Math. Struct. in Comp. Science (2009), *vol.* 19, *pp.* 1161–1189.
doi:10.1017/S0960129509990168

An expressiveness study of priority in process calculi

CRISTIAN VERSARI, NADIA BUSI and ROBERTO GORRIERI

Università di Bologna, Dipartimento di Scienze dell'Informazione
Mura Anteo Zamboni 7, 40127 Bologna, Italy
Email: {versari,gorrieri}@cs.unibo.it

Received 20 Mar 2008; revised 26 November 2008

In memory of Nadia Busi

Priority is a frequently used feature of many computational systems. In this paper we study the expressiveness of two process algebras enriched with different priority mechanisms. In particular, we consider a finite (that is, recursion-free) fragment of asynchronous CCS with global priority (FAP, for short) and Phillips' CPG (CCS with local priority), and contrast their expressive power with that of two non-prioritised calculi, namely the π-calculus and its broadcast-based version, called bπ. We prove, by means of leader-election-based separation results, that, under certain conditions, there exists no encoding of FAP in π-Calculus or CPG. Moreover, we single out another problem in distributed computing, which we call the *last man standing* problem (LMS for short), that better reveals the gap between the two prioritised calculi above and the two non-prioritised ones, by proving that there exists no parallel-preserving encoding of the prioritised calculi in the non-prioritised calculi retaining any *sincere* (complete but partially correct, that is, admitting divergence or premature termination) semantics.

1. Introduction

Priority is a frequently used feature of many computational systems. High-priority processes use more central processing unit time in workstations, or preempt the execution of low priority processes through hardware/software-driven interrupt mechanisms. In order to model such systems, many basic process algebras have been enriched with some priority mechanisms (see, for example, Baeten *et al.* (1987), Camilleri and Winskel (1995), Cleaveland and Hennessy (1990), Cleaveland *et al.* (2001) and Phillips (2001)). Priority is also implicitly used in many stochastic process calculi, where immediate actions take precedence over timed actions (see, for example, Bernardo and Gorrieri (1996), Hermanns (2002) and Bravetti and Gorrieri (2002)), or where actions are equipped with an explicit priority level (see, for example, Bernardo and Gorrieri (1998)).

In this paper (which is an extended and revised version of Versari *et al.* (2007)) we study a couple of problems in distributed systems to investigate the expressiveness of priority in (untimed) concurrent systems in order to delineate the expressive power gained by the addition of priority, and to compare the relative expressive power of different priority mechanisms.

According to the classification in Cleaveland *et al.* (2001), the basic priority mechanisms reported in the literature can be divided into two main groups: those based on *global*

priority (see, for example, Baeten *et al.* (1986) and Cleaveland and Hennessy (1990)) and those based on *local* priority (see, for example, Camilleri and Winskel (1995) and Phillips (2001)). This distinction is motivated by the *scope* of the priority effects on the system. In the case of global priority, a high-priority action is able to preempt any other low-priority action in the system, so that only higher priority processes are allowed to evolve. In the case of local priority, this effect is limited to the *location* of the process, where a location can be thought as a site on a distributed system and may be represented by the scope of some name or the guarded choice between actions with different priorities. An example may help to show the difference between the two. Consider the system S composed of five processes

$$S \triangleq \overline{a}.P \mid a.Q_1 + \underline{b}.Q_2 \mid \overline{\underline{b}}.R \mid c.T_1 + d.T_2 \mid \overline{c}.V$$

where the sum operator represents the usual choice between different actions, output actions are overlined and high-priority actions are underlined. According to the semantics of CCS^{sg} (CCS with a global notion of priority) and CCS^{sl} (CCS with local priority) reported in Cleaveland *et al.* (2001), the processes $a.Q_1 + \underline{b}.Q_2$ and $\overline{\underline{b}}.R$ are ready to perform a high-priority action on channel $\underline{b}$. In CCS^{sg} semantics this action is forced to happen before any other low priority transition in S, while in CCS^{sl} semantics, the action $\underline{b}$ only preempts the execution of the action a, so that the synchronisation on c of the last two processes may even happen first. In other words, with a global priority notion the only possible internal transition of the system S is

$$S \to \overline{a}.P \mid Q_2 \mid R \mid c.T_1 + d.T_2 \mid \overline{c}.V$$

but with local priority, we can also have the evolution

$$S \to \overline{a}.P \mid a.Q_1 + \underline{b}.Q_2 \mid \overline{\underline{b}}.R \mid T_1 \mid V.$$

As a basic representative of a calculus with global priority, in this paper we will consider a very minimal fragment of CCS (Milner 1980) without restriction and recursion but with asynchronous communication, and enrich it with static priority and global preemption (as in CCS^{sg}, which is described in Cleaveland *et al.* (2001)) where only the prefix operator on inputs is present and the asynchronous output is characterised by the possibility of assigning different priorities to the outgoing messages – we call this calculus FAP.

As a representative of a calculus with local priority, we will consider Phillips' CCS with priority guards (CPG for short) (Phillips 2001).

Moreover, we will also consider two well-known unprioritised calculi, namely the π-calculus (Milner *et al.* 1992a; Milner *et al.* 1992b) and its broadcast-based version bπ-Calculus (Ene and Muntean 1999), which we will compare with the two prioritised calculi above.

The two problems in distributed systems we will use to distinguish the expressive power of these four calculi are the *leader-election* problem (Le Lann 1977), which has already been used to study the expressiveness gap between, for example, synchronous and asynchronous π-calculus (Palamidessi 2003), and an apparently new problem, which we have called *the last man standing* problem (LMS for short), which consists of the capability

https://doi.org/10.1017/S0960129509990168 Published online by Cambridge University Press

of processes recognising the absence of other processes ready to perform synchronisations or input/output operations. In other words, the LMS problem is solvable if a process is able to check that it is the only one active in a network.

1.1. *Related work*

The election of a unique leader in distributed systems often constitutes a preliminary step during reorganisation operations that require a common agreement between the nodes of the network. Its formalisation is attributed to Le Lann in Le Lann (1977), where the leader election preceded the generation of a new token in a ring network after the loss of the previous one. This problem was considered for the first time in a process algebra framework in Bougé (1988), where it was used to study the expressive power of CSP (Hoare 1985) in relation to its communication primitives and the topology of the network. The first application of the leader election problem to the π-Calculus revealed the superior expressiveness of mixed-choice compared with separate choice (Palamidessi 1997; 2003): the relative expressive power of the two constructs is stated in terms of the impossibility of a *reasonable* encoding of mixed into separate choice, where the reasonableness is represented by the uniformity (preservation of the parallelism and symmetry of the system) of the encoding function and the preservation of the observable actions performed by the encoded processes. Many other separation results have been obtained using the same approach, each one based on the ability or inability to solve the leader election under appropriate conditions. A variant of the problem was used in Ene and Muntean (1999) to show that the broadcast-based bπ-Calculus is more expressive than the standard π-Calculus provided with point-to-point communication primitives. The result is based on the idea that broadcasting enables the leader election problem to be solved even when there is no knowledge of the number of processes participating to the election, while this is proved to be impossible in the π-Calculus.

The same approach was used in Phillips (2001; 2008) to separate CPG from CCS and from the π-Calculus by exploiting the broadcast-like power of preemption introduced by priority: as with the bπ-Calculus, it is shown that CPG can solve the leader election without any knowledge of the number of participants, while the ability of the π-Calculus to create new communication links between processes is exploited to prove the converse separation from CPG. The separation is based on the impossibility of finding a uniform encoding of CPG preserving a semantics that respects the observables of the encoded processes, and proves that local priority adds expressiveness with respect to both CCS and the π-Calculus. This is currently the only result on the expressiveness of priorities in process algebras that we are aware of.

A detailed survey of the leader election problem and the expressiveness separation results in process algebras can be found in Vigliotti *et al.* (2007).

1.2. *Contribution of this paper*

In this paper we first analyse the expressiveness of global priority, in order to check if it can be proved to be more expressive than local priority, as we would naturally expect

https://doi.org/10.1017/S0960129509990168 Published online by Cambridge University Press

it to be. As a representative of a global priority model, we choose FAP, a fragment of CCS augmented with stratified, global priority (a slight variant of the CCS with static priority and global preemption, CCS^{sg}, which was studied in Cleaveland *et al.* (2001)) where the only operators are the parallel composition of processes and the prefix on inputs, while the asynchronous output models the dispatch of messages with two different levels of priority: the delivery of high-priority messages is ensured to happen before that of low-priority ones.

We prove that this very simple language (deprived of synchronous communication, choice, recursion or replication, and hence finite) is sufficiently powerful to write programs capable of solving the leader election problem in any connected graph of processes without any knowledge of the number of processes involved in the election.

By applying the idea used in Ene and Muntean (1999) and Phillips (2001), we have as a corollary that FAP cannot be distributively encoded in the π-Calculus, but we prove that this also remains true for partially correct encodings, which introduce divergence or failure in their computations as consequences of livelocked or deadlocked conditions. This result can also be extended to the translations of bπ-Calculus and CPG in the π-Calculus, thus relaxing the encoding conditions previously stated in Ene and Muntean (1999) and Phillips (2001).

Another consequence of the above leader election result in FAP is the impossibility of encoding it in CPG under uniformity and independence-preserving conditions, which constitutes the expected result for the expressiveness gap between global and local priority in the chosen process algebra framework. It is worth considering that the separation between these two prioritised languages and the π-Calculus is stronger than that between FAP and CPG themselves, which is a first hint of the expressive power of priority for both global and local approaches.

In order to strengthen the separation between prioritised and non-prioritised languages, we then introduce a new setting, which we call the *last man standing problem*. In this setting, a bunch of n processes must realise if there is only one process participating in the LMS (and in that case, this process would be the 'last man standing'), that is, they must know if $n = 1$ or $n > 1$ in a distributed way. We prove that the LMS can be solved in both FAP and CPG (but we claim that it is also possible within other priority approaches like Camilleri and Winskel (1995) and Cleaveland *et al.* (2001)), but cannot be solved in non-prioritised languages like the π-Calculus, or even the bπ-Calculus. This result implies that there are no distributed encodings of FAP and CPG in the bπ-Calculus, and thus that the degree of expressiveness of priority does not derive from the broadcast-like power of preemption, but from the capability of processes knowing if another process is ready to perform a synchronisation on some channel *or not*. In non-prioritised calculi, it is possible to know if some process *is* ready to perform some synchronisation, but, on the contrary, knowing that this condition does not hold is not decidable. We show that in a distributed setting this simple capability is linked to priority (global or local, stratified or not) and cannot be obtained otherwise, even if we provide broadcast-like primitives and admit divergent or deadlocked computations.

https://doi.org/10.1017/S0960129509990168 Published online by Cambridge University Press

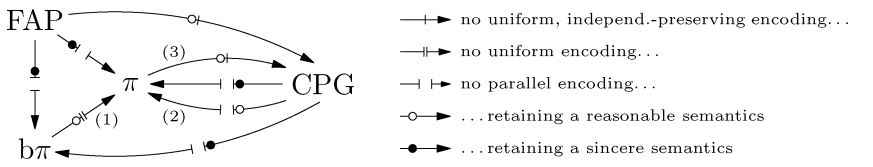

Fig. 1. Impossibility results: (1) C. Ene, T. Muntean; (2, 3) I. Phillips; the remaining ones are presented in this paper.

The above results are summarised in Figure 1.

1.3. *Structure of the paper*

The rest of the paper is structured as follows. In Section 2, we introduce the four process algebras involved in the separation results and give a brief explanation of their main features. Section 3 contains a short discussion and sets out the formal definitions of the properties of an encoding. In Sections 3.1 and 3.2, we formalise the leader election and LMS problems, and then show the separation results for them in Sections 4.1 and 4.2, respectively. Finally, we present some concluding remarks in Section 5.

2. Calculi

In this section we introduce the calculi of interest in this paper by giving their syntax and a short explanation of the way they operate. We will give a comprehensive definition of the FAP semantics, but due to lack of space, just refer to Milner (1993; 1999), Ene and Muntean (1999) and Phillips (2001; 2008) for the semantics of the other calculi.

2.1. *The π-Calculus*

The π-Calculus (Milner *et al.* 1992a; Milner *et al.* 1992b) is derived from CCS (Milner 1980) in which processes interact through synchronisation over named channels and have the ability to receive new channels and subsequently use them to interact with other processes in order to model mobility.

Definition 2.1. Let $\mathcal{N}$ be a set of names on a finite alphabet, $x, y, \ldots \in \mathcal{N}$. The syntax of the π-Calculus is defined as

$$\begin{array}{lcl} P & ::= & \mathbf{0} \;\mid\; \sum_{i\in I} \pi_i.P_i \;\mid\; P \mid Q \;\mid\; !P \;\mid\; (\nu x)P \\ \pi & ::= & \tau \;\mid\; x(y) \;\mid\; \overline{x}\langle y\rangle \end{array}$$

where:

— **0** represents the null process;
— $x(y)$ expresses the capability of performing an input on the channel x and receiving a datum that is then bound to the name y;

https://doi.org/10.1017/S0960129509990168 Published online by Cambridge University Press

— $\overline{x}\langle y\rangle$ expresses the capability of sending the name y on the channel x;

— τ is the invisible, uncontrollable action;

— $P \mid Q$ represents the parallel composition of processes;

— $!P$ stands for the unlimited replication of process P;

— $\sum_{i\in I} \pi_i.P_i$ represents the non-deterministic choice between several input/output communication capabilities, also denoted by $\pi_1.P_1 + \pi_2.P_2 + \cdots$;

— $(\nu x)P$ represents scope restriction of the name x to process P.

Definition 2.2. The congruence relation $\equiv$ on π-Calculus processes is defined as the least congruence satisfying alpha conversion, the commutative monoidal laws with respect to both $(|,\mathbf{0})$ and $(+,\mathbf{0})$ and the following axioms:

$$\begin{aligned} (\nu x)P \mid Q &\equiv (\nu x)(P \mid Q) && \text{if } x \notin \mathrm{fn}(Q) \\ (\nu x)P &\equiv P && \text{if } x \notin \mathrm{fn}(P) \\ !P &\equiv !P \mid P \end{aligned}$$

where the function fn is defined by

$$\begin{aligned} \mathrm{fn}(\tau) &\stackrel{def}{=} \varnothing \\ \mathrm{fn}(x(y)) &\stackrel{def}{=} \{x\} \\ \mathrm{fn}(\overline{x}\langle y\rangle) &\stackrel{def}{=} \{x, y\} \\ \mathrm{fn}(\mathbf{0}) &\stackrel{def}{=} \varnothing \\ \mathrm{fn}(\pi.P) &\stackrel{def}{=} \mathrm{fn}(\pi) \cup \mathrm{fn}(P) \\ \mathrm{fn}(\textstyle\sum_{i\in I} \pi_i.P_i) &\stackrel{def}{=} \bigcup_i \mathrm{fn}(\pi_i.P_i) \\ \mathrm{fn}(P \mid Q) &\stackrel{def}{=} \mathrm{fn}(P) \cup \mathrm{fn}(Q) \\ \mathrm{fn}(!P) &\stackrel{def}{=} \mathrm{fn}(P) \\ \mathrm{fn}((\nu x)P) &\stackrel{def}{=} \mathrm{fn}(P) \setminus \{x\}. \end{aligned}$$

In order to define the observables of a π-Calculus process P, we introduce the notion of barb.

Definition 2.3. Let P be a π-Calculus process. P exhibits barb π, written $P \downarrow \pi$, if and only if

— $P \equiv (\nu \tilde{y})(x(z).Q + R \mid S)$, with $\pi = x$, $x \notin \tilde{y}$; or
— $P \equiv (\nu \tilde{y})(\overline{x}\langle z\rangle.Q + R \mid S)$, with $\pi = \overline{x}$, $x \notin \tilde{y}$.

Each barb π represents one action that P is immediately ready to perform. Conversely, we write that $P \not\downarrow \pi$ if P does not exhibit barb π.

https://doi.org/10.1017/S0960129509990168 Published online by Cambridge University Press

Definition 2.4. π-Calculus semantics is given in terms of the reduction relation $\rightarrow$ described by the rules

$$\frac{}{\tau.P \;\rightarrow\; P}$$

$$\frac{}{(x(y).P + M) \mid (\overline{x}\langle z\rangle.Q + N) \;\rightarrow\; P\{z/y\} \mid Q}$$

$$\frac{P \;\rightarrow\; P'}{P \mid Q \;\rightarrow\; P' \mid Q}$$

$$\frac{P \;\rightarrow\; P'}{(\nu\, x)P \;\rightarrow\; (\nu\, x)P'}$$

$$\frac{P \equiv Q \quad P \;\rightarrow\; P' \quad P' \equiv Q'}{Q \;\rightarrow\; Q'}.$$

The fact that the λ-calculus can be encoded in the π-Calculus (Milner 1990) implies the Turing completeness of the π-Calculus. As an alternative proof, we provide the encoding of a very simple class of Random Access Machines (RAMs), which was shown to be Turing complete in Minsky (1967). Each RAM is composed of a finite set of registers $r_1, \ldots, r_n$ holding arbitrary large natural numbers and a finite set of indexed instructions $(1 : I_1), \ldots, (m : I_m)$, which represent the program executed by the RAM. Each instruction can have one of two forms:

— $(i : Inc(r_j))$: increment by 1 the contents of the register r_j and execute the next instruction; or
— $(i : DecJump(r_j, s))$: if the contents of the register r_j is not zero, decrease it by 1 and execute the next instruction, otherwise jump to instruction number s.

The internal state of a RAM is described by a configuration $(i, c_1, \ldots, c_n)$ where i is the program counter, indicating the next instruction, and $c_1, \ldots, c_n$ are the values stored in the registers.

Definition 2.5. A *Random Access Machine* (RAM) R is defined as a pair $R = (I, n)$, where n is the number of registers of R and

$$I = \{(1 : I_1), \ldots, (m : I_m)\}$$

is the set of instructions of R, with $|I| = m$ and each instruction I_i has the form

— $I_i = Inc(r_j) \quad (1 \leqslant j \leqslant n)$; or
— $I_i = DecJump(r_j, s)) \quad (1 \leqslant j \leqslant n)$.

An internal configuration C of R is defined as a state vector

$$C = (i, c_1, \ldots, c_n)$$

with i representing the program counter and $c_1, \ldots, c_n \in \mathbb{N}$ the current values stored in the n registers of R.

https://doi.org/10.1017/S0960129509990168 Published online by Cambridge University Press

The transition function $\rightsquigarrow_R : \mathbf{C} \to \mathbf{C}$ over the set of configurations $\mathbf{C}$ of a RAM R is defined as

$$(i, c_1, \ldots, c_n) \quad \rightsquigarrow_R \quad (i', c'_1, \ldots, c'_n)$$

if

— $I_i = Inc(r_j)$ and $i' = i + 1$, $c'_j = c_j + 1$, $c'_p = c_p$ for $p \neq j$;
— $I_i = DecJump(r_j, s)$, $c_j > 0$ and $i' = i + 1$, $c'_j = c_j - 1$, $c'_p = c_p$ for $p \neq j$;
— $I_i = DecJump(r_j, s)$, $c_j = c'_j = 0$ and $i' = s$, $c'_p = c_p$ for $p \neq j$.

Proposition 2.1. The π-Calculus is Turing complete.

Proof. The main program of a RAM R can be encoded as a process M (specified by means of the processes $M_1, \ldots, M_m$, one for every instruction of the RAM), and each register r_j as an independent process R_j, whose internal state is related to the value stored in the register itself. The main program M interacts with each process register R_j over a small set of channels t_j, z_j, n_j, inc_j used to:

— test the state of the register (output from M to R_j on t_j), which can be zero (corresponding to an incoming answer on z_j) or non-zero (answer on n_j), and in the latter case the register is decremented;
— increment the register (output on inc_j).

Each instruction $(i : I_i)$ corresponds to the definition of a replicated process M_i, which is spawned by an output on the channel m_i:

— in the presence of an increment operation $(i : Inc(r_j))$,

$$M_i \equiv !m_i.\overline{inc}_j.inc_j.\overline{m}_{i+1}$$

— in the presence of a 'jump if zero/decrement' operation $(i : DecJump(r_j, s))$,

$$M_i \equiv !m_i.\bar{t}_j.(z_j.\overline{m}_s + n_j.\overline{m}_{i+1}).$$

Each register R_j is encoded as a stack of processes whose length corresponds to the value stored in the register itself. The first process in the stack reacts to the instructions given by the main process, according to the following definitions:

$$\begin{aligned} Z_j &\equiv t_j.\overline{zloop_j}.\bar{z}_j + inc_j.(va)\ \left(\overline{nloop_j}\langle a\rangle.\overline{inc_j} \mid a.\overline{zloop_j}.\bar{a}\right) \\ ZL_j &\equiv !zloop_j.Z_j \end{aligned}$$

The recursive behaviour of Z_j is achieved by guarded replication on $zloop_j$ in the usual way. The first branch of the choice answers a possible query by sending one output on z_j, which signals the value zero stored in the register. One output on inc_j spawns a new process N_j (defined below) linked to a process Z_j, which represents a queue of length one. The process N_j is defined as follows:

$$\begin{aligned} N_j(a) &\equiv t_j.\bar{a}.a.\bar{n}_j + inc.(va')\ \left(\overline{nloop_j}\langle a'\rangle.\overline{inc_j} \mid a'.\overline{nloop_j}\langle a\rangle.\overline{a'}\right) \\ NL_j &\equiv !nloop_j(a).N_j(a). \end{aligned}$$

As with Z_j, each process N_j listens on t_j in order to answer on n_j that the value stored in the register is greater than zero, then the register is decremented by sending an output

https://doi.org/10.1017/S0960129509990168 Published online by Cambridge University Press

on a, which activates the next process in the stack. The increment operation is exactly the same as for Z_j.

Finally, the internal state $(i, c_1, \ldots, c_n)$ of a RAM R is encoded as

$$\left\{\!\left|\, (i, c_1, \ldots, c_n) \,\right|\!\right\}_R \quad = \quad \overline{m}_i \mid R_1^{c_1} \mid \cdots \mid R_n^{c_n} \mid ZL_1 \mid NL_1 \mid \cdots \mid ZL_n \mid NL_n$$

where

$$\begin{aligned} R_j^0 &\triangleq Z_j \\ R_j^k &\triangleq (\nu a_1, \ldots, a_k)\,\big(N_j(a_k) \mid a_k.\overline{nloop}\langle a_{k-1}\rangle.\overline{a_k} \mid \\ &\qquad\qquad \cdots \qquad\qquad \mid \\ &\qquad\qquad a_2.\overline{nloop_j}\langle a_1\rangle.\overline{a_2} \mid \\ &\qquad\qquad a_1.\overline{zloop_j}.\overline{a_1}\,\big) \qquad\qquad (k > 0). \end{aligned}$$

We have that for each configuration $\mathscr{C}'$ of a RAM R immediately reachable from $\mathscr{C}$, that is $\mathscr{C} \rightsquigarrow_R \mathscr{C}'$, there exist a sequence of reductions $M_1 \to \cdots \to M_p$ between their corresponding encodings and *vice versa*, so

$$\mathscr{C} \rightsquigarrow_R \mathscr{C}' \qquad \Longleftrightarrow \qquad \left\{\!\left|\, \mathscr{C} \,\right|\!\right\}_R \equiv M_1 \to \cdots \to M_p \equiv \left\{\!\left|\, \mathscr{C}' \,\right|\!\right\}_R \qquad (p < 8).$$

Furthermore, the encoding fully preserves the determinism of the RAM (in fact, only one reduction is possible for each encoded configuration $\{\!|\, \mathscr{C} \,|\!\}$ and for each of the above intermediate steps M_i between two encoded configurations) and, consequently, it also preserves its divergence. □

For a full treatment of the π-Calculus see Milner (1993; 1999).

2.2. *The bπ-Calculus*

The bπ-Calculus (Ene and Muntean 1999) is a variant of the π-Calculus where the point-to-point synchronisation mechanism is replaced by broadcast communication. For example, while the π-Calculus program

$$S \triangleq \overline{a}\langle b\rangle.P \mid a(x).Q \mid a(y).R \mid a(z).T$$

can evolve in one step to a system like S_1

$$S \to S_1 \triangleq P \mid Q\{b/x\} \mid a(y).R \mid a(z).T$$

where only one of Q, R, S is affected by the communication performed, in bπ-Calculus the system S evolves directly to S_2

$$S \to S_2 \triangleq P \mid Q\{b/x\} \mid R\{b/y\} \mid T\{b/z\}$$

where all the processes listening on channel a receive the broadcasted message.

So that we can give a uniform presentation of the languages analysed in this paper, we will introduce a variant of the bπ-Calculus defined in terms of reduction semantics, instead of the labelled transition system used in Ene and Muntean (1999).

https://doi.org/10.1017/S0960129509990168 Published online by Cambridge University Press

Definition 2.6. Let $\mathcal{N}$ be a set of names on a finite alphabet, $x, y, \ldots \in \mathcal{N}$. The syntax of the b$\pi$-Calculus is defined in terms of the grammar

$$P \quad ::= \quad \mathbf{0} \;\Big|\; A\langle\tilde{x}\rangle \;\Big|\; \sum_{i\in I} \pi_i.P_i \;\Big|\; P_1 \mid P_2 \;\Big|\; (\nu x)P$$

where

$$\pi_i \quad ::= \quad \tau \;\Big|\; x(y) \;\Big|\; \overline{x}\langle y\rangle$$

and each constant A is assumed to have a unique defining equation $A(\tilde{x}) \triangleq P$.

Definition 2.7. The congruence relation $\equiv$ on bπ-Calculus processes is defined as the least congruence satisfying alpha conversion, the commutative monoidal laws with respect to both $(|,\mathbf{0})$ and $(+,\mathbf{0})$ and the axioms

$$\begin{aligned} (\nu x)P \mid Q &\equiv (\nu x)(P \mid Q) && \text{if } x \notin \mathrm{fn}(Q) \\ (\nu x)P &\equiv P && \text{if } x \notin \mathrm{fn}(P) \\ A\langle\tilde{b}\rangle &\equiv P\{\tilde{b}/\tilde{a}\} \text{ if } A(\tilde{a}) \stackrel{def}{=} P \end{aligned}$$

where the function fn is defined by

$$\begin{aligned} \mathrm{fn}(\tau) &\stackrel{def}{=} \varnothing \\ \mathrm{fn}(x(y)) &\stackrel{def}{=} \{x\} \\ \mathrm{fn}(\overline{x}\langle y\rangle) &\stackrel{def}{=} \{x, y\} \\ \mathrm{fn}(\mathbf{0}) &\stackrel{def}{=} \varnothing \\ \mathrm{fn}(\pi.P) &\stackrel{def}{=} \mathrm{fn}(\pi) \cup \mathrm{fn}(P) \\ \mathrm{fn}(\textstyle\sum_{i\in I} \pi_i.P_i) &\stackrel{def}{=} \bigcup_i \mathrm{fn}(\pi_i.P_i) \\ \mathrm{fn}(P \mid Q) &\stackrel{def}{=} \mathrm{fn}(P) \cup \mathrm{fn}(Q) \\ \mathrm{fn}(A\langle\tilde{b}\rangle) &\stackrel{def}{=} \{\tilde{b}\} \\ \mathrm{fn}((\nu x)P) &\stackrel{def}{=} \mathrm{fn}(P) \setminus \{x\} \ . \end{aligned}$$

The definition of barb for the bπ-Calculus follows – it is the same as for the π-Calculus.

Definition 2.8. Let P be a bπ-Calculus process. P exhibits barb π, written $P \downarrow \pi$, if and only if

— $P \equiv (\nu\tilde{y})(x(z).Q + R \mid S)$, with $\pi = x$, $x \notin \tilde{y}$; or
— $P \equiv (\nu\tilde{y})(\overline{x}\langle z\rangle.Q + R \mid S)$, with $\pi = \overline{x}$, $x \notin \tilde{y}$.

https://doi.org/10.1017/S0960129509990168 Published online by Cambridge University Press

Definition 2.9. The bπ-Calculus semantics is given in terms of the reduction relation $\rightarrow$ described by the rules

$$\frac{R \downarrow \mu}{\prod_i(\mu(y_i).P_i + M_i) \mid (\overline{\mu}\langle z\rangle.Q + N) \mid R \quad \rightarrow \quad \prod_i P_i\{z/y_i\} \mid Q \mid R}$$

$$\frac{}{\tau.P \rightarrow P}$$

$$\frac{P \rightarrow P'}{(\nu\ x)P \rightarrow (\nu\ x)P'}$$

$$\frac{P \equiv Q \quad P \rightarrow P' \quad P' \equiv Q'}{Q \rightarrow Q'}$$

where $\prod$ represents the parallel composition of zero or more processes,

$$\prod_{i=1}^{n} P_i \quad = \quad P_1 \mid \cdots \mid P_n$$

for some $n \geqslant 1$, while for $n = 0$

$$\prod_i P_i \quad = \quad \mathbf{0}.$$

Proposition 2.2. The bπ-Calculus is Turing complete.

Proof. The encoding of a RAM into the bπ-Calculus is similar to the one given for the π-Calculus, and is shown in Figure 2. □

The encoding does not exploit the broadcast capabilities of the bπ-Calculus, but carefully forces every output to be received by at most one process. Furthermore, since the one-to-many communication of the bπ-Calculus allows the output message to be lost if some other process is not listening on the right channel at the right time (in contrast with the binary synchronisation of the π-Calculus, which requires a previous handshake between the processes involved), such an encoding ensures the correctness of the computation by forcing each input to be exhibited *before* some output may be performed on the corresponding channel.

2.3. *The CPG language*

The CPG language (Phillips 2001) is derived from CCS by extending it with a local notion of priority over actions. CPG fully embeds (syntactically and semantically) CCS so that any CCS program is still valid and is denoted by the same semantics, but, in addition, each action can be guarded by a set of names representing the actions whose co-action availability may prevent its execution. For example, in the system S,

$$S \triangleq \overline{a} : b.Q \mid \overline{b}.R,$$

the first action b is guarded by $\overline{a}$, so its execution is prevented by the presence of any parallel complementary action a. Since there is no such co-action in S, it can undergo the reduction

$$S \quad \rightarrow \quad Q \mid R.$$

https://doi.org/10.1017/S0960129509990168 Published online by Cambridge University Press

$$\begin{aligned}
\{\!|\, (i, c_1, \ldots, c_n) \,|\!\}_R &= M_i^R \mid R_1^{c_1} \mid \cdots \mid R_n^{c_n} \\
M_i^R &\triangleq \{\!|\, (i : I_i) \,|\!\}_R \quad \text{with } R = (I, n) \text{ and } (i : I_i) \in I \\
\{\!|\, (i : Inc(r_j)) \,|\!\}_R &= \overline{inc}_j.M_{i+1}^R \\
\{\!|\, (i : DecJump(r_j, s)) \,|\!\}_R &= \overline{t}_j.(z_j.M_s^R + n_j.M_{i+1}^R) \\
R_j^0 &\triangleq Z_j \\
R_j^k &\triangleq (\nu a)\big(N_j\langle a\rangle \mid a.R_j^{k-1}\big) \qquad (k > 0) \\
Z_j &\triangleq t_j.\overline{z}_j.Z_j \;+\; inc_j.(\nu a)\,\big(N_j\langle a\rangle \mid a.Z_j\big) \\
N_j(a) &\triangleq t_j.\overline{a}.\overline{n}_j \;+\; inc_j.(\nu a')\,\big(N_j\langle a'\rangle \mid a'.N_j\langle a\rangle\big)
\end{aligned}$$

Fig. 2. Definition of the function $\{\!|\, \cdot \,|\!\}$ for the encoding of RAMs into bπ-Calculus and CPG. Although different, the two calculi share a common core that is very close to CCS, and this allows us to use the same encoding function for this kind of RAM.

The system S',

$$S' \triangleq a.P \mid S,$$

cannot reduce in the same way, because of the presence of the action a in $a.P$. The locality of this effect can be oberved in the system

$$S'' \triangleq a.P \mid \overline{a} : b.Q \mid \overline{b}.R \mid c : b.T$$

where the following reduction over b is possible:

$$S'' \quad \rightarrow \quad a.P \mid \overline{a} : b.Q \mid R \mid T.$$

In fact, the process $\overline{a} : b.Q$ is still stuck in the presence of the action a, while $\overline{b}.R$ is free to synchronise with $c : b.T$, since the co-action $\overline{c}$, which would prevent the reduction with the last process, is absent.

Definition 2.10. Let $\mathcal{N}$ be a set of names on a finite alphabet, $x, y, \ldots \in \mathcal{N}$. The CPG syntax is defined in terms of the grammar

$$P \quad ::= \quad \mathbf{0} \;\Big|\; A\langle\tilde{x}\rangle \;\Big|\; \sum\nolimits_{i\in I} S_i : \alpha_i.P_i \;\Big|\; P_1 \mid P_2 \;\Big|\; (\nu x)P$$

where

$$\alpha_i \quad ::= \quad x \;\Big|\; \overline{x} \;\Big|\; \tau,$$

and each constant A is assumed to have a unique defining equation $A(\tilde{x}) \triangleq P$.

https://doi.org/10.1017/S0960129509990168 Published online by Cambridge University Press

$S_i \subseteq \mathcal{N}$ represents the set of actions whose co-action availability prevents the execution of α_i.

We now describe the reduction semantics given in Phillips (2001), in which $\mathcal{N}$ is a set of names, $\overline{\mathcal{N}}$ are the corresponding co-names, $\mathcal{U}$ is the set of names that can be used as priority guards and $\overline{\mathcal{U}}$ are the corresponding co-names. Std $= \mathcal{N} \cup \overline{\mathcal{N}}$, Pri $= \mathcal{U} \cup \overline{\mathcal{U}}$, Vis $=$ Std $\cup$ Pri, Act $=$ Vis $\cup \{\tau\}$, with: $u, v, \ldots$ ranging over Pri; $a, b, \ldots$ ranging over Vis; $\alpha, \beta, \ldots$ ranging over Act; $S, T, \ldots$ ranging over finite subsets of Vis; and $U, V, \ldots$ ranging over finite subsets of Pri.

Definition 2.11. The function $\mathrm{fn}(P) \subseteq \mathcal{N} \cup \mathcal{U}$ is defined by induction on $P \in \mathcal{P}$ as follows:

$$\begin{aligned}
\mathrm{fn}\left(\sum\nolimits_{i\in I} S_i : \alpha_i.P_i\right) &= \{n \in \mathcal{N} \cup \mathcal{U} \mid \quad \exists i \in I : \; n \in S_i \cup \{\alpha_i\} \vee \\
&\qquad\qquad\qquad\qquad\quad \overline{n} \in S_i \cup \{\alpha_i\} \vee \\
&\qquad\qquad\qquad\qquad\quad n \in \mathrm{fn}(P_i) \quad \} \\
\mathrm{fn}(P_1 \mid P_2) &= \mathrm{fn}(P_1) \cup \mathrm{fn}(P_2) \\
\mathrm{fn}((\nu a)P) &= \mathrm{fn}(P) \setminus \{a\} \\
\mathrm{fn}(A\langle a_1, \ldots, a_n\rangle) &= \{a_1, \ldots, a_n\}.
\end{aligned}$$

Definition 2.12. The congruence relation $\equiv$ on CPG processes is defined as the least congruence satisfying alpha conversion, the commutative monoidal laws with respect to both (|,**0**) and (+,**0**) and the following axioms:

$$\begin{aligned}
(\nu x)P \mid Q &\equiv (\nu x)(P \mid Q) && \text{if } x \notin \mathrm{fn}(Q) \\
(\nu x)P &\equiv P && \text{if } x \notin \mathrm{fn}(P) \\
A\langle \tilde{b}\rangle &\equiv P\{\tilde{b}/\tilde{a}\} \text{ if } A(\tilde{a}) \stackrel{def}{=} P.
\end{aligned}$$

Definition 2.13. The set $\mathrm{off}(P) \subseteq$ Pri of 'higher priority' actions 'offered' by P is defined by induction on CPG processes by the rules

$$\begin{aligned}
\mathrm{off}\left(\sum\nolimits_{i\in I} S_i : \alpha_i.P_i\right) &= \{\alpha_i : i \in I, \alpha_i \in \mathrm{Pri}, \alpha_i \notin S_i\} \\
\mathrm{off}(P_1 \mid P_2) &= \mathrm{off}(P_1) \cup \mathrm{off}(P_2) \\
\mathrm{off}(\nu a P) &= \mathrm{off}(P) \setminus \{a, \overline{a}\} \\
\mathrm{off}(A\langle \tilde{b}\rangle) &= \mathrm{off}(P\{\tilde{b}/\tilde{a}\}) \text{ if } A(\tilde{a}) \stackrel{def}{=} P.
\end{aligned}$$

Definition 2.14. Let P be a CPG process, and let $S \subseteq$ Act be finite. P *eschews* S (written P eschews S) if and only if $\mathrm{off}(P) \cap \overline{S} = \varnothing$.

https://doi.org/10.1017/S0960129509990168 Published online by Cambridge University Press

Definition 2.15. CPG semantics is given in terms of the reduction relation $\rightarrow_X$ described by the rules

$$\frac{}{S : \tau.P + M \rightarrow_{S \cap \mathrm{Pri}} P}$$

$$\frac{S : a.P + M \text{ eschews } T \quad T : \overline{a}.Q + N \text{ eschews } S}{(S : a.P + M) \mid (T : \overline{a}.Q + N) \rightarrow_{(S \cup T) \cap \mathrm{Pri}} P \mid Q}$$

$$\frac{P \rightarrow_U P' \quad Q \text{ eschews } U}{P \mid Q \rightarrow_U P' \mid Q}$$

$$\frac{P \rightarrow_U P'}{(\nu a)P \rightarrow_{U \setminus \{a, \overline{a}\}} (\nu a)P'}$$

$$\frac{Q \equiv P \quad P \rightarrow_U P' \quad P' \equiv Q'}{Q \rightarrow_U Q'}.$$

We say that $P \rightarrow Q \iff \exists X : P \rightarrow_X Q$.

The reduction relation $\rightarrow_X$ is parameterised by a set of names X representing the high-priority actions whose co-action availability would prevent the occurrence of the described reduction.

We now give the definition of barb for CPG reported in Phillips (2008).

Definition 2.16. Let P be a CPG process. P exhibits barb α, written $P \downarrow \alpha$, if and only if

— $P \equiv (\nu \tilde{y})(S : x.Q + R \mid T)$, with $\alpha = x$, $x \notin \tilde{y}$; or
— $P \equiv (\nu \tilde{y})(S : \overline{x}.Q + R \mid T)$, with $\alpha = \overline{x}$, $x \notin \tilde{y}$.

CPG Turing completeness is directly inherited from CCS.

Proposition 2.3. CPG is Turing complete.

As an alternative proof, one possible encoding of RAMs into CPG is exactly the same as that given for the bπ-Calculus in Figure 2.

For a full treatment of the CPG language, see Phillips (2001; 2008).

2.4. *The FAP language*

As previously outlined, the FAP language is a slight variant of a minimal CCS^{sg} fragment, *viz*. CCS with the addition of static, global priority (Cleaveland *et al*. 2001): keeping FAP minimal means that the expressive power of global priority can be better isolated. Only two operators are present in FAP: parallel composition and prefix. The prefix operation is only allowed after input actions, so the output can be considered asynchronous as with the asynchronous π-Calculus (Honda and Tokoro 1991; Boudol 1992). Output actions are characterised by two priority levels, meaning that high-priority output synchronisations are guaranteed to happen before low priority ones. As an example, consider the system

$$S \triangleq a.P \mid a.Q \mid b.R \mid \overline{a} \mid \underline{\overline{a}} \mid \overline{b}.$$

https://doi.org/10.1017/S0960129509990168 Published online by Cambridge University Press

The processes $\overline{a}, \underline{\overline{a}}, \overline{b}$ model messages that must be delivered to the processes listening on the appropriate channels. The message $\underline{\overline{a}}$ has higher priority than any other message in S and hence must be delivered first. Consequently, the only possible transitions of S are

$$S \rightarrow P \mid a.Q \mid b.R \mid \overline{a} \mid \overline{b} \qquad S \rightarrow a.P \mid Q \mid b.R \mid \overline{a} \mid \overline{b}$$

where the process receiving the message is chosen non-deterministically. Also, after this transition, the low-priority messages $\overline{a}$ and $\overline{b}$ can finally be delivered. To simplify the notation, inputs do not have any denotation of priority, but the results presented in this paper are completely independent of this design choice.

Definition 2.17. Let $\mathcal{N}$ be a set of names on a finite alphabet, $x, \ldots \in \mathcal{N}$. FAP syntax is defined in terms of the grammar

$$P \quad ::= \quad \mathbf{0} \;\Big|\; x.P \;\Big|\; \overline{x} \;\Big|\; \underline{\overline{x}} \;\Big|\; P \mid Q.$$

For simplicity, we will define the FAP semantics in terms of a reduction system in the style of Milner (1993).

Definition 2.18. Structural congruence for FAP is the congruence $\equiv$ generated by the equations

$$\begin{aligned} P \mid \mathbf{0} &\equiv P \\ P \mid Q &\equiv Q \mid P \\ P \mid (Q \mid R) &\equiv (P \mid Q) \mid R. \end{aligned}$$

Definition 2.19. FAP operational semantics is given in terms of the reduction systems $\mapsto$ and $\twoheadrightarrow$ described by the following rules:

$$\frac{}{x.P \mid \overline{x} \mapsto P}$$

$$\frac{}{x.P \mid \underline{\overline{x}} \twoheadrightarrow P}$$

$$\frac{P \twoheadrightarrow P'}{P \mid Q \twoheadrightarrow P' \mid Q}$$

$$\frac{P \mapsto P' \qquad P \mid Q \not\twoheadrightarrow R}{P \mid Q \mapsto P' \mid Q}$$

$$\frac{P \equiv Q \quad P \mapsto P' \quad P' \equiv Q'}{Q \mapsto Q'}$$

$$\frac{P \equiv Q \quad P \twoheadrightarrow P' \quad P' \equiv Q'}{Q \twoheadrightarrow Q'}.$$

We say that $P \rightarrow Q \iff P \mapsto Q \vee P \twoheadrightarrow Q$.

https://doi.org/10.1017/S0960129509990168 Published online by Cambridge University Press

Definition 2.20. For any process in FAP, the function fn is defined as

$$\begin{aligned} \mathrm{fn}(\mathbf{0}) &= \varnothing \\ \mathrm{fn}(\overline{x}) &= \{x\} \\ \mathrm{fn}(x.P) &= \{x\} \cup \mathrm{fn}(P) \\ \mathrm{fn}(\underline{x}) &= \{x\} \\ \mathrm{fn}(P \mid Q) &= \mathrm{fn}(P) \cup \mathrm{fn}(Q). \end{aligned}$$

As with the other languages, we define the notion of barb.

Definition 2.21. A FAP process P exhibits barb α, written $P \downarrow \alpha$, if and only if:

— $P \equiv x.Q \mid R$, $\alpha = x$; or
— $P \equiv \overline{x} \mid R$, $\alpha = \overline{x}$; or
— $P \equiv \underline{x} \mid R$, $\alpha = \underline{x}$.

Proposition 2.4. FAP is not Turing complete.

Proof. Every process $P \in$ FAP terminates – FAP has no loop operator, such as bang or recursion. □

3. Encodings

In order to provide the results previously outlined, we now formalise the relevant encoding conditions for studying the expressiveness separation between languages.

We first formalise the notion of the *observables* of a program computation, using the style of Phillips (2008).

Definition 3.1. Let L be a process language with processes $P, P_0, \ldots \in L$. A *computation* $\mathscr{C}$ *of* P is a finite or infinite sequence $P = P_0 \to P_1 \to \cdots$. We say $\mathscr{C}$ is *maximal* if it cannot be extended.

A computation of a process P is the sequence of states that P can reach during its execution. Each process P may present many different computations due to the non-determinism intrinsic in concurrent calculi.

Definition 3.2. Let L be a process language with names in $\mathscr{N}$ and having processes $P_0, \ldots, .P_i \in L$. Let $\mathscr{C}$ be a computation $P_0 \to \cdots \to P_i \cdots$. Given a set of *intended observables* $\mathrm{Obs} \subseteq \mathscr{N}$, the observables of $\mathscr{C}$ are $\mathrm{Obs}(\mathscr{C}) = \{x \in \mathrm{Obs} : \exists i\, P_i \downarrow x\}$.

The observables of a computation C are the set of all the external interactions the process may perform in the states reached during the computation.

Some of the separation results are based on the topology of the network of processes: for example, the impossibility of encoding the π-Calculus in value-passing CCS (Palamidessi 2003) is based on the hypothesis that the encoding does not increase the connectedness of the network, that is, all the processes that are independent (not sharing free names) in the source language must remain independent after the encoding. The same criterion will be necessary to separate FAP and CPG.

https://doi.org/10.1017/S0960129509990168 Published online by Cambridge University Press

Definition 3.3. Let L be a process language. Two processes $P, Q \in L$ are *independent* if they do not share any free names, that is, $\mathrm{fn}(P) \cap \mathrm{fn}(Q) = \varnothing$.

We now define the conditions an encoding may preserve, in the style of Phillips (2008).

Definition 3.4. Let L, L' be process languages. An encoding $[\![\cdot]\!] : L \to L'$ is:

1 *Observation-respecting* if $\forall P \in L$:
   — for every maximal computation $\mathscr{C}$ of P there exists a maximal computation $\mathscr{C}'$ of $[\![P]\!]$ such that $\mathrm{Obs}(\mathscr{C}) = \mathrm{Obs}(\mathscr{C}')$; and
   — for every maximal computation $\mathscr{C}$ of $[\![P]\!]$ there exists a maximal computation $\mathscr{C}'$ of P such that $\mathrm{Obs}(\mathscr{C}) = \mathrm{Obs}(\mathscr{C}')$.

2 *Weakly-observation-respecting* if $\forall P \in L$:
   — for every maximal computation $\mathscr{C}$ of P there exists a maximal computation $\mathscr{C}'$ of $[\![P]\!]$ such that $\mathrm{Obs}(\mathscr{C}) = \mathrm{Obs}(\mathscr{C}')$; and
   — for every maximal computation $\mathscr{C}$ of $[\![P]\!]$ there exists a maximal computation $\mathscr{C}'$ of P such that $\mathrm{Obs}(\mathscr{C}) \subseteq \mathrm{Obs}(\mathscr{C}')$.

3 *Distribution-preserving* if $\forall P_1, P_2 \in L$, $\quad [\![P_1 \mid P_2]\!] = [\![P_1]\!] \mid [\![P_2]\!]$.

4 *Renaming-preserving* if for any permutation σ of the source names in L there exists a permutation θ in L' such that $[\![\sigma(P)]\!] = \theta([\![P]\!])$ and the permutations are compatible on observables, that is $\sigma|_{\mathrm{Obs}} = \theta|_{\mathrm{Obs}}$.

5 *Independence-preserving* if $\forall P, Q \in L$, if P and Q are independent, then $[\![P]\!]$ and $[\![Q]\!]$ are also independent.

The observation-respecting property is the minimal requirement that any reasonable encoding should preserve: it ensures that some intended observable behaviour is preserved by the translation. The weak variant of this condition admits encodings that introduce divergence or failure deriving from livelocks or deadlocks. Under certain conditions, such as fairness or other hypotheses on scheduling or execution, the introduction of divergence or failure by the encoding may be tolerated (Nestmann 2000; Palamidessi and Herescu 2005) because it would be guaranteed not to happen anyway (or to be very unlikely).

The distribution-preserving property is a very important feature of an encoding in a concurrent framework: it implies its compositionality and above all it guarantees that the degree of parallelism of the translated system does not decrease.

The renaming-preserving property states that the encoding should not introduce asymmetries in the system. It is essential that this condition is preserved when impossibility results on problems such as the leader election are based completely on the symmetric topology of the network.

Independence-preserving represents the property of not increasing the connectedness of the network.

According to Palamidessi (2003) and Phillips (2008), a distribution- and renaming-preserving encoding is said to be *uniform*, and it is said to be *reasonable* if it also preserves the intended observables over maximal computations. We say a

https://doi.org/10.1017/S0960129509990168 Published online by Cambridge University Press

weakly-observation-respecting encoding is *sincere* if it is complete but only weakly correct, in the sense that it admits divergence or premature termination.

3.1. *The leader election problem*

An electoral system represents the situation of a bunch of processes (modelling for example a set of workstations in a network) aiming to reach a common agreement, that is, deciding which one of them (and no other) is the leader. The modelling of the election problem in process algebras requires that the system composed of the network of processes will sooner or later signal unequivocally the result of the election on some channels ω_i, which consequently become the observables of interest for the computations of the system.

Definition 3.5. Let L be a process language, and processes $P_1, \ldots, P_k \in L$. A network Net of size k, with $k \geqslant 1$, is a system $\mathsf{Net} = P_1 \mid \cdots \mid P_k$.

Definition 3.6. A network Net of size k is an *electoral system* if for every maximal computation $\mathscr{C}$ of Net $\exists i \leqslant k : \mathrm{Obs}(\mathscr{C}) = \{\omega_i\}$, where $\mathrm{Obs} = \{\omega_i : i \in \mathbb{N}\}$.

In order to keep the notation simple, the definition of an electoral system reflects the design choices made in Palamidessi (2003), Ene and Muntean (1999) and Phillips (2001), which are based on the hypothesis that the system will never perform external interactions on channels that are not intended to be observable. As in Phillips (2001; 2008), the winner process (the leader) is supposed to signal the outcome of the election, while all the other processes simply do nothing.

Definition 3.7. Given a network of size k, with $\mathsf{Net} = P_1 \mid \cdots \mid P_k$, two nodes P_i and P_j (with $i \neq j$) are *neighbours* (or connected) if they are not independent.

Definition 3.8. A network Net of size k, with $\mathsf{Net} = P_1 \mid \cdots \mid P_k$, is *connected* if one of the following conditions holds:

— $k = 1$
— $\exists i, j \leqslant k$, with $i \neq j$, such that P_i and P_j are neighbours and Net', with

$$\mathsf{Net}' = P_1 \mid \cdots \mid P_{i-1} \mid P_{i+1} \mid \cdots \mid P_k$$

is also connected.

In a connected network Net, each node is connected to at least one other node, and there exists no partition of Net into two subnetworks such that there are no cross connections between processes from one subnetwork to the other. This condition must hold when some information needs to be propagated between all the nodes.

Definition 3.9. A network Net of size k, with $\mathsf{Net} = P_1 \mid \cdots \mid P_k$, is *fully connected* if P_i and P_j are not independent, $\forall i, j \leqslant k$.

In this case each node may interact directly with any other node in the network.

https://doi.org/10.1017/S0960129509990168 Published online by Cambridge University Press

3.2. *The last man standing problem*

The last man standing problem represents a very simple situation where a bunch of n processes in a fully connected network must realise if $n = 1$ or $n > 1$ in a distributed way. The possibility or impossibility of solving the LMS problem is based on the idea that in a given language L a process P *may* or *may not* know if another process Q is ready to perform some intended action on a given channel. Usually the only way that P can know of the presence of Q is to *try* a synchronisation with it. Since the input (and often the output also) is blocking, P turns out to be blocked if the condition does not hold, or it follows another computation without any knowledge of the presence of Q. The definition of the LMS system follows.

Definition 3.10. A network N*et* of size k is a *LMS system* if for every maximal computation $\mathscr{C}$ of N*et*

— $\mathrm{Obs}(\mathscr{C}) = \{y\}$ if $k = 1$
— $\mathrm{Obs}(\mathscr{C}) = \{n\}$ if $k > 1$

where $\mathrm{Obs} = \{y, n\}$.

4. Separation results

In this section we show the separation results outlined earlier. We begin with those based on the leader election problem, and then present those based on the LMS problem.

4.1. *Leader-election-based separation results*

The bπ-Calculus and CPG have been proved capable of solving the leader election in a fully connected network without knowledge of the number of processes (Ene and Muntean 1999; Phillips 2001). Here we show that this is possible in FAP in any (not only a fully) connected network.

Theorem 4.1. Let $P_1, \ldots, P_k$ be FAP processes, $\mathsf{N}et = P_1 \mid \cdots \mid P_k$. Let

$$\begin{aligned} P_n = \overline{m}_n \mid \underline{\overline{s}}_n \mid m_n.s_n.(\omega_n \mid \underline{\overline{d}}_{n1} \mid \cdots \mid \underline{\overline{d}}_{nz_n}) \\ \mid d_{n1}.(s_n \mid \underline{\overline{d}}_{n1} \mid \cdots \mid \underline{\overline{d}}_{nz_n}) \\ \vdots \\ \mid d_{nz_n}.(s_n \mid \underline{\overline{d}}_{n1} \mid \cdots \mid \underline{\overline{d}}_{nz_n}) \end{aligned}$$

where z_n is the number of neighbours of P_n, with $1 \leqslant n \leqslant k$. Each d_{xy} represents an existing channel (which is not necessarily distinct from some other d_{wz} in the definition) that connects two neighbour processes. In other words, P_i, P_h are neighbours if and only if $\exists j, l : d_{ij} = d_{hl}$, with ω_i, s_j, m_h distinct and $\omega_i, s_j, m_h \neq d_{pq}, \forall i, j, h, p, q$. If N*et* is connected, then N*et* is an electoral system.

Proof. For $k = 1$, any maximal computation of $\mathsf{N}et = P_1$ is of the form

$$P_1 \equiv \overline{m}_1 \mid \underline{\overline{s}}_1 \mid m_1.s_1.\omega_1 \quad \mapsto \quad \underline{\overline{s}}_1 \mid s_1.\omega_1 \quad \twoheadrightarrow \quad \omega_1,$$

https://doi.org/10.1017/S0960129509990168 Published online by Cambridge University Press

hence Net is an electoral system. In fact, the first reduction is only possible by a synchronisation on channel m_1, while the second is a prioritised reduction on channel s_1. For $k \geqslant 1, \mathsf{Net} = P_1 \mid \cdots \mid P_k$, the first (low-priority) reduction can happen only on one of the m_n channels. We can suppose without loss of generality that it happens on channel m_1. Then the only possible (high-priority) reduction is on channel s_1, and the system evolves as follows:

$$\begin{aligned}\mathsf{Net} = & P_1 \mid P_2 \mid \cdots \mid P_k \mapsto \\ & Q_1 \mid P_2 \mid \cdots \mid P_k \twoheadrightarrow \\ & Q_2 \mid P_2 \mid \cdots \mid P_k = \mathsf{Net}_2\end{aligned}$$

with

$$\begin{aligned}Q_1 = \overline{s}_n \mid s_1.(\omega_1 \mid \underline{\overline{d}}_{11} \mid \cdots \mid \underline{\overline{d}}_{1z_1}) \mid d_{11}.(s_1 \mid \underline{\overline{d}}_{11} \mid \cdots \mid \underline{\overline{d}}_{1z_1}) \\ \vdots \\ \mid d_{1z_1}.(s_1 \mid \underline{\overline{d}}_{11} \mid \cdots \mid \underline{\overline{d}}_{1z_1})\end{aligned}$$

and

$$\begin{aligned}Q_2 = \omega_1 \mid \underline{\overline{d}}_{11} \mid \cdots \mid \underline{\overline{d}}_{1z_1} \mid d_{11}.(s_1 \mid \underline{\overline{d}}_{11} \mid \cdots \mid \underline{\overline{d}}_{1z_1}) \\ \vdots \\ \mid d_{1z_1}.(s_1 \mid \underline{\overline{d}}_{11} \mid \cdots \mid \underline{\overline{d}}_{1z_1}).\end{aligned}$$

As in the previous case, after the first two reductions the system already exhibits the winner. We must verify that for any subsequent computation, no other barb on $\omega_i, i \neq 1$, is exhibited. To prove this, we must ensure that every input on $s_i, i \neq 1$, is exhibited by a sequence of high-priority reductions before some other low-priority reduction on any of the channels $m_i, i \neq 1$ may occur: in this way any high-priority message $\overline{s}_i, i \neq 1$ would be extinguished and none of the remaining ω_i channels may be exhibited.
We first note that for every reduction $P \to P'$ on channels d_{nj}:

— the number of high-priority outputs on channel d_{nj} in P and P' is the same (if $d_{nj} \neq d_{nh} \forall n, j, h$, otherwise it also increases);
— the number of inputs on channel d_{nj} decreases;
— the number of inputs on channel s_i for some i increases;
— the number of high-priority outputs on channels $d_{n'h}$, with $(n', h) \neq (n, j)$, may (only) grow.

For each neighbour P_i of P_1, there exists some j, h such that $d_{1j} = d_{ih}$. This means that in Net_2, for each neighbour P_i of Q_2, there is an output $\underline{\overline{d}}_{ih}$ ready to react with the corresponding input on d_{ih} and to disclose an input on s_i. After each reduction on some d_{1j}, the number of inputs on s_1 increases, but they can only grow up to a maximum of z_1. This means that after at most $z_1 + 1$ (high-priority) reductions, an input on s_i for some i, P_i neighbour of Q_2, is exhibited. In turn, after at most another $z_i + 1 + 1$ reductions (we must take the reduction on s_i into account) another input on s_j, P_j neighbour of P_i or Q_2, is exhibited, and so on. Since the network is connected, after at most $n + \sum_{n \in N} z_n$

https://doi.org/10.1017/S0960129509990168 Published online by Cambridge University Press

high-priority reductions, Net_2 leads to a system where every s_n is exhibited, thanks to the fact that the number of high-priority outputs $\overline{d}_{nj}$ never decreases, and is thus sufficient to elide any input on d_{nj}. After this chain of high-priority reductions, a sequence of $k-1$ low-priority reductions on channels $m_i, i \neq 1$, occur, and then (because of the lack of any output $\overline{s}_i$) the computation ends before any other ω_i is exhibited. □

The next lemma (Phillips 2008) is used to prove that the above results cannot be obtained in the π-Calculus without knowing the number of processes in the electoral system, and also that the LMS problem is undecidable. As noted in Phillips (2008), this would also be true for any language having a comparable semantics for the parallel operator, such as Mobile Ambients (Cardelli and Gordon 1998).

Lemma 4.1. For any π-Calculus processes P_1, P_2, if P_i has a maximal computation with observables $O_i (i = 1, 2)$, then $P_1 \mid P_2$ has a maximal computation with observables O such that $O_1 \cup O_2 \subseteq O$.

Next we state a similar but weaker result for the bπ-Calculus, which is also needed to show the separation from FAP and CPG based on the LMS problem.

Lemma 4.2. For any bπ-Calculus processes P_1, P_2, if P_i has a maximal computation with observables $O_i (i = 1, 2)$, then $P_1 \mid P_2$ has two maximal computations $\mathscr{C}_1$ and $\mathscr{C}_2$ (which are not necessarily distinct) with observables O'_1 and O'_2, respectively, such that $O_i \subseteq O'_i$.

Proof. Let

$$P_i \to P_{i2} \to \cdots \to P_{in} \to \cdots$$

be the two maximal computations (which may be empty, finite or infinite) of P_1 and P_2, with observables O_1 and O_2, respectively. If $P_1 \not\to$ and $P_2 \not\to$ (empty computations), the sets of observables O_1 and O_2 are trivially included in the observables of $P_1 \mid P_2$.

If $P_1 \to P_{12}$, then

$$P_1 \equiv (\nu \tilde{a})(\pi.Q \mid R) \qquad \text{with} \quad \pi = \tau \quad \text{or} \quad \pi = \overline{x}\langle \tilde{y} \rangle$$

and

$$P_1 \mid P_2 \quad \to \quad P_{12} \mid P'_2$$

where P'_2 may be the same as P_2 if (at least) one of the following conditions holds:

— $\pi = \tau$
— $x \in \tilde{a}$
— $P_2 \downarrow x$.

In the same way, if $P_{12} \to P_{13}$, then

$$P_{12} \mid P'_2 \quad \to \quad P_{13} \mid P''_2,$$

and so on. By defining

$$\mathscr{C}_1 = P_1 \mid P_2 \quad \to \quad P_{12} \mid P'_2 \quad \to \quad P_{13} \mid P''_2 \quad \to \cdots,$$

https://doi.org/10.1017/S0960129509990168 Published online by Cambridge University Press

we have that $O_1 \subseteq O'_1$. Symmetrically, by defining

$$\mathscr{C}_2 = P_1 \mid P_2 \quad \rightarrow \quad P'_1 \mid P_{22} \quad \rightarrow \quad P''_1 \mid P_{23} \quad \rightarrow \cdots,$$

we have $O_2 \subseteq O'_2$. □

The next theorem follows the idea in Ene and Muntean (1999) and Phillips (2001). As discussed for the definition of sincere semantics, here the condition on the preserved observables is weaker than those considered in Ene and Muntean (1999) and Phillips (2001), but it is possible to relax them as well.

Theorem 4.2. There is no distribution-preserving and weakly-observation-respecting encoding of FAP in the π-Calculus.

Proof. Consider

$$P_n = \overline{m}_n \mid \underline{s}_n \mid m_n.s_n.(\omega_n \mid \overline{d}) \mid d.(s_n \mid \underline{d})$$

for $n = 1, 2$. By Theorem 4.1, $\mathsf{N}et_1 = P_1$, $\mathsf{N}et_2 = P_2$ and $\mathsf{N}et_{12} = P_1 \mid P_2$ are electoral systems. Let $[\![\cdot]\!]$ be a weakly-observation-respecting and distribution-preserving encoding of FAP into the π-Calculus. Hence $[\![P_1]\!]$ has a maximal computation $\mathscr{C}_1$ with observables $\mathrm{Obs}(\mathscr{C}_1) = \{\omega_1\}$, because of the weakly-observation-respecting condition. However, $[\![P_2]\!]$ also has a maximal computation $\mathscr{C}_2$ with observables $\mathrm{Obs}(\mathscr{C}_2) = \{\omega_2\}$.

So we have, for the distribution-preserving property,

$$[\![\mathsf{N}et_{12}]\!] = [\![P_1 \mid P_2]\!] = [\![P_1]\!] \mid [\![P_2]\!].$$

By Lemma 4.1, $[\![\mathsf{N}et_{12}]\!]$ has a maximal computation $\mathscr{C}_{12}$ with observables $\{\omega_1, \omega_2\} \subseteq \mathrm{Obs}(\mathscr{C}_{12})$, which are not included in the set of observables of any maximal computation of $\mathsf{N}et_{12}$, which contradicts $[\![\cdot]\!]$ being weakly-observation-preserving. □

In order to formalise the separation between FAP and CPG, we will now give the definitions pertaining to symmetric configurations of electoral systems from Phillips (2008). As previously outlined in Section 3.1, to keep the notation simple, we omit restrictions and the restriction-preserving conditions present in Phillips (2008).

Definition 4.1. Let L be a process language with names in $\mathscr{N}$. A permutation is a bijection $\sigma : \mathscr{N} \rightarrow \mathscr{N}$ such that σ preserves the distinction between observable and non-observable names, that is, $a \in \mathrm{Obs} \iff \sigma(a) \in \mathrm{Obs}$. Any permutation σ induces a mapping on processes: P is equal to $\sigma(P)$, except that any name a of P is mapped onto $\sigma(a)$ in $\sigma(P)$. Given $\mathrm{Obs} = \{\omega_i : i \in \mathbb{N}\}$, a permutation σ induces a bijection $\hat{\sigma} : \mathbb{N} \rightarrow \mathbb{N}$ defined as $\hat{\sigma}(i) = j \iff \sigma(\omega_i) = \omega_j$, thus $\sigma(\omega_i) = \omega_{\hat{\sigma}(i)}$.

Definition 4.2. Let $\mathsf{N}et = P_1 \mid \ldots \mid P_k$ be a network of size k. An *automorphism* on $\mathsf{N}et$ is a permutation σ such that $\hat{\sigma}|_{\{1,\ldots,k\}}$ is a bijection.

Definition 4.3. Let σ be an automorphism on a network of size k. For any $i \in \{1, \ldots, k\}$, the orbit $\mathscr{O}_{\hat{\sigma}}(i)$ generated by $\hat{\sigma}$ is defined as

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

https://doi.org/10.1017/S0960129509990168 Published online by Cambridge University Press

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

Definition 4.4. A network $\mathsf{Net} = P_1 \mid \ldots \mid P_k$ is *symmetric with respect to an automorphism* σ if and only if $\forall i = 1, \ldots, k\ P_{\hat{\sigma}(i)} = \sigma(P_i)$. Net is *symmetric* if it is symmetric with respect to some automorphism with a single orbit (of size k).

Definition 4.5. A *ring* is a network $\mathsf{Net} = P_1 \mid \ldots \mid P_k$ which has a single-orbit automorphism σ such that $\forall i, j < k$, if $\mathrm{fn}(P_i) \cap \mathrm{fn}(P_j) \neq \varnothing$ then $i = j$ or $\hat{\sigma}(i) = j$ or $\hat{\sigma}(j) = i$. A ring is *symmetric* if it is symmetric with respect to such an automorphism σ.

The following lemma is also stated in Phillips (2008).

Lemma 4.3. Let L, L' be process languages. If $[\![\cdot]\!] : L \to L'$ is a uniform, observation-respecting and independence-preserving encoding, then for any electoral system Net that is a symmetric ring of size k, we have $[\![\mathsf{Net}]\!]$ is also a symmetric ring of size k that is an electoral system.

Corollary 4.1. For any $k \geqslant 1$, there is a symmetric ring of size k in FAP that is an electoral system.

Proof. It is sufficient to choose a *connected* symmetric ring and apply Theorem 4.1. □

The following theorem from Phillips (2008) implies the non-encodability of the π-Calculus in CPG, but also of FAP into CPG.

Theorem 4.3. For any composite (that is, non-prime) $k \geqslant 6$, if Net is a symmetric ring of size k in CPG, then Net is not an electoral system.

Theorem 4.4. There is no uniform, observation-respecting and independence-preserving encoding of FAP into CPG.

Proof. The statement follows from Lemma 4.3, Corollary 4.1 and Theorem 4.3. □

It is worth observing that while the separation between π-Calculus and CPG arises as a result of the capability of communicating new names proper in the π-Calculus, the separation between FAP and CPG is a strict consequence of the different scope of priority in the two languages. This can be explained straightforwardly using the simple configuration of Figure 3, where a ring of size 6 is represented schematically in both FAP and CPG. As stated by Corollary 4.1 and Theorem 4.3, leader election is possible in the first system but not in the second one. In fact, the preemptive effect of global priority is not circumscribed in any way, so that the production of a single high-priority output in FAP freezes all the processes trying to perform low-priority reductions. This effect can be exploited for electing the leader in the ring of Figure 3(a), where the processes $P_1, \ldots, P_6$ are in a *low-priority state* (that is, they are ready to perform just low-priority reductions). The first process (for example, P_1) that changes its internal state can immediately become the leader by producing a high-priority output, which freezes all the other processes and propagates a chain of high-priority reductions in order to denote them as non-leader processes. However, this mechanism cannot be exploited in CPG, because the preemptive effect is bound to the scope

https://doi.org/10.1017/S0960129509990168 Published online by Cambridge University Press

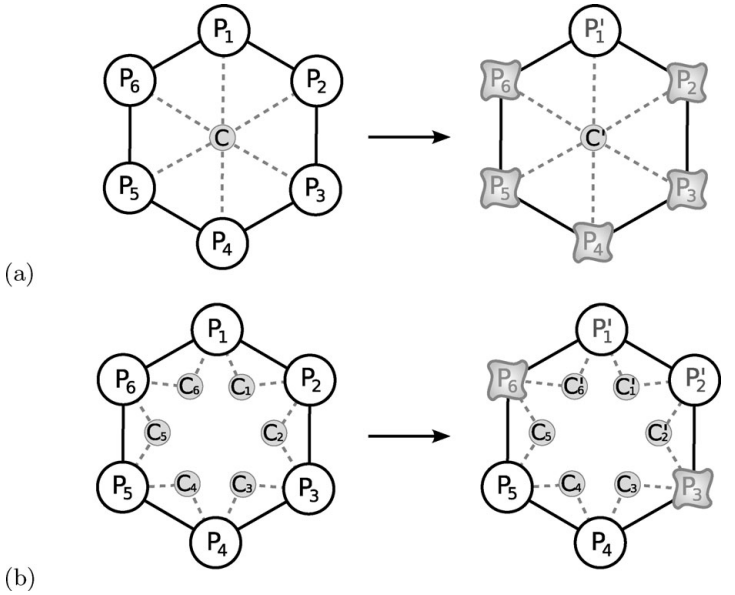

Fig. 3. Expressiveness of global (a) versus local (b) priority by leader election in a symmetric ring of size 6. The execution of high-priority reductions in languages characterised by global priority such as FAP (a) implies the freezing of *all* the low priority reductions in the system, regardless of the scope of the channel over which they occur. Conversely, languages such as CPG circumscribe the preemptive effect to the neighbour processes. This difference allows a partially distributed implementation of this kind of priority, in contrast to the complete centralisation required for the global variant.

of restricted names, so only the neighbour processes can be frozen in this way: thanks to binary synchronisation, the process P_1 in Figure 3(b) may denote process P_2 as non-leader and freeze P_3 and P_6 (the neighbours of P_2 and P_1, respectively) by exploiting local priority. However, the processes P_4 and P_5 may still attempt the same local leader election since they cannot detect in any way the state changes of the rest of the system, so the symmetry of the ring is restored and in at least one computation will never be irreversibly broken.

The price for the superior expressiveness of global priority may be revealed in a distributed implementation by the overhead introduced for the additional synchronisations needed to freeze all the low-priority processes before the execution of a high-priority reduction. A unique, central coordinator (represented by C in Figure 3(a)) should be introduced to achieve this result, while in CPG the localised effect of priority would allow the partial distribution of such coordination.

4.2. *LMS-based separation results*

In this section, we formalise the separation results based on the last man standing problem. First we show that the LMS can be solved in both FAP and CPG, and then, from this expressive capability, we derive the impossibility of encoding FAP or CPG in the π-Calculus or bπ under distribution- and weak-preservation of observables hypotheses.

https://doi.org/10.1017/S0960129509990168 Published online by Cambridge University Press

Lemma 4.4. Let P be the following FAP process:

$$P = \overline{m} \mid \underline{s} \mid \overline{q} \mid k.(s \mid q \mid \underline{\overline{k}}) \mid m.s.(s.q.(\underline{\overline{k}} \mid n) \mid \overline{l} \mid l.q.y).$$

Then $\mathsf{N}et_k = \underbrace{P \mid \cdots \mid P}_{k}$ is an LMS system of size k, $\forall k \geqslant 1$.

Proof. For $k = 1$, $\mathsf{N}et_1 = P$, $\mathsf{N}et_1$ has only one maximal computation $\mathscr{C}$

$$\begin{aligned} P \mapsto P_1 &= \underline{s} \mid \overline{q} \mid k.(s \mid q \mid \underline{\overline{k}}) \mid s.(s.q.(\underline{\overline{k}} \mid n) \mid \overline{l} \mid l.q.y) \twoheadrightarrow \\ P_2 &= \overline{q} \mid k.(s \mid q \mid \underline{\overline{k}}) \mid s.q.(\underline{\overline{k}} \mid n) \mid \overline{l} \mid l.q.y \mapsto \\ P_3 &= \overline{q} \mid k.(s \mid q \mid \underline{\overline{k}}) \mid s.q.(\underline{\overline{k}} \mid n) \mid q.y \twoheadrightarrow \\ P_4 &= k.(s \mid q \mid \underline{\overline{k}}) \mid s.q.(\underline{\overline{k}} \mid n) \mid y \end{aligned}$$

where P_4 does not allow any further transition and $\mathrm{Obs}(\mathscr{C}) = \{y\}$, as required by an LMS system of size 1.

For $k = 2$, $\mathsf{N}et_2 = P \mid P$, $\mathsf{N}et_2$ has several possible computations $\mathscr{C}$ of the same length leading to the same final state

$$Q_9 = s \mid n \mid q.y \mid s \mid q \mid \underline{\overline{k}} \mid D$$

where

$$D = s.(s.q.(\underline{\overline{k}} \mid n) \mid \overline{l} \mid l.q.y).$$

One of these computations is

$$\begin{aligned} \mathsf{N}et_2 = P \mid P &\mapsto P_1 \mid P \twoheadrightarrow P_2 \mid P \twoheadrightarrow \\ Q_3 = \overline{q} \mid k.(s \mid q \mid \underline{\overline{k}}) &\mid q.(\underline{\overline{k}} \mid n) \mid \overline{l} \mid l.q.y \mid \\ \overline{m} \mid \overline{q} \mid k.(s \mid q \mid \underline{\overline{k}}) &\mid m.D \twoheadrightarrow \\ Q_4 = k.(s \mid q \mid \underline{\overline{k}}) \mid \underline{\overline{k}} \mid n \mid \overline{l} &\mid l.q.y \mid \\ \overline{m} \mid \overline{q} \mid k.(s \mid q \mid \underline{\overline{k}}) &\mid m.D \twoheadrightarrow \\ Q_5 = s \mid q \mid \underline{\overline{k}} \mid n \mid \overline{l} &\mid l.q.y \mid \\ \overline{m} \mid \overline{q} \mid k.(s \mid q \mid \underline{\overline{k}}) &\mid m.D \twoheadrightarrow \\ Q_6 = s \mid q \mid n \mid \overline{l} &\mid l.q.y \mid \\ \overline{m} \mid \overline{q} \mid s \mid q \mid \underline{\overline{k}} &\mid m.D \twoheadrightarrow \\ Q_7 = s \mid n \mid \overline{l} &\mid l.q.y \mid \\ \overline{m} \mid s \mid q \mid \underline{\overline{k}} &\mid m.D \mapsto \\ Q_8 = s \mid n \mid q.y &\mid \\ \overline{m} \mid s \mid q \mid \underline{\overline{k}} &\mid m.D \mapsto \\ Q_9 = s \mid n \mid q.y &\mid \\ s \mid q \mid \underline{\overline{k}} &\mid D. \end{aligned}$$

We have that Q_9 does not allow any further reduction. For each computation $\mathscr{C}$ leading to Q_9, $\mathrm{Obs}(\mathscr{C}) = \{n\}$ $\forall \mathscr{C}$, as required by an LMS system of size 2. In fact, while $Q_9 \downarrow n$,

https://doi.org/10.1017/S0960129509990168 Published online by Cambridge University Press

any barb y is guarded by two inputs on channels l and q, but each high-priority output $\overline{q}$ is consumed before any low-priority output $\overline{l}$ may react.

For $k \geqslant 3$, Net_k has k^2 possible initial (low-priority) reductions on channel m that lead to congruent states. As with Net_2, the first reductions happen on channels m, s, s, q, respectively:

$$\mathsf{Net}_k \mapsto P_1 \mid \underbrace{P \mid \cdots \mid P}_{k-1} \twoheadrightarrow \cdots \twoheadrightarrow Q_4 \mid \underbrace{P \mid \cdots \mid P}_{k-2} = \mathsf{Net}_{k4}.$$

For $k \geqslant 3$, a chain of high-priority reductions happen on channels k, s, q until every input on k is consumed. After this chain of high-priority reductions, every output $\overline{s}, \overline{q}$ is consumed and no other observables $\{y, n\}$ can be exhibited. After another k low-priority transitions (one on l and $k-1$ on m), no further reduction is possible. Since for every computation no barb on y is exhibited, Net_k is an LMS system. □

Lemma 4.5. Let P be a CPG process:

$$P = a : b.\overline{a} \mid \overline{b}.(b : \tau.y \mid z : b.(\overline{b} \mid n \mid \overline{z})).$$

Then $\mathsf{Net}_k = \underbrace{P \mid \cdots \mid P}_{k}$ is an LMS system of size k, $\forall k \geqslant 1$.

Proof. For $k = 1$, $\mathsf{Net}_1 = P$, Net_1 has only one maximal computation $\mathscr{C}$:

$$P \to P_1 = \overline{a} \mid b : \tau.y \mid z : b.(\overline{b} \mid n \mid \overline{z}) \to P_2 = \overline{a} \mid y \mid z : b.(\overline{b} \mid n \mid \overline{z})$$

where P_2 does not allow any further transition and $\mathrm{Obs}(\mathscr{C}) = \{y\}$, as required by an LMS system of size 1.

For $k = 2$, $\mathsf{Net}_2 = P \mid P$, Net_2 has two possible computations $\mathscr{C}, \mathscr{C}'$ of length 2 leading to states congruent to Q_2:

$$\mathsf{Net}_2 = P \mid P \to P_1 \mid P \to$$
$$Q_2 = \overline{a} \mid b : \tau.y \mid \overline{b} \mid n \mid \overline{z} \mid a : b.\overline{a} \mid b : \tau.y \mid z : b.(\overline{b} \mid n \mid \overline{z}).$$

Q_2 is a final state and for both computations $\mathrm{Obs}(\mathscr{C}) = \mathrm{Obs}(\mathscr{C}') = \{n\}$, as required by an LMS system of size 2.

For $k \geqslant 3$, Net_k has $k \cdot (k-1)$ possible computations of length 2 leading to states congruent to Net_{k2}:

$$\mathsf{Net}_k \to P_1 \mid \underbrace{P \mid \cdots \mid P}_{k-1} \to Q_2 \mid \underbrace{P \mid \cdots \mid P}_{k-2} = \mathsf{Net}_{k2}.$$

Net_{k2} is a final state and for any computation $\mathscr{C}$ of Net_k, $\mathrm{Obs}(\mathscr{C}) = \{n\}$, as required for an LMS system of size k. □

The following is an alternative to using Theorem 4.2 to prove the separation between FAP and π-Calculus, and acts as a template for the theorems following it.

Theorem 4.5. There is no distribution-preserving and weakly-observation-respecting encoding $[\![\cdot]\!]$ of FAP in the π-Calculus.

https://doi.org/10.1017/S0960129509990168 Published online by Cambridge University Press

Proof. Suppose $[\![\cdot]\!]$ is distribution-preserving and weakly-observation-respecting. By Lemma 4.4, $\exists P : \mathsf{Net}_k$ is an LMS system for any $k \geqslant 1$, where $\mathsf{Net}_k = P \mid \ldots \mid P$. By the weakly-observation-respecting condition, $[\![\mathsf{Net}_1]\!]$ has a computation $\mathscr{C}_1$ with observables $\mathrm{Obs}(\mathscr{C}_1) = \{y\}$. By the distribution-preserving condition,

$$[\![\mathsf{Net}_k]\!] = [\![P \mid \cdots \mid P]\!] = [\![P]\!] \mid \cdots \mid [\![P]\!] = [\![\mathsf{Net}_1]\!] \mid \cdots \mid [\![\mathsf{Net}_1]\!].$$

By Lemma 4.1, there exists a maximal computation $\mathscr{C}_k$ of $[\![\mathsf{Net}_k]\!]$ such that $\mathrm{Obs}(\mathscr{C}_1) = \{y\} \subseteq \mathrm{Obs}(\mathscr{C}_k)$, while no computation of Net_k contains observable y for $k \geqslant 2$, which contradicts the weakly-observation-respecting property of the encoding function $[\![\cdot]\!]$. □

Theorem 4.6. There is no distribution-preserving and weakly-observation-respecting encoding of CPG into the π-Calculus.

Proof. By Lemma 4.5, using exactly the same reasoning as in Theorem 4.5. □

Theorem 4.7. There is no distribution-preserving and weakly-observation-respecting encoding of FAP in the bπ-Calculus.

Proof. By Lemmas 4.2 and 4.4, using exactly the same reasoning as in Theorem 4.5. □

Theorem 4.8. There is no distribution-preserving and weakly-observation-respecting encoding of CPG in the bπ-Calculus.

Proof. By Lemmas 4.2 and 4.5, using exactly the same reasoning as in Theorem 4.5. □

5. Conclusion

We have considered FAP, a finite fragment of CCS augmented with global priority, and have proved, by means of leader-election-based separation results, that it is not possible to encode it in CPG under uniformity and independence-preserving conditions on the encoding, thus providing the first expressiveness separation result between global and local priority within a process algebra framework. We then proved that FAP cannot be distributively translated into the π-Calculus even if allowing partially correct implementations, that is, encodings that may introduce divergence in computations or premature termination caused by deadlock.

We then analysed another setting, called the last man standing (LMS) problem, which allows us to considerably strengthen the separation between prioritised (with both global or local priority) languages and non-prioritised ones, by showing that even if we equip the language with broadcast-based primitives as in the bπ-Calculus, the expressiveness of priority cannot be obtained under parallel-preserving conditions.

In conclusion, we have shown that, within the context of the process algebras considered here, it is not possible to have a distribution-preserving encoding of either global or local priority in non-prioritised languages, even if we admit asymmetric translations or divergence/failure in the computation as a consequence of livelocks or deadlocks. This impossibility result does not depend on the capability of communication of names or values, synchrony or asynchrony of the output, scope extrusion, choice of the available input/outputs, recursion or replication, point-to-point or broadcast

https://doi.org/10.1017/S0960129509990168 Published online by Cambridge University Press

communication/synchronisation type. Nor does it depend on Turing completeness: in fact the non-encodability of FAP in the π-Calculus, bπ and CPG holds even if FAP is the only non-Turing complete calculus amongst the calculi considered here, and the other non-encodability results would also still hold by considering finite variants of the relevant calculi. Therefore, this impossibility depends only on the power of the instantaneous preemption characteristic of the prioritised languages analysed in this paper. As a consequence, we can see that it is impossible to have any purely distributed implementation of these kinds of priority on top of standard process calculi, even if we admit good or randomised encodings like those considered in Nestmann (2000) and Palamidessi and Herescu (2005) for the implementation of the choice operator. The strength of the separation also suggests that any encoding trying to preserve some relaxed condition on the distribution may be affected by severe performance issues due to the further synchronisations needed to preserve the constraint of instantaneous preemption.

A deeper characterisation of the expressiveness of different kinds of priority is planned as future work. In particular, further separation results between the global and local variants of priority considered here need to be investigated in order to strenghten the existing separation, as well as to determine settings where (CPG style) local priority may turn out to be more expressive.

We also plan to analyse process algebras equipped with non-instantaneous priority (in the style of the expressiveness study on PrioLinCa (Bravetti *et al.* 2005)), that is, languages where the effect of preemption is not immediate, in order to better characterise the expressive power of preemption and to identify prioritised constructs that are easier to implement in a parallel, if not distributed, framework.

Acknowledgements

The authors would like to thank the referees for their detailed remarks and helpful suggestions.

References

Baeten, J., Bergstra, J. and Klop, J. (1986) Syntax and defining equations for an interrupt mechanism in process algebra. *Fundamenta Informaticae* **IX** (2) 127–168.

Baeten, J. C. M., Bergstra, J. A. and Klop, J. W. (1987) Ready-trace semantics for concrete process algebra with the priority operator. *Comput. J.* **30** (6) 498–506.

Bernardo, M. and Gorrieri, R. (1996) Extended Markovian process algebra. In: Montanari, U. and Sassone, V. (eds) CONCUR. *Springer-Verlag Lecture Notes in Computer Science* **1119** 315–330.

Bernardo, M. and Gorrieri, R. (1998) A tutorial on empa: A theory of concurrent processes with nondeterminism, priorities, probabilities and time. *Theor. Comput. Sci.* **202** (1–2) 1–54.

Boudol, G. (1992) Asynchrony and the π-Calculus. Technical Report 1702, Department of Computer Science, INRIA Sophia-Antipolis.

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

Bravetti, M. and Gorrieri, R. (2002) The theory of interactive generalized semi-Markov processes. *Theor. Comput. Sci.* **282** (1) 5–32.

Bravetti, M., Gorrieri, R., Lucchi, R. and Zavattaro, G. (2005) Quantitative information in the tuple space coordination model. *Theor. Comput. Sci.* **346** (1) 28–57.

Camilleri, J. and Winskel, G. (1995) CCS with priority choice. *Inf. Comput.* **116** (1) 26–37.

Cardelli, L. and Gordon, A. D. (1998) Mobile ambients. In: Nivat, M. (ed.) FoSSaCS. *Springer-Verlag Lecture Notes in Computer Science* **1378** 140–155.

Cleaveland, R. and Hennessy, M. (1990) Priorities in process algebras. *Inf. Comput.* **87** (1/2) 58–77.

Cleaveland, R., Lüttgen, G. and Natarajan, V. (2001) Priority in process algebra. In: Bergstra, J., Ponse, A. and Smolka, S. (eds.) *Handbook of Process Algebra*, Elsevier Science Publishers 711–765.

Ene, C. and Muntean, T. (1999) Expressiveness of point-to-point versus broadcast communications. In: Ciobanu, G. and Paun, G. (eds.) FCT. *Springer-Verlag Lecture Notes in Computer Science* **1684** 258–268.

Hermanns, H. (2002) Interactive Markov Chains: The Quest for Quantified Quality. *Springer-Verlag Lecture Notes in Computer Science* **2428**.

Hoare, C. A. R. (1985) *Communicating sequential processes*, Prentice-Hall.

Honda, K. and Tokoro, M. (1991) An object calculus for asynchronous communication. In: America, P. (ed.) ECOOP. *Springer-Verlag Lecture Notes in Computer Science* **512** 133–147.

Le Lann, G. (1977) Distributed systems – towards a formal approach. In: IFIP Congress 155–160.

Milner, R. (1980) A Calculus of Communicating Systems. *Springer-Verlag Lecture Notes in Computer Science* **92**.

Milner, R. (1990) Functions as processes. In: *Proceedings of the seventeenth international colloquium on Automata, languages and programming*, Springer-Verlag 167–180.

Milner, R. (1993) The polyadic pi-calculus: a tutorial. In: Bauer, F. L., Brauer, W. and Schwichtenberg, H. (eds.) *Logic and Algebra of Specification*, Springer-Verlag 203–246. (Preprint version (1991) available at `citeseer.ist.psu.edu/article/milner91polyadic.html`.)

Milner, R. (1999) *Communicating and mobile systems: the π-calculus*, Cambridge University Press.

Milner, R., Parrow, J. and Walker, D. (1992a) A calculus of mobile processes, I. *Inf. Comput.* **100** (1) 1–40.

Milner, R., Parrow, J. and Walker, D. (1992b) A calculus of mobile processes, II. *Inf. Comput.* **100** (1) 41–77.

Minsky, M. (1967) *Computation: finite and infinite machines*, Prentice-Hall.

Nestmann, U. (2000) What is a "good" encoding of guarded choice? *Inf. Comput.* **156** (1–2) 287–319.

Palamidessi, C. (1997) Comparing the expressive power of the synchronous and the asynchronous pi-calculus. In: *POPL*. 256–265.

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

Palamidessi, C. and Herescu, O. M. (2005) A randomized encoding of the pi-calculus with mixed choice. *Theor. Comput. Sci.* **335** (2–3) 373–404.

Phillips, I. (2001) CCS with priority guards. In: Larsen, K. G. and Nielsen, M. (eds.) CONCUR. *Springer-Verlag Lecture Notes in Computer Science* **2154** 305–320.

Phillips, I. (2008) CCS with priority guards. *Journal of Logic and Algebraic Programming* **75** (2) 139–165.

Versari, C., Busi, N. and Gorrieri, R. (2007) On the expressive power of global and local priority in process calculi. In: Caires, L. and Vasconcelos, V. T. (eds.) CONCUR. *Springer-Verlag Lecture Notes in Computer Science* **4703** 241–255.

Vigliotti, M., Phillips, I. and Palamidessi, C. (2007) Tutorial on separation results in process calculi via leader election problems. *Theoretical Computer Science* **388** (1–3) 267–289.