Multi-π: a calculus for mobile multi-party and transactional communication

Roberto Gorrieri and Cristian Versari
Dipartimento di Scienze dell'Informazione, Università di Bologna,
Mura A. Zamboni, 7, 40127 Bologna, Italy
{gorrieri, versari}@cs.unibo.it

Multi-π is a conservative extension of π-calculus, enriched with an additional operator of strong prefixing, enabling the modeling of atomic sequences of actions. Multiparty synchronization can be implemented as an atomic sequence of binary synchronizations, where the leader of the synchronization is performing a (long) atomic sequence and all the other partners are performing one single complementary action. Transactional communication is the synchronization of two (or more) processes, each performing a (long) atomic sequence. Some classic problems are presented to show the rich expressiveness of the calculus. We also show that polyadic communication [Mil91] and polyadic synchronization [CM03] can be implemented in Multi-π thanks to its transactional synchronization. We provide two operational semantics for this calculus at two different levels of abstraction and prove that the two coincide.

1 Introduction

The π-calculus [MPW92], largely used in academia as a foundational language to study concurrent systems, is not expressive enough to model some useful behaviors in real-world systems. For instance, there is no means to express that certain sequences of actions are to be executed atomically, i.e., their execution is *all-or-nothing*; similarly, it is not possible to express multi-party synchronization where more than two processes communicate, nor a transactional synchronization where two (or more) partners perform a long synchronization on a sequence of actions.

In this paper, we present a conservative extension of π-calculus, called Multi-π, to model atomic actions as well as multiway and transactional synchronizations. This extension is based on the addition of the operator of *strong prefixing* (in opposition to normal prefixing), denoted $\underline{\mu}.P$, that is used to create atomic sequences. This operator was originally presented in [GMM90, GM$\overline{90}$] as an addition to CCS. Classical problems where multiway synchronization (or transactional sycnhronization) is required (implemented as an atomic sequence of binary synchronizations) can be now modeled faithfully. As typical examples of this class of problems, we will consider Patil's cigarette smokers' problem [Pat71], for which no deadlock-free and livelock-free solution exists in π-calculus. Moreover, we present a variation of Dijkstra's dining philosophers problem [Dij68], where each philosopher is the owner of one fork and asks for the second fork to his neighbor. Finally, we elaborate on Milner's example of mobile phones [Mil91], where the cell serving the phone can change during the conversation. These examples show the interplay between mobility and atomic sequences of binary communications.

Multi-π is a very expressive calculus. Besides the examples cited above, we show that (i) the choice operator is derivable easily from the others, (ii) polyadic π-calculus [Mil91, Mil99] is a subcalculus of Multi-π, and that (iii) polyadic synchronization [CM03] is a derived operator in Multi-π. Finally (iv) we present a problem, the so-called *last man standing* problem [VBG09] that cannot be solved in Multi-π.

Submitted to:

Multi-π is equipped with two operational semantics in terms of labeled transition systems: an abstract semantics where each transition is a transaction (labeled by a sequence) and a low-level one where each transition is labeled by one single action, but not all the states are observable; unobservable states represent the fact that the process is in the middle of an atomic action and so its partial behavior cannot be observed. We prove that the two semantics coincide on observable states, even if the low-level semantics can deadlock more often than the high-level semantics. For instance, the process $\underline{\mu}.\mathbf{0}+\alpha.\mathbf{0}$ can perform only one transition α in the abstract labeled transition system, while it can also perform a low-level transition labeled with μ, reaching, however, an unobservable state.

The paper is organized as follows. Section 2 presents the syntax of the calculus, recalling mainly the basic notation of π-calculus. In Section 3 we provide the high-level semantics of the calculus, together with some basic properties. Section 4 is devoted to the presentation of the case studies. Section 5 studies some expressiveness result for the calculus. Section 6 presents the low-level semantics for Multi-π and the proof that the two semantics coincide over observable states. Section 7 concludes the paper with a comparison with related literature and some hints for future research.

2 The Multi-π calculus: Syntax

Let $\mathscr{N}$ be a countable set of names, ranged over by $a,b,\ldots,x,y,\ldots$, and $\tau \notin \mathscr{N}$. Terms are generated from names by the following grammar:

$$p ::= \mathbf{0} \mid \mu.q \mid \underline{\mu}.q \mid p+p \quad \textit{sequential processes}$$

$$q ::= p \mid q|q \mid (\nu x)q \mid X \mid \mathrm{rec}X.q \quad \textit{processes}$$

where $\mathbf{0}$ is the terminated process, $\mu.q$ is a sequential process where action μ (that can be either an input $x(y)$ (or simply x), an output $\bar{x}y$ (or simply $\bar{x}$) or a silent move τ) is first performed and then q is ready (*normal prefixing*), $\underline{\mu}.q$ is a sequential process where action μ is *strongly prefixed* to q, meaning that μ is performed as the first action of an atomic action continuing with what q can do, $p+p'$ is the sequential process obtained by the alternative composition of sequential processes p and p', $q|q'$ is the parallel composition of q and q', $(\nu x)q$ is process q where the name x is made private (restriction), X is a recursion variable and $\mathrm{rec}X.q$ is a recursive process.

The set $\mathscr{P}$ of *processes* contains those terms which are, w.r.t. recursion variables, *closed* (all the variables occur in a *rec* binder) and *guarded* (all the variables occurring in the body q of a recursion occurs inside a *normally prefixed* subprocess $\mu.q'$ of q). With abuse of notation, $\mathscr{P}$ will be ranged over by $p,q,\ldots$.

The *free* names of a process p, $fn(p)$, are defined as usual. The basic labels in *Act* (ranged over by α) are of three different kinds: the silent action τ, the inputs of the form xy, the free and bound outputs of the form $\bar{x}y$ and $\bar{x}(y)$, respectively. These can be used for generating any sequence σ of actions. We assume that β ranges over inputs and free outputs only. Given $\beta = xy$, $\bar{\beta} = \bar{x}y$ denotes the complement of β; symmetrically, when $\beta = \bar{x}y$, $\bar{\beta} = xy$. The *free* and *bound* names of a label α, $fn(\alpha)$ and $bn(\alpha)$, are defined as follows: $fn(\tau) = bn(\tau) = \emptyset$, $fn(\bar{x}y) = fn(xy) = \{x,y\}$, $bn(\bar{x}y) = bn(xy) = \emptyset$, $fn(\bar{x}(y)) = \{x\}$, $bn(\bar{x}(y)) = \{y\}$. The names of α are $n(\alpha) = fn(\alpha) \cup bn(\alpha)$. These functions are extended to any sequence σ in the obvious way. For instance, $fn(\varepsilon) = \emptyset$, $fn(\alpha\sigma) = fn(\alpha) \cup fn(\sigma)$.

3 Multi-π: High-level Semantics

The (early) operational semantics for the Multi-π-calculus is the labelled transition system $(\mathscr{P}, \mathscr{A}, \longrightarrow)$, where the states are the processes in $\mathscr{P}$, $\mathscr{A} = Act^*$ is the set of labels (ranged over by σ), and $\longrightarrow \subseteq \mathscr{P} \times \mathscr{A} \times \mathscr{P}$ is the minimal transition relation generated by the rules listed in Table 1.

We briefly comment the rules that are less standard. Rules (S-in)–(S-out) allow for the creation of transitions labelled by nonempty sequences of actions. In order for, e.g., $\underline{\overline{x}z}.q$ to make a move, it is necessary that q can perform a transition, i.e., the rest of the transaction. Hence, $\underline{\overline{x}z}.\mathbf{0}$ cannot perform any action. If a transition is labeled by $\sigma = \alpha_1 \dots \alpha_n$, then all the actions $\alpha_1 \dots \alpha_{n-1}$ are due to strong prefixes, while α_n to a normal prefix.

Rule (Res) requires that no action in σ can contain the restricted name y. Rule (Cong) makes use of a structural congruence $\equiv$ on process terms induced by the following three equations:[1]

$$(p \mid q) \mid r \;=\; p \mid (q \mid r)$$
$$(\nu x)(p \mid q) = p \mid (\nu x)q \quad \text{if } x \notin fn(p).$$
$$(\nu x)p = (\nu y)(p\{y/x\}) \quad \text{if } y \notin fn(p).$$

(Cong) enlarges the set of transitions derivable from p, as Example 1 shows. Moreover, it simplifies the mechanism of scope extrusion (i.e., there is no need of an explicit (Close) rule [MPW92]).

Rule (Par) has a side condition that states that p can perform σ in the context of q if the bound names of σ do not clash with the free names of q, in order to avoid possible erroneous capture of names in q.

Rule (Com) has a side-condition on the possible synchronizability of sequences σ_1 and σ_2. Relation $Sync(\sigma_1, \sigma_2, \sigma)$ holds if σ is obtained from an interleaving (possibly with synchronizations) of σ_1 and σ_2, where the last action of one of the two sequences is to be synchronized, hence reflecting that the subtransaction that ends first signals this fact (i.e., *commits*) to the other subtransaction. Relation *Sync* is formally defined by the inductive rules reported in Table 2.

Rule (Open) states that whenever a process wants to perform an output of a name y which is actually restricted, then a bound output is used to label the transition to remember that some *extrusion* (i.e., enlargement of the scope of restriction) is now possible. Note that in the target state restriction disappears. The side-condition on applicability of the rule states that in σ there must be an output of y along some channel x, that function $Res(\sigma, y)$ is defined and its result is σ'. Function *Res* is formally defined in Table 3, where we assume that z and w are assumed different from y. Note that, contrary to the semantics of the π-calculus [MPW92], rule (Open) fixes the name of the bound output to the very same name of the restriction, namely y, instead of a fresh name w, as there is no need to apply rule (Close) which is subsumed by (Com) due to structural congruence. Moreover, due to (Cong), a restricted process $(\nu y)p$ may have infinitely many different transitions with a differently bound output, one for any alpha-converted $(\nu x)(p\{x/y\})$ with $x \notin fn(p)$.

The semantics we have presented is called *early* because in rules (In) and (S-in) the name z is replaced by name y at this syntactical level, and then rule (Com) – via *Sync* – simply matches that the sent values and the chosen input values are actually the same. A different discipline, called *late* semantics, could be adopted, according to which the substitution takes place only at the level of rule (Com). See, e.g., [SW01] for more details.

[1] We have taken the minimal set of equations that are needed for giving a semantics ensuring associativity of parallel composition.

(Tau)	$\tau.p \xrightarrow{\tau} p$	(In)	$x(z).p \xrightarrow{xy} p\{y/z\}$	$y \notin fn((\nu z)p)$
(S-tau)	$\dfrac{p \xrightarrow{\sigma} p'}{\underline{\tau}.p \xrightarrow{\sigma} p'}$	(S-in)	$\dfrac{p\{y/z\} \xrightarrow{\sigma} p'}{\underline{x(z)}.p \xrightarrow{xy\sigma} p'}$	$y \notin fn((\nu z)p)$
(Out)	$\bar{x}y.p \xrightarrow{\bar{x}y} p$	(S-out)	$\dfrac{p \xrightarrow{\sigma} p'}{\underline{\bar{x}y}.p \xrightarrow{\bar{x}y\sigma} p'}$	
(Sum)	$\dfrac{p \xrightarrow{\sigma} p'}{p+q \xrightarrow{\sigma} p'}$	(Rec)	$\dfrac{p\{\mathrm{rec}X.p/X\} \xrightarrow{\sigma} p'}{\mathrm{rec}X.p \xrightarrow{\sigma} p'}$	
(Cong)	$\dfrac{p \equiv p' \xrightarrow{\sigma} q' \equiv q}{p \xrightarrow{\sigma} q}$	(Res)	$\dfrac{p \xrightarrow{\sigma} p'}{(\nu y)p \xrightarrow{\sigma} (\nu y)p'}$	$y \notin n(\sigma)$
(Par)	$\dfrac{p \xrightarrow{\sigma} p'}{p \mid q \xrightarrow{\sigma} p' \mid q}$		$bn(\sigma) \cap fn(q) = \emptyset$	
(Com)	$\dfrac{p \xrightarrow{\sigma_1} p' \qquad q \xrightarrow{\sigma_2} q'}{p \mid q \xrightarrow{\sigma} p' \mid q'}$		$Sync(\sigma_1, \sigma_2, \sigma)$	
(Open)	$\dfrac{p \xrightarrow{\sigma} p'}{(\nu y)p \xrightarrow{\sigma'} p'}$		$\exists \bar{x}y \in \sigma$, and $Res(\sigma, y) = \sigma'$	

Table 1: SOS rules (symmetric rules of (Sum) and (Par) omitted)

$Sync(\beta, \bar{\beta}, \tau)$	$\dfrac{\sigma \neq \varepsilon}{Sync(\beta\sigma, \bar{\beta}, \sigma)}$	$\dfrac{\sigma \neq \varepsilon}{Sync(\beta, \bar{\beta}\sigma, \sigma)}$
$\dfrac{Sync(\sigma_1, \sigma_2, \sigma)}{Sync(\beta\sigma_1, \bar{\beta}\sigma_2, \sigma)}$	$\dfrac{Sync(\sigma_1, \sigma_2, \sigma)}{Sync(\beta\sigma_1, \sigma_2, \beta\sigma)}$	$\dfrac{Sync(\sigma_1, \sigma_2, \sigma)}{Sync(\sigma_1, \beta\sigma_2, \beta\sigma)}$

Table 2: Synchronization relation

$$Res(\varepsilon, y) = \varepsilon \qquad Res(xy\sigma, y) = \bot \qquad Res(yx\sigma, y) = \bot \qquad Res(zw\sigma, y) = zwRes(\sigma, y)$$

$$Res(\bar{z}y\sigma, y) = \bar{z}(y)\sigma \qquad Res(\bar{z}w\sigma, y) = \bar{z}wRes(\sigma, y) \qquad Res(\bar{y}x\sigma, y) = \bot$$

Table 3: Restriction relation Res – z and w are assumed different of y.

Example 1 **(Multi-party synchronization)** *Assume we have three processes that want to synchronize. This can be easily expressed in Multi-π. E.g., consider processes $p = \underline{a}.a.p'$ – which is the leader of the synchronization – $q = \bar{a}.q'$ and $r = \bar{a}.r'$ and the whole system $P = (\nu a)((p \mid q) \mid r)$. It is easy to observe that $P \xrightarrow{\tau} (\nu a)((p' \mid q') \mid r')$, so the three processes have synchronized in one single atomic transition. It is interesting to observe that $P' = (\nu a)(p \mid (q \mid r))$ could not perform the multiway synchronization if the structural congruence (Cong) were not allowed.* □

Example 2 **(Transactional synchronization)** *Assume we have two processes that wants to synchronize on a sequence of actions. This can be easily expressed in Multi-π. E.g., consider processes $p = \underline{a}.a.p'$ and $q = \underline{\bar{a}}.\bar{a}.q'$ and the whole system $P = (\nu a)(p \mid q)$. It is easy to observe that $P \xrightarrow{\tau} (\nu a)(p' \mid q')$, so the two processes have synchronized in one single atomic transaction. Of course, it is possible to define transactional multi-party synchronization as well. For instance, $p = \underline{\bar{a}}.\bar{b}.p'$ and $q = \underline{b}.\bar{a}.q'$, $r = \underline{a}.a.r'$, and the whole system $Q = (\nu a)((p \mid q) \mid r) \xrightarrow{\tau} (\nu a)((p' \mid q') \mid r')$.* □

Example 3 **(Guardedness)** *We have assumed that each recursion variable occurring in the body q of a recursion occurs inside a normally prefixed subprocess $\mu.q'$ of q. This will prevent infinitely branching sequential processes. For instance, consider the non legal process $p = recX.(\underline{a}.X + b.\mathbf{0})$. According to the operational rules, p has infinitely many transitions leading to $\mathbf{0}$, each one with a label of the form $a^n b$, for $n = 0, 1, \ldots$.* □

Processes of Multi-π are equipped with an observational semantics, given in terms of a strong bisimulation-based equivalence [Mil89], where we have to be careful about the free and bound names that are involved, in order to avoid the capture of free names.

Definition 1 *A binary symmetric relation R over the set $\mathscr{P}$ of process terms is an* early bisimulation *if $(p,q) \in R$ implies:*

- *if $p \xrightarrow{\sigma} p'$ and $bn(\sigma) \cap fn(p,q) = \emptyset$ there exists then q' such that $q \xrightarrow{\sigma} q'$ and $(p',q') \in R$.*

Two process terms p and q are bisimilar*, written $p \sim q$, if there exists an (early) bisimulation R such that $(p,q) \in R$.* □

Proposition 2 *Let $p,q \in \mathscr{P}$ be processes. If $p \equiv q$ then $p \sim q$.* □

Besides the three equations defining the structural congruence $\equiv$, other interesting laws hold for bisimulation equivalence.

Proposition 3 *Let $p,q,r \in \mathscr{P}$ be processes. Then the following holds:*

(1) $(p+q)+r \sim p+(q+r)$ (2) $p+q \sim q+p$
(3) $p+\mathbf{0} \sim p$ (4) $p \mid q \sim q \mid p$
(5) $p \mid \mathbf{0} \sim p$ (6) $(\nu x)(\nu y)p \sim (\nu y)(\nu x)p$
(7) $x(y).p \sim x(z).(p\{z/y\})$ *if* $z \notin fn(p)$ (8) $(\nu x)\mathbf{0} \sim \mathbf{0}$

Proof: The proof is standard. E.g., for (2) it is enough to prove that relation $R = \{((p+q),(q+p) \mid p,q \in \mathscr{P}\} \cup \{(p,p) \mid p \in \mathscr{P}\}$ is a strong bisimulation. □

As expected, the sum operator is associative, commutative and $\mathbf{0}$ absorbent. Commutativity of restriction (case (6) above) allows for a simplification in the notation that we usually adopt: restriction on a set of names, e.g., $(\nu x,y)p$. The parallel operator is commutative and $\mathbf{0}$ absorbent, but associative only because of the addition of the structural congruence and the associated rule (Cong).

A few properties of strong prefixing are as follows:

Proposition 4 *Let $p,q \in \mathscr{P}$ be processes. Then the following holds:*

(1) $\underline{\mu}.(p+q) \sim \underline{\mu}.p + \underline{\mu}.q$ (2) $\underline{\mu}.\mathbf{0} \sim \mathbf{0}$ (3) $\underline{\tau}.p \sim p$ □

4 Case studies

Example 4 **(Cigarette smokers' problem)** *This problem, proposed by Patil in [Pat71], is a typical instance of a problem for which no deadlock-free and livelock-free solution seems to exist in π-calculus. Here is its description, according to [Pet81]. The problem consists of four processes: an agent and three smokers. Each smoker continuously makes a cigarette and smokes it. But to make a cigarette, three ingredients are needed: tobacco, paper and matches. One of the smokers has paper, another tobacco and the third has matches. The agent has an infinite supply of all three. The agent places two of the ingredients on the table. The smokers who has the remaining ingredient can then make and smoke a cigarette, signalling the agent upon completion. The agent then puts out another two of the three ingredients and the cycle repeats. The problem is to define the code for the smoker processes to determine which of the three processes should proceed (the agent is fixed and cannot be changed). Clearly, a solution to the problem is to have an atomic transaction in which the agent synchronize only with the smoker that has the missing ingredient.*

$$
\begin{aligned}
Agent &\stackrel{def}{=} recX.\tau.(\overline{\underline{tob}}.\overline{mat}.end.X) + \tau.(\overline{\underline{mat}}.\overline{pap}.end.X) + \tau.(\overline{\underline{pap}}.\overline{tob}.end.X)\\
S_{pap} &\stackrel{def}{=} recX.\underline{tob}.mat.smoke.\overline{end}.X\\
S_{tob} &\stackrel{def}{=} recX.\underline{mat}.pap.smoke.\overline{end}.X\\
S_{mat} &\stackrel{def}{=} recX.\underline{pap}.tob.smoke.\overline{end}.X\\
Patil &\stackrel{def}{=} (\nu tob, pap, mat, end)(Agent \,|\, S_{tob} \,|\, S_{mat} \,|\, S_{pap})
\end{aligned}
$$

This solution, based on transactional synchronization, determines a finite-state labeled transition systems, but it is somehow not really compliant to the specification of the problem, where the ingredients are not treated as resources that are consumed and then reproduced. A more faithful representation of the problems is as follows:

$$
\begin{aligned}
Agent' &\stackrel{def}{=} recX.\tau.(\overline{tob}.\mathbf{0} \,|\, \overline{mat}.\mathbf{0} \,|\, end.X) + \tau.(\overline{mat}.\mathbf{0} \,|\, \overline{pap}.\mathbf{0} \,|\, end.X) + \tau.(\overline{pap}.\mathbf{0} \,|\, \overline{tob}.\mathbf{0} \,|\, end.X)\\
Patil' &\stackrel{def}{=} (\nu tob, pap, mat, end)(Agent' \,|\, S_{tob} \,|\, S_{mat} \,|\, S_{pap})
\end{aligned}
$$

where a multi-party synchronization takes place among a smoker and the two needed ingredients. This solution, however, generates an infinite-state transition system. [2]

Patil showed in [Pat71] that no sequence of P and V operations [Dij68] can correctly solve this problem. The reason is that P and V operations offer binary synchronization only, while this problem requires a transactional synchronization (first version) or a multiparty sycnhronization (second version). Following the same argument of Patil, one can give evidence supporting the claim that this problem cannot be solved in the π-calculus. A recent paper [LV10] also gives further support to our claim, as there it is proved that there is an expressivity gap between binary CCS-based synchronization and ternary synchronization, as required by (the second version of) this example. □

Example 5 **(The Polite Dining Philosophers)** *This alteration of the famous problem, originally proposed in [Dij71], is as follows. Five philosophers seat at a round table, with a private plate and where each of the five forks is private to one philosopher and can be shared, upon request, to the neighbor philosopher. Philosophers can think and eat; in order to eat, a philosopher has to acquire both forks, his private one (the left one) and the one of his right neighbor, All philosophers behave the same, so the problem is intrinsically symmetric. Clearly a naïve solution would cause deadlock exactly when all five philosophers take their private fork at their left at the same time and are waiting for the fork at their*

[2] Note, however, that, by adding the equation $p \,|\, \mathbf{0} = p$ to those defining the structural congruence $\equiv$ and by identifying congruent states, the resulting labeled transition systems would be finite.

right. A simple solution is to force atomicity on the acquisition of the two forks so that either both are taken or none. The forks can be defined as

$$fork_i \stackrel{def}{=} recX.\overline{up_i}.\overline{dn_i}.X \quad for\ i = 0,\ldots 4$$

The philosophers can be described as

$$\begin{aligned} phil_i \stackrel{def}{=}\ & recX.(think.X + \underline{up_i}.\underline{req_{i+1}}(x).\underline{x}.eat.\underline{dn_i}.\underline{rel_{i+1}}(y).\underline{y}.X \\ & + \overline{req_i}up_i.recY.(think.Y + \overline{rel_i}dn_i.X)) \end{aligned}$$

for $i = 0,\ldots 4$, where $i+1$ is computed modulo 5. For simplicity, we assume also that the release of the two forks is atomic, but this is not necessary for correctness. The whole system is

$$P_i \stackrel{def}{=} (\nu\, up_i, dn_i)(phil_i \,|\, fork_i) \quad for\ i = 0,\ldots 4$$

$$DF \stackrel{def}{=} (\nu\, L)(P_0 \,|\, P_1 \,|\, P_2 \,|\, P_3 \,|\, P_4)$$

where $L = \{req_i, rel_i \mid i = 0,\ldots 4\}$. Note that the semantics generates a finite-state labeled transition system for DF. This kind of solution to the dining philosophers problem cannot be given in a language which admits binary communication only, such as the π-calculus, according to [LR81, LV10]. □

Example 6 **(Mobile Transaction)** *Let us now consider a variation of the mobile telephone example in [Mil99]. Here a car, with a mobile phone on it, is moving and the service to the mobile phone is granted by one of the two base stations. Each base station is linked to a centre, which handles the stream of data coming from the base stations. In the following example, the mobile phone on the car modeled by CAR, is linked to BASE, so to represent a call in progress: the exchange of data between the phone and the base occurs on channel t_1. At some point, as a consequence of the movement of the car, the phone detects (on channel sgn) the presence of the (presumably stronger) wireless signal b_2 of another base station, represented by IDLEBASE. In order to continue the call in progress, it starts a transaction with the aim of switching the communication stream from the current BASE to the closer IDLEBASE. The transaction is opened by sending the detected signal b_2 (used later on as a channel) to BASE, which forwards the switch request over channel c to the CENTRE, by specifying first the channel b_1 of the current BASE, and then the channel b_2 of the IDLEBASE that is going to support the communication of the mobile phone. The CENTRE creates a new channel t over which the voice data stream between CAR and IDLEBASE is going to flow, and communicates it first over channel b_2 to IDLEBASE (which becomes active as BASE) and then over b_1 to the current BASE, which in turn forwards it to the CAR. The whole system is represented by the following code:*

$$\begin{aligned} & (\nu\, c, sgn, t_i, b_i, i = 1,2) \\ & \big(\ recCAR.(t_1.CAR + \\ & \qquad \underline{sgn(b_2)}.\overline{\underline{b_1}}\underline{b_2}.b_1(t_2).recCAR'.(t_2.CAR' + \\ & \qquad\qquad \underline{sgn(b_1)}.\overline{\underline{b_2}}\underline{b_1}.b_2(t_1).CAR)) \ \mid \\ & \quad recBASE.(\overline{t_1}.BASE + \\ & \qquad \underline{b_1(b_2)}.\underline{\bar{c}b_1}.\underline{\bar{c}b_2}.\underline{b_1(t_2)}.\overline{b_1}t_2. \\ & \qquad recIDLEBASE.(\overline{sgn}b_1.IDLEBASE + b_1(t_1).BASE)) \ \mid \\ & \quad recIDLEBASE.(\overline{sgn}b_2.IDLEBASE + b_2(t_2). \\ & \qquad recBASE.(\overline{t_2}.BASE + \underline{b_2(b_1)}.\underline{\bar{c}b_2}.\underline{\bar{c}b_1}.\underline{b_2(t_1)}.\overline{b_2}t_1.IDLEBASE)) \ \mid \\ & \quad recCENTRE.(\underline{c(b_1)}.\underline{c(b_2)}.(\nu t)(\overline{\underline{b_2}}t.\overline{b_1}t.CENTRE))\big) \end{aligned}$$

Strong prefixing allows us to represent the whole process of stream switching from one BASE to the other as an atomic transaction, so that there is no need to handle explicitly the possible race condition occurring when two distinct CARs try to switch their communication to the same IDLEBASE. □

5 Expressiveness

In this section we analyze the expressiveness of Multi-π in different respects. We first show that the strong prefixing operator is extremely powerful, in fact it allows us to derive several other operators straightforwardly. Next we show that such operator, although very expressive, is *not* powerful enough to solve the *Last Man Standing*, a problem of distributed computing introduced in [VBG09], apparently solvable only by calculi with inherent centralized implementation of parallel composition.

These results strengthen the argument that strong prefixing is the best trade-off between the expressiveness gained by a concurrent calculus and the effective parallelism that it can still retain.

5.1 Choice

Strong prefixing can be exploited to give an encoding of the choice operator equivalent with respect to strong bisimilarity, so that Multi-π turns out to be, to some extent, redundant. The encoding of the choice is based on the same idea first presented in [NP96], except that, thanks to strong prefixing, there is no need of separating input and output choices, nor of taking care of symmetry and convergence, which are all preserved. For example, if we want to encode the process $s = x(y).p' + \bar{w}z.q'$ in a process s' without choice operator, we can split the choice in two parallel processes p and q sharing a private channel c, over which a third process r acts as a semaphore:

$$s' = (\nu c)(p \,|\, q \,|\, r) = (\nu c)(\underline{c}.x(y).p' \,|\, \underline{c}.\bar{w}z.q' \,|\, \bar{c})$$

The observable behavior of s and s' is exactly the same, in fact both allow the transitions $\xrightarrow{x(y)} p'$ and $\xrightarrow{\bar{w}z} q'$. The only (formally unobservable) difference between them is the deadlocked garbage left by s'. In general, the encoding of $\mu_1.p_1 + \mu_2.p_2 + \ldots + \mu_n.p_n$ becomes $(\nu c)(\underline{c}.\mu_1.p_1 \,|\, \underline{c}.\mu_2.p_2 \,|\, \cdots \,|\, \underline{c}.\mu_n.p_n \,|\, \bar{c})$. We can formalize this encoding as follows:

Proposition 5 *Let Multi-π^{-c} be the language Multi-π without choice operator and let $[\![\cdot]\!]^c$ be the function from Multi-π to Multi-π^{-c} processes defined homomorphically with respect to all the operators, except for choice as follows:*

$$[\![p_1 + p_2]\!]^c = (\nu x)(\bar{x} \,|\, [\![p_1]\!]^c_x \,|\, [\![p_2]\!]^c_x) \qquad x \notin fn(p_1 + p_2)$$

with $[\![\cdot]\!]^c_x$ defined over sequential processes as

$$[\![\mathbf{0}]\!]^c_x = \mathbf{0} \qquad [\![\mu.q]\!]^c_x = \underline{x}.\mu.[\![q]\!]^c$$
$$[\![\underline{\mu}.q]\!]^c_x = \underline{x}.\underline{\mu}.[\![q]\!]^c \qquad [\![p_1 + p_2]\!]^c_x = [\![p_1]\!]^c_x \,|\, [\![p_2]\!]^c_x$$

Then, for every process q of Multi-π, $q \sim [\![q]\!]^c$. □

5.2 Polyadic π-calculus

Polyadic communication – consisting in the atomic transmission of an arbitrary number n of names over some channel x – can be easily obtained thanks, again, to strong prefixing. The easiest and correct encoding is (a strong prefixing-based variation of) the one considered in [Mil91] for the standard π-calculus: for example, the transmission of the pair of names a_1, a_2 over channel x (written in polyadic

form as $\bar{x}\langle a_1, a_2\rangle$ for the output and $x(z_1, z_2)$ for the input) can be implemented by first communicating a fresh, private channel name c, and then sending over c the intended names:

$$(\nu c)(\underline{\bar{x}c}.\underline{\bar{c}a_1}.\bar{c}a_2.p') \,|\, \underline{x(y)}.\underline{y(z_1)}.y(z_2).q' \xrightarrow{\tau} (\nu c)(p' \,|\, q'\{a_1/z_1, a_2/z_2\})$$

provided that c, y do not appear free in q'. The encoding of a choice between two or more polyadic actions, e.g. $\bar{x_1}\langle a_1, a_2\rangle + \bar{x_2}\langle a_3, a_4\rangle$, can be obtained by using just one private name c for all the mutually exclusive actions: this allows both the minimization of the number of private names needed for the encoding and the conformity to the grammar of Multi-π, that forbids creation of private names directly in the branches of the choice operator.

It is worth remarking that strong prefixing extinguishes any problem of channel typing, deriving from the need of preventing polyadic communication between those pairs of processes whose number of transmitted (resp. received) names does not match with the number of names of the receiver (resp. sender). For example, the encoding of the pair $p = \bar{x}\langle a_1, a_2, a_3\rangle$ and $q = x(z_1, z_2)$ would lead to the unintentional deadlock of the process p in the absence of channel typing or strong prefixing.

Formally, the encoding of polyadic communication by means of strong prefixing does not provide processes strictly equivalent, because of the different observables. However, the semantics of the original process is preserved in terms of full abstraction with respect to strong bisimilarity:

Proposition 6 *Let Multi-π^{+pc} be the language Multi-π with the addition of polyadic communication, that is with prefixes of the form $\mu' = \tau \mid x(y_1, \ldots, y_n) \mid \bar{x}\langle y_1, \ldots, y_n\rangle$ for any $n \in \mathbb{N}$ and SOS rules generalized accordingly (for details, see Appendix A). Let also $[\![\cdot]\!]^{pc}$ be the function from Multi-π^{+pc} to Multi-π processes defined homomorphically with respect to terminated process, parallel composition, restriction, process variable and recursion, and defined as follows for the remaining operators:*

$$\begin{array}{ll}
[\![p_1 + p_2]\!]^{pc} = (\nu x)([\![p_1]\!]^{pc}_x + [\![p_2]\!]^{pc}_x) & x \notin fn(p_1 + p_2) \\
[\![\mu'.q]\!]^{pc} = (\nu x)([\![\mu'.q]\!]^{pc}_x) & x \notin fn(\mu'.q),\ \mu' = \bar{z}\langle y_1, \ldots, y_n\rangle \\
[\![\underline{\mu'}.q]\!]^{pc} = (\nu x)([\![\underline{\mu'}.q]\!]^{pc}_x) & x \notin fn(\underline{\mu'}.q),\ \underline{\mu'} = \underline{\bar{z}}\langle y_1, \ldots, y_n\rangle \\
[\![\mu'.q]\!]^{pc} = [\![\mu'.q]\!]^{pc}_x & \mu' = z(y_1, \ldots, y_n) \text{ or } \mu' = \tau \\
[\![\underline{\mu'}.q]\!]^{pc} = [\![\underline{\mu'}.q]\!]^{pc}_x & \underline{\mu'} = \underline{z}(y_1, \ldots, y_n) \text{ or } \underline{\mu'} = \underline{\tau}
\end{array}$$

with $[\![\cdot]\!]^{pc}_x$ defined as

$$\begin{aligned}
[\![\tau.q]\!]^{pc}_x &= \tau.[\![q]\!]^{pc} \\
[\![\underline{\tau}.q]\!]^{pc}_x &= \underline{\tau}.[\![q]\!]^{pc} \\
[\![z(y_1, \ldots, y_n).q]\!]^{pc}_x &= \underline{z(w)}.\underline{w(y_1)}.\cdots.\underline{w(y_{n-1})}.w(y_n).[\![q]\!]^{pc} \\
[\![\underline{z}(y_1, \ldots, y_n).q]\!]^{pc}_x &= \underline{z(w)}.\underline{w(y_1)}.\cdots.\underline{w(y_{n-1})}.\underline{w(y_n)}.[\![q]\!]^{pc} \\
[\![\bar{z}\langle y_1, \ldots, y_n\rangle.q]\!]^{pc}_x &= \underline{\bar{z}x}.\underline{\bar{x}y_1}.\cdots.\underline{\bar{x}y_{n-1}}.\bar{x}y_n.[\![q]\!]^{pc} \\
[\![\underline{\bar{z}}\langle y_1, \ldots, y_n\rangle.q]\!]^{pc}_x &= \underline{\bar{z}x}.\underline{\bar{x}y_1}.\cdots.\underline{\bar{x}y_{n-1}}.\underline{\bar{x}y_n}.[\![q]\!]^{pc} \\
[\![p_1 + p_2]\!]^{c}_x &= [\![p_1]\!]^{c}_x + [\![p_2]\!]^{c}_x
\end{aligned}$$

Then, for every pair of processes q_1, q_2 of Multi-π^{+pc}, $q_1 \sim q_2 \iff [\![q_1]\!]^{pc} \sim [\![q_2]\!]^{pc}$.

It is worth noticing that the parameter x of the function $[\![\cdot]\!]^{pc}_x$ is not used in the encoding of the input and silent action.

5.3 Polyadic Synchronization

Polyadic synchronization [CM03] extends a language by structuring channels: instead of being composed of just one name, each channel can be represented by a tuple of names $x_1@x_2@\ldots@x_k$ of arbitrary length. The synchronization between two processes is consequently allowed only if each name x_i of the output action matches with the corresponding name of the input action, so that, for example, the two processes $p = \bar{x}@\bar{y}\langle a\rangle.p'$ and $q = x@y(z).q'$ are allowed to communicate, while p and $r = x@w(z).r'$ are not allowed, provided that $y \neq w$.

Polyadic synchronization can be encoded as a sequence of strong-prefixed synchronizations over the names x_i which compose the channel. In order to avoid interference with other processes, each sequence must be opened with a communication over a channel representing the arity of the synchronization (in order to avoid that a sender synchronize with more than one receiver) and closed with a synchronization check over a fresh, private name c that has been transmitted at each step of the sequence (in order to be sure that all the received messages are from the same sender). The above processes p and q can be encoded as follows:

$$(\nu c)(\underline{\overline{two}c}.\underline{\bar{x}c}.\underline{\bar{y}c}.\bar{c}a.p' \,|\, \underline{two(w)}.\underline{x(w')}.\underline{y(w'')}.\underline{w''(z)}.(\underline{w}.w.q' \,|\, \overline{w'} \,|\, \overline{w''})) \xrightarrow{\tau} (\nu c)(p' \,|\, q'\{a/z\} \,|\, \mathbf{0} \,|\, \mathbf{0})$$

with two, c, w, w', w'' not appearing free in q'. The first synchronization over the channel two ensures that the processes involved in the polyadic synchronization are ready to synchronize over the same number of names, while the last sequence of synchronizations over w, w', w'' ensures that all these names correspond to instances of the same private name w repeatedly sent by the same sender process, which excludes any interference by some other process trying to perform an output action over x and/or y.

Observationally, we obtain a similar result as for polyadic communication:

Proposition 7 *Let Multi-π^{+ps} be the language Multi-π with the addition of polyadic synchronization, that is with prefixes generalized as $\mu' = \tau \mid x_1@\ldots@x_n(y) \mid \overline{x_1@\ldots@x_n}\langle y\rangle$ for any $n \in \mathbb{N}$ and SOS rules generalized accordingly (for details, see Appendix B). Let also $[\![\cdot]\!]^{ps}$ be the function from Multi-π^{+ps} to Multi-π processes defined homomorphically with respect to terminated process, parallel composition, restriction, process variable and recursion, and defined as follows for the remaining operators:*

$$\begin{array}{ll}
[\![p_1 + p_2]\!]^{ps} = (\nu x)([\![p_1]\!]^{ps}_x + [\![p_2]\!]^{ps}_x) & x \notin fn(p_1 + p_2) \\
[\![\mu'.q]\!]^{ps} = (\nu x)([\![\mu'.q]\!]^{ps}_x) & x \notin fn(\mu'.q),\ \mu' = \overline{z_1@\ldots@z_n}\langle y\rangle \\
[\![\underline{\mu'}.q]\!]^{ps} = (\nu x)([\![\underline{\mu'}.q]\!]^{ps}_x) & x \notin fn(\underline{\mu'}.q),\ \underline{\mu'} = \underline{\overline{z_1@\ldots@z_n}\langle y\rangle} \\
[\![\mu'.q]\!]^{ps} = [\![\mu'.q]\!]^{ps}_x & \mu' = z_1@\ldots@z_n(y) \text{ or } \mu' = \tau \\
[\![\underline{\mu'}.q]\!]^{ps} = [\![\underline{\mu'}.q]\!]^{ps}_x & \underline{\mu'} = \underline{z_1@\ldots@z_n}(y) \text{ or } \underline{\mu'} = \underline{\tau}
\end{array}$$

with $[\![\cdot]\!]^{ps}_x$ defined as

$$\begin{array}{l}
[\![\tau.q]\!]^{ps}_x = \tau.[\![q]\!]^{ps} \\
[\![\underline{\tau}.q]\!]^{ps}_x = \underline{\tau}.[\![q]\!]^{ps}
\end{array}$$

$$[\![z_1@\ldots@z_n(y).q]\!]^{ps}_x = \underline{n(w_0)}.\underline{z_1(w_1)}.\cdots.\underline{z_n(w_n)}.\underline{w_n(y)}. \qquad w_i \notin fn(q)$$
$$(\underbrace{\underline{w_0}.\cdots.\underline{w_0}}_{n-1}.w_0.[\![q]\!]^{ps} \mid \overline{w}_1 \mid \cdots \mid \overline{w}_n)$$

$$[\![\underline{z_1@\ldots@z_n}(y).q]\!]^{ps}_x = \underline{n(w_0)}.\underline{z_1(w_1)}.\cdots.\underline{z_n(w_n)}.\underline{w_n(y)}. \qquad w_i \notin fn(q)$$
$$(\underbrace{\underline{w_0}.\cdots.\underline{w_0}}_{n}.[\![q]\!]^{ps} \mid \overline{w}_1 \mid \cdots \mid \overline{w}_n)$$

$$[\![\overline{z_1@\ldots@z_n}\langle y\rangle.q]\!]^{ps}_x = \overline{\underline{n}x}.\overline{\underline{z_1}x}.\cdots.\overline{\underline{z_n}x}.\bar{x}y.[\![q]\!]^{ps}$$
$$[\![\overline{\underline{z_1@\ldots@z_n}}\langle y\rangle.q]\!]^{ps}_x = \overline{\underline{n}x}.\overline{\underline{z_1}x}.\cdots.\overline{\underline{z_n}x}.\underline{\bar{x}y}.[\![q]\!]^{ps}$$
$$[\![p_1+p_2]\!]^{c}_x = [\![p_1]\!]^{c}_x + [\![p_2]\!]^{c}_x$$

Then, for every pair of processes q_1, q_2 of Multi-π^{+ps}, $q_1 \sim q_2 \iff [\![q_1]\!]^{ps} \sim [\![q_2]\!]^{ps}$.

As for polyadic communication, the parameter x of the function $[\![\cdot]\!]^{ps}_x$ is not used in the encoding of the input and silent action.

5.4 Last man standing problem

The Last Man Standing (LMS) has been introduced in [VBG09] as the first problem of distributed computing able to differentiate "truly parallel" languages from those whose underlying implementation requires, to some degree, centralization. We can informally say that the LMS can be solved in some process calculus if there exist a process p able to detect the presence or absence of other processes without generating deadlocks or introducing divergence. In order to say that the LMS can be solved in Multi-π, we would need to identify a process p such that p is able to perform an action a only when there is exactly one copy of p in the current system, while p is able to perform an action b only when there are at least two copies of p in the current system. In other words, if s_i is the system where i copies of p are enabled, we should have that

$$s_1 = p \qquad s_1 \xrightarrow{a} s_1' \qquad s_1 \not\xrightarrow{b} s_1''$$

while

$$s_2 = p \mid p \qquad s_2 \not\xrightarrow{a} s_2' \qquad s_2 \xrightarrow{b} s_2''$$
$$\ldots$$
$$s_n = \underbrace{p \mid p \mid \cdots \mid p}_{n} \qquad s_n \not\xrightarrow{a} s_n' \qquad s_n \xrightarrow{b} s_n''$$

with $a \neq b$. Rule (Par) in Table 1 allows us to easily prove that the LMS cannot be solved in Multi-π. In fact it states that any process p, able to perform some action a, can perform the same action in the presence of other processes as well, so that if $p \xrightarrow{a} p'$, then also $p \mid p \xrightarrow{a} p \mid p'$, which contradicts the requirement that $s_2 \not\xrightarrow{a} s_2'$.

6 Low-level semantics

In this section we want to offer an alternative semantics to the calculus where each transition is labeled by one single action, but where the states are not all equally observable. In more detail, we define an

enriched set of process terms, obtained by the introduction of an auxiliary state operator $*p$: if a process term is marked by $*$, then it represents an intermediate state of the transaction and so cannot be observed. Marked processes are ranged over by l, possibly indexed.

We also define a low-level transition relation $\longmapsto$ which relates extended process terms. For instance, for process $p = \underline{\bar{x}y}.\bar{z}y.q$ we can derive the following: $p \stackrel{\bar{x}y}{\longmapsto} *(\bar{z}y.q) \stackrel{\bar{z}y}{\longmapsto} q$, while in the abstract semantics of Section 3 we get $p \stackrel{\bar{x}y\bar{z}y}{\longrightarrow} q$.

The labels of the low-level transition systems are enriched by the empty label ε that is needed to label the transitions corresponding to, e.g., strongly prefixed τ's. For instance, $p' = \underline{\tau}.\bar{z}y.q$ is such that $p' \stackrel{\varepsilon}{\longmapsto} *(\bar{z}y.q) \stackrel{\bar{z}y}{\longmapsto} q$. We use γ to range over the set of possible low-level labels.

Marked processes should have priority when composed with unmarked processes, to express that the transaction is to be completed before any other activity can take place. When a process p in $p \,|\, q$ starts a transaction, it may decide that also the sibling process q should cooperate in the transaction; this is obtained by marking q. For instance, if $p \stackrel{\gamma}{\longmapsto} l$, we can derive both $p \,|\, q \stackrel{\gamma}{\longmapsto} l \,|\, q$ and $p \,|\, q \stackrel{\gamma}{\longmapsto} l \,|\, *q$; the former option represent the situation in which p performs its transaction in isolation, the latter option, instead, the case when also q is invited to join. Of course, if q is not willing to cooperate, the process $l \,|\, *q$ may be unable to complete a transaction (by reaching an observable state). As only computations that start from and arrive to observable states are considered relevant, these spurious additional low-level computations are inessential.

In the rules in Table 4 we use the following convention: observable states are normal processes p, q, possibly indexed; unobservable states are marked processes l, possibly indexed, while states (observable or not observable) are ranged over by s, possibly indexed. We define the following derivation relation:

$$\frac{p \stackrel{\gamma_1}{\longmapsto} l_1 \stackrel{\gamma_2}{\longmapsto} l_2 \ldots l_{k-1} \stackrel{\gamma_k}{\longmapsto} p'}{p \stackrel{\gamma_1 \ldots \gamma_k}{\longrightarrow_s} p'} \quad k \geq 1$$

The following proposition states the soundness and the completeness of the low-level semantics w.r.t. the abstract semantics.

Proposition 8 *For any Multi-π process p, $p \stackrel{\sigma}{\longrightarrow} p'$ if and only if $p \stackrel{\sigma}{\longrightarrow_s} p'$.*

Proof: The proof of the only if *part is by induction on the proof of transition $p \stackrel{\sigma}{\longrightarrow} p'$. The proof of the* if *part is by induction on the length of the sequence of transitions used to derive $p \stackrel{\sigma}{\longrightarrow_s} p'$ and on the depth of the proof tree of the first transition in such a sequence.* □

7 Conclusion

Multi-π is a conservative extension to the π-calculus, with essentially the same algebraic properties, but with a much richer modeling power, as the examples reported in this paper show. In particular, the polite dining philosophers problem and the mobile telephone example shows the interplay between mobility and atomic sequences of synchronization. Moreover, the extensions of polyadic communication [Mil91] and polyadic synchronization [CM03] are encoded in Multi-π thanks to its transactional communication capability.

(tau)	$\tau.p \overset{\tau}{\longmapsto} p$	(in)	$x(z).p \overset{xy}{\longmapsto} p\{y/z\}$	$y \notin fn((\nu z)p)$
(s-tau)	$\underline{\tau}.p \overset{\varepsilon}{\longmapsto} *p$	(s-in)	$\underline{x(z)}.p \overset{xy}{\longmapsto} *p\{y/z\}$	$y \notin fn((\nu z)p)$
(out)	$\bar{x}y.p \overset{\bar{x}y}{\longmapsto} p$	(s-out)	$\underline{\bar{x}y}.p \overset{\bar{x}y}{\longmapsto} *p$	
(star)	$\dfrac{s \overset{\gamma}{\longmapsto} s'}{*s \overset{\gamma}{\longmapsto} s'}$	(sum_1)	$\dfrac{p \overset{\gamma}{\longmapsto} s}{p+q \overset{\gamma}{\longmapsto} s}$	
(sum_2)	$\dfrac{q \overset{\gamma}{\longmapsto} s}{p+q \overset{\gamma}{\longmapsto} s}$	(res)	$\dfrac{s \overset{\gamma}{\longmapsto} s'}{(\nu y)s \overset{\gamma}{\longmapsto} (\nu y)s'}$	$y \notin n(\gamma)$
(cong_1)	$\dfrac{p \equiv p' \overset{\gamma}{\longmapsto} q' \equiv q}{p \overset{\gamma}{\longmapsto} q}$	(cong_2)	$\dfrac{p \overset{\gamma}{\longmapsto} l \;\; p \equiv q}{q \overset{\gamma}{\longmapsto} l}$	
(cong_3)	$\dfrac{l \overset{\gamma}{\longmapsto} p \;\; p \equiv q}{l \overset{\gamma}{\longmapsto} q}$	(rec)	$\dfrac{p\{\text{rec}X.p/X\} \overset{\gamma}{\longmapsto} s}{\text{rec}X.p \overset{\gamma}{\longmapsto} s}$	
(com_1)	$\dfrac{p \overset{\beta}{\longmapsto} p' \qquad q \overset{\bar{\beta}}{\longmapsto} q'}{p \mid q \overset{\tau}{\longmapsto} p' \mid q'}$	(com_2)	$\dfrac{l_1 \overset{\beta}{\longmapsto} l_1' \qquad l_2 \overset{\bar{\beta}}{\longmapsto} s'}{l_1 \mid l_2 \overset{\varepsilon}{\longmapsto} l_1' \mid s'}$	
(com_3)	$\dfrac{p \overset{\beta}{\longmapsto} l \qquad q \overset{\bar{\beta}}{\longmapsto} s}{p \mid q \overset{\varepsilon}{\longmapsto} l \mid s}$	(com_4)	$\dfrac{l \overset{\beta}{\longmapsto} p \qquad l' \overset{\bar{\beta}}{\longmapsto} q}{l \mid l' \overset{\tau}{\longmapsto} p \mid q}$	
(par_1)	$\dfrac{s \overset{\gamma}{\longmapsto} s'}{s \mid p \overset{\gamma}{\longmapsto} s' \mid p}$		$bn(\gamma) \cap fn(p) = \emptyset$	
(par_2)	$\dfrac{p \overset{\gamma}{\longmapsto} l}{p \mid q \overset{\gamma}{\longmapsto} l \mid *q}$		$bn(\gamma) \cap fn(q) = \emptyset$	
(par_3)	$\dfrac{l_1 \overset{\gamma}{\longmapsto} l'}{l_1 \mid l_2 \overset{\gamma}{\longmapsto} l' \mid l_2}$		$bn(\gamma) \cap fn(l_2) = \emptyset$	
(open)	$\dfrac{s \overset{\bar{x}y}{\longmapsto} s'}{(\nu y)s \overset{\bar{x}(w)}{\longmapsto} s'\{w/y\}}$		$y \neq x \;\; w \notin fn((\nu y)s')$	

Table 4: SOS low level rules – symmetric rules of (com_2)-(com_3) and (par_1)-(par_3) omitted.

7.1 Related work

7.1.1 Mobile Multi-party synchronization

In the literature, there are only a few calculi for mobility offering multi-party synchronization [EM99, HYC08].

In particular, [EM99] presents the $b\pi$-calculus (where b stands for *broadcast*) as a variant of the π-calculus where the point-to-point synchronisation mechanism is replaced by broadcast communication: the name x sent along a channel name a is received by all the processes ready to input on channel a. The form of multi-party synchronization offered by Multi-π is more flexible, as we do not need to use a shared channel name a, rather it is implemented as an atomic sequence of standard binary synchronizations. Moreover, $b\pi$ does not offer any form of atomic transactional communication and so Patil's problem cannot be solved in $b\pi$. On the other hand, in $b\pi$-calculus it is possible to solve the leader-election problem for an arbitrary number of participants, while this problem is solvable in Multi-π only for a fixed number of participants.

Paper [HYC08] describes a calculus with asynchronous multi-party sessions, where a process can perform a suitable multicast to some designated processes (that share the same channel name) and start a session by initializing some fresh session channels for the rest of the conversation. The main difference w.r.t. Multi-π is that this calculus is based on sessions, that represent a striking different abstraction of communication.

A recent interesting paper on a non-mobile calculus for multi-party synchronization is [LV10], where it is showed that there is an expressiveness gap between a CCS-based calculus with at most n-ary synchronizations and a CCS-based calculus with (at most) $n+1$-ary synchronizations by means of a generalization of the dining philosophers' problem.

7.1.2 Atomic actions

A^2CCS [GMM90] is an extension to CCS with the operator of strong prefixing. This calculus has been refined in [GV10] to Multi-CCS, by making associative its parallel composition operator and by changing its synchronization relation *Sync* over sequences.

Paper [ABD07] presents AtCCS, a CCS-like calculus with a more elaborated mechanism for atomic transactions, somehow inspired by the work on *Software Transactional Memories* (STM for short) [HM93]. Transactions are performed with an optimistic evaluation discipline. All actions are performed on a copy of the memory (the local state); when the transactions ends, there are two possible outcomes: if some concurrent writes have occurred, the transaction aborts and is then rescheduled; otherwise, the transaction commits and its effect is propagated instantaneously. In AtCCS it is possible to have multi-party communication as well.

7.1.3 Long running transactions and compensation

There is a large body of work on long running transctions for Web Services (see, e.g, [BMM05, BBF+05, BHF04], just to mention a few), equipped with some compensations. As in this setting transactions can last for a long time, it is unreasonable to demand atomicity and isolation in the execution of a transaction. Therefore, in case of failure, a compensation (a piece of code) is to be executed in order to undo most of the visible actions that have been performed.

7.2 Future work

We plan to equip Multi-π with a distributed semantics in terms of nets with inhibitor arcs, in the style of [BG09]. This net semantics offers a finite net representation for the so-called *finite-net* Multi-π processes, i.e., processes where restriction cannot occur inside recursion (but parallel composition can!) and $Sync(\sigma_1, \sigma_2, \sigma)$ is defined if σ_1 or σ_2 is a single action (a limitation that does not prevent multi-party synchronization). For instance, the system $Patil'$ of Section 4 is a typical example of a finite-net process: its net semantics is a finite and safe P/T net, in contrast to the fact that its interleaving semantics generates an infinite-state lts. According to [GV10], it can be proved that finite-net processes are expressive enough to represent all finite P/T nets, a further interesting expressiveness result for Multi-π.

Future work will be also devoted to define a more elaborate low-level *reversible* semantics, in the style of [DK04] (in turn based on an idea in [BPW94]), with the ability of backtracking when an unobservable deadlock is met.

References

[ABD07] L. Acciai, M. Boreale, S. Dal-Zilio, "A Concurrent Calculus with Atomic Transactions", in Proc. ESOP'07, LNCS 4421: 48-63, 2007.

[BPW94] J. Bergstra, A. Ponse, J.J. van Walem, "Process Algebra with Backtracking", in Proc. *REX School/Symposium*, LNCS 803:46-91, 1994.

[BMM05] R. Bruni, H.C. Melgratti, U. Montanari "Theoretical Foundations for Compensations in Flow Composition Languages", in Procs. POPL'05, ACM Press, pp. 209-220, 2005.

[BBF+05] R. Bruni, M.J. Butler, C. Ferreira, C. A. R. Hoare, H. C. Melgratti, U. Montanari "Comparing Two Approaches to Compensable Flow Composition", in Procs. CONCUR'05, LNCS 3653: 383-397, 2005.

[BHF04] M. J. Butler, C. A. R. Hoare, C. Ferreira "A Trace Semantics for Long-Running Transactions", in *25 Years Communicating Sequential Processes*, LNCS 3525: 133-150, 2004.

[BG09] N. Busi, R. Gorrieri, "Distributed semantics for the π-calculus based on Petri nets with inhibitor arcs", *J. of Logic and Alg. Programming* 78(3):138-162, 2009.

[CM03] M. Carbone, S. Maffeis, "On the Expressive Power of Polyadic Synchronisation in pi-calculus", *Nordic Journal of Computing* 10(2): 70-98, 2003.

[Dij68] E.W.Dijkstra, "Cooperating sequential processes", (F. Genuys ed.) *Programming Languages*, Academic Press, 43-112, 1968.

[Dij71] E.W.Dijkstra, "Hierarchical ordering of sequential processes", *Acta Informatica* 1(2):115-138, 1971.

[DK04] V. Danos, J. Krivine, "Reversible Communicating Systems", in Proc. CONCUR'04, LNCS 3170: 292-307, 2004.

[EM99] C. Ene, T. Muntean, "Expressiveness of point-to-point versus broadcast communications", In Procs. FCT'99, Springer-Verlag LNCS 1684:258-268, 1999.

[HM93] M. Herlihy, J.E.B. Moss, 'Transactional Memory: Architectural Support for Lock-Free Data Structures", in Procs. Int.l Symposium on Computer Architecture (ISCA'93), IEEE-CS Press, pp. 289-300, 1993.

[GM90] R. Gorrieri, U. Montanari, "Towards Hierarchical Specification of Systems: A Proof System for Strong Prefixing", *Int. Journal of Foundations of Computer Science*, 1(3):277-293, 1990.

[GMM90] R. Gorrieri, S. Marchetti, U. Montanari, "A^2CCS: Atomic Actions for CCS", *Theoretical Computer Science* 72(2-3): 203-223, 1990.

[GV10] R. Gorrieri, C. Versari, "A Process Calculus for expressing finite Place/Transition Petri Nets", Submitted, 2010.

(In)	$x(\tilde{z}).p \xrightarrow{x\tilde{y}} p\{\tilde{y}/\tilde{z}\}$	$y_1,\ldots,y_n \notin fn((\nu z_1)\ldots(\nu z_n)p)$
(S-in)	$\dfrac{p\{\tilde{y}/\tilde{z}\} \xrightarrow{\sigma} p'}{\underline{x(\tilde{z})}.p \xrightarrow{x\tilde{y}\sigma} p'}$	$y_1,\ldots,y_n \notin fn((\nu z_1)\ldots(\nu z_n)p)$
(Out)	$\bar{x}\tilde{y}.p \xrightarrow{\bar{x}\tilde{y}} p$	
(S-out)	$\dfrac{p \xrightarrow{\sigma} p'}{\underline{\bar{x}\tilde{y}}.p \xrightarrow{\bar{x}\tilde{y}\sigma} p'}$	
(Open)	$\dfrac{p \xrightarrow{\sigma} p'}{(\nu y)p \xrightarrow{\sigma'} p'}$	$\bar{x}\tilde{y} \in \sigma, \quad y = y_i$ for some i, and $Res(\sigma, y) = \sigma'$

Table 5: SOS extended rules for polyadic communication, with $\tilde{y} = y_1 \ldots y_n, \tilde{z} = z_1 \ldots z_n$ for some n.

[HYC08] Kohei Honda, Nobuko Yoshida, M. Carbone "Multiparty asynchronous session types", in Proc. POPL'08, ACM Press, pp. 273-284, 2008.

[LR81] D.J. Lehmann. M.l O. Rabin., "On the advantages of free choice: A symmetric and fully distributed solution to the dining philosophers problem", In Procs. POPL'81, pages 133-138, ACM Press, 1981.

[LV10] C. Laneve, A. Vitale, "The Expressive Power of Synchronizations", to appear in procs LICS'10, 2010.

[Mil89] R. Milner, *Communication and Concurrency*, Prentice-Hall, 1989.

[Mil91] R. Milner, "The Polyadic π–Calculus: A Tutorial", Technical Report, Dept. of Computer Science, University of Edinburgh, ECS–LFCS–91–180, Oct. 1991.

[Mil99] R. Milner. *Communicating and mobile systems: the π-calculus*, Cambridge University Press, 1999.

[MPW92] R. Milner, J. Parrow, D. Walker, "A Calculus of Mobile Processes", *Information and Computation* 100(1), 1-77, 1992.

[NP96] U. Nestmann and B.C. Pierce. Decoding Choice Encodings. In Proceedings of 7th International Conference on Concurrency Theory (CONCUR'96), Lecture Notes in Computer Science vol. 1119, Springer, Berlin, pp. 179-194, 1996.

[Pat71] S. Patil "Limitations and Capabilities of Dijkstra's Semaphore Primitives for Coordination Among Processes", Computation Structures Group Memo 57, Project MAC, MIT, 1971.

[Pet81] J.L. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, 1981.

[SW01] D. Sangiorgi, D. Walker, *The π-calculus: A Theory of Mobile Processes*, Cambridge University Press, 2001.

[VBG09] C. Versari, N. Busi, R. Gorrieri. An expressiveness study of priority in process calculi. *Mathematical Structures in Computer Science* 19(6): 1161-1189, 2009.

Appendix A: operational rules for polyadic communication

In Table 5 and 6 the SOS rules of Multi-π are extended for polyadic communication. The synchronization relation of Table 2 is basically unchanged, with proper extension of $\beta, \overline{\beta}$ to the polyadic setting.

$$
\begin{array}{rll}
Res(\varepsilon, y) = \varepsilon & & \\
Res(x\tilde{y}\sigma, y_i) = \bot & & \tilde{y} = y_1 \ldots y_n, \quad 1 \leq i \leq n \\
Res(y\tilde{x}\sigma, y) = \bot & & \tilde{x} = x_1 \ldots x_n \\
Res(z\tilde{w}\sigma, y) = z\tilde{w}Res(\sigma, y) & & \tilde{w} = w_1 \ldots w_n \\
Res(\overline{z}\tilde{y}\sigma, y_i) = \overline{z}\tilde{y}'\sigma & & \tilde{y} = y_1 \ldots y_n, \quad y' = y_1 \ldots (y_i) \ldots y_n \\
Res(\overline{z}\tilde{w}\sigma, y) = \overline{z}\tilde{w}Res(\sigma, y) & & \tilde{w} = w_1 \ldots w_n \\
Res(\overline{y}\tilde{x}\sigma, y) = \bot & & \tilde{x} = x_1 \ldots x_n
\end{array}
$$

Table 6: Restriction relation extended for polyadic communication – z, z_i and w, w_i are assumed different of y, y_i.

(In)	$\tilde{x}(z).p \xrightarrow{\tilde{x}y} p\{y/z\}$	$y \notin fn((\nu z)p)$
(S-in)	$\dfrac{p\{y/z\} \xrightarrow{\sigma} p'}{\underline{\tilde{x}(z)}.p \xrightarrow{\tilde{x}y\sigma} p'}$	$y \notin fn((\nu z)p)$
(Out)	$\bar{\tilde{x}}y.p \xrightarrow{\bar{\tilde{x}}y} p$	
(S-out)	$\dfrac{p \xrightarrow{\sigma} p'}{\underline{\bar{\tilde{x}}y}.p \xrightarrow{\bar{\tilde{x}}y\sigma} p'}$	
(Open)	$\dfrac{p \xrightarrow{\sigma} p'}{(\nu y)p \xrightarrow{\sigma'} p'}$	$\bar{\tilde{x}}\langle y\rangle \in \sigma$, and $Res(\sigma, y) = \sigma'$

Table 7: SOS extended rules for polyadic synchronization, with $\tilde{x} = x_1 @ \ldots @ x_n$.

$Res(\varepsilon, y) = \varepsilon$		
$Res(\tilde{x}y\sigma, y) = \perp$	$\tilde{x} = x_1@\ldots@x_n$	
$Res(\tilde{y}x\sigma, y_i) = \perp$	$\tilde{y} = y_1@\ldots@y_n$	$1 \leq i \leq n$
$Res(\tilde{z}w\sigma, y) = \tilde{z}wRes(\sigma, y)$	$\tilde{z} = z_1@\ldots@z_n$	
$Res(\bar{\tilde{z}}y\sigma, y) = \bar{\tilde{z}}(y)\sigma$	$\tilde{z} = z_1@\ldots@z_n$	
$Res(\bar{\tilde{z}}w\sigma, y) = \bar{\tilde{z}}wRes(\sigma, y)$	$\tilde{z} = z_1@\ldots@z_n$	
$Res(\bar{\tilde{y}}x\sigma, y_i) = \perp$	$\tilde{y} = y_1@\ldots@y_n$	$1 \leq i \leq n$

Table 8: Restriction relation extended for polyadic synchronization – z, z_i and w are assumed different of y, y_i.

Appendix B: operational rules for polyadic synchronization

In Table 7 and 8 the SOS rules of Multi-π are extended for polyadic synchronization. As for polyadic communication, the synchronization relation of Table 2 is basically unchanged, with proper extension of $\beta, \overline{\beta}$ to the polyadic synchronization setting.