Tutorial on separation results in process calculi via leader election problems

Maria Grazia Vigliotti, Iain Phillips, Catuscia Palamidessi

Tutorial on separation results in process calculi via leader election problems

Maria Grazia Vigliotti [a,b], Iain Phillips [c] and Catuscia Palamidessi [b,d]

[a] *INRIA Sophia-Antipolis, France*

[b] *INRIA Futurs, France*

[c] *Department of Computing, Imperial College London, England*

[d] *LIX Polytechnique, France*

Abstract

We compare the expressive power of process calculi by studying the problem of electing a leader in a symmetric network of processes. We consider the π-calculus with mixed choice, separate choice and internal mobility, value-passing CCS and Mobile Ambients, together with other ambient calculi (Safe Ambients, the Push and Pull Ambient Calculus and Boxed Ambients). We provide a unified approach for all these calculi using reduction semantics.

Key words: leader election, π-calculus, Mobile Ambients, CCS

1 Introduction

In this tutorial we consider expressiveness results regarding different process calculi. In the last twenty years, a great variety of concurrent calculi have been developed, and most of them are Turing complete, i.e. they can compute the same class of functions as Turing machines. However, function computability is only one possible way to evaluate the power of a concurrent language; other aspects, related to the concurrent nature of the model, should also be taken into account. Our focus is on the synchronisation capabilities of a calculus, and more precisely on the mechanisms that allow remote processes to achieve an agreement. Agreement is, in general, an important problem in distributed computing. A lot of research has been devoted to either finding algorithms to achieve agreement among remote processes, or proving the impossibility of such algorithms existing. The problem has important implications of a practical nature in the field of Distributed Systems, where the design of the operating

system has to ensure the correct interaction between remote processes when a central coordinator is not feasible.

Comparing the expressiveness of calculi can be achieved by either exhibiting an encoding or by showing that such an encoding cannot exist. For the latter, one way of proceeding is to show that there is a problem that can be solved in some calculi, but not the others. In the field of distributed algorithms [17,36], various models of computation have been compared via the *symmetric leader election problem*, which consists in requiring the members of a symmetric network to elect one of their number as their leader. This problem expresses the ability of group of processes to reach an agreement (on the leader) in a completely decentralised way. Translation of the problem into process algebra setting has proved a rather successful way of comparing various process calculi [3,11,27,26,28,30,37].

The notion of encoding in a formal setting is subject to specific and natural conditions, which guarantee the meaningfulness of the encoding. For instance, when dealing with the synchronisation problem, the subject of this tutorial, we should require that the encoding should not itself solve the problem of synchronisation; it would be like mapping Turing machines into finite automata by using a translation which adds an oracle.

For symmetric leader election problems, the computational difficulty of finding a solution lies in breaking the initial symmetry to achieve a situation which is inherently asymmetric (one is the leader and the others are not). In the case of process calculi, some of the symmetry-breaking arguments are rather sophisticated and use additional discriminations that are related to the topology of the network.

In this tutorial we shall collect, present, systematise and interpret a collection of results regarding expressiveness in process calculi obtained by means of the symmetric leader election problem. We shall provide a uniform presentation by the use of reduction semantics, and we shall highlight the similarities and differences between the various approaches to leader election problems. In particular, we shall focus on the following calculi:

- Communicating Concurrent Systems (CCS).
- The π-calculus (π_{m}) and its dialects: the π-calculus with separate choice π_{s} and the π-calculus with internal mobility π_{I}.
- Ambient calculi: Mobile Ambients (MA) and its dialects Safe Ambients (SA), the Push and Pull Ambient Calculus (PAC) and Boxed Ambients (BA).

CCS [18,20] is a simple calculus, that aims to represent concurrency with synchronous communication. Based on the concept of channels, it contains two primitives for sending and receiving which can synchronise by handshaking on

the same channel. In this paper we shall consider *value-passing* CCS, where input and output primitives carry value parameters. However, for the sake of simplicity, we shall call it CCS throughout the paper.

The π-calculus [21] enhances the CCS model by allowing processes to communicate channel names, which can also be used as channels for communication, allowing the dynamic creation of new links between processes (link mobility). In this paper we consider the π-calculus as presented in [20]. We call this version the *mixed-choice* π-calculus, which we denote by π_{m}; here the word "mixed" signifies that a choice can contain both input and output guards. The results presented this tutorial would still hold also for the π-calculus with matching and mismatching.

The asynchronous π-calculus [15,2] has become particularly popular as a model for asynchronous communication. In this fragment there is no explicit choice, and outputs have no continuation. However output prefixing and separate choice can be encoded in the asynchronous π-calculus [2,25]; separate choice is guarded choice with the restriction that input and output guards cannot be mixed in the same choice. In this tutorial we look at the *separate-choice* π-calculus, which we denote by π_{s}, rather than the asynchronous π-calculus; however the results valid for π_{s} also hold for the asynchronous π-calculus.

We shall also consider the π-calculus with *internal mobility* (which we denote by π_{I}), proposed by Sangiorgi [34]. This is a subset of π_{m} where only private names can be sent, giving a symmetry between inputs and outputs. The main advantage of this calculus is the simpler theory of bisimulation.

Finally, we shall deal with Mobile Ambients and other ambient calculi. MA [9] has been proposed to model features of computation over the Internet. This calculus is based on the simple unifying concept of *ambient*. Computation is no longer defined as exchanging values, but it is the result of ambients moving into and out of other ambients bringing along active processes and possibly other ambients.

In the last few years many dialects have been spawned: Levi and Sangiorgi's SA [16], which introduces a way of controlling the boundary of the ambients; Phillips and Vigliotti's PAC [29], which aims to model a more traditional client-server architecture and finally BA [4,5], which focuses on security properties and therefore does not permit ambients to be dissolved. Several relations among the above calculi are obvious or have been proved in the literature, addressing at least partially the issue of expressiveness. However, questions about their expressive power can still be asked:

- π_{s} is a subcalculus of π_{m}. *Does there exist an encoding from* π_{m} *into* π_{s}*, or is* π_{m} *strictly more expressive?*
- CCS with value passing and π_{I} can be viewed as subcalculi of π_{m}. Thus π_{m}

is as least as expressive as CCS. *Does an encoding exist from π_{m} into CCS or π_{l}?*
- The asynchronous π-calculus can be encoded into MA, PAC and BA. *Can MA or PAC or BA be encoded into the asynchronous π-calculus or CCS ?*

In the tutorial we shall show that the answers to the previous questions are negative, i.e. those encodings do not exist under certain conditions (Section 2.3). The proofs are based on the possibility/impossibility of solving the symmetric leader election problem.

In encodings of languages that (do not) admit a solution for leader election problems, one important requirement is that the encoding preserves the original distribution among processes. This requirement aims at avoiding that the encoding may introduce a central coordinator [26,28]. Therefore this condition makes the notion of encoding suitable for comparing the expressiveness of languages for distributed systems, where processes are expected to coordinate without the help of a centralised server.

The negative results mentioned above have been achieved in recent years as follows:

- Palamidessi [26] established that π_{m} is not encodable in π_{s};
- Phillips and Vigliotti [28,37] proved that small fragments of MA, PAC and BA are not encodable in π_{s}.

All these separation results are proved by considering the leader election problem in a fully connected (and symmetric) network. For instance, Palamidessi showed that the problem can be solved in the case of π_{m}, but not in the case of π_{s}. If there were an encoding from π_{m} to π_{s}, then the solution for π_{m} could be translated into one for π_{s}, provided that the encoding satisfied certain conditions (such as distribution preservation—see Section 2.3). Therefore no such encoding can exist.

Finer-grained separation results are obtained by considering the leader election problem in a network whose underlying graph is a ring. Those latter negative results have been achieved in recent years as follows:

- Palamidessi [26] proved that CCS and π_{l} do not admit a solution to the leader election problem for certain symmetric rings, while π_{m} does. She deduced that there is no encoding from π_{m} into CCS.
- Phillips and Vigliotti [30] proved that subcalculi of MA, PAC and BA admit solutions to the leader election problem for symmetric rings. They concluded that those calculi cannot be encoded into CCS.

The tutorial is organised as follows. We start by discussing leader election in distributed networks, and how to formalise the problem in process calculi

(Section 2). In Section 3 we define the various calculi we shall consider. We next deal with leader election problems in general symmetric networks with no restriction on topology (Section 4). We present solutions for various calculi, show that other calculi do not admit solutions, and derive separation results (Section 4.3). Then we deal with leader election problems in rings (Section 5). We present positive and negative results for various calculi, and again derive separation results (Section 5.4). We then discuss the meaning of the results presented in this tutorial and compare other notions of encoding which yield separation results among the asynchronous π-calculus, CCS and π_{I}. We end the tutorial with a history of related work and conclusions.

2 Leader Election, Electoral Systems and Encodings

After first discussing leader election informally, we show how it can be formalised in the setting of process calculi and reduction semantics. We then discuss criteria for encodings between calculi.

2.1 Leader Election Problems in Distributed Systems

In this section we introduce leader election problems as described in the field of distributed systems. We use the word 'problems' in the plural, because there are different settings that lead to diverse solutions (when solutions do exist). A network is informally a set of machines that run independently and that compute through communication. Abstractly we can think of them as processes. Processes have the same state, if they can perform intuitively the same actions. The essence of a symmetric leader election problem is to find an algorithm where, starting from a configuration (network) of processes in the *same state*, any possible computation reaches a configuration where *one* process is in the state of *leader* and the other processes are in the state *lost* (i.e. they have lost the election). In some cases a solution may be impossible, and in other cases there may be more than one algorithm, and then complexity measures can be used in order to compare the different solutions. In this tutorial, we shall not consider such issues.

The criteria common to all leader election problems are the following:

Symmetry Each process in the network has to have the *same duties* (furthermore, processes cannot be allowed to have distinct identifiers which could be used to decide the leader by e.g. allowing the process with the largest identifier to become leader). In the symmetry lies the computational difficulty of the problem. In fact, in an asymmetric configuration of processes,

one process can declare itself the winner. This is not possible in symmetric configurations, since if one process can declare itself the winner, every other process in the configuration can do the same. Thus, in symmetric networks, for the winner to be elected, the *initial symmetry* has to be somehow broken.

Distribution The computation has to be *decentralised*, in the sense that the computation has to start from any subset of processes in the network. Generally, leader election problems are run after a reconfiguration or crash of a system, in order to establish which process can start the initialisation. In this context, the configuration of processes has to be able to elect a leader without any help from outside.

Uniqueness of the leader The processes in a network reach a *final configuration* from *any* computation. In the final configuration there is *one process only* that is elected the *winner* and the other processes in the configuration have lost.

Leader election problems may vary according to the following parameters:

Topology of the network The network could be a *fully connected graph* or a *ring* or *tree* or any other graph or hyper-graph [1,36,17]. The topology of the network influences the construction of the algorithm, since it changes the information regarding the totality of the processes involved.

In this tutorial we look at general networks, where there is no restriction on topology, in Section 4, and at rings in Section 5. In the general case, our algorithms will assume that the network is fully connected, though of course this is not assumed when we state impossibility results.

Knowledge of size of the network The number of processes can be known or unknown to the processes before starting the election [36]. This parameter also influences the construction of an algorithm. In most cases we shall implement algorithms where the size of the network is known, but there is an interesting exception in the case of PAC, where we present an algorithm for a ring where the processes are defined uniformly regardless of the size of the ring.

Declaration of the leader The leader could be announced by one process only, which could be the leader itself or any other process. Alternatively every process in the configuration has to be aware of the winner. The latter requirement is considered standard, although the weaker one (the former one) is also acceptable, since the winner could inform the other processes of the outcome of the election.

We shall adopt the weaker assumption in this tutorial for simplicity. Note that in the original paper [26] Palamidessi uses the stronger requirement for her results.

We have described leader election problems as presented in the field of distributed algorithms. In this field, it is common to reason on what is known as *pseudo-code*. This means that proofs are given by using some form of 'general

enough language', that is, a mixture of an ad hoc Pascal-like language and natural language without any formalised semantics. Nestmann et al. [24] show that this approach very often hides underpinning problems and assumptions. The formal and rigorous semantics of process algebra, as presented in this tutorial, is therefore an advantage in the description of leader election problems. Formal semantics is necessary when proving that either a given algorithm is the correct solution to a leader election problem, or that no algorithm exists.

2.2 Electoral Systems

In this section we formalise the leader election problem in process calculi using reduction semantics (unlabelled transitions). Milner and Sangiorgi [22] motivated the study of reduction semantics on the grounds that it is a uniform way of describing semantics for calculi that are syntactically different from each other. Reduction semantics has been widely used due to its simplicity and ability to represent uniformly simple process calculi such as CCS [20], first- and second-order name-passing calculi such as the π-calculus and the higher-order π-calculus [22,33], and more complex calculi such as the Seal Calculus [10] and the Ambient Calculus [9]. Reduction semantics will provide a uniform framework for all calculi we shall consider.

In reduction semantics a process calculus L is identified with:

- a set of processes;
- a reduction relation; and
- an observational predicate.

First of all, we assume the existence of a set of names $\mathcal{N}$, ranged over by; the variables $m, n, x, y \ldots$ range over it. Names are meant to be atomic, and they are a useful abstraction to represent objects that in real life we do not want to view as separated, such as identifiers, sequences of bits, etc.

Some operators of a language are *binding*, in the sense that names that fall within their *scope* are called *bound*, and processes that differ in bound variables only are considered identical. Names that are not bound in a process are called *free*. These concepts will be explicitly defined for each concrete syntax considered later in this tutorial.

We assume that a language L contains at least the parallel composition operator $\mid$ and the restriction operator $\nu n\ P$. We assume that in each calculus operator $\mid$ has the same kind of semantics: it nondeterministically lets either the left- or the right-hand process execute on its own, or else it lets the two sides synchronise. Restriction $\nu n\ P$ binds n; it makes the name n *private* or bound in P. We write $\nu\vec{n}$ instead of $\nu n_1 \ldots \nu n_k$ for some list of names $n_1, \ldots, n_k$

which is not relevant in the context. In general we identify processes that differ only on their bound names, and we keep separate the set of free names and the set of bound names. Process can be identified up to the *structural congruence relation* $\equiv$. Structural congruence allows rearrangement without computation taking place. In all the calculi we shall consider, the following laws will hold (plus other laws, depending on the particular calculus):

$$P \mid Q \equiv Q \mid P \qquad \nu n\,(P \mid Q) \equiv P \mid \nu n\,Q \quad \text{if} \quad n \notin fn(P)$$
$$(P \mid Q) \mid R \equiv P \mid (Q \mid R) \quad \nu m\,\nu n\,P \equiv \nu n\,\nu m\,P$$

The computational steps for a language can be captured by a simple relation over the set of processes called the *reduction relation*, written $\rightarrow$. To model visible behaviour of programs, an *observation relation* is defined between processes and names: $P \downarrow n$ means intuitively that the process P has the observable name n. We shall see in each concrete calculus how these notions are defined.

Networks are informally compositions of processes or processes composed with the operator $|$; the size of a network is the number of processes that can be "regarded as separate units". This means that a composition of processes can be seen as one process only in counting the size of the network. A *symmetric* network is a network where components differ only on their names. Components of a network are connected if they share names, using which they can engage in communication. *Rings* are networks where each process is connected just to its left-hand and right-hand neighbours. A network elects a leader by exhibiting a special name, and an *electoral system* is a network where every possible maximal computation elects a leader.

We now make these notions precise. We assume that $\mathcal{N}$ includes a set of *observables* $\mathsf{Obs} = \{\omega_i : i \in \mathbb{N}\}$, such that for all i, j we have $\omega_i \neq \omega_j$ if $i \neq j$. The observables will be used by networks to communicate with the outside world.

Definition 2.1 *Let P be a process. A* computation $\mathcal{C}$ *of P is a (finite or infinite) sequence $P = P_0 \rightarrow P_1 \rightarrow \cdots$. It is* maximal *if it cannot be extended, i.e. either $\mathcal{C}$ is infinite, or else it is of the form $P_0 \rightarrow \cdots \rightarrow P_h$ where $P_h \not\rightarrow$.*

Definition 2.2 *Let $\mathcal{C}$ be a computation $P_0 \rightarrow \cdots \rightarrow P_h \rightarrow \cdots$. We define the* observables *of $\mathcal{C}$ to be $\mathsf{Obs}(\mathcal{C}) = \{\omega \in \mathsf{Obs} : \ \exists h \ \ P_h \downarrow \omega\}$.*

Networks are collections of processes running in parallel, as the following definition states.

Definition 2.3 *A* network Net *of size k is a pair $(A, \langle P_0, \ldots, P_{k-1}\rangle)$, where*

A is a finite set of names and $P_0, \ldots, P_{k-1}$ are processes. The process interpretation $\mathsf{Net}^\natural$ *of* Net *is the process $\nu A\,(P_0 \mid \cdots \mid P_{k-1})$. We shall always work up to structural congruence, so that the order in which the restrictions in A are applied is immaterial.*

Networks are to be seen as presentations of processes, showing how the global process is distributed to the k nodes of the network. We shall sometimes write $[P_0 \mid \cdots \mid P_{k-1}]$ instead of $\nu A\,(P_0 \mid \cdots \mid P_{k-1})$, when the globally restricted names do not need to be made explicit.

We shall tend to write networks in their process interpretation (i.e. as restricted parallel compositions), while still making it clear which process belongs to each node of the network.

Networks inherit a notion of computation from processes through the process interpretation: $\mathsf{Net} \to \mathsf{Net}'$ if $\mathsf{Net}^\natural \to \mathsf{Net}'^\natural$. Overloading notation, we shall let $\mathcal{C}$ range over network computations. Also, we define the observables of a network computation $\mathcal{C}$ to be the observables of the corresponding process computation: $\mathsf{Obs}(\mathcal{C}) = \mathsf{Obs}(\mathcal{C}^\natural)$.

The definitions that follow lead up to the formulation of symmetry in a network (Definition 2.7), capturing the notion that each process is the same apart from the renaming of free names. First we introduce the notion of permutation on names, which is a a bijective function that keeps free and bound names separated.

Definition 2.4 *A* permutation *is a bijection $\sigma : \mathcal{N} \to \mathcal{N}$ such that σ preserves the distinction between observable and non-observable names, i.e. $n \in \mathsf{Obs}$ iff $\sigma(n) \in \mathsf{Obs}$. Any permutation σ gives rise in a standard way to a mapping on processes, where $\sigma(P)$ is the same as P, except that any free name n of P is changed to $\sigma(n)$ in $\sigma(P)$, with bound names being adjusted as necessary to avoid clashes.*

A permutation σ induces a bijection $\hat{\sigma} : \mathbb{N} \to \mathbb{N}$ defined as follows: $\hat{\sigma}(i) = j$ where $\sigma(\omega_i) = \omega_j$. Thus for all $i \in \mathbb{N}$, $\sigma(\omega_i) = \omega_{\hat{\sigma}(i)}$. We use $\hat{\sigma}$ to permute the indices of processes in a network.

An *automorphism* over a network is simply a permutation over the set of processes in the network such that the induced bijection over the indexes of the processes is finite.

Definition 2.5 *Let $\mathsf{Net} = \nu\vec{n}\,(P_0 \mid \cdots \mid P_{k-1})$ be a network of size k. An* automorphism *on* Net *is a permutation σ such that (1) $\hat{\sigma}$ restricted to $\{0, \ldots, k-1\}$ is a bijection, and (2) σ preserves the distinction between free and bound names, i.e. $n \in \vec{n}$ iff $\sigma(n) \in \vec{n}$.*

In general, if a permutation is repeatedly composed with itself, then we reach the identity permutation after a finite number of iterations. As a result, also the induced bijection reaches a fixed point after a finite number of self applications. The *orbit* is the record of all the results of the self applications of the induced bijection before reaching the fixed point.

Definition 2.6 *Let σ be an automorphism on a network of size k. For any $i \in \{0, \ldots, k-1\}$ the* orbit $\mathcal{O}_{\hat{\sigma}}(i)$ *generated by $\hat{\sigma}$ is defined as follows:*

$$\mathcal{O}_{\hat{\sigma}}(i) = \{i, \hat{\sigma}(i), \hat{\sigma}^2(i), \ldots, \hat{\sigma}^{h-1}(i)\}$$

where $\hat{\sigma}^j$ represents the composition of $\hat{\sigma}$ with itself j times, and h is the least such that $\hat{\sigma}^h(i) = i$.

Definition 2.7 *Let* $\mathsf{Net} = \nu\vec{n}\,(P_0 \mid \cdots \mid P_{k-1})$ *be a network of size k and let σ be an automorphism on it. We say that* Net *is* symmetric with respect to σ *iff for each $i = 0, \ldots, k-1$ we have $P_{\hat{\sigma}(i)} = \sigma(P_i)$.*

We say that Net *is* symmetric *if it is symmetric with respect to some automorphism with a single orbit (which must have size k).*

A simpler proposal for defining symmetry of a network would be to require that, for any two processes P_i and P_j, there is a permutation σ_{ij} such that $\sigma_{ij}(P_i) = P_j$. However, while this is necessary (and implied by our definition), it is not sufficient, since the various σ_{ij} can be defined independently of each other, and we could end up classing asymmetric networks as symmetric. As a simple example, consider the network of three CCS processes given by

$$P_0 = a.b.c \qquad P_1 = b.c.a \qquad P_2 = a.c.b$$

This should not be regarded as symmetric, since P_0 and P_2 share a common initial action, leaving P_1 as the odd one out. It can be checked that indeed, according to our definition, the network is not symmetric. But if we just consider pairs of processes we can certainly define permutations σ_{ij} such that $\sigma_{ij}(P_i) = P_j$.

Intuitively an electoral system is a network which reports a unique winner, no matter how the computation proceeds.

Definition 2.8 *A network* Net *of size k is an* electoral system *if for every maximal computation $\mathcal{C}$ of* Net *there exists an $i < k$ such that* $\mathsf{Obs}(\mathcal{C}) = \{\omega_i\}$.

The concept of encoding is inherently associated to expressiveness. If there exists an encoding $[\![-]\!]$ from a source language S to a target language T, one could see the language T as 'mirroring' S. Thus, the model underpinning T is at least as expressive as the one underpinning S. At the highest level of abstraction, an encoding $[\![-]\!]$ is a function from a source language to a target language. However, not just any function $[\![-]\!]$ from source language to target language should be accepted as an encoding; some 'relevant' behaviour of the first language must be 'preserved'.

We appeal here to the intuitive meaning of the words 'relevant' and 'to preserve', but it remains to formalise the meaning of these words, by exhibiting the semantic properties that $[\![-]\!]$ must satisfy. Before giving our particular answer in the setting of electoral systems, we mention a number of properties to be found in the literature. There are two sorts of property which are most relevant to our setting: the *operational* and the *syntactic*.

We first state some common forms of operational correspondence (including observational correspondence via barbs). Assuming that $P \in S$ and $[\![P]\!] \in T$, that $\rightarrow^*$ means the reflexive and transitive closure of the reduction relation, and that $\simeq$ is a suitable equivalence relation, we have:

- Preservation of execution steps (completeness): if $P \rightarrow P'$ then $[\![P]\!] \rightarrow^* \simeq [\![P']\!]$ [25,19,8,12];
- Reflection of execution steps (soundness): if $[\![P]\!] \rightarrow^* Q$ then there is P' such that $P \rightarrow^* P'$ and $Q \rightarrow^* \simeq [\![P']\!]$ [25,19,12];
- Barb preservation (completeness): if $P \downarrow n$ then for some Q we have $[\![P]\!] \rightarrow^* Q$ and $Q \downarrow n$ [29,38];
- Barb reflection (soundness): if $[\![P]\!] \downarrow n$ then $P \downarrow n$ [29,38].

We now turn to syntactic requirements on an encoding. In their simplest form, these involve a particular operator being preserved from source to target language. The operator in question must of course be common to both languages. We give a few examples. Assuming that $|$ and ν are two operators common to S and T, then the first two statements below express that $[\![-]\!]$ preserves restriction (bound names) and distribution (parallel composition).

- Distribution preservation: $[\![P \mid Q]\!] = [\![P]\!] \mid [\![Q]\!]$ [26,30,28,12];
- Restriction preservation: $[\![\nu n\ P]\!] = \nu n\ [\![P]\!]$ [11];
- Substitution preservation: for all substitutions σ on S there exists a substitution θ on T such that $[\![\sigma(P)]\!] = \theta([\![P]\!])$ [26,37];
- Link independence: if $fn(P) \cap fn(Q) = \emptyset$ then $fn([\![P]\!]) \cap fn([\![Q]\!]) = \emptyset$ [26,30].

The list of properties given above is certainly not exhaustive, but it includes

some common properties used by the scientific community.

In general, it is not required that all of the properties above are satisfied in order for a function to be called an encoding. More specifically, there is not even a subset of these properties that is regarded as *necessary*. In fact, the conditions regarded as relevant depend on the reasons why the encoding is sought in the first place. For instance one could show that some primitives are redundant in a calculus by showing an encoding from the full set of processes to an appropriate fragment. This could be very useful for implementation purposes. This is the case for the programming language Pict [31], which is based on the asynchronous π-calculus, where input-guarded choice can be implemented [25]. One could also show that one calculus can be encoded into another in order to 'inherit' some (possibly good) properties. For instance, from the encoding of the λ-calculus into the π-calculus one could derive easily the Turing completeness of the π-calculus.

Although there is no unanimous agreement on what constitutes an encoding, it is clear that the judgement as to whether a function is an encoding relies on acceptance or rejection of the properties that hold for the encoding. That is, to give a meaning to the results that will be presented in this tutorial, the conditions on encodings we shall now present have to be accepted and considered 'reasonable'.

In dealing with leader election problems, an encoding must *preserve the fundamental criteria of the problem*, that is, the conditions for an encoding must preserve symmetric electoral systems without introducing a solution. This is what we aim to achieve with the following definition.

Definition 2.9 *Let L and L' be process languages. An encoding $[\![-]\!] : L \to L'$ is*

(1) distribution-preserving *if for all processes P, Q of L, $[\![P \mid Q]\!] = [\![P]\!] \mid [\![Q]\!]$;*
(2) permutation-preserving *if for any permutation of names σ in L there exists a permutation θ in L' such that $[\![\sigma(P)]\!] = \theta([\![P]\!])$ and the permutations are compatible on observables, in that for all $i \in \mathbb{N}$ we have $\sigma(\omega_i) = \theta(\omega_i)$, so that $\hat{\sigma}(i) = \hat{\theta}(i)$;*
(3) observation-respecting *if for any P in L,*
  (a) for every maximal computation $\mathcal{C}$ of P there exists a maximal computation $\mathcal{C}'$ of $[\![P]\!]$ such that $\mathsf{Obs}(\mathcal{C}) = \mathsf{Obs}(\mathcal{C}')$;
  (b) for every maximal computation $\mathcal{C}$ of $[\![P]\!]$ there exists a maximal computation $\mathcal{C}'$ of P such that $\mathsf{Obs}(\mathcal{C}) = \mathsf{Obs}(\mathcal{C}')$.

An encoding which preserves distribution and permutation is uniform.

The condition of preserving distribution is important in ruling out encodings which make use of a central server. That means that, if the target language

does not admit a fully distributed solution to the leader election problem, the encoding cannot introduce a spurious solution. Nestmann [23] and Prasad [32] argue that this requirement is too strong for practical purposes. We would like to defend it, on the basis that it corresponds to requiring that the degree of distribution of the processes is maintained by the translation, i.e. no coordinator is added. This condition makes the notion of encoding suitable for comparing expressiveness of languages for distributed systems, where processes are expected to coordinate without the help of a centralised control.

The second condition allows us to map symmetric networks to symmetric networks of the same size and with the same orbits. The third condition aims to preserve the uniqueness of the winner, regardless of the length of the computation. The condition is on barbs because the winner in this framework is represented with a barb.

The conditions of Definition 2.9 have been formulated with the aim of achieving the following lemma, which says that symmetric electoral systems are preserved.

Lemma 2.10 *[28] Let L and L' be process languages. Suppose $[\![-]\!] : L \to L'$ is a uniform observation-respecting encoding. Suppose that* Net *is a symmetric electoral system of size k in L with no globally bound names. Then* $[\![\mathsf{Net}]\!]$ *is a symmetric electoral system of size k in L'.* $\square$

3 Calculi

In this section we define the various calculi we shall consider.

3.1 The π-calculus with Mixed Choice

We assume the existence of names $n \in \mathcal{N}$ and co-names $\overline{n} \in \overline{\mathcal{N}}$. The set of process terms of the π-calculus with mixed choice (π_{m}) is given by the following syntax:

$$P, Q ::= \mathbf{0} \mid \textstyle\sum_{i \in I} \alpha_i.P_i \mid P \mid Q \mid \nu n\, P \mid A\langle m_1, \ldots, m_k \rangle$$

where I is a finite set. The *prefixes* of processes, ranged over by α, are defined by the following syntax:

$$\alpha ::= m(n) \mid \overline{m}\langle n \rangle$$

Summation $\sum_{i\in I}\alpha_i.P_i$ represents a finite choice among the different processes $\alpha_i.P_i$. This operator is also called *mixed choice*, since both input and output prefixes can be present in the same summation. The symbol $\mathbf{0}$, called *nil*, is the inactive process. Commonly in the π-calculus, $\mathbf{0}$ is an abbreviation for the empty choice. Although redundant, we introduce it here as a primitive for uniformity with the syntax of other calculi. We shall feel free to omit trailing $\mathbf{0}$s. Thus we write α instead of $\alpha.\mathbf{0}$. *Recursion* is handled by process identifiers with parameters; each identifier A is equipped with a defining equation $A(\vec{m}) \stackrel{\text{df}}{=} P_A$. It is common in the literature [35] on the π-calculus to handle recursion via *replication* $!\,P$. This operator simulates recursion by spinning off copies of P. Recursion and replication are equivalent in (almost) all dialects of the π-calculus. For uniformity in the presentation we have chosen to use recursion over replication. *Parallel composition* of two processes $P \mid Q$ represents P and Q computing in parallel with each other. *Restriction* $\nu n\, P$ creates a new name n in P, which is bound. The notion of the *free names* $fn(P)$ of a term P is standard, taking into account that the only binding operators are input prefix and restriction. We write $P\{n/m\}$ to mean that each free occurrence of m is substituted by n in P. We use $P\{\vec{n}/\vec{m}\}$ for the substitution of sequences of names of the same length.

We reserve η for a bijection on I; we write $\sum_{\eta(i)\in I}$ for permutation on the sub-processes in the choice operator. The *reduction relation* over the processes of π_{m} is the smallest relation satisfying the following rules:

$$\text{(Pi Comm)}\quad (m(x).P + G) \mid (\overline{m}\langle n\rangle.Q + H) \to P\{n/x\} \mid Q$$

$$\text{(Par)}\quad \frac{P \to P'}{P \mid Q \to P' \mid Q} \qquad \text{(Res)}\quad \frac{P \to P'}{\nu n\, P \to \nu n\, P'}$$

$$\text{(Str)}\quad \frac{P \equiv Q \quad Q \to Q' \quad Q' \equiv P'}{P \to P'}$$

where G, H are summations. *Structural congruence* $\equiv$ allows rearrangement of processes; it is the smallest congruence over the set of processes that satisfies the following equations:

$$P \mid \mathbf{0} \equiv P \qquad \nu n\,(P \mid Q) \equiv P \mid \nu n\, Q \quad \text{if} \quad n \notin fn(P)$$

$$P \mid Q \equiv Q \mid P \qquad \nu m\,\nu n\, P \equiv \nu n\,\nu m\, P$$

$$(P \mid Q) \mid R \equiv P \mid (Q \mid R) \qquad A\langle\vec{n}\rangle \equiv P_A\{\vec{n}/\vec{m}\} \quad \text{if } A(\vec{m}) \stackrel{\text{df}}{=} P_A$$

$$\nu n\, \mathbf{0} \equiv \mathbf{0} \qquad \textstyle\sum_{i\in I}\alpha_i.P_i \equiv \sum_{\eta(i)\in I}\alpha_{\eta(i)}.P_{\eta(i)}$$

together with α-conversion of bound names. A process P *exhibits barb* n, written as $P \downarrow n$, iff $P \equiv \nu\vec{m}\,((\overline{n}\langle x\rangle.Q + G) \mid R)$ with $n \notin \vec{m}$. We only use

barbs on outputs; input barbs are not needed, and we thereby obtain greater uniformity across the calculi we are considering.

By *public* π_{m} we mean the subcalculus without restriction.

3.2 The π-calculus with Separate Choice

The π-calculus with separate choice (π_{s}) [35] is the sub-calculus of π_{m} where summations cannot mix input and output guards. The set of processes is given by the following grammar:

$$P, Q ::= \mathbf{0} \mid \textstyle\sum_{i\in I}\alpha_i^I.P_i \mid \sum_{i\in I}\alpha_i^O.P_i \mid P|Q \mid \nu n\, P \mid A\langle m_1, \ldots, m_k\rangle$$

$$\alpha^I ::= m(n) \qquad \alpha^O ::= \overline{m}\langle n\rangle$$

The semantics of this calculus is the same as for π_{m} taking into account the syntactic restrictions.

The asynchronous π-calculus [15,2] is the fragment of π_{s} where output has no continuation, and the choice operator is not present. It has been shown [23] that separate choice can be encoded in the asynchronous π-calculus; therefore one could regard π_{s} as having the same expressive strength as the asynchronous π-calculus.

3.3 The π-calculus with Internal Mobility

The π-calculus with internal mobility (π_{I}) [34] restricts the syntax to bound output only. Thus the set of processes is the following:

$$P, Q ::= \mathbf{0} \mid \textstyle\sum_{i\in I}\alpha_i.P_i \mid P \mid Q \mid \nu n\, P \mid A\langle m_1, \ldots, m_k\rangle$$

where I is a finite set. The prefixes of processes are defined by the following syntax:

$$\alpha ::= m(n) \mid \overline{m}(n)$$

and the semantics is given by the rule above by replacing the rule (Pi Comm) with the following rule

$$\text{(Int Comm)} \quad (m(n).P + S) \mid (\overline{m}(n).Q + T) \rightarrow \nu n\, (P \mid Q)\ .$$

The set of free names $fn(P)$ of process P is defined in the standard way, taking into account that restriction, input and bounded output are the binding operators. Thus, unlike in the π-calculus, in $\overline{m}(n).P$ the name n is not free; in fact $\overline{m}(n).P$ can be thought as $\nu n\,(\overline{m}\langle n\rangle.P)$.

3.4 CCS

In this paper we shall use the version of CCS presented in [20], with the addition of value passing. As well as names $n \in \mathcal{N}$, we use co-names $\overline{n} \in \overline{\mathcal{N}}$, a set $\mathcal{V}$ of values, ranged over by $v, \ldots$, and a set $\mathcal{W}$ of variables, ranged over by $x, \ldots$. The sets $\mathcal{N}$, $\overline{\mathcal{N}}$, $\mathcal{V}$ and $\mathcal{W}$ are mutually disjoint. Processes are defined as follows:

$$P, Q ::= \mathbf{0} \;\mid\; \sum_{i\in I} \pi_i.P_i \;\mid\; P \mid Q \;\mid\; \nu n\, P \;\mid\; A\langle m_1, \ldots, m_k\rangle$$

where I is a finite set. The *prefixes* of processes, ranged over by π, are defined by the following syntax:

$$\pi ::= n(x) \;\mid\; \overline{n}\langle v\rangle.$$

The operators of the language are the same as for π_{m} apart from prefixes where in $\overline{n}\langle v\rangle.P$ we have that v is a value and not a free name. We write $P\{v/x\}$ to indicate substitution from variables to values.

The reduction relation has the rule

$$\text{(CCS Comm)} \quad (n(x).P + G) \mid (\overline{n}\langle v\rangle.Q + H) \rightarrow P\{v/x\} \mid Q$$

(where G, H are summations) together with (Par), (Res) and (Str) as for π_{m}. The notion of the *free names* $fn(P)$ of a term P is standard, taking into account that the only binding operator on names is restriction. Barbs are much as for π_{m}: a process P *exhibits barb* n, written as $P \downarrow n$, iff $P \equiv \nu\vec{m}((\overline{n}\langle v\rangle.Q{+}G) \mid R)$ with $n \notin \vec{m}$.

The difference between CCS and π_{m} may be illustrated by the π_{m} process $P \stackrel{\text{df}}{=} a(x).\overline{x}\langle b\rangle$. This is not a valid CCS process, since x cannot be used as a name in CCS. Clearly, when P is composed with $Q \stackrel{\text{df}}{=} \overline{a}\langle c\rangle.Q'$, P can acquire a new name c that may be used for future communication.

By *public* CCS we mean the subcalculus without restriction.

In the presentation of Mobile Ambients, we follow [9], except for communication, as noted below. Let $P, Q, \ldots$ range over processes and $M, \ldots$ over capabilities. We assume a set of names $\mathcal{N}$, ranged over by $m, n, \ldots$. Processes are defined as follows:

$$P, Q ::= \mathbf{0} \mid P \mid Q \mid \nu n\, P \mid !\, P \mid n[\, P \,] \mid M.P \mid (n).P \mid \langle n \rangle$$

We describe here only the operators specific to ambients: $n[\, P \,]$ is an ambient named n containing process P; $M.P$ performs capability M before continuing as P; $(n).P$ receives input on an anonymous channel, with the input name replacing free occurrences of name n in P; and finally $\langle n \rangle$ is a process which outputs name n. Notice that output is *asynchronous*, that is, it has no continuation. Restriction and input are name-binding, which naturally yields the definition of the free names $fn(P)$ of a given process P.

Capabilities are defined as follows:

$$M ::= \mathsf{in}\, n \mid \mathsf{out}\, n \mid \mathsf{open}\, n$$

Capabilities allow movement of ambients ($\mathsf{in}\, n$ and $\mathsf{out}\, n$) and dissolution of ambients ($\mathsf{open}\, n$).

We confine ourselves in this paper to communication of names, rather than full communication including capabilities (as in [9]). This serves to streamline the presentation; the results would also hold for full communication.

The *reduction* relation $\rightarrow$ is generated by the following rules:

$$\begin{array}{ll}
\text{(In)} & n[\, \mathsf{in}\, m.P \mid Q \,] \mid m[\, R \,] \rightarrow m[\, n[\, P \mid Q \,] \mid R \,] \\
\text{(Out)} & m[\, n[\, \mathsf{out}\, m.P \mid Q \,] \mid R \,] \rightarrow n[\, P \mid Q \,] \mid m[\, R \,] \\
\text{(Open)} & \mathsf{open}\, n.P \mid n[\, Q \,] \rightarrow P \mid Q \\
\text{(MA Comm)} & \langle n \rangle \mid (m).P \rightarrow P\{n/m\} \\
\text{(Amb)} & \dfrac{P \rightarrow P'}{n[\, P \,] \rightarrow n[\, P' \,]}
\end{array}$$

together with rules (Par), (Res) and (Str) as given for π_{m}. Structural congruence is the least congruence generated by the following laws:

$$\begin{array}{lll} P \mid Q \equiv Q \mid P & \nu n\, \nu m\, P \equiv \nu m\, \nu n\, P & \\ (P \mid Q) \mid R \equiv P \mid (Q \mid R) & \nu n\, (P \mid Q) \equiv P \mid \nu n\, Q & \text{if } n \notin \mathit{fn}(P) \\ P \mid \mathbf{0} \equiv P & \nu n\, m[\, P \,] \equiv m[\, \nu n\, P \,] & \text{if } n \neq m \\ !\, P \equiv P \mid !\, P & \nu n\, \mathbf{0} \equiv \mathbf{0} & \end{array}$$

together with α-conversion of bound names. Notice that movement in MA is *subjective*: ambients move themselves using the in and out capabilities. The most basic observation we can make of an MA process is the presence of an unrestricted top-level ambient. A process P *exhibits barb* n, written as $P \downarrow n$, iff $P \equiv \nu \vec{m}\, (n[\, Q \,] \mid R)$ with $n \notin \vec{m}$.

We shall be interested in various subcalculi: *pure* MA is MA without communication; *public* MA is MA without restriction; and *boxed* MA is MA without the open capability. We also use these terms with a similar meaning when discussing the other forms of ambient calculi we are about to introduce.

3.6 Safe Ambients

The calculus of Safe Ambients (SA) [16] is a variant of MA where new *co-capabilities* are added to complement the existing in, out and open capabilities. The syntax of processes is the same as for MA, except that capabilities are defined as follows:

$$M ::= \mathsf{in}\, n \mid \mathsf{out}\, n \mid \mathsf{open}\, n \mid \overline{\mathsf{in}}\, n \mid \overline{\mathsf{out}}\, n \mid \overline{\mathsf{open}}\, n$$

Structural congruence and the *reduction* relation $\rightarrow$ are defined as for MA, except that rules (In), (Out) and (Open) are replaced by the following:

$$\begin{array}{ll} \text{(Safe In)} & n[\, \mathsf{in}\, m.P \mid Q \,] \mid m[\, \overline{\mathsf{in}}\, m.R \mid S \,] \rightarrow m[\, n[\, P \mid Q \,] \mid R \mid S \,] \\ \text{(Safe Out)} & m[\, n[\, \mathsf{out}\, m.P \mid Q \,] \mid \overline{\mathsf{out}}\, m.R \mid S \,] \rightarrow n[\, P \mid Q \,] \mid m[\, R \mid S \,] \\ \text{(Safe Open)} & \mathsf{open}\, n.P \mid n[\, \overline{\mathsf{open}}\, n.Q \mid R \,] \rightarrow P \mid Q \mid R \end{array}$$

Barbs are defined slightly differently from MA. A process P *exhibits barb* n, written as $P \downarrow n$, iff $P \equiv \nu \vec{m}\, (n[\, M.Q \mid R \,] \mid S)$ with $n \notin \vec{m}$ and M either $\overline{\mathsf{in}}\, n$ or $\overline{\mathsf{open}}\, n$.

There is a standard encoding of MA into SA, as follows:

$$[\![n[\, P\,]]\!] \stackrel{\text{df}}{=} n[\, !\,\overline{\mathsf{in}}\, n \mid !\,\overline{\mathsf{out}}\, n \mid \overline{\mathsf{open}}\, n \mid [\![P]\!]\,]$$

(with $[\![-]\!]$ homomorphic on the remaining operators) [16] [1] .

3.7 The Push and Pull Ambient Calculus

The Push and Pull Ambient Calculus (PAC) [29,37] is a variant of MA where the subjective moves enabled by the in and out capabilities are replaced by objective moves whereby ambients can be pulled in or pushed out by other ambients. The syntax of processes is the same as for MA. Capabilities are defined as follows:

$$M ::= \mathsf{pull}\, n \;\mid\; \mathsf{push}\, n \;\mid\; \mathsf{open}\, n$$

The reduction rules are the same as for MA, except that (In) and (Out) are replaced by the following:

$$\text{(Pull)} \quad n[\, \mathsf{pull}\, m.P \mid Q\,] \mid m[\, R\,] \to n[\, P \mid Q \mid m[\, R\,]\,]$$
$$\text{(Push)} \quad n[\, m[\, P\,] \mid \mathsf{push}\, m.Q \mid R\,] \to n[\, Q \mid R\,] \mid m[\, P\,]$$

Barbs are defined as for MA.

3.8 Boxed Ambients

The calculus of Boxed Ambients (BA) [4,5] is derived from MA by removing the open capability and allowing parent-child communication as well as same-level communication. Processes are defined as follows:

$$P, Q ::= \mathbf{0} \;\mid\; P \mid Q \;\mid\; \nu n\, P \;\mid\; !\, P \;\mid\; n[\, P\,] \;\mid\; M.P \;\mid\; (\vec{n})^{\eta}.P \;\mid\; \langle \vec{n} \rangle^{\eta}.P$$

Here $\vec{n}$ denotes a tuple of names, and η ranges over *locations*, defined as follows:

$$\eta ::= n \;\mid \uparrow \mid\; \star$$

[1] The original version has $!\,\overline{\mathsf{open}}\, n$, but the replication can be omitted, as ambients are opened at most once.

The "local" location $\star$ is elided. Notice that output $\langle \vec{n} \rangle^{\eta}.P$ is synchronous, unlike in MA. Capabilities M are defined as for MA but without open. The reduction rules are the same as for boxed MA, except for communication, where the rule (MA Comm) is replaced by the following five rules:

$$\begin{array}{ll}
\text{(Local)} & (\vec{m}).P \mid \langle \vec{m'} \rangle.Q \to P\{\vec{m'}/\vec{m}\} \mid Q \\
\text{(Input } n\text{)} & (\vec{m})^{n}.P \mid n[\, \langle \vec{m'} \rangle.Q \mid R \,] \to P\{\vec{m'}/\vec{m}\} \mid n[\, Q \mid R \,] \\
\text{(Input } \uparrow\text{)} & n[\, (\vec{m})^{\uparrow}.P \mid Q \,] \mid \langle \vec{m'} \rangle.R \to n[\, P\{\vec{m'}/\vec{m}\} \mid Q \,] \mid R \\
\text{(Output } n\text{)} & n[\, (\vec{m}).P \mid Q \,] \mid \langle \vec{m'} \rangle^{n}.R \to n[\, P\{\vec{m'}/\vec{m}\} \mid Q \,] \mid R \\
\text{(Output } \uparrow\text{)} & (\vec{m}).P \mid n[\, \langle \vec{m'} \rangle^{\uparrow}.Q \mid R \,] \to P\{\vec{m'}/\vec{m}\} \mid n[\, Q \mid R \,]
\end{array}$$

Clearly, rule (Local) extends rule (MA Comm), so that communication in BA is at least as powerful as communication in MA. Note that pure BA is the same as pure boxed MA.

Barbs are defined as for MA.

4 Leader Election in General Symmetric Networks

We present solutions to the leader election problem for symmetric networks in a variety of calculi (Section 4.1), followed by results showing the impossibility of solutions in other calculi (Section 4.2). We conclude the section by using the preceding to obtain separation results (Section 4.3).

4.1 Calculi with Electoral Systems

In this section we present solutions to the leader election problem in symmetric networks of any finite size in some fragments of the following calculi: CCS, π_{m}, MA, PAC and SA. The solutions are of course still valid in the respective full calculi. The solutions for CCS and π_{m} are the same, since CCS is a subcalculus of π_{m} and therefore once a solution is proposed for CCS it trivially implies that there is a solution for π_{m}. This is equally true for MA and SA; however, Theorem 4.11 presents an alternative solution for SA that uses the co-capabilities. Such a solution seems only possible in SA. Below we report the suitable fragments of the calculi cited above.

Definition 4.1 *(1) Let $\pi_{\mathsf{m}}^{-\nu}$ be public π_{m}.*
(2) Let $CCS^{-\nu}$ be public CCS.
(3) Let $\mathrm{MA}^{\mathsf{io}}$ be pure public boxed MA.
(4) Let $\mathrm{PAC}^{\mathsf{pp}}$ be pure public boxed PAC.

(5) Let $\mathrm{SA}^{\mathsf{io}}$ *be pure public boxed SA.*
(6) Let $\mathrm{SA}^{\mathsf{iop}}$ *be pure public SA without the* out *capability.*

We start by defining a symmetric electoral system of size two in $\mathrm{CCS}^{-\nu}$. Let a network Net be defined as follows:

$$P_0 \stackrel{\mathrm{df}}{=} n_1 + \overline{n_0}.\overline{\omega_0} \qquad P_1 \stackrel{\mathrm{df}}{=} n_0 + \overline{n_1}.\overline{\omega_1} \qquad \mathsf{Net} \stackrel{\mathrm{df}}{=} P_0 \mid P_1$$

(Here we omit the values passed, which are just dummies which play no rôle.) The network is symmetric with respect to a single-orbit automorphism σ defined as follows:

$$\sigma(n_0) = n_1 \quad \sigma(n_1) = n_0 \quad \sigma(\omega_0) = \omega_1 \quad \sigma(\omega_1) = \omega_0.$$

with σ the identity on all other names. There are only two possible computations. One can be described as follows:

$$\mathcal{C} : \mathsf{Net} \to \overline{\omega_1} \downarrow \omega_1 \qquad \mathsf{Obs}(\mathcal{C}) = \{\omega_1\}$$

The other one is identical up to the renaming of σ. The values passed are just dummies, which can be omitted; there is a crucial use of mixed choice to break symmetry.

The previous solution can be generalised to networks of any size k. Before giving the formal definition, we provide an informal description of the algorithm.

Informal Description 4.2 *At every step a pair of processes fight each other. Winning an individual fight is achieved by sending a message to the loser. Each time, the loser drops out of the contest. Eventually only one process is left standing. It has defeated every other process and is therefore the winner. Each node is composed of two parts:*

(1) A process that either sends a message to another node and proceeds to fight the remaining processes, or receives a message and will no longer take part in the election process. In this latter case, it will announce to every other node that it has lost.
(2) A counter, which collects all the messages of loss from the other processes, and after $k-1$ *messages declares victory (so processes have to know the size of the network).*

One important feature in this implementation is the use of mixed choice, in the description of the process that runs for the election.

Let $\prod_{i<k} P_i$ stand for $P_0 \mid \cdots \mid P_{k-1}$.

Theorem 4.3 *For any $k \geq 1$, in CCS$^{-\nu}$ there exists a symmetric electoral system of size k defined by* $\mathsf{Net} \stackrel{\text{df}}{=} \prod_{i<k} P_i$ *where*

$$\begin{aligned} P_i &\stackrel{\text{df}}{=} \mathit{Elect}_i \mid \mathit{Counter}^k_{i,0} \\ \mathit{Elect}_i &\stackrel{\text{df}}{=} \overline{n_i}.\mathit{Elect}_i + \textstyle\sum_{0\leq s<k, s\neq i} n_s.(\prod_{0\leq t<k, t\neq i} \overline{\mathit{lost}_t}) \\ \mathit{Counter}^k_{i,j} &\stackrel{\text{df}}{=} \mathit{lost}_i.\mathit{Counter}^k_{i,j+1} \quad (0 \leq j < k-1) \\ \mathit{Counter}^k_{i,k-1} &\stackrel{\text{df}}{=} \overline{\omega_i}. \end{aligned}$$

□

Because CCS$^{-\nu}$ can be regarded as a subcalculus of $\pi^{-\nu}_{\mathsf{m}}$, the algorithm written above is also a solution for $\pi^{-\nu}_{\mathsf{m}}$. Hence:

Corollary 4.4 *For any $k \geq 1$, in $\pi^{-\nu}_{\mathsf{m}}$ there exists a symmetric electoral system of size k.*

Clearly, since CCS without value passing is a subcalculus of π_{I}, also π_{I} admits a electoral system.

Corollary 4.5 *For any k, in π_{I} there exists a symmetric electoral system of size k.*

We now turn to showing the existence of symmetric electoral systems in MA. In fact we can use solely the fragment MA$^{\mathsf{io}}$. Before presenting a solution for networks of arbitrary size, we present an electoral system of size two. Let

$$P_0 \stackrel{\text{df}}{=} n_0[\,\mathsf{in}\, n_1.\omega_0[\,\mathsf{out}\, n_0.\mathsf{out}\, n_1\,]\,] \qquad P_1 \stackrel{\text{df}}{=} n_1[\,\mathsf{in}\, n_0.\omega_1[\,\mathsf{out}\, n_1.\mathsf{out}\, n_0\,]\,]$$

$$\mathsf{Net} \stackrel{\text{df}}{=} P_0 \mid P_1 \ .$$

Then Net is symmetric with respect to a single-orbit automorphism σ defined as follows:

$$\sigma(n_0) = n_1 \quad \sigma(n_1) = n_0 \quad \sigma(\omega_0) = \omega_1 \quad \sigma(\omega_1) = \omega_0.$$

There are only two possible computations. We shall present the first one in detail:

$$\begin{aligned} \mathcal{C} : n_0[\,\mathsf{in}\, n_1.\omega_0[\,\mathsf{out}\, n_0.\mathsf{out}\, n_1\,]\,] \mid n_1[\,\mathsf{in}\, n_0.\omega_1[\,\mathsf{out}\, n_1.\mathsf{out}\, n_0\,]\,] &\rightarrow \\ n_1[\,n_0[\,\omega_0[\,\mathsf{out}\, n_0.\mathsf{out}\, n_1\,]\,] \mid \mathsf{in}\, n_0.\omega_1[\,\mathsf{out}\, n_1.\mathsf{out}\, n_0\,]\,] &\rightarrow \\ n_1[\,\omega_0[\,\mathsf{out}\, n_1\,] \mid n_0[\,] \mid \mathsf{in}\, n_0.\omega_1[\,\mathsf{out}\, n_1.\mathsf{out}\, n_0\,]\,] &\rightarrow \\ \omega_0[\,] \mid n_1[\,n_0[\,] \mid \mathsf{in}\, n_0.\omega_1[\,\mathsf{out}\, n_1.\mathsf{out}\, n_0\,]\,] \downarrow \omega_0 \end{aligned}$$

Thus we conclude $\mathsf{Obs}(\mathcal{C}) = \{\omega_0\}$. The other computation is identical up to renaming via σ. Notice that symmetry is broken by one ambient entering the other.

The general solution for a network of any size is more complex, and before introducing the technical solution we shall provide an informal description which covers both MA and PAC.

Informal Description 4.6 *The basic idea of the algorithm is that winning the election is achieved by having all the opponents inside. Each node in the network is composed of two ambients: one that runs for the election and the other that has the rôle of a counter. Any ambient entering another one has lost the election. It will release an ambient called lose, which will eventually appear at the top level, where the counters are. The winning ambient is left on its own, at the top level, while all the other ambients are inside the winner. The counter will declare the winner once every loser has entered it.*

Theorem 4.7 *[28] In* $\mathrm{MA}^{\mathsf{io}}$*, for any* $k \geq 1$ *there exists a symmetric electoral system of size* k*, defined by* $\mathsf{Net} \stackrel{\mathrm{df}}{=} \prod_{i<k} P_i$ *where*

$$
\begin{aligned}
P_i &\stackrel{\mathrm{df}}{=} n_i[\,\textstyle\prod_{j\neq i} \mathsf{in}\; n_j.lose_i[\,Outn\,]\,] \mid c_i[\,C_{i,i+1}\,] \\
Outn &\stackrel{\mathrm{df}}{=} \textstyle\prod_{j<k} !\,\mathsf{out}\; n_j \\
C_{i,i} &\stackrel{\mathrm{df}}{=} \omega_i[\,\mathsf{out}\; c_i\,] \\
C_{i,j} &\stackrel{\mathrm{df}}{=} \mathsf{in}\; lose_j.\mathsf{out}\; lose_j.C_{i,j+1} \qquad (j \neq i) \qquad \Box
\end{aligned}
$$

In the preceding theorem we use addition modulo k.

We dualise the construction given in the proof of Theorem 4.7, essentially replacing in by pull and out by push.

Theorem 4.8 *[28] In* $\mathrm{PAC}^{\mathsf{pp}}$*, for any* $k \geq 1$ *there exists a symmetric electoral system of size* k*, defined by* $\mathsf{Net} \stackrel{\mathrm{df}}{=} \prod_{i<k} P_i$ *where*

$$
\begin{aligned}
P_i &\stackrel{\mathrm{df}}{=} n_i[\,\textstyle\prod_{j\neq i}(\mathsf{pull}\; n_j.lose_j[\,] \mid \mathsf{push}\; lose_j)\,] \mid c_i[\,C_{i,i+1}\,] \\
C_{i,i} &\stackrel{\mathrm{df}}{=} \omega_i[\,] \mid \mathsf{push}\; \omega_i \\
C_{i,j} &\stackrel{\mathrm{df}}{=} \mathsf{pull}\; lose_j.\mathsf{push}\; lose_j.C_{i,j+1} \qquad (j \neq i) \qquad \Box
\end{aligned}
$$

We now consider electoral systems in fragments of SA. We can take the symmetric electoral system in the statement of the proof Theorem 4.7 and adapt it for $\mathrm{SA}^{\mathsf{io}}$ using the standard encoding (Section 3.6), with the one change that we omit the $\overline{\mathsf{open}}\, n$ from the encoding of $n[\,P\,]$.

Corollary 4.9 *In* $\mathrm{SA}^{\mathsf{io}}$*, for any* $k \geq 1$ *there exists a symmetric electoral system of size* k*.* $\square$

In constructing electoral systems in $\mathrm{MA}^{\mathsf{io}}$, we use the in capability to break symmetry and the out to report the winner at the top level. An interesting feature of SA is that we can also construct electoral systems using just the in and the open capabilities, with the open enabling the reporting of the winner.

Here is a symmetric electoral system of size two in $\mathrm{SA}^{\mathsf{iop}}$:

$$\begin{aligned} P_0 &\stackrel{\mathrm{df}}{=} \mathsf{open}\ n_0 \mid n_0[\,\mathsf{in}\ n_1 \mid \overline{\mathsf{in}}\ n_0.\overline{\mathsf{open}}\ n_0.\omega_0[\,\overline{\mathsf{open}}\ \omega_0\,]\,] \\ P_1 &\stackrel{\mathrm{df}}{=} \mathsf{open}\ n_1 \mid n_1[\,\mathsf{in}\ n_0 \mid \overline{\mathsf{in}}\ n_1.\overline{\mathsf{open}}\ n_1.\omega_1[\,\overline{\mathsf{open}}\ \omega_1\,]\,] \\ \mathsf{Net} &\stackrel{\mathrm{df}}{=} P_0 \mid P_1. \end{aligned}$$

The first process to perform an in reduction loses.

For the solution for arbitrary sizes we provide an informal description first.

Informal Description 4.10 *Similarly to the algorithm for MA and PAC, each node in the network is composed of two ambients: one that runs for the election and the counter. Any ambient in the network that gets entered is a loser (the opposite of what happens in the solution for two processes given above). After being entered, an ambient can be opened and then forced to release the ambient lose, which will help to decrease the counter. In fact, the counter decrements by opening the ambient lose. The winning ambient is left on its own after the losing ambients have all been opened. The counter declares the winner after having opened all the lose ambients.*

Theorem 4.11 *[28] In* $\mathrm{SA}^{\mathsf{iop}}$*, for any* $k \geq 1$ *there exists a symmetric electoral system of size* k *defined by* $\mathsf{Net} \stackrel{\mathrm{df}}{=} \prod_{i<k} P_i$ *where*

$$\begin{aligned} P_i &\stackrel{\mathrm{df}}{=} \mathsf{open}\ n_i \mid C_{i,i+1} \mid n_i[\,\overline{\mathsf{in}}\ n_i.\overline{\mathsf{open}}\ n_i.lose_i[\,\overline{\mathsf{open}}\ lose_i\,] \mid \textstyle\prod_{j\neq i} \mathsf{in}\ n_j\,] \\ C_{i,i} &\stackrel{\mathrm{df}}{=} \omega_i[\,\overline{\mathsf{open}}\ \omega_i\,] \\ C_{i,j} &\stackrel{\mathrm{df}}{=} \mathsf{open}\ lose_j.(lose_j[\,\overline{\mathsf{open}}\ lose_j\,] \mid C_{i,j+1}) \quad (j \neq i) \end{aligned}$$ $\square$

Before concluding, we present one last algorithm for $\mathrm{MA}^{\mathsf{io}}$, which is slightly simpler than the one presented in Theorem 4.7; however correctness is more difficult to prove, and it is not clear how to dualise the construction for PAC.

Informal Description 4.12 *The idea is that the processes that take part in the election can enter one another, until they form a linear stack. At this point no further movement of the main ambient is possible, and the leader is*

the ambient which is at the top of the stack. A probe ambient ω can descend to the bottom of the stack, and then ascend to the top of the stack. Finally ω emerges at the top level and declares the winner.

Theorem 4.13 *[28] In* $\mathrm{MA}^{\mathsf{io}}$*, for any $k \geq 1$ there exists a symmetric electoral system* $\mathsf{Net} \stackrel{\mathrm{df}}{=} \prod_{i<k} P_i$ *of size k, defined as follows: for $i < k$, let $S_i^k = \{n_j : j < k, j \neq i\}$, and let T_i^k be the set of all strings of length $k-1$ using the members of S_i^k exactly once each. Given an element s of T_i^k we denote by s^- the string which is s in reverse order. By* $\mathsf{in}\,(s)$ *we mean the sequence of* $\mathsf{in}\,n_j$ *capabilities for each successive $n_j \in s$ (similarly for* out*). We set:*

$$P_i \stackrel{\mathrm{df}}{=} n_i[\prod_{j\neq i} \mathsf{in}\,n_j \mid \prod_{s\in T_i^k} \omega_i[\,\mathsf{in}\,(s).\mathsf{out}\,(s^-).\mathsf{out}\,n_i\,]\,] \qquad \square$$

4.2 Calculi without Electoral Systems

In this section we shall show that there are calculi that do not admit a symmetric electoral system. We shall see that certain operators are needed to break symmetry and for a solution to be possible. For π-calculus and CCS the crucial operator is the mixed choice operator. In fact, both π-calculus and CCS with separate choice cannot solve the problem of electing a leader in any graph. For MA and SA the in capability is the symmetry-breaking operator, while for PAC the pull capability is necessary.

The proof of the impossibility of leader election in a symmetric network has different technical details according to the different formalisms, but there is a common structure. The idea is that whenever a process takes a step, this step can be imitated symmetrically by all the other processes, completing a "round", at the end of which symmetry is restored. In this way we construct a maximal computation which preserves, at the end of each round, the invariant property of being in a symmetric state. For this maximal computation, election fails either because no one declares himself the winner or, if anybody declares himself a winner, the other processes in the network can do the same, during the same round. The proof method just described is very much inherited from a classical result in distributed computing [17, Chapter 3.2].

To make this more concrete we consider an example in π_{s}.

$$P_0 \stackrel{\mathrm{df}}{=} \overline{n_0}\langle z\rangle.\overline{\omega_0} \mid n_1(z) \qquad P_1 \stackrel{\mathrm{df}}{=} \overline{n_1}\langle z\rangle.\overline{\omega_1} \mid n_0(z) \qquad \mathsf{Net} \stackrel{\mathrm{df}}{=} P_0 \mid P_1.$$

The network of size two written above is symmetric with the standard automorphism that swaps 1 and 0, but it is not an electoral system. To see this it

is sufficient to follow one maximal computation:

$$\mathcal{C}: \quad P_0 \mid P_1 \to \overline{\omega_0} \mid n_1(z) \mid \overline{n_1}\langle z\rangle.\overline{\omega_1} \to \overline{\omega_0} \mid \overline{\omega_1} \;\downarrow \omega_1, \omega_0.$$

This example shows that, after the initial step breaking symmetry made by P_0 in trying to declare himself the winner, P_1 can respond in a similar way, which leads to a symmetric network again. Finally, no leader is elected because there is more than one winner: $\mathsf{Obs}(\mathcal{C}) = \{\omega_1, \omega_0\}$. The proof for the general case follows closely such reasoning; each time a step is made by a process (or pair of processes), all other processes can mimic this step, in such a way that symmetry is reached again, and no winner is possible.

There is no solution to the leader election problem in π_{s}:

Theorem 4.14 *[26] Let* $\mathsf{Net} = [P_0 \mid \cdots \mid P_{k-1}]$ *with* $k \geq 2$ *be a symmetric network in* π_{s}*. Then* Net *cannot be an electoral system.* □

A corollary of Theorem 4.14 would be a similar result for CCS with separate choice; however, unlike π_{s}, such a calculus has never been considered, and therefore we leave out the statement. It is clear that the mixed choice operator is the key for the expressiveness result in the π-calculus. In MA and SA, the in capability is crucial in order to break the symmetry; in fact, if this is removed, the leader election problem cannot be solved. For PAC the removal of the pull defines a calculus that cannot elect a leader in any graph.

Definition 4.15 *(1) Let* $\mathrm{MA}^{-\mathsf{in}}$ *denote MA without the* in *capability.*
(2) Let $\mathrm{SA}^{-\mathsf{in}}$ *be SA without the* in *capability.*
(3) Let $\mathrm{PAC}^{-\mathsf{pull}}$ *be PAC without the* pull *capability.*

Theorem 4.16 *[28] Let* $k \geq 2$.

(1) Let $\mathsf{Net} = [P_0 \mid \cdots \mid P_{k-1}]$ *be a symmetric network in* $\mathrm{MA}^{-\mathsf{in}}$*. Then* Net *cannot be an electoral system.*
(2) Let $\mathsf{Net} = [P_0 \mid \cdots \mid P_{k-1}]$ *be a symmetric network in* $\mathrm{PAC}^{-\mathsf{pull}}$*. Then* Net *cannot be an electoral system.*
(3) Let $\mathsf{Net} = [P_0 \mid \cdots \mid P_{k-1}]$ *be a symmetric network in* $\mathrm{SA}^{-\mathsf{in}}$*. Then* Net *cannot be an electoral system.* □

4.3 Separation Results

By Lemma 2.10, a uniform observation-respecting encoding maps symmetric electoral systems (with no globally bound names) to symmetric electoral systems. So for instance we can now deduce that there can be no uniform

observation-respecting encoding from π_{m} into π_{s}, since the former has a symmetric electoral system of at least size two (from Corollary 4.4) and the latter does not (Theorem 4.14).

We can tabulate the positive results of Section 4.1 and the negative results of Section 4.2 in the following diagram:

$\mathrm{CCS}^{-\nu}$	$\pi_{\mathsf{m}}^{-\nu}$	π_{l}	$\mathrm{MA}^{\mathsf{io}}$	$\mathrm{PAC}^{\mathsf{pp}}$	$\mathrm{SA}^{\mathsf{io}}$	$\mathrm{SA}^{\mathsf{iop}}$
	π_{s}	$\mathrm{MA}^{-\mathsf{in}}$		$\mathrm{PAC}^{-\mathsf{pull}}$	$\mathrm{SA}^{-\mathsf{in}}$	

All calculi above the line have symmetric electoral systems for any finite size. Those below the line do not have symmetric electoral systems for any size greater than one. Therefore there is no uniform, observation-respecting encoding from any calculus above the line to any below the line, giving us many separation results.

On the calculi above the line we have considered the smallest fragment which solves leader election problems. Clearly, the full calculus of each fragment above solves leader election problems as well. On the other hand for the calculi below the line, we have considered the largest fragment that does not admit a solution to the problem. Our diagram above also highlights which operators in each calculus make the difference in expressiveness: for the π-calculus (and CCS) it is mixed choice; for MA and SA it is in and for PAC it is pull.

Separation results test the ability to reach an agreement in a fully distributed way, provided that every node in the network is programmed identically. While this is an important feature in distributed systems, and a valuable metric to evaluate concurrent models of computation, our method says nothing on other aspects of the model, for instance Turing completeness or expressiveness of name passing.

5 Leader Election in Symmetric Rings

In distributed computing, one standard network topology is a ring, where each process can only communicate with its left-hand and right-hand neighbours. As far as leader election is concerned, this means that algorithms which assume that all processes are directly linked to all other processes (as considered in Section 4) will no longer work. In this section we examine whether enhanced leader election algorithms which can handle rings are available for the languages we are considering. This will enable us to separate some of the languages in the top row of the diagram in Section 4.3.

One possible way to conduct leader election in rings is what we shall call the *two-phase* method. This starts by using an algorithm to create links between all processes. Symmetry is preserved during this first (or *link-creation*) phase. Once this is done, in the second (or *election*) phase a leader election algorithm devised for fully connected networks (as in Section 4) can be used to produce the leader.

The π-calculus has the power to create new links; we shall see that the link-creation phase referred to above can be carried out in π_{m} (in fact it can be done in π_{s}). Since π_{m} can solve leader election for fully connected networks, it can therefore perform leader election on rings using the two-phase method. By contrast, CCS does not have the power to create new links, and it can be shown that CCS cannot perform leader election on rings with composite (non-prime) size. We need the compositeness condition because our method depends on partitioning the ring into equal-sized sets of non-adjacent nodes.

We now consider the ambient world. In MA, SA and PAC, the communication primitives have the same operational semantics as the π-calculus, except that they are *anonymous*, in the sense that there are no channels on which communication happens (in the π-calculus one would write $m(x).P$ for an input on the channel m, while in MA one would write $(x).P$ for an anonymous input). Since communication primitives in ambients are very similar to those of the π-calculus, it would be not surprising if the two-phase method could be formulated in MA, SA and PAC, since they all solve the leader election problem in fully connected networks. It turns out that the leader election problem for symmetric rings of any size is solved without the use of communication primitives. This means that link passing, in this case, is somehow simulated, since there is no explicit way of passing names in the absence of communication. The open capability is crucial in this setting. It is, in fact, the capability that simulates link passing, since it can be shown that MA, SA and PAC without the open capability do not admit a solution for leader election problems in rings of composite size. The situation is different for BA, where the open capability is missing as a design choice. Communications between parent and child ambients are allowed, and the synchronous choice-free π-calculus can be encoded, and with that, clearly, the power of creating new links. Thus it can be shown that in BA the leader election problem in a ring of any size can be solved by converting the ring into a fully connected network and then using the algorithm of Theorem 4.7 (or that of Theorem 4.13).

5.1 Rings and Independence Preservation

We start by providing a general framework for leader election problems in rings, augmenting that presented in Section 2.2.

Given a network $\mathsf{Net} = \nu\vec{n}\,(P_0 \mid \cdots \mid P_{k-1})$, we can associate a graph with Net by letting the set of nodes be $\{0, \ldots, k-1\}$ and letting $i, j < k$ be adjacent iff $fn(P_i) \cap fn(P_j) \neq \emptyset$. A network forms a ring if the processes can be arranged in a cycle, and each node i is adjacent to at most its two neighbours in the cycle.

Definition 5.1 *A* ring *is a network* $\mathsf{Net} = \nu\vec{n}\,(P_0 \mid \cdots \mid P_{k-1})$ *which has a single-orbit automorphism* σ *such that for all* $i, j < k$*, if* $fn(P_i) \cap fn(P_j) \neq \emptyset$ *then one of* $i = j$*,* $\hat{\sigma}(i) = j$ *or* $\hat{\sigma}(j) = i$ *must hold. A ring is* symmetric *if it is symmetric with respect to such an automorphism* σ*.*

Notice that the definition bans links between non-adjacent nodes in the ring, but does not require the existence of links between adjacent nodes. Thus a completely disconnected network is in fact a ring.

Also note that the definition says nothing about the direction of communication in a ring, which can therefore be in either direction. Thus, in distributed systems terms, we are defining bidirectional rings rather than unidirectional ones. Some of our algorithms are in fact unidirectional in character, but this extra information plays no part in obtaining separation results.

Recall that an *independent set* in a graph is a set of nodes such that no two nodes of the set are adjacent.

Definition 5.2 *Two processes* P *and* Q *are* independent *if they do not share any free names:* $fn(P) \cap fn(Q) = \emptyset$*.*

Definition 5.3 *Let* σ *be an automorphism on a network* $\mathsf{Net} = \nu\vec{n}\,(P_0 \mid \cdots \mid P_{k-1})$*. Then* Net *is* independent *with respect to* σ *if every orbit forms an independent set, in the sense that if* $i, j < k$ *are in the same orbit of* $\hat{\sigma}$ *with* $i \neq j$*, then* P_i *and* P_j *are independent.*

Unlike in Section 4, in this section we shall consider encodings which map rings to rings. We therefore need a further property on top of uniformity and the preservation of the observables. This property will guarantee that the connectivity of the original network is not increased.

Definition 5.4 *An encoding is* independence-preserving *if for any processes* P*,* Q*, if* P *and* Q *are independent then* $[\![P]\!]$ *and* $[\![Q]\!]$ *are also independent.*

The property above states that such an encoding "does not increase the level of connectivity of the network". Not all encodings preserve independence. For instance, Zimmer's [39] encoding of the synchronous π-calculus without choice into pure SA introduces a new global ambient whose name is shared by all processes.

Lemma 5.5 *[30] Suppose $[\![-]\!] : L \to L'$ is a uniform, observation-respecting and independence-preserving encoding. Suppose that* Net *is a symmetric ring of size $k \geq 1$ which is an electoral system. Then $[\![\mathsf{Net}]\!]$ is also a symmetric ring of size k which is an electoral system.* □

5.2 Calculi with Electoral Systems for Rings

In this section we show that we can solve leader election on symmetric rings in π_{m} and in ambient calculi. The solution for MA can be carried over to SA by a standard encoding. There is a fundamental difference between the solution for PAC and the others: the PAC solution works for a ring of any size with a single uniform definition for each component, so that the processes do not need to know the size of the ring.

We start with a solution to the leader election problem for rings in both π_{m} and BA; we consider both calculi at once because at some level of abstraction the algorithm is the same. We provide an informal explanation first.

Informal Description 5.6 *The algorithm has two phases. In phase one the processes pass names around the ring so that every process becomes directly connected to every other process. Here there is an essential use of the π-calculus, though without any use of choice.*

We define a symmetric ring $P_0 \mid \cdots \mid P_{k-1}$ which is an electoral system. Suppose that process P_i has a channel n_i initially known only to itself, and can send messages to P_{i-1} along channel x_i. Then the names n_i are passed around the ring so that all processes share them and can use them in the election phase. We have to be careful that for each P_i the outputs occur in the same order as the inputs, so that names do not get confused. We therefore allocate to each P_i a "synchroniser" name y_i which ensures that each successive output is completed before the next one is enabled. We elide the dummy names passed along y_i.

For $0 \leq i \leq k$, we let $P_i \stackrel{\text{df}}{=} P_i^0\langle x_i, x_{i+1}, y_i, n_i\rangle$, where for $0 \leq j \leq k-2$ we let

$$P_i^j(x_i, x_{i+1}, y_i, n_i, \ldots, n_{i+j}) \stackrel{\text{df}}{=} \bar{x}_i\langle n_{i+j}\rangle.\bar{y}_i \mid x_{i+1}(n_{i+j+1}).y_i.P_i^{j+1}\langle x_i, x_{i+1}, y_i, n_i, \ldots, n_{i+j+1}\rangle$$

and $P_i^{k-1}(x_i, x_{i+1}, y_i, n_i, \ldots, n_{i-1}) \stackrel{\text{df}}{=} Q_i\langle n_i, \ldots, n_{i-1}\rangle$. Here Q_i is a process which has acquired all the n_i and is ready to carry out the election phase. Once Q_i is reached, the names x_i, x_{i+1} and y_i are no longer required.

For π_{m}, we have seen what the Q_i would look like in Theorem 4.3, and therefore we can state the following theorem:

Theorem 5.7 *(cf. [26]) For any $k \geq 1$, there is a symmetric ring of size k which is an electoral system in $\pi_{\mathsf{m}}^{-\nu}$.* $\square$

For BA, since it is possible to encode choice-free synchronous π-calculus [5], we can carry out the link-creation phase in BA. We use the following translation of the π-calculus input and synchronous output:

$$[\![x(y).P]\!] \stackrel{\text{df}}{=} (y,z)^x.(z[\,\langle\rangle\,] \mid [\![P]\!])$$
$$[\![\bar{x}\langle y\rangle.P]\!] \stackrel{\text{df}}{=} x[\,\langle y,z\rangle\,] \mid ()^z.[\![P]\!]$$

where z is fresh in the translation of output (i.e., when translating particular networks from the π-calculus into BA we choose each of the zs to be distinct from each other and from any other names used). This translation is adapted from [5], which used restriction. Note that we do not need restriction, since in our particular setting there is no harm in introducing fresh public names. We need to define the Q_i. We could use the process defined in Theorem 4.7 or the one defined in Theorem 4.13. Therefore we have the following theorem:

Theorem 5.8 *[30] For any $k \geq 1$, there is a symmetric ring of size k which is an electoral system in public BA.* $\square$

We next turn to PAC. We show that using push and pull we can build a symmetric ring of processes which can elect a leader. Moreover, the construction is such that individual processes do not know the size of the ring.

This algorithm is different from all the others because each node can be described without knowing (either in advance or as a result of the computation) the size of the network. In other words, no counter is necessary for this solution.

Informal Description 5.9 *The gist of the algorithm is that communication goes in one direction only, for instance, from left-hand neighbours to right-hand neighbours. The right-hand neighbour is pulled and opened, reducing the size of the ring; if the left-hand neighbour is opened then this means that there are no other processes left, and therefore the last-standing ambient is the winner.*

Theorem 5.10 *[30] For any $k \geq 1$, there is a symmetric ring of size k which is an electoral system in pure public PAC, defined by* $\mathsf{Net} \stackrel{\text{df}}{=} \prod_{i<k} P_i$ *where*

$$P_i \stackrel{\text{df}}{=} n_i[\,Q_i \mid \mathsf{pull}\, n_{i+1} \mid \mathsf{open}\, n_{i+1}\,]$$
$$Q_i \stackrel{\text{df}}{=} n_i[\,\omega_i[\,]\,] \mid \mathsf{push}\, \omega_i \qquad \square$$

We now discuss the solution to the leader election problem for rings in pure public MA.

Informal Description 5.11 *We use the two-phase method. In the link-creation phase we send ambients round the ring which contain the appropriate capabilities. These are opened by their intended recipients, which then can exercise these capabilities. We already know how to carry out the election phase from Theorems 4.7 and 4.13, though in fact we use a different algorithm, which is easier to set up via the link-creation phase.*

We omit the precise details of the construction, as they are quite lengthy.

Theorem 5.12 *[30] For any $k \geq 1$ there is a symmetric ring of size k which is an electoral system in pure public MA.* □

Corollary 5.13 *For any $k \geq 1$ there is a symmetric ring of size k which is an electoral system in pure public SA.*

5.3 Calculi without Electoral Systems for Rings

In this section, we consider the calculi that do not have electoral systems for symmetric rings. In this case, the failure of the election is not related to the ability of breaking the initial symmetry. In fact in CCS, π_{I} or $\mathrm{MA}^{\mathsf{io}}$, leader election problems can be solved in fully connected networks. The separation results say something regarding the possibility of creating new shared resources. In the π-calculus this phenomenon is present since channels can be values as well; in MA, SA and PAC this phenomenon is simulated via the open capability. In BA this phenomenon is simulated via the parent-child communication primitives. It turns out that CCS, π_{I}, boxed MA, boxed SA, boxed PAC and pure BA do not admit a solution to the leader election problem in rings, at least those of composite (non-prime) size.

As in the case of general networks, the proofs for the negative results differ in their technical details in each formalism, but there is a common strategy. If a ring is of composite size, then it is symmetric with respect to a permutation with multiple independent orbits of the same size (greater than one). The basic idea is to show that there is a maximal computation where, even though symmetry may be broken in the ring as a whole, symmetry is maintained within each orbit, and the nodes of each orbit remain independent. It remains an open problem whether the result presented below still holds in networks whose size is a prime number greater than three.

Theorem 5.14 *[26,30] For any composite $k > 1$, CCS does not have a symmetric ring of size k which is an electoral system. Similarly for π_{I}, boxed*

MA, boxed PAC and boxed SA. □

5.4 Separation Results

By Lemma 5.5, we can now deduce that there can be no uniform, observation-respecting and independence-preserving encoding from π_{m} into CCS, since the former has a symmetric electoral system which is a ring of size four (from Theorem 5.7) and the latter does not (Theorem 5.14).

Much as in Section 4.3, we can tabulate the results of Sections 5.2 and 5.3 as follows:

$\pi_{\mathsf{m}}^{-\nu}$	pure public MA	pure public PAC	pure public SA	public BA
CCS	π_{l}	boxed MA	boxed PAC	boxed SA

All calculi above the line have symmetric electoral systems which are rings for any finite size. Those below the line do not have symmetric electoral systems which are rings for composite sizes greater than three. Therefore there is no uniform, observation-respecting and independence-preserving encoding from any calculus above the line to any below the line.

As in Section 4.3 we have considered here the particular versions of calculi which which yield the strongest result. Each of the calculi above the line is the smallest fragment that solves leader election problems in symmetric rings; that of course implies that the full calculus does as well. On the other hand, for the calculi below the line, we have considered the largest fragment that does not admit a solution to the problem to make the results as strong as possible. The diagram above highlights which operators in each calculus are the key to the difference in expressiveness.

Our results shed light on the expressive power provided by creating new visible channels. CCS cannot solve leader election problems in rings because it does not allow the creation of new links. On the other hand, π_{l} can create new links, but they are bound and hence cannot be used outside the scope of the process. In the case of MA, PAC and SA, new links between processes in a network could be created with a process like $(x).x[\,P\,]$. However, such top-level anonymous communication does not give us enough control over which process receives which message, and it is easy to see that symmetry need not be broken. Instead, in our election algorithms we pass around new links encased in ambients. In order to use these links for interaction between processes they need to be brought to the top level; this "unleashing" is achieved using the open capability. Thus, our method says nothing about separation results between

MA with communication primitives and pure MA. In this framework one could regard them as equally expressive, since, when it comes to passing names around, pure MA can do just as well as the full calculus.

In connection with the negative result for boxed SA (Theorem 5.14), we recall that Zimmer [39] has encoded the synchronous choice-free π-calculus into pure SA. We conjecture that for boxed SA such an encoding would not be possible, even in the presence of communication. For if it were possible, then it would seem that boxed SA *could* perform election on rings, much as shown for BA (Theorem 5.8).

Observing that the algorithm devised in the previous section for PAC does not require knowledge of the size of the network, a challenge for the future is to show that calculi other than PAC either have, or cannot have, such uniform solutions, as well as exploiting any differences to obtain further separation results.

6 Discussion on approaches to expressiveness

In this section we wish to discuss and analyse other methods used in order to compare concurrent calculi. The power to solve the leader election problem is not the only way to differentiate computational models. Moreover, by using different criteria one may well obtain different hierarchies. For instance, in [25] it was shown that there exists an encoding from the separate choice π-calculus into the asynchronous π-calculus that respects weak bisimulation; however in [6] it was shown that this result does not hold if the must semantics is considered.

In this section we will review the assumptions used in our work, and analyse the strength and the weakness of other methods. We claim that whether or not expressiveness results are meaningful depends on both the criteria and the methods used to achieve them, and ultimately on the purpose of the results.

In this work, we have taken the point of view that calculi can be differentiated by their ability to solve increasingly harder problems. In particular we have considered leader election problems in symmetric networks for both cliques and rings. These are classic problems in the literature of distributed algorithms, and they concern the ability to reach agreement in fully distributed environments with different topologies of network. We know from the work in distributed algorithms [1,17] that calculi that solve leader election problems in symmetric networks cannot be implemented in a fully distributed way. Therefore our work shows that the mixed choice operator in π-calculus and CCS, as well as the in capability in ambient calculi, cannot be implemented in a fully

distributed fashion if the parallel composition operator | is interpreted as distributed parallelism. On the other hand, with calculi, such as the asynchronous π-calculus, that cannot perform leader election in symmetric networks, there is the prospect that they can be implemented in a fully distributed fashion. So our results shed light on a significant practical difference among models of computation.

An alternative approach could have been to consider a different problem, or to invent a problem from scratch, as done in [7]. In that paper, in order to show that there is no divergence-free encoding from the π-calculus with polyadic synchronisation to the standard π-calculus, a "matching" problem was specially devised. Although there is in principle nothing wrong in devising a new problem, one question is whether or not the problem is 'reasonable' or general enough. Also conditions associated to the encoding depend on the problem in hand, and whether those are also reasonable or acceptable remains to be decided in each individual case.

A completely different approach would be to separate calculi just on the basis of the semantic differences—i.e. without considering a problem. This means that one could decide that there are some properties that an encoding has to satisfy, and systematically work out which calculi can or cannot be related by such encodings. This the approach taken for example by Gorla [12]. When adopting this methodology, one always has to make arguments to establish the usefulness of the conditions considered; they should not be merely devised ad hoc to obtain a particular separation result.

In fact, with a strong enough set of conditions, one can always separate two given calculi. As an example, consider the asynchronous π-calculus and CCS. Everyone expects that the former cannot be encoded in the latter, since the π-calculus, unlike CCS, is able to receive names as values (object position) and then use them as channels (subject position). A simple-minded argument could involve defining the set of ports of a process to be those names occurring free in subject position, and requiring that no new ports can be created by the encoding. As an example, if we let $R \stackrel{\text{df}}{=} a(x).\overline{x}\langle m\rangle \mid \overline{a}\langle b\rangle$, then R has the single port a. So the CCS encoding $[\![R]\!]$ can have no port other than a by our condition. If we then impose the further condition on encodings that weak barbs are preserved (if $P \Downarrow n$ then $[\![P]\!] \Downarrow n$), then we easily get an impossibility result: clearly $R \rightarrow \overline{b}\langle m\rangle$ and so $R \Downarrow b$. However, the CCS process $[\![R]\!]$ cannot create any new ports and so $[\![R]\!] \not\Downarrow b$. But of course this result is open to objections of both a technical and more intuitive nature. Technically it mixes a notion related to strong barbs (i.e. ports) with weak barbs. From an intuitive point of view it seems to use too directly the fact that CCS cannot create new ports while the π-calculus can.

To sum up, we contend that there can be dangers in using arbitrary properties

of encodings in an ad hoc way, and that, by contrast, there are advantages in using properties designed around real computational problems such as leader election.

In Section 2.3 we observed that there is no single set of conditions that defines whether or not an encoding is good. In a similar way, perhaps it is not possible to define a unique method to separate calculi or models of computations. Each criterion is good enough for the purpose at hand.

7 Conclusions and Related Work

The first attempt to represent leader election problems in process algebra was made by Bougé [3]. He formalised the notion of the leader election problem in symmetric networks for CSP [13,14]. The most remarkable achievements are the separation results between CSP with input and output guards and CSP with input guards only, and between the latter and CSP without guards, based on the notion of *symmetric reasonable implementation*.

A similar formalisation of the notion of leader election problem was made by Palamidessi [26] for the π-calculus. Palamidessi proves *formally* that any symmetric network in the π-calculus with separate choice admits a computation that never breaks the initial symmetry. This result is used to show that there is no encoding of the π-calculus with mixed choice into the π-calculus with separate choice. In her paper Palamidessi uses a graph framework, as in the tradition of distributed algorithms [17,36,1,3], and she proves that CCS does not admit a symmetric electoral system in a ring, as opposed to the π-calculus with mixed choice. Using a similar approach, Ene and Muntean [11] show that the π-calculus with broadcasting primitives cannot be encoded in the standard π-calculus.

Finally, Phillips and Vigliotti used these proof techniques to separate MA from the separate-choice π-calculus and MA without the in capability ($\mathrm{MA}^{-\mathsf{in}}$) [28], and to separate mixed choice π-calculus and MA from CCS and MA without the open capability (boxed MA) [30]. This work was carried out in the reduction semantics framework also used in this tutorial. This framework has the advantage of uniformity across a range of process calculi. Our results say nothing, with respect to leader election, on the relationship between the mixed choice π-calculus and MA, or between CCS and boxed MA. These are still open problems.

In this tutorial we have collected together results from different papers [26,28,30], given a uniform presentation and highlighted the similarities and differences between the various approaches to leader election problems. We have omitted

proofs and lengthy details; however, those are available in the original papers.

Acknowledgements

The work of Catuscia Palamidessi and Maria Grazia Vigliotti has been partially supported by the INRIA/ARC project ProNoBiS. Maria Grazia Vigliotti thanks the Group MIMOSA at INRIA Sophia Antipolis for having allowed her to work as guest at their site. We thank the anonymous referees for their very useful and detailed comments.

References

[1] D. Angluin. Local and global properties in networks of processors. In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing*, pages 82–93. ACM, 1980.

[2] G. Boudol. Asynchrony and the π-calculus. Technical Report 1702, INRIA Sophia-Antipolis, 1992.

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

[4] M. Bugliesi, G. Castagna, and S. Crafa. Boxed ambients. In *Proceedings of 4th International Symposium on Theoretical Aspects of Computer Software (TACS'01)*, volume 2215 of *Lecture Notes in Computer Science*, pages 38–63. Springer-Verlag, 2001.

[5] M. Bugliesi, G. Castagna, and S. Crafa. Access control for mobile agents: the calculus of Boxed Ambients. *ACM Transactions on Programming Languages and Systems*, 26(1):57–124, January 2004.

[6] D. Cacciagrano, F. Corradini, and C. Palamidessi. Separation of synchronous and asynchronous communication via testing. *Theoretical Computer Science*. To appear.

[7] M. Carbone and S. Maffeis. On the Expressive Power of Polyadic Synchronisation in π-calculus. *Nordic Journal of Computing*, 10(2):70–98, 2003.

[8] L. Cardelli and A.D. Gordon. Anytime, Anywhere: Modal Logic for Mobile Ambients. In *Proceedings of the 27th ACM Symposium on Principles of Programming Languages*, pages 365–377, 2000.

[9] L. Cardelli and A.D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.

[10] G. Castagna, J. Vitek, and F. Zappa Nardelli. The Seal calculus. *Information and Computation*, 201(1):1–54, 2005.

[11] C. Ene and T. Muntean. Expressiveness of point-to-point versus broadcast communications. In *Proceedings of 12th International Symposium on Fundamentals of Computation Theory (FCT'99)*, volume 1684 of *Lecture Notes in Computer Science*, pages 258–268. Springer-Verlag, 1999.

[12] D. Gorla. On the relative expressive power of asynchronous communication primitives. In L. Aceto and A. Ingólfsdóttir, editors, *Proceedings of 9th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'06)*, volume 3921 of *Lecture Notes in Computer Science*, pages 47–62. Springer-Verlag, 2006.

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

[14] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[15] K. Honda and M. Tokoro. An object calculus for asynchronous communication. In *Proceedings of European Conference on Object-Oriented Programming (ECOOP'91)*, volume 512 of *Lecture Notes in Computer Science*, pages 133–147. Springer-Verlag, 1991.

[16] F. Levi and D. Sangiorgi. Mobile safe ambients. *ACM Transactions on Programming Languages and Systems*, 25(1):1–69, 2003.

[17] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[18] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[19] R. Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2(2):269–310, 1992.

[20] R. Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.

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

[22] R. Milner and D. Sangiorgi. Barbed Bisimulation. In *Proceedings of the 19th International Colloquium on Automata, Languages and Programming*, volume 623 of *Lecture Notes in Computer Science*, pages 685–695. Springer-Verlag, 1992.

[23] U. Nestmann. What is a 'good' encoding of guarded choice? *Information and Computation*, 156:287–319, 2000.

[24] U. Nestmann, R. Fuzzati, and M. Merro. Modeling consensus in a process calculus. In *Proceedings of 14th International Conference on Concurrency Theory (CONCUR'03)*, volume 2761 of *Lecture Notes in Computer Science*, pages 393–407. Springer-Verlag, 2003.

[25] U. Nestmann and B.C. Pierce. Decoding Choice Encodings. *Information and Computation*, 163(1):1–59, 2000.

[26] C. Palamidessi. Comparing the expressive power of the synchronous and the asynchronous π-calculi. *Mathematical Structures in Computer Science*, 13(5):685–719, 2003.

[27] I.C.C. Phillips. CCS with priority guards. In *Proceedings of 12th International Conference on Concurrency Theory (CONCUR'01)*, volume 2154 of *Lecture Notes in Computer Science*, pages 305–320. Springer-Verlag, 2001.

[28] I.C.C. Phillips and M.G. Vigliotti. Symmetric electoral systems for ambient calculi. *Information & Computation*. Accepted.

[29] I.C.C. Phillips and M.G. Vigliotti. On reduction semantics for the Push and Pull Ambient Calculus. In *Proceedings of IFIP International Conference on Theoretical Computer Science (TCS 2002)*, pages 550–562. Kluwer, 2002.

[30] I.C.C. Phillips and M.G. Vigliotti. Leader election in rings of ambient processes. *Theoretical Computer Science*, 356(3):468–494, 2006.

[31] B.C. Pierce and D.N. Turner. Pict: A programming language based on the pi-calculus. In G. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*, pages 455–494. MIT Press, 2000.

[32] K.V.S. Prasad. Broadcast Calculus Interpreted in CCS up to Bisimulation. In *Proceedings of 8th International Workshop on Expressiveness on Concurrency (EXPRESS'01)*, volume 52 of *Electronic Notes in Theoretical Computer Science*, pages 83–100. Elsevier, 2002.

[33] D. Sangiorgi. *Expressing Mobility in Process Algebra: First-Order and Higher-Order Paradigms*. PhD thesis, University of Edinburgh, 1993.

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

[35] D. Sangiorgi and D. Walker. *The π-Calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

[36] G. Tel. *Distributed Algorithms*. Cambridge University Press, 2000.

[37] M.G. Vigliotti. *Reduction Semantics for Ambient Calculi*. PhD thesis, Imperial College London, 2004.

[38] N. Yoshida. Graph Types for Monadic Mobile Processes. In *Proceedings of 16th Conference on Foundations of Software Technology and Theoretical Computer Science (FST/TCS)*, volume 1180 of *Lecture Notes in Computer Science*, pages 371–386. Springer-Verlag, 1996.

[39] P. Zimmer. On the expressiveness of pure Safe Ambients. *Mathematical Structures in Computer Science*, 13(5):721–770, 2003.