PRISMA: A Mobile Calculus with Parametric Synchronization⋆

Roberto Bruni[1] and Ivan Lanese[2]

[1] Computer Science Department, University of Pisa, Pisa, Italy
bruni@di.unipi.it
[2] Computer Science Department, University of Bologna, Bologna, Italy
lanese@cs.unibo.it

Abstract. We present PRISMA, a parametric calculus that can be instantiated with different interaction policies, defined as synchronization algebras with mobility of names (SAMs). We define both operational semantics and observational semantics of PRISMA, showing that the second one is compositional *for any* SAM. We give examples based on heterogeneous SAMs, a case study on Fusion Calculus and some simple applications. Finally, we show that basic categorical tools can help to relate and to compose SAMs and PRISMA processes in an elegant way.

1 Introduction

Since the pioneering papers by Robin Milner [16] and Tony Hoare [10], the use of process description languages has kept proliferating at an impressive rate. Though nowadays the most prominent calculus is the π-calculus [17], many variants of it exist (see, e.g., the nice commented survey in [4]), exploiting different communication primitives and focusing on different aspects, but where interaction is a key issue [18, 8, 3, 1].

While process calculi are used for modeling different kinds of systems, ranging from computer networks to biological systems, at different levels of abstraction, 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, [7] introduces a broadcast variant of π-calculus, while [6] proves that there is no uniform encoding of it into the π-calculus.

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 calculi for SOC should face the coexistence of several interaction policies within the same model. We want to overcome the limitation of previous proposals by allowing processes to interact using 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

⋆ Research supported by the Project FET-GC II IST 16004 Sensoria.

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 $S = !(x) publish\ in\langle x\rangle.(news\ out_b\langle x\rangle|S'[x])$ where $(-)$ is name restriction, $|$ is parallel composition, . is prefixing and ! is replication. Here $S'[x]$ is a generic context exploiting x. A news provider instead has the form $P = (info) publish\ out\langle info\rangle$. Consider the system $P|S|C_1|C_2$, where $C_i = (y) news\ in_b\langle y\rangle.Use_i[y]$ is a suitable client, for each i. The components P and S can interact on channel *publish*, leading to $S|(info)(news\ out_b\langle info\rangle|S'[info])|C_1|C_2$. Then, a broadcast interaction among three different components (a sender and two receivers) delivers the news to the clients, leading in one step to $S|(info)(S'[info]|Use_1[info]|Use_2[info])$.

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 specified in a general framework. For the sake of presentation, the set of primitives is kept to a minimal extent, but we conjecture that other features (e.g., locations, ambients, encryption, probability) 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 and tools (i.e., independent from the synchronization model). Finally, when expressed in a uniform framework, different synchronization models can be more easily compared and integrated (e.g., the compound use of different policies is rather straightforward). The name PRISMA (the Italian for *prism*) is intended to expose the many communication facets of our calculus.

Although PRISMA is based on name fusion, it is reductive to see it just as an extension of Fusion Calculus [18], because much more general interactions are allowed in PRISMA. We have chosen fusion as the key primitive for mobility 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 2). Anyway we think that our approach can capture most interaction calculi, and in particular the π-calculus, but this complicates the technicalities of the approach (see e.g. the approach in [3] to deal with distinctions).

SAMs improve in a crucial way *synchronization algebras* [19] (which stem from ACP communication functions [2]), which were tailored for calculi such as

CCS and CSP, to keep them in line with more sophisticated mobile calculi for scenarios such as Global Computing.

SAMs were first defined in [13], in the context of a graph transformation framework called Synchronized Hyperedge Replacement (SHR) [5, 9], to provide a uniform presentation of two existing synchronization models. PRISMA is not a mere translation of SHR. Like 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, PRISMA focuses on the linguistic aspect of interaction, and is more useful to analyze the interactions between synchronization patterns and other primitives, since process calculi can be easily extended with specific features (e.g., probabilities, pattern matching). Our presentation of SAMs is also more general and polished w.r.t. the one in [13].

As main results: (i) we prove that the observational semantics of PRISMA, called *hyperbisimilarity*, is a congruence under any SAM, (ii) we discuss how to build complex SAMs by composing basic ones and (iii) we show how to prove properties for general classes of synchronization protocols. The expressiveness of our calculus is demonstrated via original examples on a news server (already outlined), on communications with accounting, on interoperability between different synchronization policies, and via the case study on Fusion Calculus.

Structure of the paper. § 2 recalls SAMs and shows some examples. In § 3 we define the PRISMA calculus, analyze its operational and abstract semantics and prove the congruence theorem for hyperbisimilarity. The case study on Fusion Calculus is detailed in § 4, while § 5 analyzes the relationships among different SAMs using basic concepts from category theory, which can be found, e.g., in [15]. All the material in § 3–5 is original to this contribution. Finally, § 6 contains some conclusions and plans for future work. A full discussion on PRISMA calculus and related topics can be found in the Ph.D. thesis of the second author [12].

2 Synchronization algebra with mobility (SAM)

Notation. We write $A \uplus B$ to denote the disjoint union of 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 $comp(\mathrm{inj}_i(x))$ the element $\mathrm{inj}_{3-i}(x)$ in $B \uplus A$. We denote with $\underline{n}$ the set $\{1, \ldots, n\}$ (where $\underline{0} \stackrel{\mathrm{def}}{=} \emptyset$), while id_n is the identity function on it. 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 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 $\vec{v}$ while $\mathrm{Set}(\vec{v})$ is the set of elements in $\vec{v}$. 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, which are an extension of Winskel's synchronization algebras (SAs) [19] 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* $(Act, \mathrm{ar}, \epsilon)$ *where* Act *is the set of actions,* $\mathrm{ar} : Act \to \mathbb{N}$ *is the* arity function *specifying the number of arguments of each action and* $\epsilon \in Act$ *has* $\mathrm{ar}(\epsilon) = 0$.

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 of the form* $(a, b, (c, \mathrm{Mob}, \doteq))$ *where* $a, b, c \in Act$, $\mathrm{Mob} : \underline{\mathrm{ar}(c)} \to \underline{\mathrm{ar}(a)} \uplus \underline{\mathrm{ar}(b)}$ *and* $\doteq$ *is an equivalence relation on* $\underline{\mathrm{ar}(a)} \uplus \underline{\mathrm{ar}(b)}$.

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 $(a, \overline{a}, (\overline{a}, \mathrm{Mob}_1, \doteq))$.

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. 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 x and y are merged, and the result is $\overline{a}\langle y\rangle$ (if y is chosen as representative of the equivalence class).

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 the example in Figure 1) are semantically equivalent (we will show in § 5 that they are isomorphic).

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

Fig. 1. Action synchronization.

Definition 3 (Action synchronization composition).
Given $\alpha = (a_1, b_1, (c_1, \mathrm{Mob}_1, \doteq_1))$ and $\beta = (a_2, b_2, (c_2, \mathrm{Mob}_2, \doteq_2))$ with $c_1 = a_2$, the composition $\alpha \star_\mathrm{L} \beta$ of α and β is the tuple $(a_1, b_1, b_2, (c_2, \mathrm{Mob}_3, \doteq_3))$ where $\mathrm{Mob}_3 = [\mathrm{Mob}_1, id_{\mathrm{ar}(b_2)}] \circ \mathrm{Mob}_2 : \underline{\mathrm{ar}(c_2)} \to \underline{\mathrm{ar}(a_1)} \uplus \underline{\mathrm{ar}(b_1)} \uplus \underline{\mathrm{ar}(b_2)}$*, 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 least 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 $\alpha \star_\mathrm{R} \beta$ is defined when $c_1 = b_2$ instead of $c_1 = a_2$.

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

1. *$(a, b, (\epsilon, \mathrm{Mob}, \doteq)) \in AS \Rightarrow a = b = \epsilon$;*
2. *$(a, \epsilon, (c, \mathrm{Mob}, \doteq)) \in AS \Rightarrow (c = a \wedge \mathrm{Mob} = \mathrm{inj}_1 \wedge \doteq \, = \mathrm{id})$;*
3. *$(a, b, (c, \mathrm{Mob}, \doteq)) \in AS \Rightarrow (b, a, (c, \mathrm{Mob}', \doteq')) \in AS$, where for each x, y $\mathrm{Mob}'(x) = comp(\mathrm{Mob}(x))$ and $x \doteq' y$ iff $comp(x) \doteq comp(y)$;*
4. *if $\alpha_1 = (a, b, (c, \mathrm{Mob}, \doteq)) \in AS$ and $\alpha_2 = (c, d, (e, \mathrm{Mob}', \doteq')) \in AS$ then $\exists f \in Act$, $\exists \beta_1 = (b, d, (f, \mathrm{Mob}'', \doteq'')), \beta_2 = (a, f, (e, \mathrm{Mob}''', \doteq''')) \in AS$ such that $\alpha_1 \star_\mathrm{L} \alpha_2 = \beta_1 \star_\mathrm{R} \beta_2$.*

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 (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 $\mathbf{A} = (Act, \mathrm{ar}, \epsilon)$, a set $Fin \subseteq Act$ of* final actions *and an action synchronization relation AS on $\mathbf{A}$.*

Final actions are used to deal with local channels: 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 an input is not in Fin, while the result of the synchronization between one input and one output is in Fin.

We present three simple examples of SAMs, taken from [13, 14] (albeit with different notation). 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 1. 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 ranked set of labels L is given such that $L \cap \{\epsilon, \tau\} = \emptyset$, with rank $\mathrm{ar} : L \to \mathbb{N}$.

Example 1 (Milner SAM). The SAM Milner_L is given by:

- $\mathit{Act} = \{\tau, \epsilon\} \cup \bigcup_{a \in L} \{a, \overline{a}\}$, $\mathrm{ar}(\overline{a}) = \mathrm{ar}(a)$ for each $a \in L$, $\mathrm{ar}(\tau) = 0$;
- $Fin = \{\tau\}$;
- $(\lambda, \epsilon, (\lambda, MP_{\mathrm{ar}(\lambda),0}, EQ_0)) \in AS$ for each $\lambda \in \mathit{Act}$,
  $(a, \overline{a}, (\tau, MP_{0,0}, EQ_{\mathrm{ar}(a)})) \in AS$ for each $a \in L$.

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

Example 2 (Hoare SAM). The SAM Hoare_L is given by:

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

Hoare synchronization models a global agreement on the action to perform. As before, corresponding parameters are merged, but now they are carried over the result of the interaction.

Example 3 (Broadcast SAM). The SAM Bdc_L is given by:

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

The above SAM models broadcast. When used in PRISMA, it forces an output $\overline{a}$ from a sequential PRISMA process to synchronize with *all* the listening sequential processes in parallel, which have to perform an input a. 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 $(\lambda, \epsilon, (\lambda, MP_{\mathrm{ar}(\lambda),0}, EQ_0))$ for each $\lambda \in \mathit{Act}$ to AS.

We present now a more complex (and original) example: a 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 consider just one input action *in* of arity 1, but the generalizations to many actions and different arities are straightforward.

- $Act = \{in, \epsilon\} \cup \{(out, n) | n \in \mathbb{N}\} \cup \{(out+, n) | n \in \mathbb{N}\} \cup \{(out-, n) | n \in \mathbb{N}\}$;
- $\mathrm{ar}(a) = 0$ for all $a \in \{\epsilon\} \cup \{(out+, n) | n \in \mathbb{N}\}$, and $\mathrm{ar}(a) = 1$ otherwise;
- $Fin = \{(out+, n) | n \in \mathbb{N}\}$;
- $(a, \epsilon, (a, MP_{\mathrm{ar}(a),0}, EQ_0)) \in AS$ for each $a \in Act$,
  $(in, (out, n), ((out+, n), MP_{0,0}, EQ_1)) \in AS$ for each n,
  $(in, (out, n), ((out-, n), MP_{1,0}, EQ_0)) \in AS$ for each n,
  $((out, n), (out, m), ((out, n), MP_{1,0}, EQ_0)) \in AS$ for each $n \geq m$,
  $((out, n), (out-, m), ((out+, n), MP_{0,0}, EQ_1)) \in AS$ for each $n \geq m$,
  $((out, n), (out-, m), ((out-, n), MP_{0,1}, EQ_0)) \in AS$ for each $n \geq m$,
  $((out, m), (out-, n), ((out-, n), MP_{0,1}, EQ_0)) \in AS$ for each $n \geq m$,
  $((out+, n), (out, m), ((out+, n), MP_{0,0}, EQ_0)) \in AS$ for each $n \geq m$.

Fig. 2. The priority SAM *Pri*.

Example 4 (Priority SAM). The SAM *Pri* is defined in Figure 2. The basic idea is that the result of the synchronization of an action *in* and an action (out, n), i.e. an output with priority n, is guessed: either we guess that n is the highest priority, we merge the parameters and the result is $(out+, n)$, or we guess the opposite, we propagate the input variable and the result is $(out-, n)$. The first guess, if wrong, is discarded when an output with higher priority is found, the second one is checked when the channel is declared local, since $(out-, n) \notin Fin$. 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 PRISMA calculus

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

Definition 6 (PRISMA). *The syntax for PRISMA processes is:*

$P ::= 0$	*(Inaction)*		$xa\vec{y}.P$	*(Prefix)*
$P_1 \vert P_2$	*(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 $xa\vec{y}$.

In PRISMA, restriction (x) is the only binder for x. As usual, processes are taken up to α-conversion of restricted names, and $\mathrm{fn}(P)$ denotes the set of free names in P. The intrinsic compositionality of action synchronization in SAMs makes the LTS operational semantics more natural for PRISMA than semantics in the reduction style, where all possible global synchronizations, 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 PRISMA processes, in an incremental way. The rules are parametric on the SAM S that fixes

$$\frac{P_1 \xrightarrow{(Y_1)xa_1\vec{y}_1,\pi_1} P_1' \qquad P_2 \xrightarrow{(Y_2)xa_2\vec{y}_2,\pi_2} P_2' \qquad (a_1,a_2,(c,\mathrm{Mob},\doteq)) \in AS \qquad \Phi}{P_1|P_2 \xrightarrow{(W)xc\vec{w},\pi|_{\setminus(Y_1\cup Y_2)}} (\vec{s})(P_1'|P_2')\pi} \quad (2)$$

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

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 $\vec{s}$).

Table 1. Rule for synchronization.

the allowed interaction policies. Interestingly the rules exploit just α-conversion as structural law, and this simplifies the proofs of process properties. However, the kind of axioms usually used in structural congruence equate processes which are also equivalent according to our observational semantics (see Lemma 1). One could avoid α-conversion too, but this would unnecessarily complicate the inference rules. The first rule we examine is the one for prefix:

$$xa\vec{y}.P \xrightarrow{xa\vec{y},\mathrm{id}} P \quad (1)$$

The transition simply executes the corresponding action. Note that the label contains a substitution too (the identity substitution in this case): this is used to trace fusions of global names performed by the synchronization, since they must be applied to parallel processes (see, e.g., rule 9).

The most important, but also the most complex, rule allows to synchronize two actions performed by parallel processes (rule 2 in Table 1). Its complexity is due to the great degree of flexibility of 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, since a set of extruded names appears (when empty, such as in rule 1, it is deleted from the label). Extruded names are names that were bound before, but become global after being used as parameters in the label. 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 rule for synchronization 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 π, which combines the previous substitutions traced by π_1 and π_2 with the new substitution π determined by taking into account the equivalence classes defined by $\doteq$. The substitution π 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, and, as far as global names are concerned, it is traced in the label as $\pi|_{\setminus(Y_1\cup Y_2)}$. The set W is the new set of extruded names. Finally, names that were

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

$$\frac{P \xrightarrow{(Y)xa\vec{y},\pi} P' \quad z \in \mathrm{Set}(\vec{y}) \setminus \{x\} \setminus Y \quad z \notin \mathrm{Im}(\pi)}{(z)P \xrightarrow{(\{z\}\cup Y)xa\vec{y},\pi|_{\setminus\{z\}}} P'} \quad (7)$$

$$\frac{P \xrightarrow{(Z)xa\vec{y},\pi} P' \quad a \in Fin \quad x \notin \mathrm{Im}(\pi) \quad \mathrm{Set}(\vec{z}) = Z}{(x)P \xrightarrow{\surd,\pi|_{\setminus\{x\}}} (x\vec{z})P'} \quad (8)$$

$$\frac{P \xrightarrow{\surd,\pi} P'}{P|Q \xrightarrow{\surd,\pi} P'|Q\pi} \quad (9) \qquad \frac{P \xrightarrow{\surd,\pi} P' \quad x \notin \mathrm{Im}(\pi)}{(x)P \xrightarrow{\surd,\pi|_{\setminus\{x\}}} (x)P'} \quad (10)$$

Table 2. Rules for restriction.

extruded $(Y_1 \cup Y_2)$, still exist $((Y_1 \cup Y_2)\pi)$ and are no longer appearing in the label $((Y_1 \cup Y_2)\pi \setminus W)$ must be closed by inserting them into $\vec{s}$ (in any order).

Two additional aspects must be considered to deal with parallel composition. In some SAMs, such as the Milner one, no process is forced to participate to the synchronization, while in others, such as in broadcast, 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:

$$\frac{x \in \mathcal{N}}{P \xrightarrow{x\epsilon\langle\rangle,\mathrm{id}} P} \quad (3)$$

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 [7], 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:

$$\frac{x \text{ is not an active subject of } P}{P \xrightarrow{\neg x} P} \quad (4)$$

It is easy to give an inductive definition of this rule to allow proofs by induction (not shown here just for space constraints). A dedicated rule (and its symmetric) are required to allow this action to interact with a normal action:

$$\frac{P_1 \xrightarrow{(Y)xa\vec{y},\pi} P_1' \quad P_2 \xrightarrow{\neg x} P_2 \quad Y \cap \mathrm{fn}(P_2) = \emptyset}{P_1|P_2 \xrightarrow{(Y)xa\vec{y},\pi} P_1'|P_2\pi} \quad (5)$$

Restriction is dealt with rules 6–10 in Table 2. Rule 6 says that restriction on channel z does not influence actions where z is neither the subject nor a parameter. If the equivalence class of z according to π is not a singleton, we have to remove z from π, since it is not visible outside its scope. If z is one of the parameters instead (rule 7), then it is marked as extruded in the label (as in π-calculus rule (open)). Rule 8 closes the channel on which the action a is done, reintroducing the restriction for names that were extruded by a. This is allowed only if $a \in Fin$. This rule introduces a further form of label, namely $(\surd, \pi)$, which states that an action has been performed on a bound channel, and that substitution π is its effect on global names. The simpler rules 9 (and its symmetric) and 10 deal with this kind of labels.

Finally, (almost) standard rules can be added to deal with nondeterministic sum (rule 11 and its symmetric) and replication (rule 12):

$$\frac{P_1 \xrightarrow{\lambda} P_1' \quad \lambda \neq x\epsilon\langle\rangle, \mathrm{id} \quad \lambda \neq \neg x}{P_1 + P_2 \xrightarrow{\lambda} P_1'} \quad (11) \qquad \frac{P|!P \xrightarrow{\lambda} P'}{!P \xrightarrow{\lambda} P'} \quad (12)$$

In the above rules, λ denotes a general label. The only peculiarity is that actions ϵ 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.

Example 5. Take the priority SAM *Pri* of Example 4. The $\surd$-labeled transitions for the process $S = (x)(x\ in\langle y\rangle.P \mid x(out, 3)\langle z\rangle.Q \mid x(out, 2)\langle w\rangle.R)$ are:

$S \xrightarrow{\surd,\{z/y\}} (x)(P \mid Q \mid x(out, 2)\langle w\rangle.R)\{z/y\}$
$S \xrightarrow{\surd,\{w/y\}} (x)(P \mid x(out, 3)\langle z\rangle.Q \mid R)\{w/y\}$
$S \xrightarrow{\surd,\{z/y\}} (x)(P \mid Q \mid R)\{z/y\}$

together with a transition where y is chosen as representative instead of z or w. 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, (out,2), being discarded. The only other admissible transitions are from S to itself with labels of the form $u\epsilon\langle\rangle, \mathrm{id}$ or $\neg u$ for any u.

Example 6 (News server and PRISMA). The transitions described in the Introduction for the news server can be derived, with suitable labels, in PRISMA by considering a 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 SAM can also be built using a coproduct construction in the category of SAMs, as we will show in Section 5.

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 hyperbisimilarity, as done in Fusion Calculus. This is required since standard bisimilarity is not a congruence w.r.t. composition operators.

Definition 7 (Hyperbisimilarity). *A* bisimulation *is a relation $\sim_S$ such that $P \sim_S Q$ implies:*

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

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 shall drop S from the notation when clear from the context.*

We present now some properties of hyperbisimilarity. Note that properties that hold for any SAM (or for any SAM satisfying suitable requirements, see, e.g., Lemma 2) can be proved once and for all. Next lemma, in particular, shows that hyperbisimilarity abstracts away from certain syntactic features of processes which are intuitively not important from an observational point of view. (We say that an axiom $P = Q$ on processes *bisimulates* if, for each instance of the axiom, the two equated processes are hyperbisimilar.)

Lemma 1. *The axioms below bisimulate for any SAM and for any P, Q, R:*

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

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. Proofs for other axioms are standard. □

Lemma 2. *The axiom $(x)0 = 0$ bisimulates iff $\epsilon \notin Fin$.*

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

Next theorem proves that abstract semantics 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 Fusion Calculus [18]: the interesting point is that it holds for PRISMA over any SAM.

Theorem 1. *Hyperbisimilarity $\approx_S$ is a congruence for any SAM S w.r.t. all the operators in PRISMA.*

Proof (Sketch). 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, $P_1 \approx_S Q_1$ and $P_2 \approx_S Q_2$ implies $\mathrm{op}(P_1) \approx_S \mathrm{op}(Q_1)$ (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.

We show the proofs for prefix, parallel composition and replication as examples. The other cases are similar. We will not consider transitions with labels ϵ and $\neg x$ since they can always be trivially simulated.

Case prefix): We have to prove that for each SAM S, each 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 transitions to consider are 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 substitution σ, since this corresponds to choosing $P' = P\sigma$ and $Q' = Q\sigma$.

Case $|$**):** Suppose that $P_1 \approx_S Q_1$ and $P_2 \approx_S Q_2$. To show $P_1|P_2 \approx_S Q_1|Q_2$ we have three rules to check. Let us consider rule 2. 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 $(Y_1 \cup Y_2) \cap (\mathrm{fn}(P_1) \cup \mathrm{fn}(P_2)) = \emptyset$. 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 $(\vec{s})(P_1'|P_2')\sigma$ and $(\vec{s})(Q_1'|Q_2')\sigma$ are hyperbisimilar. Thanks to α-conversion, 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 under substitutions of hyperbisimilarity $(P_1'|P_2')\sigma \approx_S (Q_1'|Q_2')\sigma$. Finally, using closure under restriction contexts, $(\vec{s})(P_1'|P_2')\sigma \approx_S (\vec{s})(Q_1'|Q_2')\sigma$. The cases for rules 5 and 9 are 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. □

4 A case study: Fusion Calculus

Let $L = \{\mathrm{in}_n | n \in \mathbb{N}\}$ with $\mathrm{ar}(\mathrm{in}_n) = n$. We will show that PRISMA over Milner_L is essentially Fusion Calculus [18] (as expected), and we suggest a new channel-located semantics for it. We consider the subset of Fusion Calculus whose processes are defined by:

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

We do not allow fusion prefixes, but $\{\vec{x} = \vec{y}\}.P$ can be encoded as $(z)(z\vec{x}.P|\overline{z}\vec{y}.0)$ for $z \notin \mathrm{fn}(P)$. We denote with $\equiv$ the structural congruence on Fusion processes and with $\approx_f$ Fusion hyperbisimilarity. We refer to [18] for full details on Fusion Calculus and on its semantics.

We define the uniform encoding function $[\![-]\!]$ from Fusion processes into PRISMA processes as the homomorphic extension to the whole calculus of $[\![u\vec{x}.P]\!] = u \ \mathrm{in}_{|\vec{x}|} \ \vec{x}.[\![P]\!]$ and $[\![\overline{u}\vec{x}.P]\!] = u \ \mathrm{out}_{|\vec{x}|} \ \vec{x}.[\![P]\!]$ where in_n and $\mathrm{out}_n = \overline{\mathrm{in}_n}$ are complementary actions. The mapping can 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 shows the relationship between the behaviors of Fusion processes and of their translations into PRISMA.

Theorem 2. *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.*

Proof (Sketch). The proof is by structural induction on Fusion processes, and has a case for each operator. One must prove that Fusion transitions correspond to PRISMA transitions of the two forms above (but PRISMA can have transitions with labels ϵ and $\neg x$ too, since these ones have no Fusion correspondence).

Notably, Fusion structural congruence can be simulated since translations of structural congruent processes are hyperbisimilar (see lemmas 1 and 2).

As far as parallel composition is concerned, Fusion synchronization is simulated by rule 2 using synchronization $(a, \overline{a}, (\tau, MP_{0,0}, EQ_{\mathrm{ar}(a)}))$. Asynchronous execution of Fusion actions can be simulated instead using synchronization $(a, \epsilon, (a, MP_{\mathrm{ar}(a),0}, EQ_0))$. Finally fusion propagation can be simulated by rule 9.

For restriction operator different rules have to be chosen according to the kind of actions: rule 6 to deal with actions on other channels, rule 7 for extrusions, rule 8 to move from the representation of fusions as $x\tau\langle\rangle, \pi$ to $(\surd, \pi)$ (note in fact that $\tau \in Fin$) and 10 if the label is already in this form.

The proofs for other operators are similar. Likewise, PRISMA rules can be simulated by Fusion rules for labels which are translations of Fusion labels. $\square$

In 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 Fusion correspondence. In particular, the second kind of labels allows to identify the active names of a process.

Corollary 1. $[\![P]\!] \approx_{Milner_L} [\![P']\!] \Rightarrow P \approx_f P'$.

The main difference between Fusion Calculus and PRISMA over $Milner_L$ is that in our more general form the τ is a normal action and thus it is located. The corresponding semantics can be defined also for Fusion Calculus, by adding located fusions $x\phi$ to the set of transition labels. Rules can be updated to take care of these labels.

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 use the translation $[\![-]\!]$ to have them in the PRISMA setting).

Example 7. One can easily find two processes that are bisimilar with the standard (non-located) semantics, but not with the PRISMA one, e.g.:

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

The right process can perform unlocated actions $\surd$ by making y and $\overline{y}$ react, while the left one cannot. A similar example can be written using replication:

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

(In both the examples the two members have the same set of active names.)

We think that the located semantics for τ can be useful, since, for accounting or performance reasons, synchronizations performed on different free channels may not be 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 than a process performing the same synchronization using x. Local channels are instead all equivalent, since any interaction, including traffic check (see Example 8), must happen inside the scope of the channel. For a more detailed description of a semantics of this kind see [11].

5 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 [15] to this end.

SAs form a category $\mathcal{SA}$ [19] 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.

Definition 8 (Morphism between action signatures).
Let $\mathbf{A}_A = (Act_A, \mathrm{ar}_A, \epsilon_A)$ *and* $\mathbf{B}_B = (Act_B, \mathrm{ar}_B, \epsilon_B)$ *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.* 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 9 (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 of parameters and merges among them) on the remaining parameters.

Lemma 3. *SAMs with asynchronous morphisms form the category $\mathcal{ASYNC}$, SAMs with synchronous morphisms form the subcategory $\mathcal{SYNC}$ of $\mathcal{ASYNC}$.*

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 10. *Given a morphism $H = (h, \{h_a\}_{a \in Act})$, we define the corresponding translation of 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 4. *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 2. *Let P and Q be two processes and $H(P)$ and $H(Q)$ be their translations according to SAM isomorphism $H : S_1 \to S_2$. Then $P \approx_{S_1} Q$ iff $H(P) \approx_{S_2} H(Q)$.*

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

Lemma 5. *Let $((Act_1, \mathrm{ar}_1, \epsilon_1), Fin_1, AS_1)$ and $((Act_2, \mathrm{ar}_2, \epsilon_2), Fin_2, AS_2)$ be two SAMs. The product in $\mathcal{ASYNC}$, which we call* asynchronous product*, has the form $((Act_\otimes, \mathrm{ar}_\otimes, \epsilon_\otimes), Fin_\otimes, AS_\otimes)$ where:*

- $Act_\otimes = Act_1 \times Act_2$ *with* $\mathrm{ar}_\otimes((a, b)) = \mathrm{ar}_1(a) + \mathrm{ar}_2(b)$;
  - *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;*
- $\epsilon_\otimes = (\epsilon_1, \epsilon_2)$;
- $Fin_\otimes = Fin_1 \times Fin_2$;
- $AS_\otimes = \{((a_1, a_2), (b_1, b_2), ((c_1, c_2), \mathrm{Mob}_\otimes, \doteq_\otimes)) |$ *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 $\mathcal{ASYNC}$ by choosing the different elements as described in the lemma. $\square$

Lemma 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 (ϵ_A, ϵ_B).*

Proof (Sketch). The proof is an easy modification of the one above. $\square$

Lemma 7. *Let $((Act_1, \mathrm{ar}_1, \epsilon_1), Fin_1, AS_1)$ and $((Act_2, \mathrm{ar}_2, \epsilon_2), Fin_2, AS_2)$ be two SAMs. The coproduct in $\mathcal{ASYNC}$ coincides with that in $\mathcal{SYNC}$ and it has the form $((Act_+, \mathrm{ar}_+, \epsilon_+), Fin_+, AS_+)$ where:*

- $Act_+ = ((Act_1 \setminus \{\epsilon_1\}) \uplus (Act_2 \setminus \{\epsilon_2\}) \cup \{\epsilon_+\})$ *with* $\mathrm{ar}_+(\mathrm{inj}_i(a)) = \mathrm{ar}_i(a)$*;*
- $a \in Fin_i \Rightarrow \mathrm{inj}_i(a) \in Fin_+$*,* $\epsilon_+ \in Fin_+$ *iff* $\exists i \in \{1,2\}.\epsilon_i \in Fin_i$*;*
- $(a, b, (c, \mathrm{Mob}, \doteq)) \in AS_i \Rightarrow (\mathrm{inj}{+}_i(a), \mathrm{inj}{+}_i(b), (\mathrm{inj}{+}_i(c), \mathrm{Mob}, \doteq)) \in AS_+$ *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 $\mathcal{ASYNC}$ and $\mathcal{SYNC}$ 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 asynchronous product of two Milner 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 8. *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 $P \approx_{S_1} Q$ iff $H(P) \approx_{S_1+S_2} H(Q)$.*

For instance, the SAM used in Example 6 is a coproduct of two SAMs, one isomorphic to $Milner_{\{in\}}$ and the other to $Bdc_{\{in_b\}}$. The coproduct of $Milner_{\{in_i | i \in \underline{254}\}}$ and $Bdc_{\{in_{255}\}}$ can be used to model normal 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 8 (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 9 (Using Fusion in a priority scenario). Consider an infrastructure built for priority communication as specified in Example 4. Suppose that one wants to run a Fusion process P_F in that framework. Suppose for simplicity that P_F uses just unary prefixes. We will show how P_F 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 Fusion to PRISMA can be used to have a corresponding PRISMA process P_M on the SAM $\mathit{Milner}_{\{\mathrm{in}_1\}}$. In the category $\mathcal{ASYNC}$ there is a morphism $H_n : \mathit{Milner}_{\{\mathrm{in}_1\}} \to \mathit{Pri}$ that maps in_1 to in, out_1 to (out, n) and τ to $(\mathit{out}{+}, n)$ 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 SAM (by considering the coproduct with other copies of itself with different actions) and then apply the same procedure.

6 Conclusion

We have presented PRISMA, a SAM-based process calculus with parametric communication patterns. This helps the modeling phase, when the desired synchronization policy can be specified directly instead of being implemented using "low-level" primitives. Different domain-specific SAMs can provide the right level of abstraction for prototyping and analysis. We have also shown that simple categorical tools allow to compare and compose SAMs. Furthermore, interoperability analysis can be easily performed, since different SAMs can be embedded in the same framework using the coproduct construction and related using morphisms. 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 congruence result holds for any SAM.

As future work, we want to test our model on some case studies taken from real distributed protocols. On a more theoretical side, we want to exploit PRISMA to compare processes based on different synchronization models. Furthermore, we want to see how other existing calculi can be related to PRISMA, starting from bπ-calculus [7].

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, M. G. Buscemi, and U. Montanari. D-fusion: A distinctive fusion calculus. In *Proc. of APLAS'04*, LNCS 3302, pages 296–310. Springer, 2004.
4. S. Dal Zilio. Mobile processes: A commented bibliography. In *MOVEP'00*, LNCS 2067, pages 206–222. Springer, 2000.
5. P. Degano and U. Montanari. A model for distributed systems based on graph rewriting. *Journal of the ACM*, 34(2):411–449, 1987.
6. C. Ene and T. Muntean. Expressiveness of point-to-point versus broadcast communications. In *Proc. of FCT'99*, LNCS 1684, pages 258–268. Springer, 1999.
7. C. Ene and T. Muntean. A broadcast-based calculus for communicating systems. In *Proc. of IPDPS'01*. IEEE Computer Society, 2001.
8. 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.
9. D. Hirsch and U. Montanari. Synchronized hyperedge replacement with name mobility. In *Proc. of CONCUR'01*, LNCS 2154, pages 121–136. Springer, 2001.
10. C.A.R. Hoare. A model for communicating sequential processes. In *On the Construction of Programs*. Cambridge University Press, 1980.
11. I. Lanese. Concurrent and located synchronizations in π-calculus. In *Proc. of SOFSEM'07*, LNCS, to appear.
12. I. Lanese. *Synchronization strategies for global computing models*. PhD thesis, Computer Science Department, University of Pisa, Pisa, Italy, 2006.
13. I. Lanese and U. Montanari. Synchronization algebras with mobility for graph transformations. In *Proc. of FGUC'04*, ENTCS 138, pages 43–60. Elsevier Science, 2004.
14. I. Lanese and E. Tuosto. Synchronized hyperedge replacement for heterogeneous systems. In *Proc. of COORDINATION'05*, LNCS 3454, pages 220–235. Springer, 2005.
15. S. MacLane. *Categories for the Working Mathematician*. Springer, 1971.
16. R. Milner. *A Calculus of Communicating Systems*, LNCS 92. Springer, 1980.
17. 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.
18. J. Parrow and B. Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *Proc. of LICS '98*, pages 176–185. IEEE, Computer Society Press, 1998.
19. G. Winskel. Synchronization trees. *Theoret. Comput. Sci.*, 34:33–82, 1984.