The Expressive Power of Synchronizations

Cosimo Laneve Antonio Vitale
Dipartimento di Scienze dell'Informazione
Università di Bologna
laneve@cs.unibo.it

Abstract

A synchronization is a mechanism allowing two or more processes to perform actions at the same time. We study the expressive power of synchronizations gathering more and more processes simultaneously. We demonstrate the non-existence of a uniform, fully distributed translation of Milner's CCS with synchronizations of $n + 1$ processes into CCS with synchronizations of n processes that retains a "reasonable" semantics. We then extend our study to CCS with symmetric *synchronizations allowing a process to perform both inputs and outputs at the same time. We demonstrate that synchronizations containing more than three input/output items are encodable in those with three items, while there is an expressivity gap between three and two.*

1. Introduction

Process calculi propose several different synchronization mechanisms. In CCS and pi calculus, the synchronization is between two processes, one sending a message and the other receiving it [15, 16]. In CSP, the synchronization is among all the processes that share a common channel name [4]. In join calculus there is a programmable synchronization mechanism – the *joint inputs* – that allows one to define the channels whose messages must be handled simultaneously [8]. Other calculi use joint input mechanisms, such as smooth orchestrators [12] – an extension of asynchronous pi calculus to program web services orchestrations – and strand algebras – a recent formalism proposed for DNA computing [6].

In [8], Fournet and Gonthier translated the generic synchronization pattern of join calculus to a basic one consisting of binary synchronizations. Their encoding manifests two problems. First, it introduces divergence and, in fact, it has been proved correct with respect to bisimulation, which is unsensible to divergent computations. Second, the encoding cannot be considered "truly distributed" because it relies on the locality principle that constrains co-defined channels to be co-located at the same node.

It is folklore in the Concurrency Theory community that some expressivity gap between the different forms of synchronization must exists. For example, in a calculus *à la* CCS without mixed choice it is not possible to elect one leader in a (symmetric) network of processes without introducing divergence [19]. On the contrary, if one extends CCS with a prefix $[a_1, a_2].P$ that enables P if there are two outputs on a_1 and a_2, then leader election in a network consisting of two processes P_1 and P_2 has the following simple solution:

$$P_1 \;=\; \overline{a_1} \mid [a_1, a_2].\overline{out_1} \qquad P_2 \;=\; \overline{a_2} \mid [a_2, a_1].\overline{out_2}$$

(this solution may be elaborated a little bit to let the two processes declare the same leader).

Our study takes inspiration from the above example. We extend CCS with the input prefix $[a_1, \cdots, a_n].P$ that gets simultaneously the outputs on $a_1, \cdots, a_n$ and transits to P. This allows us to define a family of process calculi, called CCS^n, that retains input prefixes up-to n names. In order to demonstrate the presence of an expressivity gap between CCS^n and CCS^{n-1}, we consider a well-known problem of resource condivision: the dining philosophers problem. In fact, we study a variant of it – the dining philosophers problem in the n-hypercube – where the philosophers sit at the vertices of an hypercube of dimension n, forks are at the edges, and philosophers can grab forks at their adjacent edges only. We demonstrate that the problem has solutions in CCS^n but not in CCS^{n-1} as long as philosophers have identical codes and initially retain a same number of forks. Technically we prove that CCS^{n-1} codes may get to a deadlock where all the philosophers are blocked on the attempt of grabbing forks. The proof uses a well-known result in discrete mathematics due to Alspach, Bermond and Sotteau: there are $\lfloor n/2 \rfloor$ edge-disjoint *Hamiltonian cycles* in the n-hypercube [1]. (An Hamiltonian cycle is a path traversing all the vertices of the hypercube without repetition of edges; two paths are edge-disjoint if they have no edge in common.) In our setting this means that each philosopher may grab up-to $\lfloor n/2 \rfloor$ forks without hindering the progress of the adjacent philosophers. As a result, once all the philosophers have got their forks, the system will eventually either deadlock, because no one can collect all his adjacent forks, or backtrack. In the other cases, *i.e.* a philosopher initially grabs more than $\lfloor n/2 \rfloor$ forks (and less than n forks), it is possible to define a reachable deadlock configuration.

The synchronization pattern of CCS^n is *many-to-one*: there are many outputting processes and exactly one receptor. This is inadequate when more flexibility is required. For example, consider the mixed-guarded choice $\sum_{i \in I} \alpha_i.P_i$, where α_i may be either input or output and where the progress of at most one addend process $\alpha_i.P_i$ is allowed. An implementation of mixed-guarded choice in a choice-free calculus would require a symmetric management of the addends, regardless the fact they are inputting or outputting. We are not aware of any such translation in CCS^n. (The input-guarded choice $\sum_{i \in I} a_i.P_i$ may be easily translated in CCS^2.) This problem paves the way for a slightly different calculus: CCS with n-joint prefixes, called CCS^{n+}, that has only one prefix $[\alpha_1, \cdots, \alpha_n].P$ (again, with α_i either input or output). In CCS^{2+} there is a simple encoding of mixed choice:

$$(\ell)(\prod_{i \in I} [\alpha_i, \ell].[\![P_i]\!] \mid \overline{\ell}) .$$

In opposition to what happens with joint inputs, we demonstrate that 3-joint prefixes are expressive enough for encoding any n-joint prefix, with $n > 3$, whilst 2-joint prefixes are less expressive than 3-joint ones. These conclusions follow by the remark that joint-prefixes permit the synchronization of an arbitrary number of processes. Still, for 2-joint prefixes the overall effect of synchronizations is to exhibit at most two labels, which is too restrictive when more than two resources must be grabbed at once.

Related Works. The question about the expressive power of synchronization mechanisms dates back (at least) to the eighties when Francez and Rodeh proposed a distributed, deterministic solution to the dining philosophers problem in CSP [9] and Lehmann and Rabin demonstrated that such a solution does not exist in a language with a synchronization *à la* CCS [14]. After these results, our problem slept for about two decades, till a contribution by Nestmann on the expressive power of joint inputs in pi calculus [17]. Nestmann demonstrated that it is possible to encode the pi calculus with mixed choice into a pi calculus with joint inputs; however he only conjectured the absence of an encoding from pi-calculus with joint inputs into one with 2-ary joins. Almost contemporary to Nestmann's paper, there is a study of concurrent primitives in ML by Panangaden and Reppy [21] where they claim that it is not possible to implement an $n+1$-way synchronous communication using an n-way synchronous communication. However they pointed to a forthcoming paper for the proof of this result that we have not been able to find.

The techniques in this paper follow the style of [19], where synchronous pi calculus has been proved more expressive than the asynchronous one, and of [5, 10], where synchronizations between two processes using structured channel names have been studied.

Apart from these contributions in process calculi, there are close results in formalisms for biology. Actually, our initial motivation for studying the expressive power of synchronizations has been the implementation of biologically inspired languages in pi calculus. In [13], we demonstrated that it is not possible to implement the κ-calculus into one admitting at most two reactants. In that paper, the expressive power was demonstrated by analyzing the models of a biologically relevant reaction (the homeo-trimerization).

Structure of the paper. In Section 2 we define CCS with joint inputs. In Section 3 we define the dining philosophers problem in the n-hypercube and we study basic properties. In Section 4 we introduce the notions of edge assignments and define edge assignments that saturate the forks of the n-hypercube. In Section 5 the expressive power of CCS^n is analyzed. In Section 6, CCS is extended with joint prefixes and its expressive power is studied. We conclude in Section 7.

2. CCS with joint inputs

The syntax of CCS^n, called n-join CCS, uses a countable set of *names* $\mathcal{N}$, ranged over by $a, b, c, \cdots$, a countable set of *co-names* $\overline{\mathcal{N}}$, ranged over by $\overline{a}, \overline{b}, \overline{c}, \cdots$, and a countable set of *variables* $\mathcal{V}$, ranged over by $x, y, z, \cdots$.

CCS^n *processes* $P, Q, \cdots$ are defined by the grammar:

P	$::=$	0	*inaction*
	$\mid$	$\overline{a}.P$	*output*
	$\mid$	$[a_1, \cdots, a_m].P$	*joint input* $(1 \le m \le n)$
	$\mid$	$(a)P$	*restriction*
	$\mid$	$P \mid P$	*parallel*
	$\mid$	x	*variable*
	$\mid$	$\texttt{rec}\, x.\, P$	*recursion*

The term 0 defines the terminated process; $\overline{a}.P$ defines a process that sends a message on a and continues as P; the joint input $[a_1, \cdots, a_m].P$ defines a process that receives simultaneously messages on $a_1, \cdots, a_m$ and continues as P. The parallel allows processes to interact. We often abbreviate the parallel composition of P_i for $i \in I$, where I is a finite set, with $\prod_{i \in I} P_i$. The restriction $(a)P$ limits the scope of a to P; the name a is said to be *bound* in $(a)P$. This is the only binding operator of names in CCS^n. When A is a set $\{a_1, \cdots, , a_m\}$, we write $(A)P$ for $(a_1) \cdots (a_n)P$ (the order of restrictions is irrelevant). The *free names* in P, denoted $\texttt{fn}(P)$, are the names in P with a non-bound occurrence either in a joint-input or in an output. The term $\texttt{rec}\,x.P$ defines a recursive process: a (free) occurrence of the variable x in P stands for the whole $\texttt{rec}\,x.P$. We assume that variables are always bound in processes. Ending 0 will be omitted.

The calculus CCS^1 is Milner's CCS without relabelling and choice [15]. Inputs in CCS^n have *at most* n items. One might be stricter on this point, by admitting inputs of CCS^n with *exactly* n messages. This constraint is in fact unimportant, since it is easy to encode inputs with less than n elements into a calculus using only n-joint inputs. For example $[a_1, \cdots, a_{n-1}].P$ may be encoded as $(\ell)(\overline{\ell} \mid [a_1, \cdots, a_{n-1}, \ell].P)$, with ℓ fresh.

CCS^n retains an operational semantics defined by a *labelled transition system.* Let $\mu, \mu', \cdots$ range over either sequences of co-names or sequences of names or the special symbol τ. The predicate $a \in \mu$ is true if either a or $\overline{a}$ occurs in the sequence μ, otherwise it is false. Let $a_1, \cdots, a_m \setminus a$ be the function returning

- τ, if $a_1, \cdots, a_m = a$,
- $a_2, \cdots, a_m$, if $a_1 = a$ and $m \geq 2$,
- $a_1, (a_2, \cdots, a_m \setminus a)$, if $a_1 \neq a$ and $a \in a_2, \cdots, a_m$.

The function $a_1, \cdots, a_m \setminus a$ is partial: it is not defined if $a \notin a_1, \cdots, a_m$. This function is lifted to arguments that are both sequences: $a_1, \cdots, a_m \setminus b_1, \cdots, b_\ell = (\cdots (a_1, \cdots, a_m \setminus b_1) \setminus \cdots) \setminus b_\ell$. Similarly $\in$ is lifted to sequences: $a_1, \cdots, a_m \in b_1, \cdots, b_\ell$ if, for every i, $a_i \in (b_1, \cdots, b_\ell \setminus a_1, \cdots, a_{i-1})$ (this is multiset containment).

The transition relation of CCS^n is defined by the following rules (plus the symmetric ones for $|$).

$$\overline{a}.P \xrightarrow{\overline{a}} P \qquad [a_1, \cdots, a_m].P \xrightarrow{a_1,\cdots,a_m} P$$

$$\frac{P \xrightarrow{\mu} Q \quad a \notin \mu}{(a)P \xrightarrow{\mu} (a)Q} \qquad \frac{P\{^{\texttt{rec}\,x.P}/_{x}\} \xrightarrow{\mu} Q}{\texttt{rec}\,x.P \xrightarrow{\mu} Q}$$

$$\frac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q} \qquad \frac{P \xrightarrow{\overline{a_1},\cdots,\overline{a_\ell}} P' \quad Q \xrightarrow{\overline{b_1},\cdots,\overline{b_m}} Q' \quad \ell + m \leq n}{P \mid Q \xrightarrow{\overline{a_1},\cdots,\overline{a_\ell},\overline{b_1},\cdots,\overline{b_m}} P' \mid Q'}$$

$$\frac{P \xrightarrow{a_1,\cdots,a_m} Q \quad P' \xrightarrow{\overline{b_1},\cdots,\overline{b_\ell}} Q' \quad b_1, \cdots, b_\ell \in a_1, \cdots, a_m}{P \mid P' \xrightarrow{a_1,\cdots,a_m \setminus b_1,\cdots,b_\ell} Q \mid Q'}$$

The rules are standard except the last two. Processes performing outputs are collected till a joint input containing a larger sequence is found. Then the label is updated according to the output processes that have been recruited. If the sequences of outputs and inputs match then the synchronization process terminates and a τ-labelled transition is yielded. For example, $(a)([a,b].P \mid \overline{a}.Q) \mid \overline{b}.R \xrightarrow{\tau} (a)(P \mid Q) \mid R$ because $(a)([a,b].P \mid \overline{a}.Q) \xrightarrow{b} (a)(P \mid Q)$ and $\overline{b}.R \xrightarrow{\overline{b}} R$. Also $(a)([a,b].P \mid (\overline{a}.Q \mid \overline{b}.R)) \xrightarrow{\tau} (a)(P \mid Q) \mid R$ because $[a,b].P \xrightarrow{a,b} P$ and $\overline{a}.Q \mid \overline{b}.R \xrightarrow{\overline{a},\overline{b}} Q \mid R$. It is worth to remark that, in this last example, the synchronization between $[a,b].P$ and $\overline{a}.Q \mid \overline{b}.R$ should have not been possible without a label as $\overline{a}, \overline{b}$. As usual, we use the following abbreviations of computations: $P \xrightarrow{\tau}{}^{*} \xrightarrow{\mu} \xrightarrow{\tau}{}^{*} Q$ is abbreviated with $P \overset{\mu}{\Longrightarrow} Q$ and $P \overset{\mu_1}{\Longrightarrow} \cdots \overset{\mu_n}{\Longrightarrow} Q$ is abbreviated with $P \overset{\mu_1 \cdots \mu_n}{\Longrightarrow} Q$; we omit the final process Q when it is not relevant. A computation that cannot be further extended is called *maximal*. In particular, infinite computations are maximal. A computation $P \overset{\widetilde{\mu}}{\Longrightarrow} Q$ *is of type* A if labels in $\widetilde{\mu}$ use names in $A \cup \{\tau\}$.

Let ρ be a *renaming*, that is a map $\mathcal{N} \rightarrow \mathcal{N}$, and let $\rho(\overline{a}) = \overline{\rho(a)}$. Then $\rho(P)$ is the process where ρ has been applied homomorphically to every CCS^n operator in P. Renamings are ranged over by $\rho, \sigma, \cdots$.

3. The dining philosophers problem in the hypercube

The usual description of Dijkstra's dining philosophers problem is the following [7]. There are m philosophers sitting around a table with exactly one fork in between each adjacent pair of them. Philosophers go indefinitely through the following cycle: thinking, trying to eat, and eating. In order to eat, a philosopher needs both the forks on his left and right sides; when the two forks at his sides are free, the philosopher grabs them – one after the other –, eats, and releases them (and starts thinking). The difficulty of the problem is when every philosopher grabs the fork at his left. Then, to escape the deadlock, someone has to release the fork. This fact, assuming that philosophers are identical, may get back to the initial state, thus yielding a cycle.

The dining philosophers problem has been studied for a long time. Solutions have been proposed that totally order either forks or philosophers [3] or that use powerful operators, such as the CSP synchronization [9], or probabilistic algorithms [14]. It is worth to observe that the presence of, even simple, solutions of the problem does not invalidate Lehmann and Rabin's theorem in [14] (*there is no deterministic, deadlock-free, truly distributed and symmetric solution to the dining philosophers problem*). But, rather, as they already pointed out for the CSP solution in [9], that there is no truly distributed *implementation* of the synchronization operators we are studying.

Dining philosophers in a hypercube network. We consider a generalization of the dining philosophers problem where the philosophers sit at the vertices of an n*-hypercube* and forks are at the edges. Thus every philosopher has n neighbour philosophers and n adjacent forks. In this general problem, a philosopher eats if he grabs all the n adjacent forks. The Figure 1 illustrates the 3-hypercube and the

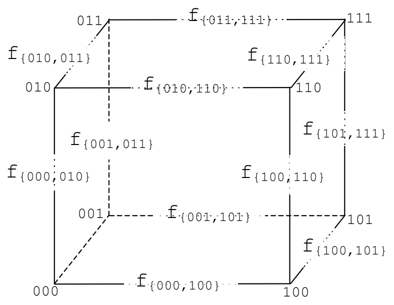

Figure 1. The (3-hyper)cube and the forks

forks; philosophers, represented by 3 bits, label the vertices and forks label the edges. In the case of the n-hypercube, the philosophers are represented by n bits – they are 2^n – and the forks are represented by unordered pairs of neighbour philosophers representations – they are $n \times 2^{n-1}$. It is worth to notice that two philosophers are neighbours if their representations differ for exactly one bit.

A few preliminary notions follow. We use $b, b', \cdots$ to range over $\{0,1\}$ and $\widetilde{b}, \widetilde{b'}, \cdots$ to range over $\{0,1\}^n$ (n bits). Let $0 \oplus 0 = 0$, $0 \oplus 1 = 1 \oplus 0 = 1$ and $1 \oplus 1 = 0$ and let $b_1 \cdots b_n \oplus 1$, the *set of neighbours* of $b_1 \cdots b_n$, be $\{(b_1 \oplus 1)b_2 \cdots b_n,\ b_1(b_2 \oplus 1)b_3 \cdots b_n, \cdots, b_1 b_2 b_3 \cdots (b_n \oplus 1)\}$. Let also F^n, the set of forks of the n-hypercube, be $\{f_{\{\widetilde{b},\widetilde{b'}\}} \mid \widetilde{b} \in \{0,1\}^n$ and $\widetilde{b'} \in \widetilde{b} \oplus 1\}$ and $F_{\widetilde{b}}$, the set of forks that are adjacent to the philosopher $\widetilde{b}$, be $\{f_{\{\widetilde{b},\widetilde{b'}\}} \mid \widetilde{b'} \in \widetilde{b} \oplus 1\}$, For example, when $n = 3$, the set F_{000} is $\{f_{\{000,001\}}, f_{\{000,010\}}, f_{\{000,100\}}\}$; the set notation as index of a fork allows us to equate the forks $f_{\{000,001\}}$ and $f_{\{001,000\}}$. Finally, let $E^n = \{eat_{\widetilde{b}} \mid \widetilde{b} \in \{0,1\}^n\}$.

A *network* $\mathbf{N}$ is a term $[P_1^{(0)} \mid \cdots \mid P_m^{(0)}]$. The computations of the network $\mathbf{N}$ have the form:

$$\begin{array}{lll} [P_1^{(0)} \mid \cdots \mid P_m^{(0)}] & \xrightarrow{\mu_1} & [P_1^{(1)} \mid \cdots \mid P_m^{(1)}] \\ & \xrightarrow{\mu_2} & [P_1^{(2)} \mid \cdots \mid P_m^{(2)}] \\ & \cdots & \\ & \xrightarrow{\mu_h} & [P_1^{(h)} \mid \cdots \mid P_m^{(h)}]\,. \end{array}$$

Given a computation $\zeta = \mathbf{N} \overset{\widetilde{\mu}}{\Longrightarrow} \mathbf{N}'$, it is possible to define the projection to the i-th component, written $\mathtt{proj}_i(\mathbf{N} \overset{\widetilde{\mu}}{\Longrightarrow} \mathbf{N}')$ to be the computation $P_i^{(0)} \xrightarrow{\nu_1} \cdots \xrightarrow{\nu_k} P_i^{(k)}$ that consists of the transitions performed by the i-th component in ζ.

When components are philosophers (and forks) of the n-hypercube, the computations of philosophers have the following properties: (i) *a fork is released only if it is retained*, (ii) *a fork is grabbed only if it is not retained.* In order to formalize these two constraints, we define the notion of (F, F')*-properness* of computations of the philosopher at $\widetilde{b}$, with $F, F' \subseteq F_{\widetilde{b}}$. The sets F and F' represent the forks retained in the initial and final states of the computation, respectively:

- the empty computation is (F, F)*-proper*;
- if $P \overset{\widetilde{\mu}}{\Longrightarrow} P'$ is (F, F')*-proper* and $P' \xrightarrow{\tau} P''$ is a synchronization whose transitions have labels not occurring in $F_{\widetilde{b}} \setminus F'$ then $P \overset{\widetilde{\mu}}{\Longrightarrow} P' \xrightarrow{\tau} P''$ is (F, F')*-proper* – internal transitions cannot concern forks that are not retained;
- if $P \overset{\widetilde{\mu}}{\Longrightarrow} P'$ is (F, F')*-proper* and $f_1, \cdots, f_k \in F'$ then $P \overset{\widetilde{\mu}}{\Longrightarrow} P' \xrightarrow{\overline{f_1}, \cdots, \overline{f_k}} P''$ is $(F, F' \setminus \{f_1, \cdots, f_k\})$*-proper* – a fork is released only if it is retained;
- if $P \overset{\widetilde{\mu}}{\Longrightarrow} P'$ is (F, F')*-proper* and $f_1, \cdots, f_k \in F_{\widetilde{b}} \setminus F'$ then $P \overset{\widetilde{\mu}}{\Longrightarrow} P' \xrightarrow{f_1, \cdots, f_k} P''$ is $(F, F' \cup \{f_1, \cdots, f_k\})$*-proper* – a fork is grabbed only if it is not retained.

A computation $P \overset{\widetilde{\mu}}{\Longrightarrow} P'$ of the philosopher at $\widetilde{b}$ of type $F_{\widetilde{b}} \cup \overline{F_{\widetilde{b}}}$, is F*-proper* ($F \subseteq F_{\widetilde{b}}$) if there exists an F' such that it is (F, F')-proper; $P \overset{\widetilde{\mu}}{\Longrightarrow} P' \xrightarrow{\overline{eat_{\widetilde{b}}}} P'' \overset{\widetilde{\mu}}{\Longrightarrow} P'''$ is F*-proper* if $P \overset{\widetilde{\mu}}{\Longrightarrow} P'$ is $(F, F_{\widetilde{b}})$-proper and $P'' \overset{\widetilde{\mu}}{\Longrightarrow} P'''$ is $F_{\widetilde{b}}$-proper. We observe that, according to this notion of properness, computations releasing forks that have been just grabbed are considered proper. Such behaviours are displayed by codes $[f_1, f_2, f_3].(\overline{f_1} \mid \overline{f_3} \mid P)$ or $[f_1, f_2].(\overline{f_2} \mid [f_2].(\overline{f_1} \mid P))$ that in fact are undistinguishable from $[f_2].P$ (since the communication through the forks is asynchronous [2]). We also observe that F-properness does not imply that a philosopher will eventually perform a $\overline{eat_{\widetilde{b}}}$-transition. This feature will follow by the foregoing deadlock-freeness condition.

Definition 3.1 *A* philosopher system of size n *is a pair* $(\mathbf{N}, (G_{\widetilde{0}}, \cdots, G_{\widetilde{1}}))$*, where* $\mathbf{N}$ *is a network* $[P_{\widetilde{0}} \mid \cdots \mid P_{\widetilde{1}} \mid \prod_{f \in F \subseteq F^n} P_{\overline{f}}]$ *with* $\mathtt{fn}(P_{\widetilde{b}}) \subseteq F_{\widetilde{b}} \cup \{eat_{\widetilde{b}}\}$ *and* $G_{\widetilde{0}}, \cdots, G_{\widetilde{1}}$ *is a partition of* $F^n \setminus F$ *such that:*

1. *for every* $\mathbf{N} \overset{\widetilde{\mu}}{\Longrightarrow} \mathbf{N}'$ *of type* E^n *and every* $\widetilde{b}$, $\mathtt{proj}_{\widetilde{b}}(\mathbf{N} \overset{\widetilde{\mu}}{\Longrightarrow} \mathbf{N}')$ *is* $G_{\widetilde{b}}$*-proper;*
2. *every maximal computation of* $P_{\overline{f}}$ *only manifests the label* $\overline{f}$*.*

A philosopher system explicitly represents the distribution in the system of its components and the fact that forks may be accessed only by adjacent philosophers.

Proposition 3.2 *1. Let $(\mathbf{N}, (G_{\widetilde{0}}, \cdots, G_{\widetilde{1}}))$ be a philosopher system of size n and let $\mathbf{N} \overset{\widetilde{\mu}}{\Longrightarrow} \mathbf{N}'$ of type E^n. Let $G'_{\widetilde{0}}, \cdots, G'_{\widetilde{1}}$ be such that $\texttt{proj}_{\widetilde{b}}(\mathbf{N} \overset{\widetilde{\mu}}{\Longrightarrow} \mathbf{N}')$ is $(G_{\widetilde{b}}, G'_{\widetilde{b}})$-proper. Then $G'_{\widetilde{0}}, \cdots, G'_{\widetilde{1}}$ are disjoint subsets of F^n.*

2. Let $P_{\widetilde{0}}, \cdots, P_{\widetilde{1}}$ be processes such that $\texttt{fn}(P_{\widetilde{b}}) \subseteq F_{\widetilde{b}} \cup \{eat_{\widetilde{b}}\}$ and let

$$(\!| \, G \, |\!) = (\prod_{f \in F_{\widetilde{b}} \setminus G} \texttt{rec}\, x.\overline{f}.f.x) \mid (\prod_{f \in G} \texttt{rec}\, x.f.\overline{f}.x) \; .$$

Let $G_{\widetilde{0}}, \cdots, G_{\widetilde{1}}$ be a partition of $F^n \setminus F$ such that, for every $\widetilde{b}$ and every $\zeta = [P_{\widetilde{b}} \mid (\!| \, G_{\widetilde{b}} \, |\!)] \overset{\mu}{\Longrightarrow} \mathbf{N}$ of type $\{\overline{eat_{\widetilde{b}}}\}$, the computation $\texttt{proj}_1(\zeta)$ is $G_{\widetilde{b}}$-proper. Then $([P_{\widetilde{0}} \mid \cdots \mid P_{\widetilde{1}} \mid \prod_{f \in F} \overline{f}], (G_{\widetilde{0}}, \cdots, G_{\widetilde{1}}))$ is a philosopher system of size n.

The first statement of Proposition 3.2 guarantees a relevant invariant of philosopher systems: two adjacent philosophers never hold the same fork. The second statement defines a sufficient condition for processes to take part in a philosopher system. This condition may be verified on a process basis by analyzing interactions with a "fasting" philospher that grabs forks (one at a time) and releases them immediately after. A process P that meets the conditions of Proposition 3.2(2) with the set $G \subseteq F_{\widetilde{b}}$ – namely $\texttt{fn}(P_{\widetilde{b}}) \subseteq F_{\widetilde{b}} \cup \{eat_{\widetilde{b}}\}$ and, for every $\zeta = [P \mid (\!| \, G \, |\!)] \overset{\mu}{\Longrightarrow} \mathbf{N}$ of type $\{\overline{eat_{\widetilde{b}}}\}$, the computation $\texttt{proj}_1(\zeta)$ is G-proper – will be called *a philosopher process at $\widetilde{b}$ with set of forks G*.

Philosopher systems that we will study satisfy the (natural) constraint that philosophers have all identical code. This is formalized by assuming that philosophers' codes are equal up-to *bijective renamings*. (In this paper, the bijective renamings between philosopher processes always map forks to forks and *eat* names to *eat* names.) A system with this property will be called *symmetric*. We notice that, in general, symmetries are broken by transitions, which may yield configurations where philosophers are in different states (*i.e.* philosophers retain different numbers of forks). We also notice that, in a symmetric system, philosophers not necessarily retain an empty set of forks. For example they may all retain one fork and, in general, more than one (see below).

Deadlocks and protocols. The definition of philosopher system admits computations without *eat*-transitions. These computations are consequences of philosophers releasing forks before eating or philosophers starving for forks that are taken by others. The following definition aims at excluding such (mis)behaviours.

Definition 3.3 *A philosopher system of size n is* deadlock-free *if every maximal computation of type $\overline{E^n}$ has infinitely many transitions with labels in $\overline{E^n}$.*

(Deadlock-freeness is less demanding than *livelock freedom* where every philosopher is guaranteed to eventually eat.)

It is possible that symmetric philosopher systems of size 2 (for example) never manifest a deadlock even if philosophers' codes are written in CCS^1. This is the case when philosophers use *different protocols* for gathering forks. Such as those at even positions grab their right fork before the left one and those at odd positions grab forks in the other way around. While philosophers are all identical up-to (bijective) renamings to

$$\texttt{rec}\, x.[f].[f'].\overline{eat}.(x \mid \overline{f} \mid \overline{f'}) \; ,$$

the competition between those at `00` and `01` and between those at `11` and `10` already allows the progress of only two philosophers. The two "winning" philosophers either will progress grabbing the remaining forks or will compete on a free fork. In any case, the overall system progresses.

Understanding whether philosophers use a same protocol or not is difficult because it requires a thorough analysis of renamings in philosopher systems. However, we argue that such analysis is not necessary. In fact, what makes the difference between CCS^1 and CCS^2 is that *every* CCS^1 philosopher code may be renamed in a way that the resulting philosopher system is not deadlock-free. On the contrary, *there are* CCS^2 codes, as $\texttt{rec}\, x.[f, f'].\overline{eat}.(x \mid \overline{f} \mid \overline{f'})$, such that every philosopher system yielded by bijective renamings is deadlock free (because all the adjacent forks are grabbed at once, therefore the renamings have all the same effect). In general, a deadlock-free philosopher system of size n in CCS^n is $([Ph_{\widetilde{0}}, \cdots, Ph_{\widetilde{1}}, \prod_{f \in F^n} \overline{f}], (\emptyset, \cdots, \emptyset))$, where

$$Ph_{\widetilde{b}} = \texttt{rec}\, x.[F_{\widetilde{b}}].\overline{eat}_{\widetilde{b}}.(x \mid \prod_{f \in F_{\widetilde{b}}} \overline{f}) \; .$$

Said in a more speculative way, the formulation of the dining philosophers problem suggests a high degree of locality: the behaviour of one philosopher should influence in no way the behaviour of his colleagues. Given that the philosopher codes are identical, this means that a code is a good solution as long as one may blindly install it in the other vertices of the hypercube and still obtain a deadlock-free philosopher system. In the above argument, these blind installations are formalized by imposing the non existence of renamings that break the deadlock-freeness.

4 Edge assignments in the n-hypercube

An *Hamiltonian cycle* in the n-hypercube is a path traversing all the vertices of the hypercube without repetition of edges. Two cycles are *edge-disjoint* if they do not share any edge. The following result is due to Alspach, Bermond and Sotteau.

Theorem 4.1 ([1]) *In the n-hypercube there exist $\lfloor n/2 \rfloor$ edge-disjoint Hamiltonian cycles.*

This theorem guarantees the progress of every philosopher in the n-hypercube when everyone initially grabs m forks, with $m \leq \lfloor n/2 \rfloor$, in m edge-disjoint Hamiltonian cycles. Since nobody can complete the grabbing and since all have identical code, the philosophers are bound to either deadlock or backtrack (to an initial configuration).

When philosophers initially grab m forks, with $m > \lfloor n/2 \rfloor$, Theorem 4.1 cannot be used. In this case, we explicitly define a grabbing that causes a deadlock (to be escaped, in case, by backtracking). The notion below of *edge assignment* represents philosophers' grabbings in the n-hypercube.

Definition 4.2 *An m-*edge assignment *in the n-hypercube ($m \leq n$) is a map χ from nodes to edges such that $\chi(\widetilde{b}) \subseteq F_{\widetilde{b}}$ and*

- *$\chi(\widetilde{b})$ is either empty or has cardinality m;*
- *for every pairs of neighbours $\widetilde{b}, \widetilde{b'}$, $\chi(\widetilde{b}) \cap \chi(\widetilde{b'}) = \emptyset$ (an edge is assigned to at most one node).*

An m-edge assignment χ in the n-hypercube

- *is* maximal *if, for every m-edge assignment χ' such that, for every $\widetilde{b}$, $\chi(\widetilde{b}) \subseteq \chi'(\widetilde{b})$, then $\chi = \chi'$ (no further m-edge assignment can be done);*
- *(when $m < n$) is* saturated *if it is maximal and, for every $\chi(\widetilde{b}) \neq \emptyset$, there exist $\widetilde{b'} \in \widetilde{b} \oplus 1$ such that $F_{\widetilde{b}} \setminus \chi(\widetilde{b}) \cap F_{\widetilde{b'}} \neq \emptyset$ (some missing edge has been assigned to an adjacent node).*

The following statements guarantee the existence of edge assignments in a constructive way (the edge assignments are explicitly defined). Since the arguments are inductive, the difficulty is to find basic edge assignements whose composition in the inductive step satisfies the constraints of the corrisponding statement. One basic edge assignment is the maximal n-edge assignment in the n-hypercube. (For details, the reader is referred to the full paper available at the web-site of the first author.)

Lemma 4.3 *There exists a maximal n-edge assignment in the n-hypercube.*

Lemmas 4.4 and 4.5 guarantee the existence of saturated m-edge assignments in the n-hypercube when $m = n-1$ and $\lfloor n/2 \rfloor + 1 \leq m \leq n-2$, respectively.

Lemma 4.4 *There exists a saturated $(n-1)$-edge assignment in the n-hypercube.*

Lemma 4.5 *Let $1 \leq m \leq n-2$. There exists a saturated m-edge assignment in the n-hypercube.*

5 The expressivity gap between CCS^{n-1} and CCS^n

Every preliminary is set for demonstrating our main result: the non existence of a philosopher process that may be freely installed in the vertices of a hypercube without hindering deadlock-freeness.

Theorem 5.1 *Let P be a CCS^{n-1} philosopher process at $\widetilde{b}$ with set of forks $\emptyset$. Let also $P_{\overline{f}}$ ($f \in F^n$) be a CCS^{n-1} code with maximal computations that only manifest the label $\overline{f}$ and that have finitely many transitions. There exists a symmetric philosopher system $([P_{\widetilde{0}}, \cdots, P, \cdots, P_{\widetilde{1}}, \prod_{f \in F^n} P_{\overline{f}}], (\emptyset, \cdots, \emptyset))$ having P at position $\widetilde{b}$ that is not deadlock-free.*

Proof: Without loss of generality, we assume P to be the process at vertex $\widetilde{0}$.

It is easy to prove that P is a philosopher process at $\widetilde{0}$ with set of forks $\emptyset$. As a consequence of this fact, if $\rho_{\widetilde{b}}$ is a bijection mapping $F_{\widetilde{0}}$ to $F_{\widetilde{b}}$ and $eat_{\widetilde{0}}$ to $eat_{\widetilde{b}}$, $\rho_{\widetilde{b}}(P)$ is a philosopher process at $\widetilde{b}$ with set of forks $\emptyset$.

As a consequence of Proposition 3.2(2), the proof reduces to defining a set $\{\rho_{\widetilde{b}} \mid \widetilde{b} \in \{0,1\}^n\}$ of bijective renamings of P such that the resulting philosopher system $([\rho_{\widetilde{0}}(P), \cdots, \rho_{\widetilde{1}}(P), \prod_{f \in F^n} P_{\overline{f}}], (\emptyset, \cdots, \emptyset))$ is not deadlock-free ($\rho_{\widetilde{0}}$ being the identity, i.e. $P = \rho_{\widetilde{0}}(P)$). Let Γ be the set of computations ζ such that:

- $\zeta = \mathtt{proj}_1([P, (\!|\, \emptyset \,|\!)] \overset{\tau}{\Longrightarrow})$ is infinite;
- $\zeta = \mathtt{proj}_1([P, (\!|\, \emptyset \,|\!)] \overset{\tau}{\Longrightarrow} [P', Q])$ is finite and $P' \not\longrightarrow$;
- $\zeta = \mathtt{proj}_1([P, (\!|\, \emptyset \,|\!)] \overset{\tau}{\Longrightarrow} [P', Q])$ is finite and $P' \xrightarrow{\overline{eat_{\widetilde{0}}}}$ is the unique possible transition.

If Γ has an infinite computation then every symmetric system containing P is not deadlock free. In fact, in this case, there is a divergent computation of P. If Γ has a finite computation where, in the last state P', the philosopher is blocked then let $\{f^{\widetilde{0}}_1, \cdots, f^{\widetilde{0}}_m\}$ be the forks retained by P'. The argument is similar to the one below (and depends on the value of m).

Otherwise, ζ may be decomposed into $P \overset{\widetilde{\mu}}{\Longrightarrow} P' \xrightarrow{f^{\widetilde{0}}_{m+1}, \cdots, f^{\widetilde{0}}_n} P'' \overset{\tau}{\Longrightarrow} P'''$ with $P \overset{\widetilde{\mu}}{\Longrightarrow} P'$ being $(\emptyset, \{f^{\widetilde{0}}_1, \cdots, f^{\widetilde{0}}_m\})$-proper.

Take the computation ζ in Γ with longest prefix $P \overset{\widetilde{\mu}}{\Longrightarrow} P'$.

A few remarks are in order:

i. The computation $P \overset{\widetilde{\mu}}{\Longrightarrow} P'$ is leading the philosopher at $\widetilde{0}$ to the state *precedent to the one where his grabbing is complete*. This state must exist because the computation ζ is $(\emptyset, F_{\widetilde{0}})$-proper.

ii. The transitions of the computation $P'' \stackrel{\tau}{\Longrightarrow} P'''$ cannot use labels in $F_{\widetilde{b}} \cup \overline{F_{\widetilde{b}}}$, otherwise the computation $P \stackrel{\widetilde{\mu}}{\Longrightarrow} P'$ is not the longest one.

iii. There may be several possible computations of the same length. We are considering one of them.

We reason by cases on m.

$1 \leq m \leq n-2$: By Lemma 4.5, there exists a saturated m-edge assignment χ in the n-hypercube. Let $\{f_1^{\widetilde{0}}, \cdots, f_m^{\widetilde{0}}\} = \chi(\widetilde{0})$ and let $\rho_{\widetilde{b}}$ be a bijective renaming such that

whenever $\chi(\widetilde{b}) \neq \emptyset$, $\{\rho_{\widetilde{b}}(f_1^{\widetilde{0}}), \cdots, \rho_{\widetilde{b}}(f_m^{\widetilde{0}})\} = \chi(\widetilde{b})$ and $\{\rho_{\widetilde{b}}(f_{m+1}^{\widetilde{0}}), \cdots, \rho_{\widetilde{b}}(f_n^{\widetilde{0}})\} = F_{\widetilde{b}} \setminus \chi(\widetilde{b})$;

otherwise $\{\rho_{\widetilde{b}}(f_1^{\widetilde{0}}), \cdots, \rho_{\widetilde{b}}(f_n^{\widetilde{0}})\} = F_{\widetilde{b}}$.

Let $\mathbf{N} = [\rho_{\widetilde{0}}(P) \mid \cdots \mid \rho_{\widetilde{1}}(P) \mid \prod_{f \in F^n} P_{\overline{f}}]$. Since χ is saturated and because of the hypothesis on $P_{\overline{f}}$, there exists a computation $\mathbf{N} \Longrightarrow \mathbf{N}'$, where the philosopher at $\widetilde{b}$ is either in the state $\rho_{\widetilde{b}}(P')$ or in a state reachable from $\rho_{\widetilde{b}}(P)$ where a (strict) subset of forks in $F_{\widetilde{b}}$ has been grabbed. (This second scenario is not possible when $1 \leq m \leq \lfloor n/2 \rfloor$ because, in this case, for every $\widetilde{b}$, $\chi(\widetilde{b}) \neq \emptyset$.) In both cases, $\rho_{\widetilde{b}}(P)$ blocks either on the transition $\xrightarrow{\rho_{\widetilde{b}}(g_1^{\widetilde{0}}), \cdots, \rho_{\widetilde{b}}(g_{m'}^{\widetilde{0}})}$ or on another input-labelled transition (because he cannot grab more than $m-1$ forks). So the network $\mathbf{N}'$ is deadlocked.

$m = n-1$: Let χ be the saturated $(n-1)$-fork assignment of Lemma 4.4. We define the renaming $\rho_{\widetilde{b}}$ from P in such a way that $\{\rho_{\widetilde{b}}(f_1^{\widetilde{0}}), \cdots, \rho_{\widetilde{b}}(f_{n-1}^{\widetilde{0}})\} = \chi(\widetilde{b})$. The proof is similar to the previous case. □

Theorem 5.1 is the main ingredient of the proof that CCS^n is not encodable into CCS^{n-1}. Still, before this proof, we need to set the requirements for the notion of encoding. Following Palamidessi [22], let $[\![\cdot]\!]$ be *uniform* if

- it is *compositional*;
- it is *name invariant*, namely for every injective renaming θ on names of P there exists an injective renaming θ' such that $[\![(\theta)(P)]\!] = (\theta')([\![P]\!])$

Compositionality ensures that the encoding of a compound process must be expressed in terms of the encoding of its components. However, in a distributed context (as the one of philosophers systems), the requirement of compositionality is usually strengthened by requiring the *homomorphism with respect to* " $|$", namely

$$[\![P \mid Q]\!] = [\![P]\!] \mid [\![Q]\!] \ .$$

That is, the encoding must preserve the degree of distribution of the processes (parallel processes remain in parallel) *and* cannot introduce additional processes that might act as coordinators.

The requirement of name invariance means that the encoding does not depend on the identity of free (channel) names, which is understandable as long as one wants liberal installations of processes in the network. We assume that the renamings θ and θ' map names into names (the encoding uses a *strict renaming policy* [10]). In addition, as in [19], Section 7, the name invariance constraint is strengthened into $\theta(a) = \theta'(a)$, for every *relevant free name*. This further restriction aims at substantiating that external resources must be accessed in the same way by the process and its encoding. In philosopher systems, the relevant free names are the forks and names *eat*. Therefore we will assume $\theta(f_{\widetilde{b},\widetilde{b}'}) = \theta'(f_{\widetilde{b},\widetilde{b}'})$ and $\theta(eat_{\widetilde{b}}) = \theta'(eat_{\widetilde{b}})$.

Finally, let $[\![\cdot]\!]$ be *semantically reasonable* if it preserves the relevant observables and the termination properties. Observables that are usually relevant in programming languages are defined in terms of *tests*. In sequential languages, the tests amount to verify the termination of programs when they are supplied with arguments (*cf.* Morris' semantics). In concurrent languages, the tests verify the presence of interactions on given names when the process is plugged into a parallel context. In our case, we will assume the encoding $[\![\cdot]\!]$ be *success-sensitive* [10], namely the tester process O contains a special name ✓ and, for every P, O, $P \mid O \stackrel{\tau}{\Longrightarrow}\xrightarrow{\checkmark}$ if and only if $[\![P]\!] \mid [\![O]\!] \stackrel{\tau}{\Longrightarrow}\xrightarrow{\checkmark}$. As regards termination, reasonableness constrains the encoding in not introducing divergence.

Corollary 5.2 *There exists no uniform, semantically reasonable encoding of* CCS^n *into* CCS^{n-1}.

Proof: Assume to the contrary that $[\![\cdot]\!]$ is a uniform, semantically reasonable encoding of CCS^n into CCS^{n-1}. Take the code in CCS^n:

$$\begin{aligned} P = & [f_1, \cdots, f_n].\overline{eat_{\widetilde{0}}}. \\ & (\textstyle\prod_{f \in F_{\widetilde{0}}} \overline{f} \mid \mathtt{rec}\, x.[f_1, \cdots, f_n].\overline{eat_{\widetilde{0}}}.(\prod_{f \in F_{\widetilde{0}}} \overline{f} \mid x)) \end{aligned}$$

where $f_1, \cdots, f_n$ are the forks adjacent to the philosopher at $\widetilde{0}$. It is easy to demonstrate that P is a philosopher process at $\widetilde{0}$ with set of forks $\emptyset$.

Consider the CCS^{n-1} processes $[\![P]\!]$ and $[\![\overline{f}]\!]$. We first demonstrate that $[\![P]\!]$ is a philosopher process at $\widetilde{0}$ with set of forks $\emptyset$. If this is not the case, let ζ be a computation of $[[\![P]\!] \mid (\!| \emptyset |\!)]$ such that $\mathtt{proj}_1(\zeta) = P \xrightarrow{\eta_1} P_1 \xrightarrow{\eta_2} P_2 \xrightarrow{\eta_3} \cdots \xrightarrow{\eta_h} P_h$ is not $\emptyset$-proper and $P \xrightarrow{\eta_1} P_1 \xrightarrow{\eta_2} P_2 \xrightarrow{\eta_3} \cdots \xrightarrow{\eta_{h-1}} P_{h-1}$ is $(\emptyset, F)$-proper, for some F. Let $\widetilde{\eta}$ be the subsequence of $\eta_1 \cdots \eta_{h-1}$ without labels τ and let ν be η_h, if $\eta_h \neq \tau$, ν be $\overline{f}$ if η_h is a synchronization between transitions with names in $F_{\widetilde{0}} \setminus F$ and $f \in F_{\widetilde{0}} \setminus F$ is one of the names involved in the synchronization. (There is no other case because of $(\emptyset, F)$-properness.)

Define $\mathcal{O}(\widetilde{\mu})$ as follows:

$$\begin{array}{rcl}
\mathcal{O}(\varepsilon) & = & \checkmark \\
\mathcal{O}(\overline{eat_{\widetilde{0}}}) & = & eat_{\widetilde{0}} \\
\mathcal{O}((f_1, \cdots, f_k)\,\widetilde{\mu}) & = & (a)(\prod_{i\in 1..k} \overline{f_i}.\overline{a} \\
 & & \quad \mid \underbrace{a.\cdots.a}_{k \text{ times}}.\mathcal{O}(\widetilde{\mu})) \\
\mathcal{O}((\overline{f_1}, \cdots, \overline{f_k})\,\widetilde{\mu}) & = & f_1.\cdots.f_k.\mathcal{O}(\widetilde{\mu})
\end{array}$$

Notice that $\mathcal{O}(\widetilde{\mu})$ is a term in CCS^1.

It turns out that $[\![P]\!] \mid \mathcal{O}(\widetilde{\eta}\nu) \overset{\checkmark}{\Longrightarrow}$ while $P \mid \mathcal{O}(\widetilde{\eta}\nu) \overset{\checkmark}{\not\Longrightarrow}$. Therefore $[\![P]\!]$ has to be a philosopher process at $\widetilde{0}$ with set of forks $\emptyset$. We also remark that every maximal $\emptyset$-proper computation of $[\![P]\!]$ must have infinitely many transitions labelled $\overline{eat_{\widetilde{0}}}$.

As regards the process $[\![\overline{f}]\!]$, since the computations of $\overline{f}$ are finite and have label $\overline{f}$, then the computations of $[\![\overline{f}]\!]$ must retain the same properties.

By Theorem 5.1, there exists a family $\rho_{\widetilde{b}}$ of bijective renamings such that the system

$$([\rho_{\widetilde{0}}([\![P]\!]), \cdots, \rho_{\widetilde{1}}([\![P]\!]), \prod_{f\in F^n} [\![\overline{f}]\!]], (\emptyset, \cdots, \emptyset))$$

(with $\rho_{\widetilde{0}}$ being the identity) is a symmetric philosopher system that is not deadlock free. Because the encoding is name invariant and the domain of $\rho_{\widetilde{b}}$ are the relevant free names $F_{\widetilde{0}}$ and $eat_{\widetilde{0}}$, then $\rho_{\widetilde{b}}([\![P]\!]) = [\![\rho_{\widetilde{b}}(P)]\!]$ (the two renamings of the constraint "renaming preserving" are identical). We may conclude by observing that

$$\begin{array}{l}
[\rho_{\widetilde{0}}([\![P]\!]) \mid \cdots \mid \rho_{\widetilde{1}}([\![P]\!]) \mid \prod_{f\in F^n} [\![\overline{f}]\!]] \\
\qquad = \quad [\![[\rho_{\widetilde{0}}(P) \mid \cdots \mid \rho_{\widetilde{1}}(P) \mid \prod_{f\in F^n} \overline{f}]]\!]
\end{array}$$

and that $([\rho_{\widetilde{0}}(P) \mid \cdots \mid \rho_{\widetilde{1}}(P) \mid \prod_{f\in F^n} \overline{f}], (\emptyset, \cdots, \emptyset))$ is a deadlock-free symmetric philosopher system. Therefore $[\![\cdot]\!]$ cannot be uniform and semantically reasonable. $\Box$

We notice that a similar result may be proved for mobile calculi à la pi-calculus with joint inputs that are reminiscent of join calculus [8]. One such language is described in [12, 17].

6. CCS with joint prefixes

The synchronization pattern of CCS^n matches many output processes with exactly one receptor. It induces an unidirectional information flow as soon as the language is enriched with messages. This lack of flexibility may be inadequate when different multi-party synchronizations are required. We discuss this issue by analyzing the encodings of the (guarded) choice in CCS. In the following $[a].P$ is abbreviated into $a.P$.

The *input guarded choice* in CCS, written $\sum_{i\in I} a_i.P_i$, is defined by the transition rule $\sum_{i\in I} a_i.P_i \xrightarrow{a_j} P_j$. There is a straightforward encoding of this choice in CCS^2, which has been already used in join calculus:

$$[\![\sum_{i\in I} a_i.P_i]\!] \; = \; (\ell)(\prod_{i\in I} [a_i, \ell].[\![P_i]\!] \; \mid \; \overline{\ell})$$

where $\ell \notin \mathtt{fn}(\sum_{i\in I} a_i.P_i)$. (The encoding is uniform and semantically reasonable: it is possible to demonstrate that $P \xrightarrow{\alpha} P'$ if and only if $[\![P]\!] \xrightarrow{\alpha}\equiv [\![P']\!]$, with $\equiv$ being the structural congruence, which is stronger than the corresponding statement in [18].) A more expressive operation then input guarded choice is *mixed choice* [18], written $\sum_{i\in I} \alpha_i.P_i$, where $\alpha_i \in \mathcal{N} \cup \overline{\mathcal{N}}$. In this case, the synchronization pattern has to coordinate the addends, regardless the fact they are inputs or outputs. This requirement of symmetry in the synchronizations makes things go wrong: in fact we are not aware of any uniform and semantically reasonable translation of mixed choice in CCS^n. It is worth to observe that the extension of CCS^2 with *output-guarded choices* bears the following simple translation of mixed choice. Let $I = I' \cup I''$ such that $\{\alpha_i \mid i \in I'\} \subseteq \mathcal{N}$ and $\{\alpha_i \mid i \in I''\} \subseteq \overline{\mathcal{N}}$. Then

$$[\![\sum_{i\in I} \alpha_i.P_i]\!] \; = \; (\ell)(\prod_{i\in I'} [\alpha_i, \ell].[\![P_i]\!] \; \mid \; (\overline{\ell} + \sum_{j\in I''} \alpha_j.[\![P_j]\!]))$$

(the correctness of the encoding may be proved similarly to the case of input-guarded choice).

The above remarks pave the way for a calculus that is alternative to CCS^n. It turns out there is a solution of the problem of encoding mixed choice in a choice-free calculus by replacing joint inputs with *joint prefixes*.

Let α, possibly indexed, range over $\mathcal{N} \cup \overline{\mathcal{N}}$ and let $\overline{\overline{a}} = a$. The calculus CCS^{n+}, called CCS with n-joint prefixes, is CCS^n where outputs and inputs are replaced by the *joint prefix*

$$[\alpha_1, \cdots, \alpha_m].P$$

with $1 \leq m \leq n$. As for CCS^n, the term $\alpha_1, \cdots, \alpha_m$ represents a sequence (now of names and co-names). With an abuse of notation, let μ, η range over sequences $\alpha_1, \cdots, \alpha_n$ and τ; let ν range over sequences $\alpha_1, \cdots, \alpha_n$ or ε (the empty sequence). Let $\overline{\nu} = \overline{\alpha_1}, \cdots, \overline{\alpha_n}$ if $\nu = \alpha_1, \cdots, \alpha_n$ and $\overline{\nu} = \varepsilon$ if $\nu = \varepsilon$. Let also $\alpha \in \nu$ if α occurs in ν. When $\alpha \in \nu$, let $\nu \setminus \alpha$ be

- ε if $\nu = \alpha$;
- $\alpha_2, \cdots, \alpha_m$, if $\nu = \alpha, \alpha_2, \cdots, \alpha_m$;
- $\alpha_1, (\alpha_2, \cdots, \alpha_m) \setminus \alpha$, otherwise.

The operations $\in$ and $\setminus$ are extended to sequences of prefixes as follows:

– $\varepsilon \in \nu$ and $\nu \setminus \varepsilon = \nu$;

– $\nu \setminus (\alpha_1, \cdots, \alpha_m) = (\cdots (\nu \setminus \alpha_1) \setminus \cdots \setminus \alpha_m)$;

– $\alpha_1, \cdots, \alpha_m \in \nu$ if, for every i, $\alpha_i \in (\nu \setminus (\alpha_1, \cdots, \alpha_{i-1}))$.

The operational semantics of CCS^{n+} is defined by the following rules (plus the symmetric ones for $|$ and where we are letting $\varepsilon, \varepsilon = \tau$ and $\varepsilon, \nu = \nu, \varepsilon = \nu$):

$$[\alpha_1, \cdots, \alpha_n].P \xrightarrow{\alpha_1, \cdots, \alpha_n} P$$

$$\frac{P \xrightarrow{\mu} Q \quad a \notin \mu}{(a)P \xrightarrow{\mu} (a)Q} \qquad \frac{P\{{}^{\mathtt{rec}\, x.P}/_{x}\} \xrightarrow{\mu} Q}{\mathtt{rec}\, x.P \xrightarrow{\mu} Q}$$

$$\frac{P \xrightarrow{\mu} Q}{P \mid P' \xrightarrow{\mu} Q \mid P'} \qquad \frac{\begin{array}{c} P \xrightarrow{\mu} Q \qquad P' \xrightarrow{\eta} Q' \\ \mu \neq \tau \neq \eta \quad \nu \in \mu \quad \overline{\nu} \in \eta \end{array}}{P \mid P' \xrightarrow{\mu \setminus \nu, \eta \setminus \overline{\nu}} Q \mid Q'}$$

The main difference with the transition relation of CCS^n is that there is no collector of the synchronizing processes – this role was played by the input process in CCS^n – but they aggregate two by two in a more symmetric way. This pairwise aggregation may not cause any synchronization, which is the case when $\nu = \varepsilon$ – similarly to the aggregation of outputs in CCS^n. The sychronization is complete when the two processes in parallel show up matching sequences. For example, the process $(a)([\overline{a}, b].P \mid [a, \overline{c}].Q) \mid [\overline{b}, c].R$ has one τ transition into $(a)(P \mid Q) \mid R$ that follows by collecting first $[\overline{a}, b].P$ and $[a, \overline{c}].Q$ and then $[\overline{b}, c].R$. The collection of the first two processes produces a "residual" label $b, \overline{c}$ that matches with the label of the third process. No other process needs to be collected because the result of the match is ε, ε, which represents a τ move.

Back to the issue of encoding the mixed choice $\sum_{i \in I} \alpha_i.P_i$, there is the following translation in CCS^{2+}:

$$[\![\sum_{i \in I} \alpha_i.P_i]\!] \;=\; (\ell)(\prod_{i \in I} [\alpha_i, \ell].[\![P_i]\!] \;\mid\; \overline{\ell}) \qquad (1)$$

Therefore $[\![a.P + \overline{b}.Q]\!]$ is $(\ell)([a, \ell].[\![P]\!] \mid [\overline{b}, \ell].[\![Q]\!] \mid \overline{\ell})$.

A relevant difference between CCS^{2+} and CCS^2 is that, in the former, more than three processes may synchronize. For example the encoding of $(a.P + \overline{b}.Q) \mid (\overline{a}.P' + b.Q')$ is

$$\begin{array}{l} (\ell)([a, \ell].[\![P]\!] \mid [\overline{b}, \ell].[\![Q]\!] \mid \overline{\ell}) \\ \mid (\ell')([\overline{a}, \ell'].[\![P']\!] \mid [b, \ell'].[\![Q']\!] \mid \overline{\ell'}) \end{array}$$

that requires the cooperation of four parallel processes (for sorting out the right choices). Another difference between CCS^n and CCS^{n+} is that the hierarchy CCS^{n+} flattens after 3.

Proposition 6.1 *Let $n \geq 3$. There exists a uniform, semantically reasonable encoding of CCS^{n+} into CCS^{3+}.*

Proof: The encoding $[\![\cdot]\!]$ is homomorphic with respect to every operation except the prefix $[\alpha_1, \cdots, \alpha_n].P$ whose definition is:

$$\begin{array}{rl} [\![[\alpha_1, \cdots, \alpha_n].P]\!] = (\ell_1, \cdots, \ell_n)(& [\ell_1, \alpha_1, \overline{\ell_2}].0 \\ & \mid [\ell_2, \alpha_2, \overline{\ell_3}].0 \\ & \cdots \\ & \mid [\ell_n, \alpha_n, \overline{\ell_1}].[\![P]\!] \;) \end{array}$$

where $\ell_1, \cdots, \ell_n$ are fresh names. It is easy to prove that $P \xrightarrow{\mu} P'$ if and only if $[\![P]\!] \xrightarrow{\mu}\equiv [\![P']\!]$, where $\equiv$ is the structural congruence [15]. $\square$

Yet, an expressivity gap remains between two and three. This gap may be shown by the dining philosophers problem in the cube. The point is that, while in CCS^{2+} it is possible to synchronize as many processes as needed, the overall effect of synchronizations is to exhibit *at most* two labels. This is too restrictive when more than two resources must be grabbed at once. The proof is omitted because it is similar to those of Theorem 5.1 and Corollary 5.2.

Proposition 6.2 *There exists no uniform, semantically reasonable encoding of* CCS^{3+} *into* CCS^{2+}.

The relationship between CCS^{2+} and the hierarchy CCS^n remains an open issue. While it is possible to encode the mixed choice into CCS^{2+} – see (1) –, we have not been able to define a uniform, semantically reasonable encoding of mixed choice in CCS^n. On the contrary, there are solutions of the dining philosophers problem in the cube in CCS^3 and there are not in CCS^{2+}.

7 Conclusions

We have proved the non-existence of a uniform, fully distributed translation of synchronizations of $n + 1$ processes into synchronizations of n processes that retains a "reasonable" semantics. As pointed out by Lehmann and Rabin [14], this implies that there is no truly distributed implementation of operators synchronizing more than three processes.

In [14], the dining philosophers problem is solved by means of a probabilistic protocol, which is correct as long as every fork is shared by exactly two philosophers [11]. Since this is the case for philosophers in the n-hypercube, we conjecture that there is a probabilistic protocol written in CCS^{n-1} for the dining philosophers problem in the n-hypercube. This probabilistic solution, if any, in turns can bring to a CCS^{n-1} implementation of CCS^n, as proved in [20].

Our final comment is about the problem of translating $n+1$ synchronizations in stochastic calculi into n synchronizations. As a consequence of our results, the corresponding protocols should match one transition with a (possible infinite) sequence of transitions. However, this matching might hardly preserve the stochastic semantics for two reasons. First it is not clear the rate of transitions in sequences that are infinite. Second, the stochastic semantics defines an exponential law controlling the waiting time before a transition can be fired (the so-called sojourn time). Matching an exponential distribution with a sum of exponential distributions is, in general, not possible.

Acknowledgements. We thank Roberto Gorrieri and Gianluigi Zavattaro for the helpful comments and discussions about the operational semantics of CCS^n. We also thank Ivan Lanese and the reviewers of LICS for the excellent work of revision. The authors are member of the joint FOCUS Research Team INRIA/Università di Bologna.

References

[1] B. Alspach, J.-C. Bermond, and D. Sotteau. Decomposition into cycles I: Hamilton decompositions. In *Proceedings of 1987 Cycles and Rays Colloquium, Montréal*, pages 9–18. NATO ASI Ser. C, Kluwer Academic Publishers, Dordrech, 1990.

[2] Roberto M. Amadio, Ilaria Castellani, and Davide Sangiorgi. On bisimulations for the asynchronous pi-calculus. *Theor. Comput. Sci.*, 195(2):291–324, 1998.

[3] K. Mani Chandy andiln Jayadev Misra. The drinking philosopher's problem. *ACM Trans. Program. Lang. Syst.*, 6(4):632–646, 1984.

[4] Stephen D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *J. ACM*, 31(3):560–599, 1984.

[5] Marco Carbone and Sergio Maffeis. On the expressive power of polyadic synchronisation in pi-calculus. *Nord. J. Comput.*, 10(2):70–98, 2003.

[6] Luca Cardelli. Strand algebras for DNA computing. In *DNA Computing and Molecular Programming'09*, volume 5877 of *Lecture Notes in Computer Science*, pages 12–24. Springer, 2009.

[7] Edsger W. Dijkstra. Hierarchical ordering of sequential processes. *Acta Inf.*, 1:115–138, 1971.

[8] Cédric Fournet and Georges Gonthier. The reflexive CHAM and the join-calculus. In *POPL*, pages 372–385, 1996.

[9] Nissim Francez and Michael Rodeh. A distributed abstract data type implemented by a probabilistic communication scheme. In *FOCS*, pages 373–379. IEEE, 1980.

[10] Daniele Gorla. Towards a unified approach to encodability and separation results for process calculi. In *CONCUR'08*, volume 5201 of *Lecture Notes in Computer Science*, pages 492–507. Springer, 2008.

[11] Oltea Mihaela Herescu and Catuscia Palamidessi. On the generalized dining philosophers problem. In *Proceedings of the 20th ACM Symposium on Principles of Distributed Computing*, pages 81–89, 2001.

[12] Cosimo Laneve and Luca Padovani. Smooth orchestrators. In *FoSSaCS'06*, volume 3921 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 2006.

[13] Cosimo Laneve and Antonio Vitale. Expressivity in the kappa family. *Electr. Notes Theor. Comput. Sci.*, 218:97–109, 2008.

[14] Daniel J. Lehmann and Michael O. Rabin. On the advantages of free choice: A symmetric and fully distributed solution to the dining philosophers problem. In *POPL*, pages 133–138, 1981.

[15] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.

[16] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, i and ii. *Inf. Comput.*, 100(1):1–77, 1992.

[17] Uwe Nestmann. On the expressive power of joint input. *Electr. Notes Theor. Comput. Sci.*, 16(2), 1998.

[18] Uwe Nestmann and Benjamin C. Pierce. Decoding choice encodings. *Inf. Comput.*, 163(1):1–59, 2000.

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

[20] Catuscia Palamidessi and Oltea Mihaela Herescu. A randomized encoding of the pi-calculus with mixed choice. *Theor. Comput. Sci.*, 335(2-3):373–404, 2005.

[21] Prakash Panangaden and John Reppy. The Essence of Concurrent ML. In Flemming Nielson, editor, *ML with Concurrency*, pages 5–29. Springer Verlag, 1997.

[22] Maria Grazia Vigliotti, Iain Phillips, and Catuscia Palamidessi. Separation results via leader election problems. In *FMCO*, volume 4111 of *Lecture Notes in Computer Science*, pages 172–194. Springer, 2006.