Parametric Synchronizations in Mobile Nominal Calculi [⋆]

Roberto Bruni [a] and Ivan Lanese [b,∗]

[a] *Dipartimento di Informatica, Università di Pisa, Italia*

[b] *Dipartimento di Scienze dell'Informazione, Università di Bologna, Italia*

Abstract

We present and compare P-PRISMA and F-PRISMA, two parametric calculi that can be instantiated with different interaction policies, defined as synchronization algebras with mobility of names (SAMs). In particular, P-PRISMA is based on name transmission (P-SAM), like π-calculus, and thus exploits directional (input-output) communication only, while F-PRISMA is based on name fusion (F-SAM), like Fusion calculus, and thus exploits a more symmetric form of communication. However, P-PRISMA and F-PRISMA can easily accommodate many other high-level synchronization mechanisms than the basic ones available in π-calculus and Fusion, hence allowing for the development of a general meta-theory of mobile calculi. We define for both the operational and the observational semantics, showing that the latter is compositional *for any* SAM. We give several examples based on heterogeneous SAMs, we investigate the case studies of π-calculus and Fusion calculus giving correspondence theorems, and we show how P-PRISMA can be encoded in F-PRISMA. Finally, we show that basic categorical tools can help to relate and to compose SAMs and PRISMA processes in an elegant way.

Key words: Process calculi, name mobility, synchronization algebra with mobility, operational semantics, bisimulation congruences, π-calculus, Fusion calculus.

[⋆] Research supported by the Project FET-GC II IST 16004 SENSORIA.

[∗] Corresponding author address: Mura Anteo Zamboni n.7, I-40127 Bologna, Italia. Tel. +39.339.5861783. Fax +39.051.2094510

Email addresses: `bruni@di.unipi.it` (Roberto Bruni), `lanese@cs.unibo.it` (Ivan Lanese).

URLs: `http://www.di.unipi.it/~bruni` (Roberto Bruni), `http://www.cs.unibo.it/~lanese` (Ivan Lanese).

1 Introduction

Process calculi are mathematical abstractions for experimenting with and validating novel principles, mechanisms and interaction primitives of concurrent processes, and for possibly exporting them to modern concurrent programming languages. Though nowadays the most prominent calculus is the π-calculus [21], many other variants exist (see, e.g., the nice commented survey in [9]), exploiting different communication primitives and focusing on different aspects [1,4,13,22]. While process calculi are used to model different kinds of systems at different levels of abstraction, ranging from computer networks to biological systems, typically each calculus relies on just one fixed communication mechanism. When a different communication protocol is needed, either it is encoded using the available mechanism, and this may be quite difficult and may obfuscate the model, or a new ad hoc calculus providing this primitive is developed. For instance, [12] introduces a broadcast variant of π-calculus, while [11] proves that there is no uniform encoding of it into the π-calculus.

There are many situations in which one would like even different communication mechanisms to be available in the same model. For example, in Service Oriented Computing (SOC) it is commonly understood that services come with their own invocation policies (e.g., one-way or request-response), so that the current development of calculi for SOC [3,6–8,19] poses the problem of allowing the coexistence of several interaction policies, possibly seen at the same level of granularity. Furthermore, as new interaction policies can emerge, it would be convenient to have a uniform formal framework, easy to extend for accommodating and comparing different policies. We want to overcome the limitation of previous proposals in the sense explained above, by defining a calculus where processes can interact using the most convenient synchronization models tailored to the specific application in mind.

For instance, take a news server S that interacts with news providers using a message passing protocol, but then uses broadcast to send the news to subscribed recipients. We consider basic actions of the form $xa\vec{y}$ where x is the channel where the interaction is performed, a an action specifying the contribution to the interaction and $\vec{y}$ a tuple of parameters. Note the separation between the channel name and the action executed on it, which is a distinctive feature of our approach. In particular, we consider actions *in* and *out* respectively as input and output primitives for message passing, and in_b and out_b for broadcast. Also, we use *publish* and *news* as communication channels: the first is used for the interaction between the news provider and the server, and the latter for sending news to their recipients. Channel *info* is used as news value instead. Thus the server can be modeled as $S = !publish\ in\langle x\rangle.(news\ out_b\langle x\rangle | S'[x])$ where $|$ is parallel composition, . is prefixing and ! is replication. Here $S'[x]$ is a generic context exploiting x. A news provider instead has the form $P = (\mathit{info})\mathit{publish}\ \mathit{out}\langle \mathit{info}\rangle$ where $(-)$ is name restriction. Consider the system $P|S|C_1|C_2$, where $C_i = \mathit{news}\ \mathit{in}_b\langle y\rangle.\mathit{Use}_i[y]$ is a suitable client, for each i. The components P and S can interact on channel *publish*, leading to $S|(\mathit{info})(\mathit{news}\ \mathit{out}_b\langle \mathit{info}\rangle|S'[\mathit{info}])|C_1|C_2$. Then, a broadcast interaction concerning three different components (one sender and two receivers) delivers the news to the clients, leading in one step to $S|(\mathit{info})(S'[\mathit{info}]|\mathit{Use}_1[\mathit{info}]|\mathit{Use}_2[\mathit{info}])$.

In the paper we show that many different protocols can be formalized as *synchronization algebras with mobility* (SAMs) and how the above defined interactions can be exploited as action prefixes in a process calculus with name mobility, called PRISMA. The name PRISMA (the Italian for *prism*) is intended to expose the many communication facets of our calculus. For the sake of presentation, the set of primitives is kept to a minimal extent, but we conjecture that other features (e.g., ambients, encryption) can be likely transferred from the literature. The main advantage of having a uniform framework for expressing high-level synchronization mechanisms is that their formalization becomes simpler, since SAMs are tools dedicated to that purpose, and there is no need of, e.g., introducing special processes implementing the required synchronization patterns on top of the available ones. Also, PRISMA allows for developing general theories that are highly independent from the synchronization model. Finally, when expressed in a uniform framework, different synchronization models can be more easily compared and integrated (e.g., we shall show that the compound use of different policies as done in the news server is rather straightforward).

SAMs improve in a crucial way *synchronization algebras* [24] (which stem from ACP communication functions [2]), which were tailored for calculi such as CCS and CSP, to keep them in line with nowadays more sophisticated mobile calculi. SAMs were first defined in [17], in the context of a graph transformation framework called Synchronized Hyperedge Replacement (SHR) [10,14], to provide a uniform presentation of two existing synchronization models. They are what we call F-SAMs in this paper, to make explicit that the basic interaction is name fusion, as opposed to the novel class of P-SAMs introduced here, that are based on the more classic directional (input-output) name passing. Correspondingly, we distinguish two main variants of PRISMA, called P-PRISMA and F-PRISMA.

Even if P-SAMs might appear just as a more constrained form of SAMs, the difference between F-SAM and P-SAM is not inessential and is motivated by the different contexts in which fusions (of nodes, ports and resources) and i/o communications (flow of data and message passing) arise.

We remark that F-PRISMA is not a mere translation of SHR. Like F-PRISMA, also SHR is a unifying framework for modeling systems, but SHR is more suitable for architectural models since the structure of the system is explicitly represented. Instead, F-PRISMA (and P-PRISMA) focuses on the linguistic aspect of interaction, and are more useful to analyze the interactions between synchronization patterns and other primitives. Our presentation of F-SAMs is also more general and polished w.r.t. the one in [17]. We have chosen fusion as the key primitive in F-SAMs since Fusion calculus inherits the expressive power of π-calculus, while making the communication primitive more symmetric and easy to generalize. In fact, in π-calculus, input and output are treated ad hoc, and this gives no hint on how to deal with actions that are neither inputs nor outputs, such as in the Hoare SAM (Example 10). In this sense, one may expect that F-SAMs offer a more general setting than P-SAMs, an intuition that is in fact sustained also by our results in Section 5. Note that it would be very limited to see P-PRISMA and F-PRISMA just as extensions of π-calculus and Fusion calculus [22], because SAMs allow for much more general interactions, as shown by many examples in this paper.

As main results: (i) we prove that the observational semantics of P-PRISMA and F-PRISMA, called respectively *full P-bisimilarity* and *full F-bisimilarity*, are congruences under any SAM (Theorems 19 and 25), (ii) we show how to prove properties for general classes of synchronization protocols (Lemmas 17 and 18), (iii) we show an operational encoding of P-PRISMA in F-PRISMA (Theorem 30) and (iv) we discuss how to build complex SAMs by composing basic ones (Section 6). The expressiveness of our calculi is demonstrated via examples on a news server (already outlined), on communications with accounting, on interoperability between different synchronization policies, and via the case studies on π-calculus (Section 3.1) and on Fusion calculus (Section 4.1).

This paper is an extended version of [5], where F-PRISMA was presented (under the name PRISMA, which is used here in a broader sense). Instead all the results about P-SAMs and P-PRISMA are original to this contribution.

Structure of the paper. Section 2 introduces P-SAMs and F-SAMs and shows some examples. In Section 3 we define the P-PRISMA calculus, analyze its operational and abstract semantics and prove the congruence theorem for bisimilarity. The case study on π-calculus is detailed in Section 3.1. Section 4 introduces F-PRISMA and presents the related results (including the case study on Fusion calculus in Section 4.1). Section 5 compares the two calculi, while Section 6 analyzes the relationships among different SAMs using basic concepts from category theory, which can be found, e.g., in [20]. Finally, Section 7 contains some conclusions and plans for future work. A full discussion on F-PRISMA calculus and related topics can be found in the Ph.D. thesis of the second author [15].

2 Synchronization algebra with mobility (SAM)

Notation. Given a function f we denote with $\mathrm{dom}(f)$ its domain and with $\mathrm{Im}(f)$ its image. Also, the function $f|_S$ (resp. $f|_{\setminus S}$) is obtained by restricting f to set S (resp. to $\mathrm{dom}(f) \setminus S$). When set operations (e.g., $\cup$) are used on function f, it is implicitly assumed that f is seen as a set of pairs $(a, f(a))$. We use $\circ$ to denote the standard composition of functions, i.e. $(g \circ f)(x) = g(f(x))$. Given a vector $\vec{v}$ and an integer i we denote with $\vec{v}[i]$ the i-th element of $\vec{v}$, $\mathrm{Set}(\vec{v})$ is the set of elements in $\vec{v}$ and $|\vec{v}|$ is its length. We write $A \uplus B$ to denote the disjoint union of sets A and B, with $\mathrm{inj}_1 : A \to A \uplus B$ and $\mathrm{inj}_2 : B \to A \uplus B$ the left and right inclusions, respectively. When no confusion can arise we write $\mathrm{inj}_i(x)$ simply as x. If $\mathrm{inj}_i(x) \in A \uplus B$ we denote with $\mathit{comp}(\mathrm{inj}_i(x))$ the element $\mathrm{inj}_{3-i}(x)$ in $B \uplus A$. Given two functions $f : A \to C$ and $g : B \to D$ we denote with $[f, g] : A \uplus B \to C \uplus D$ the function that applies f to the elements in A and g to the ones in B. We denote with $\underline{n}$ the set $\{1, \ldots, n\}$ (where $\underline{0} \overset{\mathrm{def}}{=} \emptyset$), while id_n is the identity function on it. Given a set S, S^* is the set of strings on alphabet S and λ is the empty string. Finally, we denote with $\mathrm{mgu}(E)$ any idempotent substitution resulting from computing the most general unifier on the set of equations E, when it exists.

In this section we present SAMs, an extension of Winskel's synchronization algebras (SAs) [24] able to deal with name mobility, local resource handling and nondeterminism. SAMs can be used to specify the interactions among different actions, each carrying a tuple of arguments which are names of channels. Each allowed synchronization pattern is modeled by an action synchronization triple, whose first and second components are the interacting actions and whose third component is the result of the synchronization. This is composed by three different fields: (1) the resulting action, (2) a function specifying how the arguments attached to the resulting action are computed (the component Mob in Definition 2), (3) a relation determining which names are merged (the component $\doteq$ in Definition 2).

Definition 1 (Action signature)
An action signature $\mathbf{A}$ *is a tuple* $(\mathit{Act}, \mathrm{mod}, \epsilon)$ *where* Act *is the set of actions,* $\mathrm{mod} : \mathit{Act} \to \mathcal{MOD}^*$ *is the* modality function *specifying the modality of the arguments of each action (here* $\mathcal{MOD}$ *is a set of modalities) and* $\epsilon \in \mathit{Act}$ *has* $\mathrm{mod}(\epsilon) = \lambda$.

For each action a, the length of the string $\mathrm{mod}(a)$, denoted as $\mathrm{ar}(a)$, is the number of arguments of a. We use directional modalities where $\mathcal{MOD} = \{\mathit{in}, \mathit{out}\}$ (in for input, out for output) and symmetric modalities where $\mathcal{MOD} = \{\bullet\}$. In the first case, $\mathrm{mod}(a)[i] = \mathit{in}$ means that the i-th parameter is an input parameter, while $\mathrm{mod}(a)[i] = \mathit{out}$ means that it is an output parameter. If $\mathcal{MOD} = \{\bullet\}$ we may write simply $\mathbf{A} = (\mathit{Act}, \mathrm{ar}, \epsilon)$, the modalities being obvious. The action ϵ stands for "not taking part in synchronization", and it allows to deal in a uniform way with synchronization and with asynchronous execution of actions, the latter being modeled as synchronization with ϵ.

Definition 2 (Action synchronization set)
An action synchronization set AS *on* $\mathbf{A}$ *is a set of triples* $(a, b, (c, \mathrm{Mob}, \doteq))$ *where* $a, b, c \in Act$*,* $\mathrm{Mob} : \underline{\mathrm{ar}(c)} \to \underline{\mathrm{ar}(a)} \uplus \underline{\mathrm{ar}(b)}$ *and* $\doteq$ *is an equivalence relation on* $\underline{\mathrm{ar}(a)} \uplus \underline{\mathrm{ar}(b)}$*.*

If $\mathcal{MOD} = \{in, out\}$*, for each triple* $(a, b, (c, \mathrm{Mob}, \doteq))$*,* Mob *and* $\doteq$ *should satisfy the following conditions:*

(1) for each equivalence class defined by $\doteq$*, at most one element may have modality out;*
(2) the image of function Mob *contains at least one element for each equivalence class according to* $\doteq$ *composed by only input elements;*
(3) $\mathrm{mod}(c)[i] = in$ *iff the equivalence class according to* $\doteq$ *of* $\mathrm{Mob}(i)$ *contains no element of modality out.*

The Mob component assigns to each argument of c an argument of either a or b, i.e. it specifies how the arguments of the resulting action are obtained from the arguments of the component actions. Since actual arguments are not known at SAM-definition time, the correspondence is defined according to the positions in the tuple: for instance $\mathrm{Mob}(1) = \mathrm{inj}_2(1)$ means that the first parameter of the resulting action comes from the first parameter of the second action, as it is in the left part of Figure 1, that represents the action synchronization $(a, \overline{a}, (\overline{a}, \mathrm{Mob}_1, \doteq))$.

To be more precise, the parameter is defined by the equivalence relation according to $\doteq$ of the argument individuated by Mob. Note that $\doteq$ uses a positional notation too. In the directional case, if the class contains only input parameters then they are all merged, and a representative is used as argument for the synchronized action (thanks to condition 2 it can not disappear). Notice that it must be still an input parameter. Otherwise if an output parameter is present, then it is propagated. For instance, according to the action synchronization in the left part of Figure 1, $a\langle x\rangle$ can interact with $\overline{a}\langle y\rangle$. Then y (the output parameter, denoted by a full bullet) is substituted for x.

Action synchronizations $(a, b, (c, \mathrm{Mob}_1, \doteq))$ and $(a, b, (c, \mathrm{Mob}_2, \doteq))$ such that $\mathrm{Mob}_1(n) \doteq \mathrm{Mob}_2(n)$ for each n (see, e.g., Figure 1) are semantically equivalent.

Next definition introduces a notion of composition on action synchronizations. In the general case, synchronization among n different processes must be specified. However, SAMs guarantee that the order in which synchronization is achieved is not important. Our approach allows to specify synchronization in a compositional way, i.e. by considering the interaction between two processes at the time. In particular, in order to express associativity we find it con-

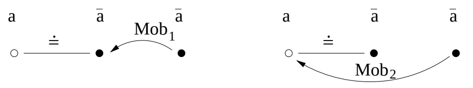

Fig. 1. Action synchronization.

venient to consider the synchronization of three actions, which arises as the composition of two binary synchronizations.

Definition 3 (Left-composition of action synchronization) *Given* $\beta_1 = (a_1, b_1, (c_1, \text{Mob}_1, \doteq_1))$ *and* $\beta_2 = (a_2, b_2, (c_2, \text{Mob}_2, \doteq_2))$ *with* $c_1 = a_2$*, the composition* $\beta_1 \star_{\text{L}} \beta_2$ *of* β_1 *and* β_2 *is the tuple* $(a_1, b_1, b_2, (c_2, \text{Mob}_3, \doteq_3))$ *where* $\text{Mob}_3 = [\text{Mob}_1, id_{\text{ar}(b_2)}] \circ \text{Mob}_2 : \underline{\text{ar}(c_2)} \to \underline{\text{ar}(a_1)} \uplus \underline{\text{ar}(b_1)} \uplus \underline{\text{ar}(b_2)}$*, and the equivalence relation* $\doteq_3$ *on* $\underline{\text{ar}(a_1)} \uplus \underline{\text{ar}(b_1)} \uplus \underline{\text{ar}(b_2)}$ *is defined as the projection on the above specified domain of the least equivalence relation* R *on* $\underline{\text{ar}(a_1)} \uplus \underline{\text{ar}(b_1)} \uplus \underline{\text{ar}(c_1)} \uplus \underline{\text{ar}(b_2)}$ *such that* $x \,\text{R}\, y$ *if* $x \doteq_1 y \lor x \doteq_2 y \lor \text{Mob}_1(x) = y$*.*

A similar *right-composition* $\beta_1 \star_{\text{R}} \beta_2$ is defined when $c_1 = b_2$ instead of $c_1 = a_2$.

Definition 4 (Action synchronization relation) *Given an action signature* $\mathbf{A} = (Act, \text{mod}, \epsilon)$*, an* action synchronization relation AS *on* $\mathbf{A}$ *is an action synchronization set such that:*

(1) $(a, b, (\epsilon, \text{Mob}, \doteq)) \in AS \Rightarrow a = b = \epsilon$*;*
(2) $(a, \epsilon, (c, \text{Mob}, \doteq)) \in AS \Rightarrow (c = a \land \text{Mob} = \text{inj}_1 \land \doteq = \text{id})$*;*
(3) $(a, b, (c, \text{Mob}, \doteq)) \in AS \Rightarrow (b, a, (c, \text{Mob}', \doteq')) \in AS$*, where for each* x*,* y $\text{Mob}'(x) = comp(\text{Mob}(x))$ *and* $x \doteq' y$ *iff* $comp(x) \doteq comp(y)$*;*
(4) if $\beta_1 = (a, b, (c, \text{Mob}, \doteq)) \in AS$ *and* $\beta_2 = (c, d, (e, \text{Mob}', \doteq')) \in AS$ *then* $\exists f \in Act$*,* $\exists \gamma_1 = (b, d, (f, \text{Mob}'', \doteq'')), \gamma_2 = (a, f, (e, \text{Mob}''', \doteq''')) \in AS$ *such that* $\beta_1 \star_{\text{L}} \beta_2 = \gamma_1 \star_{\text{R}} \gamma_2$*.*

Condition 1 (already present in SAs) specifies that no action can disappear producing ϵ. Also, interaction of ϵ with any action just propagates the other action and its parameters (condition 2). Conditions 3 and 4 ensure commutativity and associativity of synchronization respectively, by specifying that the composed actions take the same parameters and force the same merges.

Definition 5 (SAM) *A* synchronization algebra with mobility *is a triple* $S = (\mathbf{A}, Fin, AS)$ *which includes an action signature* $\mathbf{A} = (Act, \text{mod}, \epsilon)$*, a set* $Fin \subseteq Act$ *of* final actions *such that* $\epsilon \notin Fin$ *and an action synchronization relation* AS *on* $\mathbf{A}$*. We require that actions in* Fin *have no input parameters. We call P-SAM any SAM with directional modality and F-SAM any SAM with symmetric modality.*

Final actions are used to deal with local resources: since no process from outside can interact with a bound channel, only actions corresponding to successful interactions that do not require additional contributions can take place on bound channels. Those actions are in Fin. For instance in message passing synchronization (see Example 7) an input is not in Fin, while the result of the synchronization between one input and one output is in Fin. In the directional case the action can have only output parameters, since input parameters would be unable to get a value from an output.

We present a few simple examples of SAMs, extending the ones in [17,18]. Below, $MP_{i,j}$ (for message passing) is a shorthand for the function from $\underline{\max(i,j)}$ to (any superset of) $\underline{i} \uplus \underline{j}$ such that $MP_{i,j}(m) = \mathrm{inj}_1(m)$ if $m \leq i$, and $\mathrm{inj}_2(m)$ otherwise, while EQ_i denotes the least equivalence relation on (any superset of) $\underline{i} \uplus \underline{i}$ containing $\{(\mathrm{inj}_1(m), \mathrm{inj}_2(m)) | m \leq i\}$.

Remark 6 *From now on, to simplify the presentation, we will not write explicitly the triples obtained by commutativity and we assume that the triple* $(\epsilon, \epsilon, (\epsilon, MP_{0,0}, EQ_0))$ *is omnipresent. We also assume that a set of labels* L *is given such that* $L \cap \{\epsilon, \tau\} = \emptyset$. *In the directional case, we assume* $\mathrm{mod} : L \to \{in\}^*$ *given, otherwise we just assume the arities given. Finally, given a modality* $\mu \in \mathcal{MOD}^*$ *we denote with* μ^c *the modality obtained by swapping in and out everywhere in* μ.

Example 7 (Milner SAMs) *The Milner P-SAM* Pi_L *is given by:*

- $Act = \{\tau, \epsilon\} \cup \bigcup_{l \in L} \{l, \bar{l}\}$, $Fin = \{\tau\}$;
- $\mathrm{mod}(\bar{l}) = \mathrm{mod}(l)^c$ *for each* $l \in L$, $\mathrm{mod}(\tau) = \lambda$;
- $(a, \epsilon, (a, MP_{\mathrm{ar}(a),0}, EQ_0)) \in AS$ *for each* $a \in Act$,
  $(l, \bar{l}, (\tau, MP_{0,0}, EQ_{\mathrm{ar}(l)})) \in AS$ *for each* $l \in L$.

Pi_L represents message passing *à la* π-calculus: one input l interacts with one output $\bar{l}$, and the parameters of the output are assigned to the input variables. Action τ represents a complete message exchange, and thus belongs to Fin. Here we are more general than π-calculus, since it allows just one output action, while we allow many (each with corresponding input), and this corresponds to introducing a simple form of typing.

From a P-SAM we can always derive a F-SAM, as described in the following definition.

Definition 8 *Given a P-SAM* S, *the corresponding F-SAM* $\mathrm{symm}(S)$, *is obtained from* S *by changing only the modality function. If* $\mathrm{ar}(a) = n$ *then the new modality is* $\mathrm{mod}(a) = \bullet^n$.

The Milner F-SAM, $Fusion_L = \mathrm{symm}(Pi_L)$ models Fusion calculus synchronization instead, where input and output parameters are treated in the same way. This will be clearer when we will show the calculi and their semantics.

Example 9 (Broadcast SAM) *The P-SAM* Bdc_L *is given by:*

- $Act = \{\epsilon\} \cup \bigcup_{l \in L}\{l, \overline{l}\}$, $Fin = \bigcup_{a \in L}\{\overline{a}\}$;
- $\mathrm{mod}(\overline{l}) = \mathrm{mod}(l)^c$ *for each* $l \in L$;
- $(l, \overline{l}, (\overline{l}, MP_{\mathrm{ar}(l),\mathrm{ar}(\overline{l})}, EQ_{\mathrm{ar}(l)})) \in AS$ *for each* $l \in L$,
  $(l, l, (l, MP_{\mathrm{ar}(l),\mathrm{ar}(l)}, EQ_{\mathrm{ar}(l)})) \in AS$ *for each* $l \in L$.

The above SAM models broadcast. When used in P-PRISMA, it forces an output $\overline{l}$ from a sequential P-PRISMA process to synchronize with *all* the listening sequential processes in parallel, which have to perform an input l. Notice that, if one wants to have a multicast Mul_L, where some listening process may not synchronize with the output, it is enough to add the triples $(a, \epsilon, (a, MP_{\mathrm{ar}(a),0}, EQ_0))$ for each $a \in Act$ to AS. The corresponding F-SAM can be defined as usual, with the expected behavior.

Interestingly, sometimes the constraints imposed by modalities are too strong, and some useful SAMs can only be defined in the symmetric setting. This is the case for instance of Hoare SAM.

Example 10 (Hoare SAM) *The F-SAM* $Hoare_L$ *is given by:*

- $Act = Fin = \{\epsilon\} \cup L$;
- $(a, a, (a, MP_{\mathrm{ar}(a),\mathrm{ar}(a)}, EQ_{\mathrm{ar}(a)})) \in AS$ *for each* $a \in Act$.

Hoare SAM is inspired by CSP synchronization, extended with name mobility. It models a global agreement on the action to perform. Since the interacting actions are all equal, it is not possible to divide the parameters into input and output ones. From a technical point of view, using only output parameters forbids synchronization. Using only input parameters (which are bound) makes all of them α-convertible, so they would be all equal for the outside world. In the symmetric setting, corresponding parameters are just merged.

A P-SAM featuring binary symmetric interactions can be defined using actions with two parameters, the first used for output and the second for input.

Example 11 (Exchange SAM) *The P-SAM* $Exch_X$ *is given by:*

- $Act = X \cup \{\tau, \epsilon\}$, $Fin = \{\tau\}$;
- $\mathrm{mod}(l) = \langle out, in \rangle$ *for each* $l \in X$, $\mathrm{mod}(\tau) = \lambda$;
- $(a, \epsilon, (a, MP_{\mathrm{ar}(a),0}, EQ_0)) \in AS$ *for each* $a \in Act$,
  $(l, l, (\tau, MP_{0,0}, \chi)) \in AS$ *for each* $l \in X$;
- $\chi = \{(\mathrm{inj}_1(1), \mathrm{inj}_2(2)), (\mathrm{inj}_2(1), \mathrm{inj}_1(2))\}$.

We present now a more complex example: a P-SAM for *communication with priority* that allows many senders to synchronize with just one receiver, which takes only the message with the highest priority. This SAM can be used, e.g., to model communication in sensor networks, where the base station acquires at each step the most important available information. In the example we

- $Act = \{i, \epsilon\} \cup \{(o, n) | n \in \mathbb{N}\} \cup \{(o^+, n) | n \in \mathbb{N}\} \cup \{(o^-, n) | n \in \mathbb{N}\}$;
- $\mathrm{mod}(a) = \lambda$ for all $a \in \{\epsilon\} \cup \{(o^+, n) | n \in \mathbb{N}\}$, $\mathrm{mod}(i) = in$, $\mathrm{mod}(a) = out$ otherwise;
- $Fin = \{(o^+, n) | n \in \mathbb{N}\}$;
- The action synchronizations are:

(1) $(a, \epsilon, (a, MP_{\mathrm{ar}(a),0}, EQ_0)) \in AS$ for each $a \in Act$,
(2) $(i, (o, n), ((o^+, n), MP_{0,0}, EQ_1)) \in AS$ for each n,
(3) $(i, (o, n), ((o^-, n), MP_{1,0}, EQ_0)) \in AS$ for each n,
(4) $((o, n), (o, m), ((o, n), MP_{1,0}, EQ_0)) \in AS$ for each $n \geq m$,
(5) $((o, n), (o^-, m), ((o^+, n), MP_{0,0}, EQ_1)) \in AS$ for each $n \geq m$,
(6) $((o, n), (o^-, m), ((o^-, n), MP_{0,1}, EQ_0)) \in AS$ for each $n \geq m$,
(7) $((o, m), (o^-, n), ((o^-, n), MP_{0,1}, EQ_0)) \in AS$ for each $n \geq m$,
(8) $((o^+, n), (o, m), ((o^+, n), MP_{0,0}, EQ_0)) \in AS$ for each $n \geq m$.

Fig. 2. The priority SAM *Pri*.

consider just one input action i of arity 1, but the generalizations to many actions and different arities are straightforward.

Example 12 (Priority SAM) *The P-SAM Pri is defined in Figure 2. The basic idea is that the result of the synchronization of an action i and an action (o, n), i.e. an output with priority n, is guessed: either the guess is that n is the highest priority or the opposite. The former case corresponds to action synchronization 2 that performs the communication of the parameter and produces (o^+, n) as a result. The latter corresponds to action synchronization 3 that propagates the input variable and produces (o^-, n). The first guess, if wrong, is discarded when an output with higher priority is found since (o^+, n) can not interact with actions with priority greater than n. The second guess is checked when the channel is bound, since $(o^-, n) \notin Fin$. Here nondeterminism is useful for the guess, but also needed to choose which output to propagate when two with the same priority interact.*

3 The P-PRISMA calculus

We can now present the syntax and the semantics of P-PRISMA. Action prefixes in P-PRISMA are parametric on a P-SAM $S = ((Act, \mathrm{mod}, \epsilon), Fin, AS)$.

Definition 13 (P-PRISMA syntax) *The syntax for P-PRISMA processes on P-SAM $S = ((Act, \mathrm{mod}, \epsilon), Fin, AS)$ is:*

$P ::= 0$	*(Inaction)*	$xa\vec{y}.P$	*(Prefix)*
$P_1 \| P_2$	*(Parallel composition)*	$P_1 + P_2$	*(Nondeterministic sum)*
$(x)P$	*(Restriction)*	$!P$	*(Replication)*

where x is a channel name, $a \in Act$ is an action and $\vec{y}$ is a vector of channel names whose length is $\mathrm{ar}(a)$*. Channel x is the subject of $xa\vec{y}$.*

There are two binders in P-PRISMA: restriction (x) and prefixes. In particular, in prefix $xa\vec{y}.P$ the names in $\vec{y}$ which are input parameters according to $\mathrm{mod}(a)$ are bound with scope P (and must be all distinct). As usual, $\mathrm{fn}(P)$, $\mathrm{bn}(P)$ and $\mathrm{n}(P)$ denote respectively the sets of free names, bound names and all the names in P and processes are taken up to α-conversion of restricted names (denoted by $\equiv_\alpha$). The intrinsic compositionality of action synchronization in SAMs makes the LTS operational semantics more natural for P-PRISMA than semantics in the reduction style, where all possible global synchronizations, possibly involving an unbound number of processes, should be considered explicitly (think, e.g., of broadcast). Roughly, reductions would correspond to "closed" synchronizations, according to Fin.

We present now the inference rules defining the semantics of P-PRISMA processes, in an incremental way. The rules are parametric on the P-SAM S that fixes the allowed interaction policies. We annotate the transition arrow with $P(S)$ where P identifies P-PRISMA and S is the used P-SAM. We use α to range over labels. Interestingly the rules exploit just α-conversion of bound names as structural law, and this simplifies the proofs of process properties. This corresponds to have the rule:

$$\frac{P \equiv_\alpha P' \qquad P \xrightarrow{\alpha}_{P(S)} Q \qquad Q \equiv_\alpha Q'}{P' \xrightarrow{\alpha}_{P(S)} Q'} \quad (\text{PP-A}\textsc{lpha})$$

However, the kind of axioms usually used in structural congruence equate processes which are also equivalent according to our observational semantics (see Lemma 17). One could avoid α-conversion too, but this would unnecessarily complicate the inference rules. Similarly to π-calculus, we consider the early semantics. The rule for input prefix is:

$$\frac{|\vec{z}| = |\vec{y}| \qquad \mathrm{mod}(a)[i] = \mathit{out} \Rightarrow \vec{z}[i] = \vec{y}[i]}{xa\vec{y}.P \xrightarrow{xa\vec{z}}_{P(S)} P\{\vec{z}/\vec{y}\}} \quad (\text{PP-P}\textsc{ref})$$

The transition simply executes the action prefix $xa\vec{y}$. Output parameters are copied to the label, while for input parameters the received value is guessed according to the early semantics, and the corresponding substitution is applied to the continuation.

The most important, but also the most complex, rule allows to synchronize two actions performed by parallel processes (rule (PP-S<span style="font-variant:small-caps">ynchr</span>) in Table 1). Its complexity is due to the great degree of flexibility of P-PRISMA, which allows to specify both action synchronization and name mobility patterns. Also, we deal with slightly more complex actions than the ones seen so far,

Table 1
Rule for synchronization.

$$\frac{P_1 \xrightarrow{(Y_1)xa_1\vec{y}_1}_{P(S)} P_1' \qquad P_2 \xrightarrow{(Y_2)xa_2\vec{y}_2}_{P(S)} P_2' \qquad \Phi}{P_1|P_2 \xrightarrow{(W)xc\vec{w}}_{P(S)} (\vec{s})(P_1'|P_2')} \quad (\text{PP-}\textsc{Synchr})$$

where the premise Φ is the conjunction of the following six side conditions:

correctness of the synchronization: $(a_1, a_2, (c, \text{Mob}, \doteq)) \in AS$;

freshness of extruded names: $Y_1 \cap Y_2 = \emptyset$, $(Y_1 \cup Y_2) \cap (\text{fn}(P_1) \cup \text{fn}(P_2)) = \emptyset$;

correctness of the guesses: $\text{inj}_{i_1}(j_1) \doteq \text{inj}_{i_2}(j_2) \Rightarrow \vec{y}_{i_1}[j_1] = \vec{y}_{i_2}[j_2]$;

arguments of c: $\vec{w}[k] = \vec{y}_i[j]$ iff $\text{Mob}(k) = \text{inj}_i(j)$;

names extruded by c: $W = \text{Set}(\vec{w}) \cap (Y_1 \cup Y_2)$;

closed names: $\text{Set}(\vec{s}) = (Y_1 \cup Y_2) \setminus W$ (any order can be chosen for $\vec{s}$).

since a set of extruded names appears (when empty, as in rule (PP-Pref), it is deleted from the label). Bound names are extruded when used as output parameters in the label, thus becoming free. Extruded names must be traced, since when they are removed from the tuple of parameters, restrictions for them have to be reintroduced (as in π-calculus (close) rule).

The synchronization rule (PP-Synchr) allows two actions a_1 and a_2 performed on the same channel x to synchronize producing action c. Also, if some parameters are merged by $\doteq$, then their values should coincide: this amounts to say that the values guessed for inputs are correct when the input is merged with an output and compatible when two inputs are merged. The action c propagates and its parameters are computed according to Mob. The set W is the new set of extruded names. Finally, names that were extruded $(Y_1 \cup Y_2)$ and no longer appear in the label $((Y_1 \cup Y_2) \setminus W)$ must be closed by inserting them into $\vec{s}$ (in any order). Notice that according to condition 2 in Definition 2 these are output names.

Two additional aspects must be considered to deal with parallel composition. In some P-SAMs, such as the Pi one (Example 7), no process is forced to participate to the synchronization, while in others, such as in broadcast (Example 9), the processes in a given set *must* participate. This is specified in SAMs by allowing or disallowing the interaction with ϵ, which can be executed for free by any process using rule:

$$\frac{x \in \mathcal{N}}{P \xrightarrow{x\epsilon\langle\rangle}_{P(S)} P} \quad (\text{PP-}\textsc{Epsilon})$$

Table 2
Rules for restriction.

$$\frac{P \xrightarrow{(W)za\vec{y}}_{P(S)} P' \quad a \in Fin \quad \mathrm{Set}(\vec{w}) = W}{(z)P \xrightarrow{\surd}_{P(S)} (z\vec{w})P'} \quad (\text{PP-}\textsc{Res})$$

$$\frac{P \xrightarrow{(Y)xa\vec{y}}_{P(S)} P' \quad z \in \mathrm{Set}(\vec{y}) \setminus \{x\} \setminus Y \quad z \text{ has modality } in \text{ in } \vec{y}}{(z)P \xrightarrow{(\{z\}\cup Y)xa\vec{y}}_{P(S)} P'} \quad (\text{PP-}\textsc{Open})$$

$$\frac{P \xrightarrow{\alpha}_{P(S)} P' \quad z \notin \mathrm{n}(\alpha)}{(z)P \xrightarrow{\alpha}_{P(S)} (z)P'} \quad (\text{PP-}\textsc{Pass}) \qquad \frac{P \xrightarrow{\surd}_{P(S)} P'}{P|Q \xrightarrow{\surd}_{P(S)} P'|Q} \quad (\text{PP-}\textsc{TauPar})$$

However, also in broadcast, we want to allow processes which are not interested in the synchronization to stay idle. We consider that a process is interested in a synchronization at x if it has an active prefix with subject x. Thus, following the approach of [12], we introduce a label $\neg x$ which can be executed by any process which has no active prefix with subject x. This can be modeled with:

$$\frac{x \text{ is not an active subject of } P}{P \xrightarrow{\neg x}_{P(S)} P} \quad (\text{PP-}\textsc{Indip})$$

An inductive definition of this rule can be found in Appendix A. This allows proofs by induction. A dedicated rule (and its symmetric) are required to allow this action to interact with a normal action:

$$\frac{P_1 \xrightarrow{(Y)xa\vec{y}}_{P(S)} P_1' \quad P_2 \xrightarrow{\neg x}_{P(S)} P_2 \quad Y \cap \mathrm{fn}(P_2) = \emptyset}{P_1|P_2 \xrightarrow{(Y)xa\vec{y}}_{P(S)} P_1'|P_2} \quad (\text{PP-}\textsc{Par})$$

Restriction is dealt with rules in Table 2.

Rule (PP-Res) closes the channel z on which the action a is done, reintroducing the restriction for names extruded by a. This is allowed only if $a \in Fin$. This rule introduces a further form of label, namely $\surd$, which states that a final action has been performed on a bound channel. If instead the restricted name z is one of the parameters with output modality (rule (PP-Open)), then it is marked as extruded in the label (as in π-calculus rule (open)). Rule (PP-Pass) says that restriction on channel z does not influence actions where z does not appear. The simple rule (PP-TauPar) (and its symmetric) deals with labels of the form $\surd$ (restriction is dealt with by rule (PP-Pass) as usual).

Finally, (almost) standard rules can be added to deal with nondeterministic sum (rule (PP-Sum) and its symmetric) and replication (rule (PP-Repl)):

$$\frac{P_1 \xrightarrow{\alpha}_{P(S)} P_1' \quad \alpha \neq x\epsilon\langle\rangle \quad \alpha \neq \neg x}{P_1 + P_2 \xrightarrow{\alpha}_{P(S)} P_1'} \quad (\text{PP-S}\textsc{um})$$

$$\frac{P|!P \xrightarrow{\alpha}_{P(S)} P'}{!P \xrightarrow{\alpha}_{P(S)} P'} \quad (\text{PP-R}\textsc{epl})$$

The only peculiarity is that actions $x\epsilon\langle\rangle$ and $\neg x$, which can be executed for free and thus do not represent real process activities, should not force the choice of one branch of a sum.

We show here a simple example to clarify how the semantic rules and SAMs are actually used.

Example 14 *Take the priority P-SAM Pri of Example 12. The $\surd$-labeled transitions for the process $P' = (x)(x\ i\langle y\rangle.P \mid x(o,3)\langle z\rangle.Q \mid x(o,2)\langle w\rangle.R)$ are:*

$P' \xrightarrow{\surd}_{P(Pri)} (x)P\{z/y\} \mid Q \mid x(o,2)\langle w\rangle.R$

$P' \xrightarrow{\surd}_{P(Pri)} (x)P\{w/y\} \mid x(o,3)\langle z\rangle.Q \mid R$

$P' \xrightarrow{\surd}_{P(Pri)} (x)P\{z/y\} \mid Q \mid R$

The only other admissible transitions are from P' to itself with labels of the form $u\epsilon\langle\rangle$ or $\neg u$ for any u, because all the topmost action prefixes in P' operate on the restricted channel x. Here the last transition is the most interesting, since it features an interaction between two outputs and one input, with the output with the lowest priority, $(o,2)$, being discarded. We give here some highlights on the derivation. Using rule (PP-Pref)*, the transition:*

$x\ i\langle y\rangle.P \xrightarrow{x\ i\langle y'\rangle}_{P(Pri)} P\{y'/y\}$

can be derived for each y'. Similarly the transitions:

$x(o,3)\langle z\rangle.Q \xrightarrow{x(o,3)\langle z\rangle}_{P(Pri)} Q \qquad x(o,2)\langle w\rangle.R \xrightarrow{x(o,2)\langle w\rangle}_{P(Pri)} R$

can be derived. Let us consider the interaction between the first two sub-terms (considering before the interaction between the last two should give the same result because of condition 4 in Definition 4). They can interact according to the action synchronizations $(i,(o,3),((o^+,3),MP_{0,0},EQ_1))$ or $(i,(o,3),((o^-,3),MP_{1,0},EQ_0))$. In the first case we obtain label $(o^+,3)$ and to satisfy the correctness of the guesses in rule (PP-Synchr) *we need $y' = z$. Then the action is synchronized with the lower priority output using rule* (PP-Synchr) *again via $((o^+,3),(o,2),((o^+,3),MP_{0,0},EQ_0))$. It is finally closed* *using rule* (PP-Res) *since* $(o^+, 3) \in Fin$ *and it gives rise to the desired transition. Notice that* $(o^-, 3)$ *can only interact with* $(o, 2)$ *using action synchronization* $((o, 2), (o^-, 3), ((o^-, 3), MP_{0,1}, EQ_0))$ *(or, more precisely, its symmetric), but since* $(o^-, 3) \notin Fin$ *this can not be extended to a derivation of a transition for the whole process.*

We comment here the example on news server outlined in the Introduction.

Example 15 (News server) *The transitions described in the Introduction for the news server can be derived, with suitable labels, in P-PRISMA by considering a P-SAM with six actions: in and out interacting using Milner synchronization and producing* τ *as a result,* in_b *and* out_b *interacting using broadcast synchronization, and* ϵ*. Such a P-SAM can also be built using a coproduct construction in the category of P-SAMs, as we will show in Section 6.*

Also, more complex scenarios can be considered. For instance the broadcast action can be tagged with some additional information on the content of the news, and different input actions can be chosen to receive only some of them. For instance we can have actions $out - CS$ *for computer science news and* $out - math$ *for mathematical news. Correspondingly we can have actions* $in - CS$ *and* $in - math$*, retrieving the corresponding news, and* $in - all$ *retrieving both of them. A process interested only in some kind of news must however explicitly use actions to discard the others, since broadcast enforces reception of the information by all the listening processes.*

We study the observational properties of processes using bisimilarity, as done in π-calculus. We resort to full bisimilarity (substitution-closed bisimilarity) to get a congruence.

Definition 16 (P-PRISMA bisimilarity) *A* P-bisimulation *is a relation* $\mathrm{R}_{P(S)}$ *such that* $P \,\mathrm{R}_{P(S)}\, Q$ *implies:*

- $P \xrightarrow{\alpha}_{P(S)} P' \wedge \mathrm{bn}(\alpha) \cap \mathrm{fn}(Q) = \emptyset \Rightarrow Q \xrightarrow{\alpha}_{P(S)} Q' \wedge P' \,\mathrm{R}_{P(S)}\, Q'$*;*
- *vice versa.*

A full P-bisimulation *is a substitution-closed P-bisimulation. We denote with* $\sim_{P(S)}$ *the maximal P-bisimulation (called P-bisimilarity) and with* $\approx_{P(S)}$ *the maximal full P-bisimulation (called full P-bisimilarity).*

We present now some properties of P-bisimilarity. Note that properties that hold for any SAM can be proved once and for all. Similarly, properties of processes can be related to properties of SAMs. We will show these two cases in the following lemmas. The first one shows that P-bisimilarity abstracts away from certain syntactic features of processes which are intuitively not important from an observational point of view.

Lemma 17 *The two sides of each axiom below are full P-bisimilar for each P-SAM S and each P, Q, R processes.*

$$P|Q = Q|P \qquad (P|Q)|R = P|(Q|R) \qquad P|0 = P \qquad P + P = P$$

$$P + Q = Q + P \qquad (P + Q) + R = P + (Q + R) \qquad P + 0 = P$$

$$(x)(y)P = (y)(x)P \qquad (x)P|Q = (x)(P|Q) \textit{ if } x \notin \mathrm{fn}(Q) \qquad (x)0 = 0$$

PROOF. [Sketch] Each axiom requires a coinductive proof. Axioms concerning parallel composition exploit the properties of SAMs. The proof for $P + 0 = P$ uses the fact that transitions with source 0 cannot force a branch of the sum to be taken. Axiom $(x)0 = 0$ exploits the fact that $\epsilon \notin Fin$. Proofs for other axioms are standard. □

The following lemma shows that in many P-SAMs we can compute bisimulations more easily.

Lemma 18 *Suppose that S is a P-SAM such that for each action a there is an action synchronization allowing to synchronize a and ϵ. Then $P \xrightarrow{\neg x}_{P(S)} P$ iff there is no transition of the form $P \xrightarrow{(Y)xa\vec{y}}_{P(S)} P'$.*

In these SAMs (such as Pi SAM) labels $\neg x$ do not influence bisimilarity, since they can be derived from other labels, thus can be disregarded when applying the definition of P-bisimulation.

Next theorem proves that full P-bisimilarity is compositional. This result is fundamental to compute the abstract semantics of large complex systems from the abstract semantics of their components. It extends in a non-trivial way an analogous result for π-calculus: the interesting point is that it holds for P-PRISMA over any P-SAM.

Theorem 19 *Full P-bisimilarity $\approx_{P(S)}$ is a congruence for any P-SAM S w.r.t. all the operators in P-PRISMA.*

PROOF. [Sketch] For each unary (resp. binary) operator op, we have to prove that for each P-SAM S and for each P_1, P_2, Q_1, Q_2 processes, $P_1 \approx_{P(S)} Q_1$ and $P_2 \approx_{P(S)} Q_2$ implies $\mathrm{op}(P_1) \approx_{P(S)} \mathrm{op}(Q_1)$ (resp. $\mathrm{op}(P_1, P_2) \approx_{P(S)} \mathrm{op}(Q_1, Q_2)$). The proof is by rule induction on the derivation of the transition of $\mathrm{op}(P_1)$ (resp. $\mathrm{op}(P_1, P_2)$), and each step requires a coinductive proof. However, rule induction is needed just for replication, while in the other cases it is enough to consider each operator in isolation.

We show below the cases for prefix, parallel composition and replication as examples, without considering transitions with labels $x\epsilon\langle\rangle$ and $\neg x$ whose treatment is trivial. The cases for the remaining operators are similar.

Case prefix) We have to prove that for each P-SAM S, prefix $xa\vec{y}$ and pair of processes P and Q such that $P \approx_{P(S)} Q$ we have $xa\vec{y}.P \approx_{P(S)} xa\vec{y}.Q$. Thus we have to prove that, for each substitution σ, $(xa\vec{y}.P)\sigma$ and $(xa\vec{y}.Q)\sigma$ can perform the same transitions, going into full P-bisimilar processes. The only transitions to consider are the ones from rule (PP-Pref), which have the same label as required and lead to states $P\sigma\{\vec{z}/\vec{y}\}$ and $Q\sigma\{\vec{z}/\vec{y}\}$ which are full P-bisimilar by hypothesis. In general, we do not need to consider explicitly the substitution σ, since this corresponds to choosing $P' = P\sigma$ and $Q' = Q\sigma$.

Case $|$) Suppose that $P_1 \approx_{P(S)} Q_1$ and $P_2 \approx_{P(S)} Q_2$. We have three rules to check to show $P_1|P_2 \approx_{P(S)} Q_1|Q_2$. Let us consider rule (PP-Synchr). Most of the conditions deal only with the labels, thus they are verified for P_1 and P_2 iff they are verified for Q_1 and Q_2. The only condition to check is $(Y_1 \cup Y_2) \cap (\text{fn}(P_1) \cup \text{fn}(P_2)) = \emptyset$. This can be satisfied since names in $Y_1 \cup Y_2$ are bound, thus they can be α-converted if necessary. We have to prove that the two resulting processes, namely $(\vec{s})(P_1'|P_2')$ and $(\vec{s})(Q_1'|Q_2')$ are full P-bisimilar. Thanks to α-conversion, we can suppose that $\vec{s}$ is the same in both the cases. By hypothesis $P_1' \approx_{P(S)} Q_1'$ and $P_2' \approx_{P(S)} Q_2'$. By coinductive hypothesis, $P_1'|P_2' \approx_{P(S)} Q_1'|Q_2'$. Finally, using closure under restriction contexts, $(\vec{s})(P_1'|P_2') \approx_{P(S)} (\vec{s})(Q_1'|Q_2')$. The cases for rules (PP-Par) and (PP-TauPar) are simpler than the one just shown.

Case !) We have to prove that if $P \approx_{P(S)} Q$, then $!P \approx_{P(S)} !Q$. We have to use rule induction for that case. If $!P \xrightarrow{\alpha}_{P(S)} P'$, then we also have $P|!P \xrightarrow{\alpha}_{P(S)} P'$, which is a premise. By inductive hypothesis on the context $\bullet|!\bullet$, $Q|!Q \xrightarrow{\alpha} Q'$ with $Q' \approx_{P(S)} P'$. Since also $!Q$ has the same transition, the thesis follows. □

Corollary 20 *The structural congruence defined by axioms in Lemma 17 is a full P-bisimulation.*

PROOF. The axioms P-bisimulate thanks to Lemma 17, congruence holds thanks to Theorem 19, and reflexivity, symmetry and transitivity are trivial. □

3.1 A case study: π-Calculus

Let $L = \{\text{in}_n | n \in \mathbb{N}\}$ with $\text{mod}(\text{in}_n) = in^n$. We will show that P-PRISMA over Pi_L is essentially π-calculus (as expected). The obtained semantics is not the standard one, but the channel-located semantics described in [16].

The standard semantics can be obtained by slightly changing the definition of bisimulation. We consider the following syntax for π-calculus processes:

$$P ::= 0 \mid u(\vec{x}).P \mid \overline{u}\vec{x}.P \mid P_1|P_2 \mid P_1 + P_2 \mid (u)P \mid !P$$

For simplicity, we do not allow τ prefix, but it can easily be obtained via synchronization on a local channel.

We denote with $\equiv_\pi$ the structural congruence on π processes. Also, we write $P \xrightarrow{\alpha}_\pi P'$ if the transition is derived using standard π-calculus early semantics (see, e.g., [23]) and $P \xrightarrow{\alpha}_{l\pi} P'$ if it is derived using the channel-located semantics (see [16]). Similarly, we denote with $\sim_\pi$ standard early bisimilarity and with $\sim_{l\pi}$ the channel-located bisimilarity for π-calculus described in [16]. Roughly, the channel-located semantics makes observable the name of the free channel where a synchronization is performed (i.e., the label $u\tau$ denotes a τ on the free channel u, while the label τ denotes a synchronization on a restricted channel) and is thus more distinguishing than the standard one. We refer to [16] for a detailed comparison between the two semantics.

Definition 21 *We define the uniform encoding function $[\![-]\!]$ from π processes into P-PRISMA processes over Pi_L as the homomorphic extension to the whole calculus of:*

$$[\![u(\vec{x}).P]\!] = u \ \mathrm{in}_{|\vec{x}|} \ \vec{x}.[\![P]\!] \qquad [\![\overline{u}\vec{x}.P]\!] = u \ \mathrm{out}_{|\vec{x}|} \ \vec{x}.[\![P]\!]$$

where $\mathrm{out}_n = \overline{\mathrm{in}_n}$. *The mapping can be extended to channel located labels by defining:*

$$[\![u\vec{x}]\!] = u \ \mathrm{in}_{|\vec{x}|} \ \vec{x} \qquad [\![(\vec{y})\overline{u}\vec{x}]\!] = (\mathrm{Set}(\vec{y}))u \ \mathrm{out}_{|\vec{x}|} \ \vec{x} \qquad [\![u\tau]\!] = u\tau\langle\rangle \qquad [\![\tau]\!] = \surd$$

The translation $[\![(\vec{y})\overline{u}\vec{x}]\!]$ loses the order of extruded names, but this is unimportant, since in π-calculus all different orderings can be obtained thanks to structural congruence.

The following theorem shows the relationship between the behaviors of π processes and of their translations into P-PRISMA.

Theorem 22 *Let P be a π process. $P \xrightarrow{\alpha}_{l\pi} P'$ iff $[\![P]\!] \xrightarrow{[\![\alpha]\!]}_{P(Pi_L)} [\![P_1]\!]$ and $P_1 \equiv_\pi P'$.*

PROOF. [Sketch] The proof is by structural induction on π processes, and it has a case for each operator. One must prove that π transitions correspond to P-PRISMA transitions whose labels are translations of π labels (but P-PRISMA can have transitions with labels $x\epsilon\langle\rangle$ and $\neg x$ too, since these ones have no correspondence in π-calculus).

Notably, the structural congruence of π-calculus can be simulated since translations of structural congruent processes are P-bisimilar (see corollary 20).

We will consider parallel composition in more detail, and sketch the remaining operators.

In π-calculus parallel processes can interact according to rules (par), (com) and (close). Suppose that $Q_1|Q_2 \xrightarrow{\alpha}_{l\pi} Q_1'|Q_2'$.

If the transition is derived using rule (par) then by hypothesis $Q_1 \xrightarrow{\alpha}_{l\pi} Q_1'$, $Q_2 = Q_2'$ and $\mathrm{bn}(\alpha) \cap \mathrm{fn}(Q_2) = \emptyset$. We have $[\![Q_1|Q_2]\!] = [\![Q_1]\!]|[\![Q_2]\!]$ and by inductive hypothesis $[\![Q_1]\!] \xrightarrow{[\![\alpha]\!]}_{P(Pi_L)} [\![Q_1']\!]$. We have two cases corresponding to $[\![\alpha]\!] = \surd$ and $[\![\alpha]\!] \neq \surd$. In the first case we can use rule (PP-TauPar) to derive the desired transition. In the second case using rule (PP-Epsilon) we can derive $[\![Q_2]\!] \xrightarrow{x\epsilon\langle\rangle}_{P(Pi_L)} [\![Q_2]\!]$ where x is the subject of α. We can then use rule (PP-Synchr) to derive the desired transition using action synchronization $(a, \epsilon, (a, MP_{\mathrm{ar}(a),0}, EQ_0))$.

If the transition is derived using either rule (com) or (close) then we can use rule (PP-Synchr) using action synchronization $(a, \overline{a}, (\tau, MP_{0,0}, EQ_{\mathrm{ar}(a)}))$. This ensures that the resulting label is $x\tau\langle\rangle$ where x is the subject of the two interacting actions, and that the guesses for the parameters are correct. Also, if the output is bound (and rule (close) is used in π-calculus) then a restriction for the bound name is added because of the clause about closed names.

Notice that no other transitions with labels that are transitions of π-calculus labels can be derived in P-PRISMA, thus the correspondence holds in both directions.

For restriction, rule (PP-Pass) corresponds to rule (res), rule (PP-Open) to rule (open) and rule (PP-Res) to rule (tau).

The proofs for other operators are similar. $\square$

In P-PRISMA each process can always do idle steps to itself, with labels of the form $x\epsilon\langle\rangle$ or $\neg x$, which have no correspondence in π-calculus. These transitions do not influence P-bisimilarity, thus the following corollary holds.

Corollary 23 *$[\![P]\!] \sim_{Pi_L} [\![P']\!] \Leftrightarrow P \sim_{l\pi} P' \Rightarrow P \sim_{\pi} P'$.*

A similar result holds also for the corresponding congruences.

4 From directional to symmetric communication

In this section we introduce F-PRISMA, a generalization of P-PRISMA based on fusions instead of input/output communications. F-PRISMA can also be seen as the generalization of Fusion calculus [22] with SAMs.

Moving from P-PRISMA to F-PRISMA we drop action modalities, i.e. we use F-SAMs. The semantics has to be changed to take into account the main advantage of Fusion synchronization: the scope of fusions is determined by the restriction operators, thus it can affect also parallel components. This allows, e.g., to model a shared state.

The syntax for F-PRISMA processes is the same of P-PRISMA processes (see Definition 13), the only difference being that now prefixes are not binders as in P-PRISMA: the only binder is restriction.

Even if the effect of the generalization on the syntax is minimal, the effect on the semantics is important. The LTS semantics is in Table 3. We decorate the transitions with $F(S)$ where S is the used F-SAM.

The main difference w.r.t. the semantics of P-PRISMA is that now all the labels (but $\neg x$) carry an additional information: the current substitution. This is necessary since input is no more bound, thus the effect of the synchronization should be propagated and applied to parallel terms until the substituted variable is bound (see rules (PF-Par) and (PF-TauPar)).

Rule (PF-Pref) is now simpler, since the prefix is just copied into the label: the substitution will be computed later, by rule (PF-Synchr). In fact, this rule computes the most general unifier of the set of equations equating names to be merged, and the substitution is applied to the term, to the label, and, as far as free names are concerned, tracked in the label.

When a name z is restricted, one has to check that it has not been chosen as representative of a non trivial equivalence class, and this is done by checking that $z \notin \mathrm{Im}(\pi)$. If the condition is not satisfied a derivation using a different mgu π has to be used.

Rule (PF-TauRes) has been added to take care of labels $(\surd, \pi)$.

Most of the results shown in Section 3 can be extended to this generalized setting. We start by generalizing the definition of bisimilarity (cfr. Definition 16).

Definition 24 (F-PRISMA bisimilarity) *A* F-bisimulation *is a relation* $\mathrm{R}_{F(S)}$ *such that* $P\,\mathrm{R}_{F(S)}\,Q$ *implies:*

Table 3
F-PRISMA semantics.

$$\frac{}{xa\vec{y}.P \xrightarrow{xa\vec{y},\mathrm{id}}_{F(S)} P}\ (\text{PF-Pref}) \qquad \frac{P' \equiv_\alpha P \xrightarrow{\alpha}_{F(S)} Q \equiv_\alpha Q'}{P' \xrightarrow{\alpha}_{F(S)} Q'}\ (\text{PF-Alpha})$$

$$\frac{P_1 \xrightarrow{(Y_1)xa_1\vec{y}_1,\pi_1}_{F(S)} P_1' \qquad P_2 \xrightarrow{(Y_2)xa_2\vec{y}_2,\pi_2}_{F(S)} P_2' \qquad \Phi}{P_1|P_2 \xrightarrow{(W)xc\vec{w},\pi|_{\setminus(Y_1\cup Y_2)}}_{F(S)} (\vec{s})(P_1'|P_2')\pi}\ (\text{PF-Synchr})$$

$$\frac{x \in \mathcal{N}}{P \xrightarrow{x\epsilon\langle\rangle,\mathrm{id}}_{F(S)} P}\ (\text{PF-Epsilon}) \qquad \frac{x \text{ is not an active subject of } P}{P \xrightarrow{\neg x}_{F(S)} P}\ (\text{PF-Indip})$$

$$\frac{P_1 \xrightarrow{(Y)xa\vec{y},\pi}_{F(S)} P_1' \quad P_2 \xrightarrow{\neg x}_{F(S)} P_2 \quad Y \cap \mathrm{fn}(P_2) = \emptyset}{P_1|P_2 \xrightarrow{(Y)xa\vec{y},\pi}_{F(S)} P_1'|P_2\pi}\ (\text{PF-Par})$$

$$\frac{P \xrightarrow{(Y)xa\vec{y},\pi}_{F(S)} P' \quad z \notin \mathrm{Set}(\vec{y}) \cup \{x\} \quad z \notin \mathrm{Im}(\pi)}{(z)P \xrightarrow{(Y)xa\vec{y},\pi|_{\setminus\{z\}}}_{F(S)} (z)P'}\ (\text{PF-Pass})$$

$$\frac{P \xrightarrow{(Y)xa\vec{y},\pi}_{F(S)} P' \quad z \in \mathrm{Set}(\vec{y}) \setminus \{x\} \setminus Y \quad z \notin \mathrm{Im}(\pi)}{(z)P \xrightarrow{(\{z\}\cup Y)xa\vec{y},\pi|_{\setminus\{z\}}}_{F(S)} P'}\ (\text{PF-Open})$$

$$\frac{P \xrightarrow{(W)xa\vec{y},\pi}_{F(S)} P' \quad a \in Fin \quad z \notin \mathrm{Im}(\pi) \quad \mathrm{Set}(\vec{w}) = W}{(z)P \xrightarrow{\surd,\pi|_{\setminus\{z\}}}_{F(S)} (z\vec{w})P'}\ (\text{PF-Res})$$

$$\frac{P \xrightarrow{\surd,\pi}_{F(S)} P'}{P|Q \xrightarrow{\surd,\pi}_{F(S)} P'|Q\pi}\ (\text{PF-TauPar}) \qquad \frac{P \xrightarrow{\surd,\pi}_{F(S)} P' \quad z \notin \mathrm{Im}(\pi)}{(z)P \xrightarrow{\surd,\pi|_{\setminus\{z\}}}_{F(S)} (z)P'}\ (\text{PF-TauRes})$$

$$\frac{P_1 \xrightarrow{\alpha}_{F(S)} P_1' \quad \alpha \neq x\epsilon\langle\rangle,\mathrm{id} \quad \alpha \neq \neg x}{P_1 + P_2 \xrightarrow{\alpha}_{F(S)} P_1'}\ (\text{PF-Sum})$$

$$\frac{P|!P \xrightarrow{\alpha}_{F(S)} P'}{!P \xrightarrow{\alpha}_{F(S)} P'}\ (\text{PF-Repl})$$

where the premise Φ in rule (PF-Synchr) is the conjunction of the six conditions:

correctness of the synchronization: $(a_1, a_2, (c, \mathrm{Mob}, \doteq)) \in AS$;

freshness of extruded names: $Y_1 \cap Y_2 = \emptyset$, $(Y_1 \cup Y_2) \cap (\mathrm{fn}(P_1) \cup \mathrm{fn}(P_2)) = \emptyset$;

forced fusions: $\pi = \mathrm{mgu}(\{\vec{y}_{i_1}[j_1] = \vec{y}_{i_2}[j_2] | \mathrm{inj}_{i_1}(j_1) \doteq \mathrm{inj}_{i_2}(j_2)\} \cup \{x = y | x\pi_1 = y\pi_1 \vee x\pi_2 = y\pi_2\})$ where we choose elements not in $Y_1 \cup Y_2$ as representatives for the equivalence classes of names in π, whenever possible;

arguments of c: $\vec{w}[k] = (\vec{y}_i[j])\pi$ iff $\mathrm{Mob}(k) = \mathrm{inj}_i(j)$;

names extruded by c: $W = \mathrm{Set}(\vec{w}) \cap (Y_1 \cup Y_2)$;

closed names: $\mathrm{Set}(\vec{s}) = (Y_1 \cup Y_2)\pi \setminus W$ (any order can be chosen for $\vec{s}$).

- $P \xrightarrow{\alpha}_{F(S)} P' \wedge \mathrm{bn}(\alpha) \cap \mathrm{fn}(Q) = \emptyset \Rightarrow Q \xrightarrow{\alpha}_{F(S)} Q' \wedge P' \,\mathrm{R}_{F(S)}\, Q'$;
- *vice versa.*

A full F-bisimulation *is a substitution-closed F-bisimulation. We denote with* $\sim_{F(S)}$ *the maximal F-bisimulation (called F-bisimilarity) and with* $\approx_{F(S)}$ *the maximal full F-bisimulation (called full F-bisimilarity).*

The analogous of Lemma 17 holds for full F-bisimilarity (i.e., all the axioms in Lemma 17 are valid w.r.t. $\approx_{F(S)}$) and the analogous of Lemma 18 holds when $\rightarrow_{F(S)}$ is considered instead of $\rightarrow_{P(S)}$. The proofs are similar and thus omitted.

Also Theorem 19 can be extended to the new setting.

Theorem 25 *Full F-bisimilarity* $\approx_{F(S)}$ *is a congruence for any SAM S w.r.t. all the operators in F-PRISMA.*

PROOF. [Sketch] The proof is similar to the one of Theorem 19. The main differences are that now closure under substitutions is needed in rules (PF-Synchr), (PF-Par) and (PF-TauPar).

4.1 A case study: Fusion calculus

The results in Section 3.1 can be extended too, obtaining an analogous correspondence between F-PRISMA and Fusion calculus [22]. The main differences are that now the F-SAM $Fusion_L$ should be used. Since a channel-located semantics for Fusion in the style of [16] has never been studied in detail, we directly state the correspondence result with standard Fusion semantics.

We remember here Fusion syntax and refer to [22] for the description of Fusion semantics. We use subscript f to denote Fusion transitions, structural congruence, and bisimilarity relations.

$$P ::= 0 \mid u\vec{x}.P \mid \overline{u}\vec{x}.P \mid P_1|P_2 \mid P_1 + P_2 \mid (u)P \mid !P$$

Fusion transitions are divided into communication actions, i.e. inputs $u\vec{x}$ and outputs $(\vec{y})\overline{u}\vec{x}$, and fusion actions ϕ where ϕ is an equivalence relation among names with a finite number of non-singleton equivalence classes. The encoding of Fusion processes and communication labels into F-PRISMA processes and labels is analogous to the one in Definition 21. The correspondence result is however a bit different, since in F-PRISMA we use substitutions instead of fusions and we apply them immediately instead of when the corresponding names are restricted.

Theorem 26 *Let P be a Fusion process. $P \xrightarrow{\alpha}_f P'$ iff:*

(1) α is a communication action, $[\![P]\!] \xrightarrow{[\![\alpha]\!]}_{F(Fusion_L)} [\![P_1]\!]$ and $P_1 \equiv_f P'$ or;
(2) α is a fusion action, $[\![P]\!] \xrightarrow{\alpha'}_{F(Fusion_L)} [\![P_1\pi]\!]$ and $P_1\pi \equiv_f P'\pi$ where α' can be either $(\surd, \pi)$ or $(x\tau\langle\rangle, \pi)$ for some $x \in \mathrm{fn}([\![P]\!])$ and where π is a mgu of α.

Corollary 27 *$[\![P]\!] \approx_{Fusion_L} [\![P']\!] \Rightarrow P \approx_f P'$.*

We refer to [5] for more details about the correspondence.

5 P-PRISMA vs F-PRISMA

In the previous sections we have introduced two parametric calculi P-PRISMA and F-PRISMA. In this section we compare them.

First, the two calculi differ only for mobility, not for synchronization, thus they are equivalent in the particular case where there are no parameters. Notice in particular that a SAM S where all actions have arity 0 can be considered both a P-SAM and a F-SAM, i.e. the mapping symm (see Definition 8) is the identity. Similarly P-PRISMA and F-PRISMA processes on S coincide.

Lemma 28 *Let S be a SAM where all actions have arity 0. Let P be a process on S. $P \xrightarrow{\alpha}_{P(S)} P'$ iff $P \xrightarrow{\alpha'}_{F(S)} P'$ where $\alpha' = \alpha$ if $\alpha' = \neg x$, $\alpha' = \alpha, \mathrm{id}$ otherwise.*

PROOF. Trivial, just by checking that the inference rules for $\rightarrow_{P(S)}$ and $\rightarrow_{F(S)}$ coincide under the hypothesis on S. $\square$

When mobility is introduced, the correspondence follows the idea of the mapping from π-calculus into Fusion calculus [22].

Definition 29 *The encoding $[\![-]\!]$ from P-PRISMA into F-PRISMA is the homomorphic extension to the whole calculus of the function that maps each prefix $xa\vec{y}.P$ to $(\vec{w})xa\vec{y}.[\![P]\!]$ where $\vec{w}$ contains all the elements in $\vec{y}$ which have input modality.*

The operational correspondence is described below. Notice that the semantics of P-PRISMA is early, while the semantics of F-PRISMA is late, thus the correspondence is a bit more complex than expected. We have preferred these styles of semantics since they are the most used for π-calculus and Fusion calculus respectively.

Theorem 30 *Let S be a P-SAM and $T = \mathrm{symm}(S)$ the corresponding F-SAM. Let P be a P-PRISMA process. We have $P \xrightarrow{\alpha}_{P(S)} P'$ iff $Q \xrightarrow{\alpha'}_{F(T)} Q'$ and there exists a substitution π which is the identity on all names but the input names in α such that $Q'\pi = [\![P']\!]$ and:*

- *$\alpha' = \alpha$ if $\alpha' = \neg x$;*
- *$\alpha' = (\mathrm{Set}(\vec{w}) \cup Y)xa\vec{y}, \mathrm{id}$ if $\alpha = (Y)xa\vec{y}\pi$ and $\vec{w}$ contains all the names in $\vec{y}$ which have input type;*
- *$\alpha' = (\surd, \mathrm{id})$ if $\alpha = \surd$.*

PROOF. The proof of the only if is by induction on the derivation of the P-PRISMA transition. We have a case for each rule. We show just the most relevant.

(PP-Pref): $xa\vec{y}.P \xrightarrow{xa\vec{y}\pi}_{P(S)} P\pi$ where $\pi = \{\vec{z}/\vec{y}\}$ is the identity but on input names in $\vec{y}$. We have $[\![xa\vec{y}.P]\!] = (\vec{w})xa\vec{y}.[\![P]\!]$. Using rule (PF-PREF) we have $xa\vec{y}.[\![P]\!] \xrightarrow{xa\vec{y},\mathrm{id}}_{F(T)} [\![P]\!]$. We can then apply rule (PF-OPEN) for each name in $\vec{w}$ obtaining $(\vec{w})xa\vec{y}.[\![P]\!] \xrightarrow{(\mathrm{Set}(\vec{w}))xa\vec{y},\mathrm{id}}_{F(T)} [\![P]\!]$ as required.

(PP-Synchr): $P_1|P_2 \xrightarrow{(W)xc\vec{w}}_{P(S)} (\vec{s})(P_1'|P_2')$ if all the conditions in Φ are satisfied, $P_1 \xrightarrow{(Y_1)xa_1\vec{y}_1}_{P(S)} P_1'$ and $P_2 \xrightarrow{(Y_2)xa_2\vec{y}_2}_{P(S)} P_2'$. We have $[\![P_1|P_2]\!] = [\![P_1]\!]|[\![P_2]\!]$. By inductive hypothesis we deduce $[\![P_1]\!] \xrightarrow{(\mathrm{Set}(\vec{w}_1)\cup Y_1)xa_1\vec{z}_1}_{F(T)} Q_1'$ and $[\![P_2]\!] \xrightarrow{(\mathrm{Set}(\vec{w}_2)\cup Y_2)xa_2\vec{z}_2}_{F(T)} Q_2'$ where $\vec{w}_1$ and $\vec{w}_2$ contain the elements of $\vec{y}_1$ and $\vec{y}_2$ respectively which have input types and there exist substitutions π_1 and π_2 which are the identity on all names but the ones in $\vec{w}_1$ and $\vec{w}_2$ respectively such that $\vec{y}_1 = \vec{z}_1\pi_1$, $\vec{y}_2 = \vec{z}_2\pi_2$, $Q_1'\pi_1 = [\![P_1']\!]$ and $Q_2'\pi_2 = [\![P_2']\!]$. We can apply rule (PF-SYNCHR) using the same synchronization used in (PP-SYNCHR). The freshness of extruded names is satisfied for output names because of the corresponding condition in (PP-SYNCHR) and for input names since they can be chosen fresh. Let us consider the forced fusions. We know from the correctness of the guesses that for corresponding names $(\vec{z}_{i_1}[j_1])\pi_{i_1} = (\vec{z}_{i_2}[j_2])\pi_{i_2}$. Since the domains of π_{i_1} and π_{i_2} are disjoint then $\pi_1 \cup \pi_2$ is a unifier. Notice that $\pi_1 \cup \pi_2$ restricted to the variables in non-singleton equivalence classes is an mgu since all the merged variables are different by hypothesis. However, for equivalence classes containing only inputs this is not relevant, since it introduces new names (corresponding to the output to be guessed). Thus we can pick a relevant mgu π' choosing for these variables any representative inside the equivalence class and choosing the output for the other classes, and have a substitution π'' which is the identity but on input variables such that $\pi'\pi'' = \pi_1 \cup \pi_2$. Note that $\mathrm{dom}(\pi')$ is a subset of the input variables in $\vec{w}$. In this way, for each k, in the F-PRISMA setting $(\vec{w}_f[k])\pi'\pi'' = (\vec{w}_f[k])(\pi_1 \cup \pi_2) = (\vec{z}_i[j])\pi_i = \vec{y}_i[j]$ as required. Notice that $\pi'|_{\setminus(\mathrm{Set}(\vec{w}_1)\cup Y_1 \cup \mathrm{Set}(\vec{w}_2)\cup Y_2)} = \mathrm{id}$. Input names in the label are still bound as required, since they are obtained from bound names, thus the chosen representative was bound in the synchronizing labels. Similarly for names whose only output component was extruded. Names which are no more extruded are bound. Notice that thanks to the condition 2 in Definition 2 no input name can be removed from the label unless it is merged with an output one, thus the set of names to be restricted is the same of the P-PRISMA scenario. Thus the bound names are $\mathrm{Set}(\vec{w}_3) \cup W$ where $\vec{w}_3$ contains the input names in $\vec{w}$. Thus we have $[\![P_1|P_2]\!] \xrightarrow{(\mathrm{Set}(\vec{w}_3)\cup W)xc\vec{w}_f,\mathrm{id}}_{F(T)} (\vec{s})(Q'_1|Q'_2)\pi'$ with $(\vec{s})(Q'_1|Q'_2)\pi'\pi'' = [\![(\vec{s})(P'_1|P'_2)]\!]$ as required.

(PP-Res): $(z)P \xrightarrow{\surd}_{P(S)} (z\vec{w})P'$ if $P \xrightarrow{(W)za\vec{y}}_{P(S)} P'$, $a \in Fin$ and $\mathrm{Set}(\vec{w}) = W$. By inductive hypothesis $[\![P]\!] \xrightarrow{(\mathrm{Set}(\vec{x})\cup W)za\vec{u},\mathrm{id}}_{F(T)} Q'$ and there exists π such that $\vec{u}\pi = \vec{y}$ and $Q'\pi = [\![P']\!]$. Since $a \in Fin$ we know that $\vec{x}$ is empty, thus also $\pi = \mathrm{id}$. Thus we can apply rule (PF-Res) to derive $[\![(z)P]\!] = (z)[\![P]\!] \xrightarrow{\surd,\mathrm{id}}_{F(T)} (z\vec{w})Q'$ as required since $(z\vec{w})Q'\,\mathrm{id} = (z\vec{w})[\![P']\!] = [\![(z\vec{w})P']\!]$.

The proof for the opposite direction is similar, by induction on the derivation of the F-PRISMA transition. We mainly have to check that, when applied to labels which are translation of P-PRISMA labels, F-PRISMA rules produce only the desired transitions. □

6 A category of SAMs

We want to analyze now how different SAMs can be combined and interact, in order to allow interoperability among calculi based on different synchronization primitives. We use basic tools from category theory [20] to this end.

SAs form a category $\mathcal{SA}$ [24] whose objects are SAs and whose morphisms are functions $h : Act_A \to Act_B$ such that $h(\epsilon_A) = \epsilon_B$ and $(a,b,c) \in AS_A \Rightarrow (h(a),h(b),h(c)) \in AS_B$. The morphism h is called *synchronous* (strict using Winskel's terminology) if $h(a) = \epsilon_B \Leftrightarrow a = \epsilon_A$. SAs with synchronous morphisms form the subcategory $s\mathcal{SA}$ of $\mathcal{SA}$. We want to extend these definitions to SAMs. P-SAMs and F-SAMs give rise to different but similar categories.

Definition 31 (Morphism between action signatures)
Let $\mathbf{A}_A = (Act_A, \mathrm{mod}_A, \epsilon_A)$ *and* $\mathbf{A}_B = (Act_B, \mathrm{mod}_B, \epsilon_B)$ *be action signatures. An* asynchronous morphism $H : \mathbf{A}_A \to \mathbf{A}_B$ *is a function* $h : Act_A \to Act_B$ *such that* $h(\epsilon_A) = \epsilon_B$, *together with a family of functions* $h_a : \underline{\mathrm{ar}(h(a))} \to \underline{\mathrm{ar}(a)}$ *indexed by actions such that* $\mathrm{mod}_A(a)[h_a(i)] = \mathrm{mod}_B(h(a))[i]$. Synchronous

morphisms *additionally require that* $h(a) = \epsilon_B \Leftrightarrow a = \epsilon_A$*.*

Each component of identity morphisms is an identity. We define morphism composition as $(h, \{h_a\}_{a \in Act_A}); (k, \{k_b\}_{b \in Act_B}) = (k \circ h, \{h_a \circ k_{h(a)}\}_{a \in Act_A})$. Note that the functions $\{h_a\}_{a \in Act_A}$ and morphism H are in opposite directions.

Definition 32 (Morphism between SAMs)
Let $(\mathbf{A}_A, Fin_A, AS_A)$ *and* $(\mathbf{A}_B, Fin_B, AS_B)$ *be two SAMs. A morphism* H *from the first to the second is a morphism* $H : \mathbf{A}_A \to \mathbf{A}_B$ *between the corresponding action signatures such that:*

(1) $a \in Fin_A \Rightarrow h(a) \in Fin_B$*;*
(2) $(a_1, a_2, (c, \mathrm{Mob}_A, \doteq_A)) \in AS_A \Rightarrow (h(a_1), h(a_2), (h(c), \mathrm{Mob}_B, \doteq_B)) \in AS_B$ *and*
- *if* $\mathrm{Mob}_A(h_c(n)) = \mathrm{inj}_i(m)$ *then* $\exists j, m'$ *such that* $\mathrm{Mob}_B(n) = \mathrm{inj}_j(m')$ *and* $\mathrm{inj}_j(h_{a_j}(m')) \doteq_A \mathrm{inj}_i(m)$*;*
- $\mathrm{inj}_i(n) \doteq_B \mathrm{inj}_j(m)$ *if and only if* $\mathrm{inj}_i(h_{a_i}(n)) \doteq_A \mathrm{inj}_j(h_{a_j}(m))$*.*

A SAM morphism is synchronous *iff the corresponding morphism between action signatures is synchronous.*

Essentially actions are mapped to other actions implementing them, and a mapping between parameters (in the opposite direction) is provided. Morphisms can remove parameters or add new synchronizations, but they must provide corresponding elements for the existing ones, preserving their behavior (i.e., action composition, computation and modality of parameters and merges among them) on the remaining parameters.

Lemma 33 *P-SAMs with asynchronous morphisms form the category* $P - \mathcal{ASYNC}$*, P-SAMs with synchronous morphisms form the subcategory* $P - \mathcal{SYNC}$ *of* $P - \mathcal{ASYNC}$*. Similarly, F-SAMs form the category* $F - \mathcal{ASYNC}$*, with the subcategory* $F - \mathcal{SYNC}$*.*

Lemma 34 *Function* symm *can be extended to give rise to forgetful functors from* $P - \mathcal{ASYNC}$ *to* $F - \mathcal{ASYNC}$ *and from* $P - \mathcal{SYNC}$ *to* $F - \mathcal{SYNC}$*.*

Processes on a SAM S_1 can be translated into processes on a SAM S_2 according to a morphism $H : S_1 \to S_2$.

Definition 35 *Given a morphism* $H = (h, \{h_a\}_{a \in Act})$*, we define the corresponding translation of P-PRISMA processes (resp. F-PRISMA processes) as the homomorphic extension of the prefix translation mapping* $xa\vec{y}$ *to* $xh(a)\vec{w}$ *where* $\vec{w}[i] = \vec{y}[h_a(i)]$*.*

In general morphisms neither preserve nor reflect process behavior, but some classes of them, such as isomorphisms, do.

Lemma 36 *An isomorphism between SAMs can only rename actions, permute their parameters, and change for each action synchronization triple the representative chosen by* Mob *inside a* $\doteq$*-equivalence class.*

Corollary 37 *Let P and Q be two processes and $H(P)$ and $H(Q)$ be their translations according to SAM isomorphism $H : S_1 \rightarrow S_2$. Then $P \approx_{P(S_1)} Q$ iff $H(P) \approx_{P(S_2)} H(Q)$ (resp. $P \approx_{F(S_1)} Q$ iff $H(P) \approx_{F(S_2)} H(Q)$).*

Products and coproducts exist and can be used to combine SAMs.

Lemma 38 *Let $((Act_1, \mathrm{mod}_1, \epsilon_1), Fin_1, AS_1)$, $((Act_2, \mathrm{mod}_2, \epsilon_2), Fin_2, AS_2)$ be two SAMs. The product in $P - \mathcal{ASYNC}$ (resp. $F - \mathcal{ASYNC}$), which we call* P-asynchronous product *(resp.* F-asynchronous product*), has the form $((Act_\otimes, \mathrm{mod}_\otimes, \epsilon_\otimes), Fin_\otimes, AS_\otimes)$ where:*

- *$Act_\otimes = Act_1 \times Act_2$ with $\mathrm{mod}_\otimes((a, b)) = \mathrm{mod}_1(a) \cdot \mathrm{mod}_2(b)$ where $\cdot$ is string concatenation;*
  - *without loss of generality, we can assume that for each (a_1, a_2), the first $\mathrm{ar}(a_1)$ parameters correspond to the ones of a_1, and the other ones are from a_2;*
- *$\epsilon_\otimes = (\epsilon_1, \epsilon_2)$;*
- *$Fin_\otimes = Fin_1 \times Fin_2$;*
- *$AS_\otimes = \{((a_1, a_2), (b_1, b_2), ((c_1, c_2), \mathrm{Mob}_\otimes, \doteq_\otimes)) |$ for each $i \in \{1, 2\}$ there is $(a_i, b_i, (c_i, \mathrm{Mob}_i, \doteq_i)) \in AS_i\}$;*
  - *$\mathrm{Mob}_\otimes$ and $\doteq_\otimes$ are defined as the union of the corresponding relations in the component objects on the respective parameters.*

The two projection maps are the obvious ones.

PROOF. [Sketch] If we consider just the part of morphisms that deals with actions, then we have a product in the category of sets and functions, which is cartesian product. If we fix an action and we consider its images, as far as parameters are concerned we obtain a coproduct diagram in the category of finite sets and functions, and this coproduct is the disjoint union. These diagrams can be extended to diagrams in $P - \mathcal{ASYNC}$ (resp. $F - \mathcal{ASYNC}$) by choosing the different elements as described in the lemma. $\square$

By a slight modification of the proof above, we can define products even in the synchronous cases.

Lemma 39 *The product in $P - \mathcal{SYNC}$ (resp. $F - \mathcal{SYNC}$), which we call* P-synchronous product *(resp.* F-synchronous product*), is like the asynchronous one, but it has no actions of the form (a_A, ϵ_B) and (ϵ_A, b_B) except (ϵ_A, ϵ_B).*

Notably, the notion of coproduct yields the same construction for the asynchronous and synchronous cases.

Lemma 40 *Let* $((Act_1, \mathrm{mod}_1, \epsilon_1), Fin_1, AS_1)$, $((Act_2, \mathrm{mod}_2, \epsilon_2), Fin_2, AS_2)$ *be two SAMs. The coproduct in* $P-\mathcal{ASYNC}$ *(resp.* $F-\mathcal{ASYNC}$*) coincides with that in* $P-\mathcal{SYNC}$ *(resp.* $F-\mathcal{SYNC}$*) and it is* $((Act_+, \mathrm{mod}_+, \epsilon_+), Fin_+, AS_+)$ *where:*

- $Act_+ = ((Act_1 \setminus \{\epsilon_1\}) \uplus (Act_2 \setminus \{\epsilon_2\}) \cup \{\epsilon_+\})$ *with* $\mathrm{mod}_+(\mathrm{inj}_i(a)) = \mathrm{mod}_i(a)$;
- $a \in Fin_i \Rightarrow \mathrm{inj}_i(a) \in Fin_+$, $\epsilon_+ \notin Fin_+$;
- $(a, b, (c, \mathrm{Mob}, \doteq)) \in AS_i \Rightarrow (\mathrm{inj}{+}_i(a), \mathrm{inj}{+}_i(b), (\mathrm{inj}{+}_i(c), \mathrm{Mob}, \doteq)) \in AS_+$ *where* $\mathrm{inj}{+}_i(x) = \mathrm{inj}_i(x)$ *for each* $x \neq \epsilon_i$, $\mathrm{inj}{+}_i(\epsilon_i) = \epsilon_+$.

The two injection maps are the obvious ones.

PROOF. [Sketch] Here we have as underlying diagram a coproduct diagram in the category of pointed sets and point-preserving functions (where ϵ is the point). The coproduct is the disjoint union with merged points. This diagram can be extended to diagrams in both $P-\mathcal{ASYNC}$ and $P-\mathcal{SYNC}$ (resp. $F-\mathcal{ASYNC}$ and $F-\mathcal{SYNC}$) by choosing the different elements as described in the lemma. $\square$

We provide now some examples on how to exploit these constructions. Products have pairs of actions with one element for each of the component SAMs as actions, with the union of parameters. For instance, the P-asynchronous product of two Pi SAMs is a message passing communication where at most two communications can be performed at each step. Also, the synchronous product of $Hoare_{L_1}$ and $Hoare_{L_2}$ is $Hoare_{L_1 \times L_2}$. Coproduct allows to merge two SAMs in a unique one preserving the behavior of each action, as proved by the following lemma.

Lemma 41 *Let* P, Q *be processes and* $H(P)$, $H(Q)$ *be their translations according to SAM injection* $H : S_1 \to S_1 + S_2$*. Then we have* $P \approx_{P(S_1)} Q$ *iff* $H(P) \approx_{P(S_1+S_2)} H(Q)$ *(resp.* $P \approx_{F(S_1)} Q$ *iff* $H(P) \approx_{F(S_1+S_2)} H(Q)$*).*

For instance, the SAM used in Example 15 is a coproduct of two P-SAMs, one isomorphic to $Pi_{\{in\}}$ and the other to $Bdc_{\{in_b\}}$. The coproduct of $Pi_{\{in_i | i \in \underline{254}\}}$ and $Bdc_{\{in_{255}\}}$ can be used to model TCP/IP protocol, where address 255 is used for broadcast. Clearly this is just an intuition, since far more refined techniques are needed to model TCP/IP in full details.

We conclude by presenting some interesting applications of our framework. Notice that in the examples the modeling effort is required only to choose a suitable SAM to model the desired interaction. After that, the primitives available in the model are in strict correspondence with the desired ones.

Example 42 (Introducing accounting on synchronization)
Take the SAM account with actions $\{\epsilon, c\}$ *of arity* 0*, with* $Fin = \{c\}$ *and where* $(c, \epsilon, (c, \mathrm{Mob}_{0,0}, \doteq_0))$ *is the only non trivial synchronization. The asynchronous product of account with any SAM* S *allows a controller process* P_c *to count the number of synchronizations performed by a process* P*. Not accounted actions can be added via a coproduct with another SAM. Let* P *be a process without restrictions and let* x *be one of its free names. Let* H *be the inclusion morphism mapping each action* a *from* S *to* (a, ϵ)*. Suppose that* S *contains an action* \$ *of arity* 0*. Then* $(x)(H(P)|!x(\epsilon, c)\langle\rangle.y(\$, c)\langle\rangle.0)$ *with the product synchronization behaves as* $(x)P$ *(up to translation of actions) with the synchronization specified by* S*, but it sends a message* $(\$, c)$ *on channel* y *for each synchronization performed by* P *on channel* x*. In fact, synchronization with* (ϵ, c) *is required to get a final action on* x*.*

Example 43 (Using π-calculus in a priority scenario) *Consider an infrastructure built for priority communication as specified in Example 12. Suppose that one wants to run a* π *process* P_π *in that framework. Suppose for simplicity that* P_π *uses just unary prefixes. We will show how* P_π *can be made to interact with the other processes (although, clearly, it will not be able to fully exploit the priority mechanism). The translation from* π*-calculus to P-PRISMA can be used to have a corresponding P-PRISMA process* P_M *on the P-SAM* $Pi_{\{\mathrm{in}_1\}}$*. In the category* $P - \mathcal{ASYNC}$ *there is a morphism* $H_n : Pi_{\{\mathrm{in}_1\}} \to Pri$ *that maps* in_1 *to* i*,* out_1 *to* (o, n) *and* τ *to* (o^+, n) *for any statically chosen priority* n*. The corresponding translation allows to automatically produce a priority process* $P_P = H_n(P_M)$*. The process essentially has all the outputs at the fixed priority* n*, and it inputs the message with the highest priority as specified by the priority synchronization. Notice that to have priority communication with many different actions we can just extend the priority P-SAM (by considering the coproduct with other copies of itself with different actions) and then apply the same procedure.*

7 Conclusion and future work

We have introduced a formal framework to deal with parametric synchronization and mobility, and we have shown that such features can be handy to model complex systems in a direct way. We have also shown how to extend some relevant part of the theory of π-calculus and Fusion calculus (operational and abstract semantics) to the parametric setting. Other parts of the theory can be generalized too (e.g., open semantics), but have been left outside of this work for space constraints.

Options for future work include the application of PRISMA to particular domains (e.g., service oriented computing, sensor networks) to test its expressive power and find the SAMs that better allows to model and reason about such a kind of systems. Also, it would be interesting to extend analysis techniques (typing, up to techniques for bisimulation, ...) to the parametric setting.

References

[1] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proc. of POPL'01*, pages 104–115. ACM Press, 2001.

[2] J.C.M. Baeten and W.P. Weijland. *Process algebra.* Cambridge University Press, 1990.

[3] M. Boreale, R. Bruni, L. Caires, R. De Nicola, I. Lanese, M. Loreti, F. Martins, U. Montanari, A. Ravara, D. Sangiorgi, V. Vasconcelos, and G. Zavattaro. SCC: a service centered calculus. In *Proc. of WS-FM 2006*, volume 4184 of *Lect. Notes in Comput. Sci.*, pages 38–57. Springer, 2006.

[4] M. Boreale, M. G. Buscemi, and U. Montanari. D-fusion: A distinctive fusion calculus. In *Proc. of APLAS'04*, volume 3302 of *Lect. Notes in Comput. Sci.*, pages 296–310. Springer, 2004.

[5] R. Bruni and I. Lanese. PRISMA: A mobile calculus with parametric synchronization. In *Proc. of TGC'06*, Lect. Notes in Comput. Sci. Springer, 2007. To appear.

[6] M. G. Buscemi and U. Montanari. Cc-pi: A constraint-based language for specifying service level agreements. In *Proc. of ESOP'07*, Lect. Notes in Comput. Sci. Springer, 2007. To appear.

[7] N. Busi, R. Gorrieri, C. Guidi, R. Lucchi, and G. Zavattaro. SOCK: a calculus for service oriented computing. In *Proc. of ICSOC'06*, volume 4294 of *Lect. Notes in Comput. Sci.*, pages 327–338. Springer, 2006.

[8] M. Carbone, K. Honda, N. Yoshida, and R. Milner. Structured communication-centred programming for web services. In *Proc. of ESOP'07*, Lect. Notes in Comput. Sci. Springer, 2007. To appear.

[9] Silvano Dal Zilio. Mobile processes: A commented bibliography. In *MOVEP'00*, volume 2067 of *Lect. Notes in Comput. Sci.*, pages 206–222. Springer, 2000.

[10] P. Degano and U. Montanari. A model for distributed systems based on graph rewriting. *Journal of the ACM*, 34(2):411–449, 1987.

[11] C. Ene and T. Muntean. Expressiveness of point-to-point versus broadcast communications. In *Proc. of FCT'99*, volume 1684 of *Lect. Notes in Comput. Sci.*, pages 258–268. Springer, 1999.

[12] C. Ene and T. Muntean. A broadcast-based calculus for communicating systems. In *Proc. of IPDPS'01*. IEEE Computer Society, 2001.

[13] C. Fournet and G. Gonthier. The reflexive chemical abstract machine and the Join calculus. In *Proc. of POPL'96*, pages 372–385. ACM Press, 1996.

[14] D. Hirsch and U. Montanari. Synchronized hyperedge replacement with name mobility. In *Proc. of CONCUR'01*, volume 2154 of *Lect. Notes in Comput. Sci.*, pages 121–136. Springer, 2001.

[15] I. Lanese. *Synchronization strategies for global computing models*. PhD thesis, Computer Science Department, University of Pisa, Pisa, Italy, 2006.

[16] I. Lanese. Concurrent and located synchronizations in pi-calculus. In *Proc. of SOFSEM'07*, volume 4362 of *Lect. Notes in Comput. Sci.*, pages 388–399. Springer, 2007.

[17] I. Lanese and U. Montanari. Synchronization algebras with mobility for graph transformations. In *Proc. of FGUC'04*, volume 138 of *Elect. Notes in Th. Comput. Sci.*, pages 43–60. Elsevier Science, 2004.

[18] I. Lanese and E. Tuosto. Synchronized hyperedge replacement for heterogeneous systems. In *Proc. of COORDINATION 2005*, volume 3454 of *Lect. Notes in Comput. Sci.*, pages 220–235. Springer, 2005.

[19] A. Lapadula, R. Pugliese, and F. Tiezzi. A calculus for orchestration of web services. In *Proc. of ESOP'07*, Lect. Notes in Comput. Sci. Springer, 2007. To appear.

[20] S. MacLane. *Categories for the Working Mathematician*. Springer, 1971.

[21] R. Milner, J. Parrow, and J. Walker. A calculus of mobile processes, I and II. *Inform. and Comput.*, 100(1):1–40,41–77, 1992.

[22] J. Parrow and B. Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *Proc. of LICS '98*. IEEE, Computer Society Press, 1998.

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

[24] G. Winskel. Synchronization trees. *Theoret. Comput. Sci.*, 34:33–82, 1984.

A Inductive definition of transitions labeled $\neg x$

An inductive definition of transitions with labels $\neg x$ is in Table A.1.

Table A.1
Inductive definition of label $\neg x$.

$$0 \xrightarrow{\neg x} 0 \qquad \frac{x \neq z}{xa\vec{y}.P \xrightarrow{\neg z} xa\vec{y}.P}$$

$$\frac{P \xrightarrow{\neg x} P \quad x \neq y}{(y)P \xrightarrow{\neg x} (y)P} \qquad (x)P \xrightarrow{\neg x} (x)P$$

$$\frac{P_1 \xrightarrow{\neg x} P_1 \qquad P_2 \xrightarrow{\neg x} P_2}{P_1|P_2 \xrightarrow{\neg x} P_1|P_2}$$

$$\frac{P_1 \xrightarrow{\neg x} P_1 \qquad P_2 \xrightarrow{\neg x} P_2}{P_1 + P_2 \xrightarrow{\neg x} P_1 + P_2} \qquad \frac{P \xrightarrow{\neg x} P}{!P \xrightarrow{\neg x} !P}$$