. They also pointed out the necessity of the condition about connectivity in Corollary 7.2. Furthermore I would like to express my gratitude to the anonymous reviewers, whose careful and thorough reports have helped enormously to improve the paper. Last but not least, I would like to thank the editors of this special issue of *MSCS*, Luca Aceto, Giuseppe Longo and Björn Victor, for the excellent job they did as editors.

References

Amadio, R. M., Castellani, I., and Sangiorgi, D. (1998). On bisimulations for the asynchronous π-calculus. *Theoretical Computer Science*, 195(2):291–324. An extended abstract appeared in *Proceedings of CONCUR '96*, LNCS 1119: 147–162.

Boreale, M. (1998). On the expressiveness of internal mobility in name-passing calculi. *Theoretical Computer Science*, 195(2):205–226. A preliminary version of this paper appeared in the *Proceedings of CONCUR'96*, volume 1119 of *LNCS*.

Boreale, M. and Sangiorgi, D. (1998). Some congruence properties for π-calculus bisimilarities. *Theoretical Computer Science*, 198(1,2):159–176.

Boudol, G. (1992). Asynchrony and the π-calculus (note). Rapport de Recherche 1702, INRIA, Sophia-Antipolis.

Bougé, L. (1988). On the existence of symmetric algorithms to find leaders in networks of communicating sequential processes. *Acta Informatica*, 25(2):179–201.

Francez, N. and Rodeh, M. (1980). A distributed abstract data type implemented by a probabilistic communication scheme. In *Proc. 21st Ann. IEEE Symp. on Foundations of Computer Science*, pages 373–379.

He, J., Josephs, M. B., and Hoare, C. A. R. (1990). A theory of synchrony and asynchrony. In Broy, M. and Jones, C. B., editors, *Programming Concepts and Methods, Proc. of the IFIP WG 2.2/2.3, Working Conf. on Programming Concepts and Methods*, pages 459–478. North-Holland.

Herescu, O. M. and Palamidessi, C. (2000). Probabilistic asynchronous π-calculus. In Tiuryn, J., editor, *Proceedings of FOSSACS 2000 (Part of ETAPS 2000)*, Lecture Notes in Computer Science, pages 146–160. Springer-Verlag. Postscript available at `http://cse.psu.edu/~catuscia/papers/Prob_asy_pi/fossacs.ps`.

Hoare, C. (1978). Communicating sequential processes. *Communications of the ACM*, 21(8):666–677.

Hoare, C. A. R. (1985). *Communicating Sequential Processes.* Prentice-Hall.

Honda, K. and Tokoro, M. (1991). An object calculus for asynchronous communication. In America, P., editor, *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, volume 512 of *Lecture Notes in Computer Science*, pages 133–147. Springer-Verlag.

Milner, R. (1989). *Communication and Concurrency.* International Series in Computer Science. Prentice Hall.

Milner, R. (1993). The polyadic pi-calculus: a tutorial. In Bauer, F. L., Brauer, W., and Schwichtenberg, H., editors, *Logic and Algebra of Specification*, pages 203–246. Springer-Verlag.

Milner, R., Parrow, J., and Walker, D. (1992). A calculus of mobile processes, I and II. *Information and Computation*, 100(1):1–40 & 41–77.

Milner, R., Parrow, J., and Walker, D. (1993). Modal logics for mobile processes. *Theoretical Computer Science*, 114(1):149–171.

Nestmann, U. (2000). What is a 'good' encoding of guarded choice? *Journal of Information and Computation*, 156:287–319. An extended abstract appeared in the *Proceedings of EXPRESS'97*, volume 7 of *ENTCS*.

Nestmann, U. and Pierce, B. C. (2000). Decoding choice encodings. *Journal of Information and Computation*, 163:1–59. An extended abstract appeared in the *Proceedings of CONCUR'96*, volume 1119 of *LNCS*.

Palamidessi, C. (1997). Comparing the expressive power of the synchronous and the asynchronous π-calculus. In *Conference Record of POPL '97: The 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 256–265, Paris, France.

Parrow, J. and Sjödin, P. (1992). Multiway synchronization verified with coupled simulation. In Cleaveland, W. R., editor, *CONCUR '92: Third International Conference on Concurrency Theory*, volume 630 of *Lecture Notes in Computer Science*, pages 518–533, Stony Brook, New York. Springer-Verlag.

Pierce, B. C. and Turner, D. N. (1998). Pict: A programming language based on the pi-calculus. In Plotkin, G., Stirling, C., and Tofte, M., editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*. The MIT Press.

Pugliese, R. (1997). Personal communication.

Sangiorgi, D. (1996). π-calculus, internal mobility and agent-passing calculi. *Theoretical Computer Science*, 167(1,2):235–274.

Sangiorgi, D. and Walker, D. (2001). *The π-calculus: a Theory of Mobile Processes.* Cambridge University Press.