PRISMA: A parametric calculus based on synchronization algebras with mobility

Roberto Bruni and Ivan Lanese

Computer Science Department, University of Pisa, Italy.
{bruni,lanese}@di.unipi.it

Abstract. We present PRISMA, a parametric framework for modeling distributed and mobile systems with different communication policies, defined as synchronization algebras with mobility (SAMs). This allows to choose each time the SAM that corresponds to the real synchronization used, and also to prove properties for different synchronization models at once. The paradigm we propose is based on process calculi to facilitate the reuse of techniques and results from the vast literature. We define both the operational semantics and a compositional observational semantics for PRISMA. We present a detailed case study on Fusion Calculus and many different SAMs as examples. Finally, we exploit basic categorical constructions to relate and compose SAMs.
Keywords. Synchronization algebras, mobile calculi, bisimilarity congruences, Fusion Calculus.

1 Introduction

Since the pioneering papers by Robin Milner [22] and Tony Hoare [16], the use of process algebras and calculi has kept proliferating at an impressive rate. Though nowadays the most prominent process calculus is the π-calculus [23], many variants of it exist (see, e.g., the nice commented survey in [10]), exploiting different communication primitives and focusing on different aspects, but where interaction is always the key issue [17, 24, 20, 12, 7, 14, 5, 4, 8]. The main features of process calculi are roughly: a clean syntax that focuses on carefully chosen primitives, an operational semantics (in the LTS or reduction style) and abstract observational equivalences (like trace equivalence or bisimulation) that quotients out syntactically different processes that offer the same interaction patterns to the outside world. The widespread use of process calculi is probably due to the reconciliation of the two faces of computer science: rigorous mathematical modeling with a close look toward implementation [25, 13, 9, 6, 26]. As process calculi approach the third decade of their existence, they offer a full-fledged framework for the specification, analysis, and verification of interactive systems, also supported by some industry-level tools and prototype platforms [1–3].

Analysis of relationships among calculi have been carried out to study and compare their expressiveness, and primitives used for extending one specific calculus have often been transferred to other calculi, but surprisingly enough, the semantics of any process calculus has always been studied in isolation from the others. For example, behavioral equivalences which are not congruences are often closed w.r.t. process embedding in suitable contexts expressed in the same calculus, which can cause strange situations to arise, like in the case of the asynchronous fragment of π-calculus, whose processes have different abstract semantics depending on whether they are seen as π-calculus processes or not (because different contextual closures are chosen). In general, whenever a process calculus is designed, it is implicitly assumed that all actors (i.e., processes) will play (i.e., communicate) accordingly to the same script (i.e., communication pattern).

In the last decades, modern distributed computing has evolved as well, thanks to new technological advancements. Global computing (GC) and Service Oriented Computing (SOC) are centered around the idea of platform independence and maximum interoperability. Therefore, emerging and future interactive systems should be designed according to the foreseeable scenario where heterogeneous processes can share the same world and find a fruitful way of interact together.

Our belief is that process calculi are mature enough to move one step further in the direction sketched above. Imagine a join calculus process exchanging information with a π-calculus agent within the same ambient, or a fusion process cooperating with some old-fashioned CCS agent. After all, in principle, this is already possible for many languages that have been inspired by those calculi. The more challenging aspect is that any process, in any calculus, should be ready to communicate even with processes from calculi that are not invented and designed yet, as this is the "open world" vision of SOC.

Our present contribution addresses interoperability by defining a parametric setting, called PRISMA calculus, where different communication mechanisms can be suitably embedded and studied. PRISMA can act both as a unifying framework and as communication monitor, disciplining the possible interactions between processes with different vocabularies. For the sake of presentation, the set of primitives is kept to the minimal extent needed to study communication and interaction, but we conjecture that other features (e.g., locations, encryption, probability) can be transferred from the literature without too much effort. More precisely, the operators we consider are action prefix, parallel composition, restriction, external choice and replication. The key point is that prefixes are parametric w.r.t. the *synchronization algebra with mobility* (SAM for short) under consideration, which establishes the possible communication patterns and those that are disallowed. The name of the calculus, PRISMA, is intended to expose the many different faces it could be looked at.

An important source of inspiration for our approach is the work of Glynn Winskel on *synchronization algebras* (SAs) [27], which we extend with the concept of *mobility*, as required to model modern GC systems. We follow the approach of Fusion Calculus [24] to mobility, which extends the one of π-calculus with the concept of name fusion. We have chosen this approach because, contrary to the case of π-calculus, input and output are symmetric in fusion, thus its communication pattern is more easily generalizable to a case where we may have actions that are neither inputs nor outputs.

SAMs were first defined in [18], in the context of Synchronized Hyperedge Replacement (SHR) [11, 15], a graph transformation framework. The introduction of SAMs was motivated by their provision of a uniform presentation of the two existing synchronization models for SHR. Like PRISMA, also SHR provides a unifying framework for modeling systems, but SHR is more suitable for architectural models since it represents explicitly the structure of the system. Instead, PRISMA is aimed at studying the linguistic aspect of interoperability, since a process calculus can be easily extended with different primitives.

The proposed extension with mobility of SAs improves a lot the pre-existing framework of [27], which was tailored for calculi such as CCS and CSP, to keep it in line with current research, that concentrates on more sophisticated mobile calculi such as π-calculus and its variants, and with current fields of application, such as SOC and GC. Our framework, while recovering all the results of SAs, allows to model different mobility patterns, apt to deal with many real problems of distributed applications. This is shown via many examples, and the case study on Fusion Calculus.

As main results, we prove that hyperbisimilarity is a congruence for PRISMA with any SAM, we show how to build SAMs by composing basic ones and how to prove properties for general classes of synchronizations. Furthermore, our presentation of SAMs is more general and more refined w.r.t. the one in [18].

Structure of the paper. In the paper, we assume the reader has some familiarity with process calculi. § 2 presents a simple calculus with SAs without mobility as a warm-up for the general case, § 3 defines SAMs and shows some examples. In § 4 we define the PRISMA calculus and analyze its operational and abstract semantics. This section also contains the main theorem of the paper, which proves that hyperbisimilarity is a congruence for PRISMA. The extensive case study on Fusion Calculus is detailed in § 5, while § 6 analyzes the relationships among different SAMs. This section uses some basic concepts from category theory, which can be found, e.g., in [21]. Finally, § 7 contains some conclusions and traces for future work. For referees' convenience, Appendix A recalls the semantics of Fusion Calculus, while main proofs are in Appendix B.

2 Channels and synchronization algebras

Our first step toward the definition of a parametric calculus based on synchronization algebras with mobility (SAMs) is to analyze first a simpler case without mobility patterns, which can also help the reader to gradually understand the more general and complex communication mechanism of PRISMA.

We assume to have a set $\mathcal{N}$ of channel names ranged over by x, y, z and a disjoint set *Act* of actions ranged over by a, b, c. First of all we consider a calculus with syntax $P ::= 0 \mid xa.P \mid P|P$, i.e. with a null process 0, (channel-located) action prefixes xa, and parallel composition $|$. The basic interaction primitive is the execution of some action a on some channel x. Interaction among actions is modeled using synchronization algebras (SAs), which were first defined by Winskel [27].

Definition 2.1 (Winskel's synchronization algebra). *A synchronization algebra is a tuple* $(Act, \varepsilon, 0, \bullet)$ *where Act is a set of actions with two constants* ε *and* 0*, plus a function* $\bullet : Act \times Act \to Act$ *for action composition, such that* $\bullet$ *is associative and commutative,* 0 *is the absorbing element for* $\bullet$ *(i.e.,* $0 \bullet a = 0$*) and* $a \bullet b = \varepsilon$ *iff* $a = b = \varepsilon$*.*

Intuitively, $a \bullet b = c$ means that action a can synchronize with action b yielding action c as a result: commutativity and associativity are assumed to guarantee that the order in which the same actions synchronize is not important. If actions a and b cannot

$$xa.P \xrightarrow{xa} P \qquad \frac{x \in \mathcal{N}}{P \xrightarrow{x\varepsilon} P} \qquad \frac{P_1 \xrightarrow{xa_1} P_1' \qquad P_2 \xrightarrow{xa_2} P_2' \qquad (a_1,a_2,c) \in ActCmp}{P_1|P_2 \xrightarrow{xc} P_1'|P_2'}$$

Fig. 1. Rules for prefix and $|$.

synchronize, then $a \bullet b = 0$. For instance, in CCS we have that action a synchronizes with action $\overline{a}$ giving τ, i.e. that $a \bullet \overline{a} = \tau$.

The action ε, which stands for "not taking part in the synchronization", allows for the uniform definition of both synchronization and asynchronous execution of actions. Thus the synchronization of any a and ε always corresponds to the asynchronous execution of a. Last condition ensures that no synchronization can disappear yielding ε.

Even in the case without mobility we use a slightly different and more general presentation than Winskel's one: we consider a set of triples generalizing function $\bullet$ to a relation *ActCmp*. As a direct consequence, nondeterministic synchronizations are allowed. Furthermore, 0 is no longer needed, since synchronization between a and b is automatically forbidden letting $(a,b,c) \notin ActCmp$ for all c. Commutativity and associativity of $\bullet$ have to be rephrased in the new setting. In particular, associativity amounts to say that if $(a,b,c),(c,d,e) \in ActCmp$ then there exists $f \in Act$ such that $(b,d,f),(a,f,e) \in ActCmp$.

The self-explaining operational rules for this simple calculus are in Figure 1. Note that the labels of the LTS are *channel-located* actions xa for $x \in \mathcal{N}$ and $a \in Act$.

Remark 2.2. In many simple cases, like those considered above, channels can simply be managed as part of the action. In fact, if we eliminate channels and we consider new action pairs of the form (x,a) where x is the old channel and a the old action, we can preserve the semantics by substituting each triple (a,b,c) with the set of triples $\{((x,a),(x,b),(x,c))|x \text{ is a channel}\}$. We prefer the first presentation since it is more intuitive and it has a more distributed flavor (in the second one essentially we are simulating all the channels with a single centralized one). Furthermore, the second presentation is less suitable to be extended, as we will see shortly.

By changing the SA, the simple calculus above can behave as (fragments of) CCS or CSP, with the added feature of channel names. For CCS, we need just one channel, and a SA with only triples of the form $(a,\overline{a},\tau),(\lambda,\varepsilon,\lambda)$ and their symmetric $(\overline{a},a,\tau),(\varepsilon,\lambda,\lambda)$, where a is any input action and λ can be any action. For CSP, we consider instead the SA with triples (a,a,a) for each action a (our parallel composition $|$ corresponds to $||_{Act}$ in CSP).

As a second step toward PRISMA, we add a restriction construct (x) that binds channel x, declaring it as local. To manage this extension we have to add a new part to the SA: a set $Fin \subseteq Act$ of final actions, which are the actions that can traverse a bound channel. Intuitively, we want to express that some actions, being part of a synchronization that is not yet concluded, are meaningful only if the missing part of the synchronization can still be provided. But when we restrict a channel no more actions can be synchronized on it, thus unmatched actions are not allowed to propagate further: *Fin* contains all the other ones. For instance, for CCS, we would have $Fin = \{\tau\}$, while, for CSP, $Fin = Act$.

$$\frac{P \xrightarrow{xa} P' \qquad x \neq z}{(z)P \xrightarrow{xa} (z)P'} \qquad \frac{P \xrightarrow{xa} P' \quad a \in Fin}{(x)P \xrightarrow{\surd} (x)P'} \qquad \frac{P \xrightarrow{\surd} P'}{P|Q \xrightarrow{\surd} P'|Q} \qquad \frac{P \xrightarrow{\surd} P'}{(x)P \xrightarrow{\surd} (x)P'}$$

Fig. 2. Rules for restriction.

The rules for operational semantics of restriction are in Figure 2. The distinguished symbol $\surd \notin Act$ notifies that an action has been performed on a hidden channel. The action $\surd$ can only be propagated by requiring parallel components to stay idle, thus enforcing interleaving. By comparison, in CCS interleaving is forced when propagating τ. Here however, restrictions are not on actions as in CCS, but on channels. Thus we want to distinguish between an action performed on a free channel (which is thus visible) like $x\tau$ and an action performed on a local channel (which is not visible) like $\surd$. Thus, in our setting, $x\tau$ is used to force interleaving in CCS-style synchronization on global channel x, while $\surd$ is used, with any synchronization model, on restricted channels.

Note that, after having added restriction, simulating channels by modifying the SA as in Remark 2.2 is not easy, since the restriction operator explicitly refers a channel. Furthermore, the behavior of the restriction operator depends on both the channel and the action in the label (whether it is in Fin or not).

3 Synchronization algebra with mobility (SAM)

PRISMA calculus further generalizes the prefixes presented in § 2. In fact, PRISMA prefixes are based on a SAM, which allows actions to carry some channel names as arguments, and which prescribes how to combine them.

Essentially, we use name mobility in the π-calculus style. In order to do that, we have to define for each action a its arity $\mathrm{ar}(a)$, which is the number of channel names that can be carried by a. When synchronization is performed we have to specify two things: (1) how the arguments attached to the resulting action are computed (the component Mob in Definition 3.3), (2) which names are merged (the component $\doteq$ in Definition 3.3). We have chosen channel fusion as the key primitive for mobility as first done in Fusion Calculus [24], one of the variants of π-calculus. The choice is motivated by the fact that Fusion Calculus inherits the expressive power of π-calculus (which can be encoded into fusion), while adding fusion primitive, which also allows to update a shared state since substitutions can affect parallel processes. Furthermore this extension is obtained by simplifying and making more symmetric the π-calculus, a change that helps to generalize the semantics to different synchronization models. In fact, in π-calculus, input and output are treated ad hoc, and this gives no hint on how to deal with general actions (which can be neither inputs nor outputs).

Notation. We shall use the following conventions. We use $\uplus$ to denote disjoint set union. We denote with inj_1 and inj_2 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 $comp(x)$ the element $\mathrm{inj}_{3-i}(x)$ in $B \uplus A$. We denote with $\underline{n}$ the set $\{1,\ldots,n\}$ (where $\underline{0} \stackrel{\text{def}}{=} \emptyset$) while id_n is the identity function on it.

Fig. 3. Action synchronization.

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. Given a function f, the function $f|_S$ (resp. $f|_{\setminus S}$) is obtained by restricting f to S (resp. to $\mathrm{dom}(f) \setminus S$). Also, when set operations (e.g., $\cup$) are used on function f, it is implicitly assumed that it is seen as set of pairs $(a, f(a))$. We use $\circ$ to denote the standard composition of functions, i.e. $(f \circ g)(x) = g(f(x))$. Given a vector $\vec{v}$ and an integer i we denote with $\vec{v}[i]$ the i-th element of $\vec{v}$ while $\mathrm{Set}(\vec{v})$ is the set of elements in $\vec{v}$. We denote with $\mathrm{mgu}(E)$ an idempotent substitution resulting from computing the most general unifier on the set of equations E, when it exists. Finally, the transitive closure of relation R is R^+.

Before defining SAMs, we still need some preliminary definitions.

Definition 3.1 (Action signature). *An* action signature $\mathbf{A}$ *is a triple* $(Act, \mathrm{ar}, \varepsilon)$ *where* Act *is the set of actions,* $\varepsilon \in Act$*, and* $\mathrm{ar} : Act \to \mathbb{N}$ *is the arity function satisfying* $\mathrm{ar}(\varepsilon) = 0$.

Definition 3.2 (Action composition).
Let $\alpha = (a_1, b_1, (c_1, \mathrm{Mob}_1, \doteq_1))$ *and* $\beta = (a_2, b_2, (c_2, \mathrm{Mob}_2, \doteq_2))$ *with* $\mathrm{Mob}_i : \underline{\mathrm{ar}(c_i)} \to \underline{\mathrm{ar}(a_i)} \uplus \underline{\mathrm{ar}(b_i)}$ *a function and* $\doteq_i$ *an equivalence relation on* $\underline{\mathrm{ar}(a_i)} \uplus \underline{\mathrm{ar}(b_i)}$ *for* $i = 1, 2$ *and where* $c_1 = a_2$*. The composition of* α *and* β *is the tuple* $(a_1, b_1, b_2, (c_2, \mathrm{Mob}_3, \doteq_3))$ *where the function* $\mathrm{Mob}_3 : \underline{\mathrm{ar}(c_2)} \to \underline{\mathrm{ar}(a_1)} \uplus \underline{\mathrm{ar}(b_1)} \uplus \underline{\mathrm{ar}(b_2)}$ *is defined by* $\mathrm{Mob}_3 = \mathrm{Mob}_2 \circ [\mathrm{Mob}_1, id_{\underline{\mathrm{ar}(b_2)}}]$*, and the equivalence relation* $\doteq_3$ *on* $\underline{\mathrm{ar}(a_1)} \uplus \underline{\mathrm{ar}(b_1)} \uplus \underline{\mathrm{ar}(b_2)}$ *is defined as the projection on the above specified domain of the minimal equivalence relation* R *on* $\underline{\mathrm{ar}(a_1)} \uplus \underline{\mathrm{ar}(b_1)} \uplus \underline{\mathrm{ar}(c_1)} \uplus \underline{\mathrm{ar}(b_2)}$ *such that* $x \mathrm{R} y$ *if* $x \doteq_1 y \vee x \doteq_2 y \vee \mathrm{Mob}_1(x) = y$.

A similar composition is defined when $c_1 = b_2$ *instead of* $c_1 = a_2$.

Definition 3.3 (Action synchronization relation).
Given an action signature $\mathbf{A} = (Act, \mathrm{ar}, \varepsilon)$*, an action synchronization relation* $ActCmp$ *on* $\mathbf{A}$ *is a set of triples of the form* $(a, b, (c, \mathrm{Mob}, \doteq))$ *where* $a, b, c \in Act$*,* $\mathrm{Mob} : \underline{\mathrm{ar}(c)} \to \underline{\mathrm{ar}(a)} \uplus \underline{\mathrm{ar}(b)}$ *is a function and* $\doteq$ *is an equivalence relation on* $\underline{\mathrm{ar}(a)} \uplus \underline{\mathrm{ar}(b)}$ *such that:*

1. $(a, b, (c, \mathrm{Mob}, \doteq)) \in ActCmp \Rightarrow (c = \varepsilon \Leftrightarrow a = b = \varepsilon)$;
2. $(a, b, (c, \mathrm{Mob}, \doteq)) \in ActCmp \Rightarrow (b, a, (c, \mathrm{Mob}', \doteq')) \in ActCmp$*, with* $\mathrm{Mob}'(x) = comp(\mathrm{Mob}(x))$ *for each* x*, and* $x \doteq' y$ *iff* $comp(x) \doteq comp(y)$;
3. $(a, b, (c, \mathrm{Mob}, \doteq)), (c, d, (e, \mathrm{Mob}', \doteq')) \in ActCmp \Rightarrow$
 $\exists f \in Act, \exists (b, d, (f, \mathrm{Mob}'', \doteq'')), (a, f, (e, \mathrm{Mob}''', \doteq''')) \in ActCmp$ *such that the composition of* $(a, b, (c, \mathrm{Mob}, \doteq))$ *and* $(c, d, (e, \mathrm{Mob}', \doteq'))$ *coincides with the composition of* $(b, d, (f, \mathrm{Mob}'', \doteq''))$ *and* $(a, f, (e, \mathrm{Mob}''', \doteq'''))$.

The Mob component serves to specify where the arguments of the resulting action are taken from, i.e. from which of the arguments of the component action. Since actual arguments are not known, 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 3. For $\doteq$ the idea is to define equivalence classes over incoming parameters: parameters in the same class are then merged. Again, a positional notation is used. Condition 1 is taken from standard SAs. Condition 2 extends commutativity of $\bullet$, dealing also with parameters. Last condition extends associativity, by specifying that the composed action takes the same parameters and forces the same merges.

Action synchronizations $(a,b,(c,\mathrm{Mob}_1,\doteq))$ and $(a,b,(c,\mathrm{Mob}_2,\doteq))$ that differ only because $\mathrm{Mob}_1(n) \neq \mathrm{Mob}_2(n)$ but with $\mathrm{Mob}_1(n) \doteq \mathrm{Mob}_2(n)$ (see the example in Figure 3) are semantically equivalent (we will show in § 6 that they are isomorphic). In fact, they will give the same allowed transitions for the PRISMA calculus we shall introduce in the next section.

Definition 3.4 (Synchronization algebra with mobility). *A* synchronization algebra with mobility *over the action signature* $\mathbf{A}$ *is a triple* $(\mathbf{A}, Fin, ActCmp)$ *that includes a set* $Fin \subseteq Act$ *of* final actions *and an action synchronization relation* $ActCmp$ *on* $\mathbf{A}$.

We present three simple examples of SAMs, taken from [18, 19] (but with different notations). In the examples we use the following shorthands: $MP_{i,j}$ is 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 $m \leq i$, $\mathrm{inj}_2(m)$ otherwise. We denote with $EQ_{i,j}$ the equivalence relation on any superset S of $\underline{i} \uplus \underline{j}$ given by $\mathrm{id}_S \cup \{(\mathrm{inj}_{n_1}(m), \mathrm{inj}_{n_2}(m)) | n_1, n_2 \in \{1,2\}, m \leq \min(i,j)\}$.

Remark 3.5. From now on, to simplify the presentation, we stop writing explicitly the triples obtained by commutativity and the omnipresent triple $(\varepsilon, \varepsilon, (\varepsilon, MP_{0,0}, EQ_{0,0}))$.

Example 3.6 (Milner SAM). Given a set of labels inp, the SAM $Milner_{inp}$ is given by:

- $Act = \{\tau, \varepsilon\} \cup \bigcup_{a \in inp} \{a, \overline{a}\}$, $Fin = \{\tau\}$;
- $\mathrm{ar}(\overline{a}) = \mathrm{ar}(a)$ for each $a \in inp$, $\mathrm{ar}(\tau) = 0$;
- $(\lambda, \varepsilon, (\lambda, MP_{\mathrm{ar}(\lambda),0}, EQ_{0,0})) \in ActCmp$ for each $\lambda \in Act$,
  $(a, \overline{a}, (\tau, MP_{0,0}, EQ_{\mathrm{ar}(a),\mathrm{ar}(\overline{a})})) \in ActCmp$ for each $a \in inp$.

$Milner_{inp}$ represents message passing *à la* π-calculus: one input a interacts with one output $\overline{a}$ with the same arity, and parameters in the same position are merged.

Example 3.7 (Hoare SAM). Given a set of labels inp, the SAM $Hoare_{inp}$ is given by:

- $Act = Fin = \{\varepsilon\} \cup inp$;
- $(\lambda, \lambda, (\lambda, MP_{\mathrm{ar}(\lambda),\mathrm{ar}(\lambda)}, EQ_{\mathrm{ar}(\lambda),\mathrm{ar}(\lambda)})) \in ActCmp$ for each $\lambda \in Act$.

Hoare synchronization models an agreement on the action to perform.

Example 3.8 (Broadcast SAM). Given a set of labels inp, the SAM Bdc_{inp} is given by:

- $Act = \{\varepsilon\} \cup \bigcup_{a \in inp} \{a, \overline{a}\}$, $Fin = \bigcup_{a \in inp} \{\overline{a}\}$;
- $\mathrm{ar}(\overline{a}) = \mathrm{ar}(a)$ for each $a \in inp$;
- $(a, \overline{a}, (\overline{a}, MP_{\mathrm{ar}(a),\mathrm{ar}(\overline{a})}, EQ_{\mathrm{ar}(a),\mathrm{ar}(\overline{a})})) \in ActCmp$ for each $a \in inp$,
  $(a, a, (a, MP_{\mathrm{ar}(a),\mathrm{ar}(a)}, EQ_{\mathrm{ar}(a),\mathrm{ar}(a)})) \in ActCmp$ for each $a \in inp$.

This SAM implements broadcast. When used in PRISMA, it forces an output $\overline{a}$ from a sequential PRISMA process to synchronize with *all* the other sequential processes in parallel, which have to perform an input a. Notice that, if one wants to have a multicast Mul_{inp}, where some process may not synchronize with the output, it is enough to add the triples $(a,\varepsilon,(a,MP_{\mathrm{ar}(a),0},EQ_{0,0}))$ for each $a \in Act$ to $ActCmp$.

We present now a more complex (and original) example: a SAM for *communication with priority* that allows many senders to communicate with just one receiver, which receives the message with the highest priority.

Example 3.9 (Priority SAM). The SAM *Pri* is defined as follows:

- $Act = \{in,\varepsilon\} \cup \{(out,n)|n \in \mathbb{N}\} \cup \{(out+,n)|n \in \mathbb{N}\} \cup \{(out-,n)|n \in \mathbb{N}\}$;
- $\mathrm{ar}(a) = 1$ for all $a \in Act, a \neq \varepsilon$;
- $(a,\varepsilon,(a,MP_{\mathrm{ar}(a),0},EQ_{0,0})) \in ActCmp$ for each $a \in Act$,
  $(in,(out,n),((out+,n),MP_{1,1},EQ_{1,1})) \in ActCmp$ for each n,
  $(in,(out,n),((out-,n),MP_{1,0},EQ_{0,0})) \in ActCmp$ for each n,
  $((out,n),(out,m),((out,\max(n,m)),MP_{1,0},EQ_{0,0})) \in ActCmp$,
  $((out,n),(out-,m),((out+,n),MP_{1,1},EQ_{1,1})) \in ActCmp$ for each $n \geq m$,
  $((out,n),(out-,m),((out-,n),MP_{0,1},EQ_{0,0})) \in ActCmp$ for each $n \geq m$,
  $((out+,n),(out,m),((out+,n),MP_{0,1},EQ_{0,0})) \in ActCmp$ for each $n \geq m$;
- $Fin = \{(out+,n)|n \in \mathbb{N}\}$.

This definition is a bit tricky: the idea is that when an *in* and an (out,n) synchronize we have a guess: either we guess that n is the highest priority and thus perform the merge on parameters and the result is $(out+,n)$ or we guess the opposite and we propagate the input variable (discarding the parameter of the output) while the result is $(out-,n)$. When the channel is restricted only correct guesses survive. A simpler approach, without guesses and that simply merges the parameters of input and output and chooses the parameter of the output with highest priority when two synchronize, does not work, since there is no way of undoing the merge of the input and of an output with low priority. Note that here non-determinism is useful for the guess, but also, in a more intuitive way, to decide which output to propagate when two with the same priority interact.

4 The PRISMA calculus

We can now present the syntax and the semantics of PRISMA.

PRISMA is parametric on a SAM, but it also takes a set *pref* of actions allowed in prefixes as additional parameter: some actions are just used to compute the synchronization, and processes cannot directly execute them. For instance, in PRISMA over the SAM with priority, processes can be prefixed with input *in* and outputs (out,n), but they cannot execute guesses $(out+,n)$ or $(out-,n)$ as prefixes.

Definition 4.1. *The syntax for processes is the following one:*

$$\begin{array}{llll} P ::= 0 & \textit{(Inaction)} & xa\vec{y}.P & \textit{(Prefix)} \\ \quad P_1|P_2 & \textit{(Parallel composition)} & P_1 + P_2 & \textit{(Nondeterministic sum)} \\ \quad (x)P & \textit{(Restriction)} & !P & \textit{(Replication)} \end{array}$$

where x is a channel name, $\vec{y}$ is a vector of channel names and a is an action in pref.

$$xa\vec{y}.P \xrightarrow{xa\vec{y},\mathrm{id}} P \quad (1) \qquad \frac{x \in \mathcal{N}}{P \xrightarrow{x\varepsilon\langle\rangle,\mathrm{id}} P} \quad (2)$$

$$\frac{P_1 \xrightarrow{(Y_1)xa_1\vec{y}_1,\pi_1} P_1' \qquad P_2 \xrightarrow{(Y_2)xa_2\vec{y}_2,\pi_2} P_2' \qquad \Phi}{P_1|P_2 \xrightarrow{(W)xc\vec{w},\pi} (\vec{s})(P_1'|P_2')\sigma} \quad (3)$$

$$\frac{P \xrightarrow{(Y)xa\vec{y},\pi} P' \quad z \notin \mathrm{Set}(\vec{y}) \cup \{x\} \quad z \notin \mathrm{Im}(\pi)}{(z)P \xrightarrow{(Y)xa\vec{y},\pi|_{\setminus\{z\}}} (z)P'} \quad (4)$$

$$\frac{P \xrightarrow{(Z)xa\vec{y},\pi} P' \quad a \in Fin \quad x \notin \mathrm{Im}(\pi) \quad \mathrm{Set}(\vec{z}) = Z}{(x)P \xrightarrow{\surd,\pi|_{\setminus\{x\}}} (x\vec{z})P'} \quad (5)$$

$$\frac{P \xrightarrow{(Y)xa\vec{y},\pi} P' \quad z \in \mathrm{Set}(\vec{y}) \setminus \{x\} \setminus Y \quad z \notin \mathrm{Im}(\pi) \quad z' \notin (\mathrm{fn}(P') \cup \{x\} \cup \mathrm{Set}(\vec{y})) \setminus \{z\}}{(z)P \xrightarrow{(\{z'\}\cup Y)xa\vec{y}\{z'/z\},\pi|_{\setminus\{z\}}} P'\{z'/z\}} \quad (6)$$

$$\frac{P \xrightarrow{\surd,\pi} P'}{P|Q \xrightarrow{\surd,\pi} P'|Q\pi} \quad (7) \qquad \frac{P \xrightarrow{\surd,\pi} P' \quad x \notin \mathrm{Im}(\pi)}{(x)P \xrightarrow{\surd,\pi_{\setminus\{x\}}} (x)P'} \quad (8)$$

$$\frac{P_1 \xrightarrow{\lambda} P_1' \quad \lambda \neq x\varepsilon\langle\rangle,\mathrm{id}}{P_1 + P_2 \xrightarrow{\lambda} P_1'} \quad (9) \qquad \frac{P|!P \xrightarrow{\lambda} P'}{!P \xrightarrow{\lambda} P'} \quad (10)$$

Φ in the premise of rule 3 is the conjunction of:

- $Y_1 \cap Y_2 = \emptyset$, $(Y_1 \cup Y_2) \cap (\mathrm{fn}(P_1) \cup \mathrm{fn}(P_2)) = \emptyset$;
- $(a_1, a_2, (c, \mathrm{Mob}, \doteq)) \in ActCmp$;
- $\sigma = \mathrm{mgu}(\{\vec{y}_{i_1}[j_1] = \vec{y}_{i_2}[j_2] | \mathrm{inj}_{i_1}(j_1) \doteq \mathrm{inj}_{i_2}(j_2)\} \cup \{x = y | x\pi_1 = y\pi_1 \vee x\pi_2 = y\pi_2\})$ where we choose elements not in $Y_1 \cup Y_2$ as representatives for the equivalence classes of names in σ, whenever possible;
- $\pi = \sigma|_{\setminus Y_1 \cup Y_2}$;
- $\vec{w}[k] = (\vec{y}_i[j])\sigma$ iff $\mathrm{Mob}(k) = \mathrm{inj}_i(j)$;
- $W = \mathrm{Set}(\vec{w}) \cap (Y_1 \cup Y_2)$;
- $\mathrm{Set}(\vec{s}) = (Y_1 \cup Y_2) \setminus W$ (any order can be chosen).

Fig. 4. Operational semantics of PRISMA.

In PRISMA, (x) is the only binder for x. We define as usual the function $\mathrm{fn}(P)$ that computes the set of free names in process P.

The operational semantics of PRISMA is the least labelled transition system generated from the inference rules in Figure 4. We also need the symmetric versions of rules 9 and 7. Note that rule 3 instead is self-symmetric.

We have two kinds of labels: the first one has the form $(Y)xa\vec{y},\pi$ and models prefix execution, but it adds to prefix $xa\vec{y}$ two additional pieces of information: a set of extruded names Y and a name substitution π. First of all, when a bound name is communicated outside its scope (i.e., extruded), we want to trace that its scope is growing, and thus put it in Y, since, when it is removed from the tuple of communicated names (e.g., because of some synchronization), we want to reintroduce the restriction. This is analogous to what is performed in π-calculus by rule (close). Note that Y is a subset of $\mathrm{Set}(\vec{y})$ and it does not contain x. Furthermore, fusions performed on global names must be traced by π, since they must be applied to parallel processes. This is performed by rule 7.

The second kind of label has the form $(\surd, \pi)$ and it is used to force interleaving. Substitution π is as before. We denote with λ a general label.

Rules 1, 2, 7, 8, 9 and 10 are straightforward. The most complex rule is the rule for synchronization (rule 3), which allows two actions a_1 and a_2 performed on the same channel x to synchronize. The main effect of the synchronization is to produce a new substitution σ, which combines the previous substitutions traced by π_1 and π_2 with the new substitution determined by the equivalence classes defined by $\doteq$. The substitution σ is applied to the two interacting processes P_1' and P_2' and to the tuple $\vec{w}$ of parameters of the resulting action c (computed according to Mob), and, as far as global names are concerned, it is traced in the label by new substitution π. W is the new set of extruded variables (a variable is extruded when it appears in the label, and it was bound before the transition). Finally, names that were extruded ($Y_1 \cup Y_2$) and are no longer in the label ($(Y_1 \cup Y_2) \setminus W$) must be closed by inserting them into $\vec{s}$ (in any order).

The rules for restriction deserve some explanations too. Rule 4 says that restriction on channel z does not influence actions on channel x. If the equivalence class of z according to π is not a singleton, we have to remove it from π, since it is not visible outside its scope. We require that it is not a representative (this is always possible, since we have a transition for each choice of the representatives among the free variables in the equivalence class), and delete the binding for it. Rule 5 closes the channel on which the action a is done. This is allowed iff $a \in Fin$, and all the extruded names are closed. Finally, rule 6 corresponds to the rule (open) of π-calculus: if the restricted name appears in the tuple of parameters of the action, then it is marked as extruded. Note that the extruded name can be α-converted (the choices that allow to build a full derivation are the ones where the chosen name does not appear in other components of the process).

We introduce here a notion of observational equivalence, namely hyperbisimulation.

Definition 4.2. *A bisimulation is a relation $\sim_S$ on terms such that $P \sim_S Q$ implies:*

- $P \xrightarrow{\surd,\pi} P' \Rightarrow Q \xrightarrow{\surd,\pi} Q' \wedge P' \sim_S Q'$,
- $P \xrightarrow{(Y)xa\vec{y},\pi} P' \wedge Y \cap \mathrm{fn}(Q) = \emptyset \Rightarrow Q \xrightarrow{(Y)xa\vec{y},\pi} Q' \wedge P' \sim_S Q'$

and vice versa, where all the transitions are derived using SAM S. A hyperbisimulation is a substitution-closed bisimulation. We denote with $\approx_S$ the maximal hyperbisimulation. If $P \approx_S Q$, we say that P and Q are hyperbisimilar. We will drop S from the notation when clear from the context.

Next lemma shows that hyperbisimilarity abstracts from some syntactic features of processes which are intuitively not important from an observational point of view.

We say that an axiom $P_1 = P_2$ on processes bisimulates if for each instantiation, the two equated processes are hyperbisimilar.

Lemma 4.3. *The following axioms bisimulate for each SAM:*
$P_1|P_2 = P_2|P_1 \quad (P_1|P_2)|P_3 = P_1|(P_2|P_3) \quad P|0 = P \quad (x)P = (y)P\{x/y\}$ *if* $y \notin \mathrm{fn}(P)$
$P_1 + P_2 = P_2 + P_1 \quad (P_1 + P_2) + P_3 = P_1 + (P_2 + P_3) \quad P + 0 = P \quad (x)(y)P = (y)(x)P$

We present here a simple lemma to show how SAMs can be used to derive properties of families of calculi at once.

Lemma 4.4. *The axiom* $(x)P|Q = (x)(P|Q)$ *if* $x \notin \mathrm{fn}(Q)$ *bisimulates if for each action a the relation ActCmp contains* $(a, \varepsilon, (a, MP_{\mathrm{ar}(a),0}, EQ_{0,0})) \in ActCmp$.

Milner synchronization satisfies the hypothesis of previous lemma, while broadcast do not. In fact with broadcast synchronization $ua\vec{y}.0|(v)v\overline{a}\vec{z}.0 \xrightarrow{\surd,\mathrm{id}} ua\vec{y}.0$ while $(v)ua\vec{y}.0|v\overline{a}\vec{z}.0$ has no $\surd$ transitions.

Lemma 4.5. *The axiom* $(x)0 = 0$ *bisimulates iff* $\varepsilon \notin Fin$.

In fact, including ε in Fin corresponds to observe internal idle steps as $\surd$.

Next theorem proves that abstract semantics is compositional, thus context closure, which depends on the set of allowed processes, is not requested. This result is fundamental to compute the abstract semantics of large complex systems from the abstract semantics of their components, and it holds for PRISMA over any SAM.

Theorem 4.6. *Hyperbisimilarity is a congruence w.r.t. all the operators in PRISMA.*

5 A case study: Fusion Calculus

We will show that PRISMA over $Milner_{\{\mathrm{in}_n | n \in \mathbb{N}\}}$ is essentially Fusion Calculus [24] (as expected), and we propose a new channel-located semantics for it. We consider the subset of Fusion Calculus whose processes are defined by:

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

i.e. containing the null process, communication prefixes, parallel composition, summation, restriction and replication. Note that we do not allow fusion prefixes, but $\{\vec{x} = \vec{y}\}.P$ can be encoded as $(z)(za\vec{x}.P|z\overline{a}\vec{y})$ for $z \notin \mathrm{fn}(P)$. We denote with $\equiv$ the structural congruence on fusion processes. For referees' convenience, the operational semantics and the abstract semantics $\approx_f$ of Fusion Calculus are recalled in Appendix A.

We define the uniform encoding function $[\![-]\!]$ from fusion processes into PRISMA processes as the homomorphic extension to the whole calculus of $[\![u\vec{x}.P]\!] = u\mathrm{in}_{|\vec{x}|}\vec{x}.[\![P]\!]$, $[\![\overline{u}\vec{x}.P]\!] = u\mathrm{out}_{|\vec{x}|}\vec{x}.[\![P]\!]$ where in_n and out_n are complementary actions. The mapping can also be extended to communication labels by defining $[\![(\vec{y})u\vec{x}]\!] = (\mathrm{Set}(\vec{y}))u\mathrm{in}_{|\vec{x}|}\vec{x}, \mathrm{id}$ and similarly for outputs. The translation loses the order of extruded names, but this is unimportant, since in fusion all different orderings can be obtained thanks to structural congruence.

The following theorem can be proved by structural induction on fusion processes.

Theorem 5.1. *Let P be a fusion process.* $P \xrightarrow{\alpha} P'$ *iff:*

1. α *is a communication action,* $[\![P]\!] \xrightarrow{[\![\alpha]\!]} [\![P_1]\!]$ *and* $P_1 \equiv P'$ *or;*
2. α *is a fusion action,* $[\![P]\!] \xrightarrow{\lambda} [\![P_1\pi]\!]$ *and* $P_1\pi \equiv P'\pi$ *where* λ *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.

Note also that in PRISMA each process can always do an idle step to itself, which has no fusion correspondence (but since any process can do it, and there are no other ε moves, it does not influence the abstract semantics).

Corollary 5.2. $[\![P]\!] \approx_{Milner} [\![P']\!] \Rightarrow P \approx_f P'$.

The main difference between Fusion Calculus and PRISMA over *Milner* is that in our more general form the τ is a normal action and is thus located. The corresponding semantics can be defined also for Fusion Calculus, by adding located fusions $x\phi$ to the set of transition labels. Furthermore, we have to change rule COM into the one below, and to add two new rules SCOPE' and SCOPE" to deal with the new labels:

$$\text{(COM)} \quad \frac{P \xrightarrow{u\vec{x}} P' \quad Q \xrightarrow{\overline{u}\vec{y}} Q' \quad |\vec{x}| = |\vec{y}|}{P|Q \xrightarrow{u\{\vec{x}=\vec{y}\}} P'|Q'}$$

$$\text{(SCOPE')} \quad \frac{P \xrightarrow{u\phi} P' \quad z\phi x \quad z \neq x \quad z \neq u}{(z)P \xrightarrow{u\phi \backslash z} P'\{x/z\}} \qquad \text{(SCOPE")} \quad \frac{P \xrightarrow{z\phi} P' \quad z\phi x \quad z \neq x}{(z)P \xrightarrow{\phi \backslash z} P'\{x/z\}}$$

The corresponding hyperbisimilarity is in general finer than the standard one, as the following examples illustrate (the examples are written in the Fusion Calculus syntax, but one can readily use the translation $[\![-]\!]$ to have them in the PRISMA setting).

Example 5.3.

$$x\vec{y}|\overline{x}\vec{z} \approx_f x\vec{y}.\overline{x}\vec{z} + \overline{x}\vec{z}.x\vec{y} + u\{\vec{y} = \vec{z}\}$$

These two processes are bisimilar with the standard semantics (where the fusion prefix is not located) but not with the located one. This example uses the prefix $u\{\vec{y} = \vec{z}\}$ containing just a fusion. However we can modify the example to:

$$x|\overline{x} \approx_f (y)(x+y)|(\overline{x}+\overline{y})$$

Here again we have two behaviors. The left process can just perform actions at x, while the right one can also perform unlocated actions $\surd$. A similar example can be written using replication instead of sum:

$$\overline{x}|!x.\overline{x} \approx_f (y)y|\overline{y}|\overline{x}|!x.\overline{x}$$

As shown in the example, located hyperbisimilarity is finer because it allows to distinguish between actions on restricted channels and actions on global channels. The two semantics are equivalent just in the simple case without sum and replication and without τ in *pref*, since in this case we can count the number of synchronizations, and the ones that are not allowed by synchronization of visible actions are the ones on restricted channels.

We think that, in general, located hyperbisimilarity can be useful, since, for accounting or performance reasons, actions performed on different channels are not equivalent. Suppose for instance that channel x is provided by some company while channel y is local and owned by the user: a process performing a synchronization on y is cheaper (and thus not equivalent) to a process performing the same synchronization using channel x. Local channels are instead all equivalent, since any interaction, included traffic check, is required to happen inside the scope of the channel.

6 A category of SAMs

We want analyze now how different SAMs can be combined and interact, in order to study interoperability among calculi based on different synchronization primitives. We use basic category theory [21] to this end.

Winskel's SAs form a category $\mathcal{SA}$ [27] with morphisms that are functions $h : Act_A \to Act_B$ such that (using our notations) $h(\varepsilon_A) = \varepsilon_B$ and $(a,b,c) \in ActCmp_A \Leftrightarrow (h(a),h(b),h(c)) \in ActCmp_B$.

Furthermore a morphism is called *synchronous* (strict in Winskel's terminology) if $h(a) = \varepsilon_B \Leftrightarrow a = \varepsilon_A$. SAs with synchronous morphisms form the subcategory $s\mathcal{SA}$ of $\mathcal{SA}$.

We want to extend these definitions to our more general setting.

Definition 6.1 (Morphism between action signatures).
Let $(Act_A, \mathrm{ar}_A, \varepsilon_A)$ and $(Act_B, \mathrm{ar}_B, \varepsilon_B)$ be two action signatures. An asynchronous morphism *H from the first to the second is a function $h : Act_A \to Act_B$ such that $h(\varepsilon_A) = \varepsilon_B$, together with a family of functions $h_a : \underline{\mathrm{ar}(h(a))} \to \underline{\mathrm{ar}(a)}$ indexed by actions.* Synchronous morphisms *are defined as asynchronous ones, but we require additionally that $h(a) = \varepsilon_B \Leftrightarrow a = \varepsilon_A$.*

In identity morphisms each component of the morphism is an identity. We define morphism composition as $(h, \{h_a\}_{a \in Act_A}) \circ (k, \{k_b\}_{b \in Act_B}) = (h \circ k, \{k_{h(a)} \circ h_a\}_{a \in Act_A})$.

Note that the functions $\{h_a\}_{a \in Act_A}$ are contravariant w.r.t. the direction of H.

Definition 6.2 (Morphism between SAMs).
Let $(\mathbf{A}_A, Fin_A, ActCmp_A)$ and $(\mathbf{A}_B, Fin_B, ActCmp_B)$ 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. *$(a_1, a_2, (c, \mathrm{Mob}_A, \doteq_A)) \in ActCmp_A \Rightarrow (h(a_1), h(a_2), (h(c), \mathrm{Mob}_B, \doteq_B)) \in ActCmp_B$ and*
   - *$\mathrm{Mob}_A(h_c(n)) = \mathrm{inj}_i(m) \Rightarrow \exists j, m'. \mathrm{Mob}_B(n) = \mathrm{inj}_j(m') \wedge h_{a_j}(m') \doteq_A m$;*
   - *$\mathrm{inj}_i(n) \doteq_B \mathrm{inj}_j(m) \Leftrightarrow \mathrm{inj}_i(h_{a_i}(n)) \doteq_A \mathrm{inj}_j(h_{a_j}(m))$.*

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

Essentially actions are mapped to other actions implementing them, and a mapping between parameters (in the reverse 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, merges among parameters and parameters computation) on the remaining parameters.

SAMs with asynchronous morphisms form the category $\mathcal{ASYNC}$, while SAMs with synchronous morphisms form the subcategory $\mathcal{SYNC}$ of $\mathcal{ASYNC}$.

One can consider just the subcategory where $(a_1, a_2, (c, \mathrm{Mob}_A, \doteq_A)) \in ActCmp_A \Leftrightarrow (h(a_1), h(a_2), (h(c), \mathrm{Mob}_B, \doteq_B)) \in ActCmp_B$, which is the most direct generalization of Winskel's one. Since we want to be more general we consider the larger category.

Lemma 6.3. *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.*

We show now that isomorphisms preserve the behavior of processes. Note that a morphism $H = (h, \{h_a\}_{a \in Act})$ can be extended to a translation of PRISMA prefixes by mapping the prefix $xa\vec{y}$ to $xh(a)\vec{w}$ where $\vec{w}[i] = \vec{y}[h_a(i)]$. The translation can be extended homomorphically to processes.

Lemma 6.4. *Let* P *and* Q *be two processes and* $H(P)$ *and* $H(Q)$ *be the corresponding translations according to SAM isomorphism* $H : S_1 \to S_2$. *Then* $P \approx_{S_1} Q$ *iff* $H(P) \approx_{S_2} H(Q)$.

Lemma 6.5. *Let* $((Act_1, \mathrm{ar}_1, \varepsilon_1), Fin_1, ActCmp_1)$ *and* $((Act_2, \mathrm{ar}_2, \varepsilon_2), Fin_2, ActCmp_2)$ *be two SAMs. The product in* $\mathcal{ASYNC}$*, which we call* asynchronous product*, has the form* $((Act_\otimes, \mathrm{ar}_\otimes, \varepsilon_\otimes), Fin_\otimes, ActCmp_\otimes)$ *where:*

- $Act_\otimes = Act_1 \times Act_2$;
- $\mathrm{ar}_\otimes((a,b)) = \mathrm{ar}_1(a) + \mathrm{ar}_2(b)$;
  - *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;
- $\varepsilon_\otimes = (\varepsilon_1, \varepsilon_2)$;
- $Fin_\otimes = Fin_1 \times Fin_2$;
- $ActCmp_\otimes = \{((a_1,a_2),(b_1,b_2),((c_1,c_2),\mathrm{Mob}_\otimes,\doteq_\otimes)) \mid$
  $(a_1, b_1, (c_1, \mathrm{Mob}_1, \doteq_1)) \in ActCmp_1 \wedge (a_2, b_2, (c_2, \mathrm{Mob}_2, \doteq_2)) \in ActCmp_2\}$;
  - $\mathrm{Mob}_\otimes$ *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.

Lemma 6.6. *The product in* $\mathcal{SYNC}$*, which we call* 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* $(\varepsilon_A, \varepsilon_B)$.

Lemma 6.7. *Let* $((Act_1, \mathrm{ar}_1, \varepsilon_1), Fin_1, ActCmp_1)$ *and* $((Act_2, \mathrm{ar}_2, \varepsilon_2), Fin_2, ActCmp_2)$ *be two SAMs. The coproduct in* $\mathcal{ASYNC}$ *coincides with that in* $\mathcal{SYNC}$ *and it has the form* $((Act_+, \mathrm{ar}_+, \varepsilon_+), Fin_+, ActCmp_+)$ *where:*

- $Act_+ = ((Act_1 \setminus \{\varepsilon_1\}) \uplus (Act_2 \setminus \{\varepsilon_2\}) \cup \{\varepsilon_+\})$;
- $\mathrm{ar}_+(\mathrm{inj}_i(a)) = \mathrm{ar}_i(a)$;
- $a \in Fin_i \Rightarrow \mathrm{inj}_i(a) \in Fin_+$, $\varepsilon_+ \in Fin_+$ *iff* $\forall i \in \{1,2\}.\varepsilon_i \in Fin_i$;
- $(a, b, (c, \mathrm{Mob}, \doteq)) \in ActCmp_i \Rightarrow (\mathrm{inj}_i(a), \mathrm{inj}_i(b), (\mathrm{inj}_i(c), \mathrm{Mob}, \doteq)) \in ActCmp_+$.

The two injection maps are the obvious ones.

We discuss now how these constructions can be applied, providing some examples. Products have pairs of actions with one element for each of the component SAMs as actions, with the union of parameters. For instance, the asynchronous product of two Milner SAMs is a message passing communication where at most two communications can happen at each step. Instead, the synchronous product of $Milner_{inp_1}$ and $Milner_{inp_2}$ is $Milner_{inp_1 \times inp_2}$.

Coproduct allows to merge two SAMs in a unique one preserving the behavior of each action. For instance, the coproduct of $Milner_{\{a,b\}}$ and $Hoare_{\{c,d\}}$ is a SAM where actions a and b (or, more precisely, their images along the morphism) interact using Milner synchronization, while actions c and d interact using Hoare synchronization. The coproduct of $Milner_{\{in_i | i \in \underline{254}\}}$ and $Bdc_{\{in_{255}\}}$ models normal TCP/IP protocol, where address 255 is used for broadcast.

We conclude by presenting a slightly more complex example.

Example 6.8 (SAMs with accounting). Consider the simple SAM *account* with actions $\{\varepsilon, c\}$ of arity 0 with $Fin = \{c\}$ and where $(c, \varepsilon, (c, \mathrm{Mob}_{0,0}, \doteq_{0,0}))$ is the only allowed kind of synchronization. The asynchronous product of *account* with any SAM S allows a controller process P_c to count the number of synchronizations performed on a channel. Let P be any process and x be the channel under accounting. Then $(x)P|!xc\langle\rangle.y\$\langle\rangle$ with the product synchronization behaves as $(x)P$ (up to translation of actions) with normal synchronization, but it sends a message \$ on channel y for each synchronization performed. Not accounted actions can be added by using coproduct with another SAM.

7 Conclusion

We have presented PRISMA, a process calculus with parametric communication patterns, which are specified in terms of a SAM. This helps the modeling phase, when the desired synchronization policy can be specified directly, instead of having to be implemented using the synchronization policy provided by the model (in normal process calculi, there is usually just one synchronization policy allowed, and it is often very simple). Furthermore, interoperability analysis can be performed, since different SAMs can be embedded in the same framework using the coproduct construction.

We have defined an observational semantics for PRISMA which is a congruence w.r.t. all the operators in the language, thus allowing compositional analysis of system behavior. Note that the result holds for any SAM. As side results, we have provided a new semantics for Fusion Calculus, where synchronizations performed on different channels (which may have different properties) are distinguished. Finally, we have shown that simple categorical tools allow to compare and compose different SAMs.

As future work, we want to put at work our model on some case studies taken from real distributed protocols. On a more theoretical side, we want to use the defined categories of SAMs to compare processes based on different synchronization models. Furthermore, we want to further study the properties of our calculus, by analyzing its expressiveness, which clearly depends on the chosen SAM. We also want to define a suitable weak observational semantics and analyze its properties.

References

1. Concurrency workbench of the new century. http://www.cs.sunysb.edu/~cwb/.
2. The edinburgh concurrency workbench. http://homepages.inf.ed.ac.uk/perdita/cwb/.
3. Profundis web. http://jordie.di.unipi.it:8080/pweb.
4. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proc. of POPL'01*, pages 104–115. ACM Press, 2001.

5. M. Abadi and A. Gordon. A calculus for cryptographic protocols: The spi calculus. *Inform. and Comput.*, 148(1):1–70, 1999.
6. L. Bettini, R. De Nicola, R. Pugliese, and G. L. Ferrari. Interactive mobile agents in x-klaim. In *Proc. of WETICE'98*, pages 110–117. IEEE Computer Society Press, 1998.
7. 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.
8. L. Cardelli and A. Gordon. Mobile ambients. In *Proc. of FoSSaCS'98*, volume 1378 of *Lect. Notes in Comput. Sci.*, pages 140–155. Springer, 1998.
9. S. Conchon and F. Le Fessant. Jocaml: Mobile agents for objective-caml. In *Proc. of ASA/MA'99*, pages 22–29. IEEE Computer Society Press, 1999.
10. S. Dal Zilio. Mobile processes: A commented bibliography. In *MOVEP'00*, volume 2067 of *Lect. Notes in Comput. Sci.*, pages 206–222. Springer, 2000.
11. P. Degano and U. Montanari. A model for distributed systems based on graph rewriting. *Journal of the ACM (JACM)*, 34(2):411–449, 1987.
12. 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.
13. P. Gardner, C. Laneve, and L. Wischik. The fusion machine. In *Proc. of CONCUR'02*, volume 2421 of *Lect. Notes in Comput. Sci.*, pages 418–433. Springer, 2002.
14. M. Hennessy and J. Riely. A typed language for distributed mobile processes (extended abstract). In *Proc. of POPL'98*, pages 378–390. ACM Press, 1998.
15. 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.
16. C. Hoare. A model for communicating sequential processes. In *On the Construction of Programs*. Cambridge University Press, 1980.
17. K. Honda and M. Tokoro. An object calculus for asynchronous communication. In *Proc. of ECOOP'91*, volume 512 of *Lect. Notes in Comput. Sci.*, pages 1–20. Springer, 1991.
18. I. Lanese and U. Montanari. Synchronization algebras with mobility for graph transformations. In *Proc. of FGUC'04*, Elect. Notes in Th. Comput. Sci., 2004. To appear.
19. 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. To appear.
20. C. Laneve and B. Victor. Solos in concert. In *Proc. of ICALP'99*, volume 1644 of *Lect. Notes in Comput. Sci.*, pages 513–523. Springer, 1999.
21. S. MacLane. *Categories for the Working Mathematician*. Springer, 1971.
22. R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lect. Notes in Comput. Sci.* Springer, 1980.
23. 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.
24. J. Parrow and B. Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *Proc. of LICS '98*. IEEE, Computer Society Press, June 1998.
25. B. C. Pierce and D. N. Turner. Pict: A programming language based on the pi-calculus. In *Proof, Language and Interaction: Essays in Honour of Robin Milner*, pages 455–494. MIT Press, 2000.
26. P. Sewell and A. Unyapoth. Nomadic pict: correct communication infrastructure for mobile computation. In *Proc. of POPL'01*, pages 116–127. ACM Press, 2001.
27. G. Winskel. Synchronization trees. *Theoret. Comput. Sci.*, 34:33–82, 1984.

A The Fusion Calculus

We recall here the semantics of Fusion Calculus. In our work we consider only a sub-calculus of the Fusion Calculus, which has no match and no mismatch operator. We suppose to have an infinite set $\mathcal{N}$ of names ranged over by $u, v, \ldots, z$. Names represent communication channels. We use $\vec{x}$ to denote a vector of names and ϕ to denote an equivalence relation on $\mathcal{N}$, called fusion, which can be represented by a finite set of equalities.

Definition A.1. *The free actions* α *and the processes* P *are defined by:*

$$\begin{array}{rll} \alpha ::= & u\vec{x} & \textit{(Input)} \\ & \overline{u}\vec{x} & \textit{(Output)} \\ & \phi & \textit{(Fusion)} \end{array} \qquad \begin{array}{rll} P ::= & 0 & \textit{(Inaction)} \\ & \alpha.P & \textit{(Prefix)} \\ & P_1|P_2 & \textit{(Composition)} \\ & (x)P & \textit{(Scope)} \\ & P_1+P_2 & \textit{(Sum)} \\ & !P & \textit{(Replication)} \end{array}$$

The scope operator is the only binder for names, thus x is bound in $(x)P$. We define the functions $\mathrm{fn}(-)$, $\mathrm{bn}(-)$ and $\mathrm{n}(-)$ that given a process or an action compute the set of free, bound and all names respectively (the names in a fusion are the names in non singleton equivalence classes). Processes are considered up to structural axioms defined as follows.

Definition A.2. *The structural congruence* $\equiv$ *between processes is the least congruence satisfying the* α*-conversion law for names, the abelian monoid laws for summation and composition (associativity, commutativity and 0 as identity), the scope laws* $(x)0 \equiv 0$, $(x)(y)P \equiv (y)(x)P$ *and the scope extrusion law* $P|(z)Q \equiv (z)(P|Q)$ *where* $z \notin \mathrm{fn}(P)$.

Note that $\mathrm{fn}(-)$ can be trivially extended to equivalence classes of processes.

Definition A.3. *A bound action is of the form* $(\vec{z})a\vec{x}$ *where* $|\vec{z}| > 0$ *and all elements in* $\vec{z}$ *are also in* $\vec{x}$. *Names in* $\vec{z}$ *are bound names. The actions consist of the free actions and the bound actions.*

For convenience we define $\phi \setminus z$ to mean $\phi \cap (\mathcal{N} \setminus \{z\})^2 \cup \{(z,z)\}$.

Definition A.4 (SOS semantics for Fusion Calculus). *The operational semantics of Fusion Calculus is the least labelled transition system generated by the inference rules in Figure 5.*

Definition A.5 (Abstract semantics for Fusion Calculus). *A bisimulation is a relation* $\sim_f$ *on terms such that* $P \sim_f Q$ *implies:*

– $P \xrightarrow{\gamma} P' \wedge \mathrm{bn}(\gamma) \cap \mathrm{fn}(Q) = \emptyset \Rightarrow Q \xrightarrow{\gamma} Q' \wedge P'\sigma \sim_f Q'\sigma$ *where* σ *is an mgu of* γ *if* γ *is a fusion, the identity otherwise*

and vice versa. A hyperbisimulation is a substitution-closed bisimulation. We denote with $\approx_f$ *the maximal hyperbisimulation.*

$$\text{(PREF)}\ \alpha.P \xrightarrow{\alpha} P \qquad \text{(SUM)}\ \frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'}$$

$$\text{(PAR)}\ \frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q} \qquad \text{(COM)}\ \frac{P \xrightarrow{u\vec{x}} P' \quad Q \xrightarrow{\overline{u}\vec{y}} Q' \quad |\vec{x}| = |\vec{y}|}{P|Q \xrightarrow{\{\vec{x}=\vec{y}\}} P'|Q'}$$

$$\text{(SCOPE)}\ \frac{P \xrightarrow{\phi} P' \quad z\phi x \quad z \neq x}{(z)P \xrightarrow{\phi\backslash z} P'\{x/z\}} \qquad \text{(OPEN)}\ \frac{P \xrightarrow{(\vec{y})a\vec{x}} P' \quad z \in \vec{x} \backslash \vec{y} \quad a \notin \{z, \overline{z}\}}{(z)P \xrightarrow{(z\vec{y})a\vec{x}} P'} \qquad \text{(PASS)}\ \frac{P \xrightarrow{\alpha} P' \quad z \notin \mathrm{n}(\alpha)}{(z)P \xrightarrow{\alpha} (z)P'}$$

$$\text{(REPL)}\ \frac{P|!P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P'} \qquad \text{(STRUCT)}\ \frac{P \xrightarrow{\alpha} P' \quad P \equiv Q \quad P' \equiv Q'}{Q \xrightarrow{\alpha} Q'}$$

Fig. 5. SOS rules for Fusion Calculus.

B Proofs

We present here a lemma that is used in the proof of Theorem 4.6, in the case of parallel composition.

Lemma B.1. *If we have a transition $P \xrightarrow{(Y)xa\vec{y}} P'$ with $y \in Y$ in PRISMA, then we also have a transition $P \xrightarrow{(Y\{y'/y\})xa\vec{y}\{y'/y\},\pi} P'\{y'/y\}$ for each $y' \notin \mathrm{fn}(P') \cup \{x\} \cup \mathrm{Set}(\vec{y})$.*

Proof. The proof is by rule induction on the derivation of the transition.

Proof (of Theorem 4.6). For each unary (resp. binary) operator op, we have to prove that for each SAM S and for each P_1, P_2, Q_1, Q_2 processes such that $P_1 \approx_S Q_1$ and $P_2 \approx_S Q_2$ we have $\mathrm{op}(P_1) \approx_S \mathrm{op}(Q_1)$ (resp. $\mathrm{op}(P_1,P_2) \approx_S \mathrm{op}(Q_1,Q_2)$). 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 (in a suitable order). Thus we will use rule induction just for replication.

The first cases of the proof are fully discussed to show the general idea, the last ones are less detailed. Note that, when referring to rule n, we mean rule n in Figure 4.

Case prefix) We have to prove that for each SAM S, each (allowed) prefix $xa\vec{y}$ and each pair of processes P and Q such that $P \approx_S Q$ we also have $xa\vec{y}.P \approx_S xa\vec{y}.Q$. 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 hyperbisimilar states. The only allowed transitions are the ones from rule 2, which are to the process itself and thus verify the thesis (by coinductive hypothesis), and the ones from rule 1, which have the same label as required and lead to states $P\sigma$ and $Q\sigma$ which are hyperbisimilar by hypothesis. In general, we have not to consider explicitly the cases of all the σ, since we can always choose $P' = P\sigma$ and $Q' = Q\sigma$ and reason without an explicit σ. Also, we have not to reason explicitly about transition labelled by ε, since they can always be trivially simulated.

Case $+$) Suppose that $P_1 \approx_S Q_1$ and $P_2 \approx_S Q_2$. We have to prove that $P_1 + P_2 \approx_S Q_1 + Q_2$. Each (non ε) transition $P_1 + P_2 \xrightarrow{\lambda} M$ is derived from rule 9 (or its symmetric, and the two cases are analogous), thus we have that $P_1 \xrightarrow{\lambda} M$. By hypothesis also $Q_1 \xrightarrow{\lambda} M'$ with $M \approx_S M'$. Since we can derive $Q_1 + Q_2 \xrightarrow{\lambda} M'$ the thesis follows.

Case (x)) Suppose that $P \approx_S Q$. We have to prove that $(x)P \approx_S (x)Q$. We have different cases according to which rule is used for deriving the transition. Notice that, in all the cases, the majority of the side conditions deal with the label and the restricted variable, thus they hold for P iff they hold for Q.Thus we can use the premise of the rule used to derive the transition for P, to derive the transition for Q using the same rule. We have to check that: (1) the derived labels are the same and (2) the resulting processes are hyperbisimilar. For rule 4, the first part is straightforward, while the second follows by coinduction. The same happens for rule 5. As far as rule 6 is concerned, we have the condition that $z' \notin \mathrm{fn}(P')$ which involves P', but notice that z' is an extruded name, thus for definition of bisimulation we have to consider just the cases where $z' \notin \mathrm{fn}(Q)$. Since z' is not an extruded name in the premise (because $z' \notin \mathrm{Set}(\vec{y})$), then it is not a free name of Q' and the condition is satisfied also for the derivation for Q. Hyperbisimilarity of the resulting processes follows from the closure by substitutions. The proof for rule 8 is straightforward.

Case $|$) Suppose that $P_1 \approx_S Q_1$ and $P_2 \approx_S Q_2$. We have to prove that $P_1|P_2 \approx_S Q_1|Q_2$. We have two rules to check here. Let us consider rule 3 first. As before, 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 $(Y_1 \cup Y_2) \cap (\mathrm{fn}(P_1) \cup \mathrm{fn}(P_2)) = \emptyset$. By definition of bisimulation we have that we need to bisimulate only transitions such that $W \cap \mathrm{fn}(Q_1|Q_2) = \emptyset$. Thus the only names that can cause problems are the ones in $(Y_1 \cup Y_2) \setminus W$. These are extruded names that will be restricted in the resulting process. Thanks to Lemma B.1, we have infinite choices for which names to choose in the premise, thus we can find a choice that will not use names in $\mathrm{fn}(Q_1) \cup \mathrm{fn}(Q_2)$. Any choice is good, since these names are restricted afterward, and the α-conversion axiom bisimulates thanks to Lemma 4.3. We still have to prove that the two resulting processes, namely $(\vec{s})(P_1'|P_2')\sigma$ and $(\vec{s})(Q_1'|Q_2')\sigma$ are hyperbisimilar. For what said before, we can suppose that $\vec{s}$ is the same in both the cases. By hypothesis $P_1' \approx_S Q_1'$ and $P_2' \approx_S Q_2'$. By coinductive hypothesis, $P_1'|P_2' \approx_S Q_1'|Q_2'$. Thanks to the closure by substitutions of hyperbisimilarity, $(P_1'|P_2')\sigma \approx_S (Q_1'|Q_2')\sigma$. Finally, for what already proved for restriction, $(\vec{s})(P_1'|P_2')\sigma \approx_S (\vec{s})(Q_1'|Q_2')\sigma$. The case for rule 7 is simpler than the one just shown.

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

Proof (of Theorem 5.1). The proof is by structural induction on fusion processes. We have a case for each operator. We have to prove that the fusion process has a transition if and only if its translation has a transition of the wanted form. As far as the only if part is concerned, note that translations of fusion processes have just three kinds of transitions: the two described by the theorem, and ε transitions to itself. Thus completeness is proved by checking at each point that no other kinds of transitions can be obtained, and that these ones are in correspondence with fusion ones. Note that, when referring to rule n, we mean rule n in Figure 4.

Case 0**)** We have no rules for 0 in Fusion Calculus, and we can have only ε transitions in PRISMA over *Milner*. This proves the thesis.

Case prefix) We have to show that if the thesis holds for P, then it also holds for $xa\vec{y}.P$. The two rules for prefix are the same, up to the translation. The thesis follows trivially.

Case $|$**)** We have to show that if the thesis holds for P and for Q, then it also holds for $P|Q$. We have two applicable rules here. Let us consider rule (PAR) first. If α is a communication label, by hypothesis $[\![P]\!] \xrightarrow{[\![\alpha]\!]} [\![P_1]\!]$ and $P_1 \equiv P'$. Then we can use rule 2 to derive $[\![Q]\!] \xrightarrow{x\varepsilon\langle\rangle} [\![Q]\!]$ and rule 3 using synchronization $(a, \varepsilon, (a, MP_{\mathrm{ar}(a),0}, EQ_{0,0}))$ to derive $[\![P|Q]\!] \xrightarrow{[\![\alpha]\!]} [\![P_1|Q]\!]$ since we have $\sigma = \mathrm{id}$. The thesis follows because $P_1|Q \equiv P'|Q$. If the action is a fusion action, its translation can be either of the form $x\tau\langle\rangle, \pi$ or of the form $\surd, \pi$. In the first case we have by hypothesis that $[\![P]\!] \xrightarrow{\lambda} [\![P_1\pi]\!]$, and we can use the same rules as before to deduce $[\![P|Q]\!] \xrightarrow{\lambda} [\![(P_1\pi|Q)\pi]\!] = [\![(P_1|Q)\pi]\!]$, since $\vec{s}$ is empty and $\sigma = \pi$ (up to a change of representatives). Last step is justified by idempotence of π. The thesis follows again. If the label has the form $\surd, \pi$, then the transition can be derived using rule 7.

Let us now consider rule (COM). By hypothesis we have two transitions $[\![P]\!] \xrightarrow{[\![u\vec{x}]\!]} [\![P_1]\!]$ and $[\![Q]\!] \xrightarrow{[\![\overline{u}\vec{y}]\!]} [\![Q_1]\!]$ with $P_1 \equiv P'$ and $Q_1 \equiv Q'$. Since the length of the two vectors coincide, the two translations use complementary actions and can thus interact using the triple $(\mathrm{in}_n, \mathrm{out}_n, (\tau, MP_{0,0}, EQ_{n,n})$. Thus we can derive a transition of the form $[\![P|Q]\!] \xrightarrow{u\tau\langle\rangle,\pi} (P_1|Q_1)\pi$ as required. Notice in fact that we have no extruded names, thus $\vec{s}$ is empty and that $\pi_1 = \pi_2 = \mathrm{id}$, thus the new σ is just given by the merge on parameters, and thus it is a mgu of $\{\vec{x} = \vec{y}\}$. Furthermore $\pi = \sigma$ since we have no bound names again. Notice that the fusion rule (COM) deals only with actions with no extruded names, while rule 3 in the semantics of PRISMA is more general. However, the more general case can be simulated by fusion, in fact extruded names can be avoided by moving restrictions to the outside using structural congruence. The transitions are the same in both the cases since the axiom for scope extrusion bisimulates.

Case (x)**)** Let us consider restriction now. Here we have three rules. Rule (SCOPE) is simulated by rule 8 if the PRISMA action is of the form $\surd, \pi$, by rule 4 if it is of the form $z\tau\langle\rangle, \pi$ with $z \neq x$ and by rule 5 if it is of the form $x\tau\langle\rangle, \pi$. In that case, the rule can be applied because $\tau \in Fin$, and we have $Z = \emptyset$ because $\mathrm{ar}(\tau) = 0$. Notice also that in all the cases the binding for x in π is deleted, and there is no need to apply it as done in fusion, since it has already been applied. The condition that $x \notin \mathrm{Im}(\pi)$ is not restrictive since we have similar transitions for each choice of the representatives.

Rule (PASS) is an instance of rule 4 when $\pi = \mathrm{id}$ for communication actions and where x is not in π for fusion actions.

Rule (OPEN) is an instance of rule 6 when $\pi = \mathrm{id}$. Note that here the substitution is applied in PRISMA while it is just traced in the label in Fusion Calculus. Also note that in fusion just x is used as extruded variable since it can be converted using α conversion and rule (STRUCT), while in PRISMA we generate any allowed name.

In order to have the other implication we have to check that PRISMA rules do not allow further transitions. PRISMA has four rules for restriction. Rule 4 is simulated by rule (PASS) when $z \notin \mathrm{n}(\pi)$ and by (SCOPE) otherwise (the restriction can be removed since z is no more used as name, furthermore the renaming is not necessary in PRISMA since a transition with a different name can be used as a premise). Rule 5 is simulated by rule (PASS) in Fusion Calculus, and it switches between the two kinds of representations of fusion actions in PRISMA. Rule 6 is simulated by rule (OPEN) (notice that here π is always the identity if the transition has been generated by the translation of a fusion process). Finally, the same observations done for rule 4 hold for rule 8, when the PRISMA label has the form $\surd, \pi$.

Case ! and $+$) The rules are the same in the two frameworks up to the translation, thus the proof is trivial. Note just that in PRISMA we require that ε transitions cannot force any choice.

Structural congruence) The first part of rule (STRUCT) is modeled since translations of equivalent fusion terms have the same allowed transitions since the corresponding axioms bisimulate (see lemmas 4.3, 4.4 and 4.5). The second part is simulated since in the theorem we just require to have the translation of an equivalent process.

Proof (of Lemma 6.5). We have to show that $S_\otimes$ is the product of S_1 and S_2 in $\mathcal{ASYNC}$ with the obvious projections P_i. In order to do that, we have to show that, for any SAM S with morphisms $H_i : S \to S_i$ we have a unique morphism $H : S \to S_\otimes$ such that $H_i = H \circ P_i$.

If we just consider the part of the morphisms that deals with actions, we have a product in the category of sets and functions, which is the cartesian product of sets. Fixing an action $a_S \in Act_S$ and considering also its images in S_1, S_2 and $S_\otimes$ along morphisms, the mappings among parameters form a coproduct diagram in the category of finite sets and functions. In that category, coproducts are disjoint unions.

For each diagram, we thus have an underlying diagram on action sets. Vice versa, let us fix a commuting diagram of this shape in the category of sets with Act_1 and Act_2 as targets of the span. For each choice of two SAMs S_1 and S_2 with Act_1 and Act_2 as action sets, we can always extend the diagram to a diagram in $\mathcal{ASYNC}$ (it is enough to use in S actions with enough parameters and without synchronizations, and to have $Fin = \emptyset$). Thus the product, if it exists, must have the cartesian product of action sets as action set. Notice that in the category of sets, we have different isomorphic choices for the cartesian product, since all the elements are equivalent and just their cardinality matters. The elements in $Act_\otimes$ instead are not equivalent, thus we have less isomorphisms (e.g. isomorphisms have to preserve ε) and not all the choices of cartesian product are good, but we have to carefully choose one product in the equivalence class. In particular, the distinguished element $\varepsilon_\otimes$, must be the pair $(\varepsilon_1, \varepsilon_2)$ for the diagram commute. Furthermore, given a triple $(a_S, b_S, (c_S, \mathrm{Mob}_S, \doteq_S))$, corresponding triples $(h_i(a_S), h_i(b_S), (h_i(c_S), \mathrm{Mob}_i, \doteq_i))$ for $i = 1, 2$ must exist (and this gives further constraints on the choice of the product).

We have to consider the parameter part now. Again, for a fixed action, each coproduct in the category of finite sets and functions can be extended to a parameter mapping in a product diagram of SAMs with fixed S_1 and S_2, thus we must have disjoint union on the parameters. Given that, the conditions on Mob and $\doteq$ are satisfied by definition, and we have no other choice for Mob and $\doteq$ that makes the diagram to commute.

Finally, a final action must have final actions as projections. Also, all that pairs of actions must be final, since otherwise we may find a SAM S with a final action $a \in Fin_S$ which is mapped to $h_1(a) \in Fin_1$ and $h_2(a) \in Fin_2$ for $(h_1(a), h_2(a)) \notin Fin_\otimes$, and in that case H will not be a SAM morphism.

Proof (of Lemma 6.6). The proof is similar to the one of Lemma 6.6, but now the only action that can be mapped to ε_i is $\varepsilon_\otimes$ because morphisms are synchronous, and not all diagrams on action sets can be lifted to diagrams on SAMs (we need that both ε_i are mapped to the same element, and no other elements are mapped there). For the non ε elements, the above reasoning applies. For ε, we must have just one element ε_+. In fact, since the only element in Act_S that can be mapped to ε_i is ε_S, and it must also be mapped to ε_{3-i}, then the diagram commutes anyway (and if we add other elements, we loose the uniqueness).

Proof (of Lemma 6.7). Here it is not enough to consider as underlying diagrams all the diagrams in the category of sets and functions, since only the ones that map ε_1 and ε_2 to the same element can be extended to diagrams in $\mathcal{ASYNC}$ or in $\mathcal{SYNC}$. But each diagram in the category of pointed sets (with ε as point) and point-preserving functions can be extended to a diagram in $\mathcal{ASYNC}$ or in $\mathcal{SYNC}$. Since, in the category of pointed sets, the coproduct is the disjoint union of the two sets with merged points, if a coproduct exists in $\mathcal{ASYNC}$ or in $\mathcal{SYNC}$, then it must have this underlying set. Furthermore h must be defined as $[h_1, h_2]$, apart from ε_+ that is mapped to ε_S. We have that $ActCmp_+$ must contain all the images of the action synchronization tuples from the two components in order to make the diagram commute, and no more, since otherwise we would have no morphism into some S not containing them.

As far as parameters are concerned, fixing one action $\mathrm{inj}_i(a_i) \in Act_+$, its parameters coincide with the parameters of a_i. The diagram commutes with $h_{\mathrm{inj}_i(a_i)} = h_{ia_i}$, i.e. the arguments of $h(inj_i(a_i))$ are mapped to the parameters of $inj_i(a_i)$ in the same way as they are mapped to the arguments of a_i by h_{ia_i}. Note that this definition is the unique (up to isomorphisms) that makes the diagram commute. The conditions on Mob and $\doteq$ are satisfied by definition.

Finally, actions that are final are injected into final actions. The distinguished action ε_+ is final if at least one of the ε_i is final (in the synchronous case, both of them are final if one is, otherwise we cannot have both the morphisms h_1 and h_2).