Synchronous Multiparty Synchronizations and Transactions*

Ivan Lanese[1] and Hernán Melgratti[2]

[1] Dipartimento di Scienze dell'Informazione, Università di Bologna,
lanese@cs.unibo.it
[2] Departamento de Computación, FCEyN, Universidad de Buenos Aires,
hmelgra@dc.uba.ar

Dedicated to Ugo Montanari
*on the occasion of his 65th birthday***

In this paper we analyze how a powerful synchronization mechanism such as synchronous multiparty synchronizations, which is able to specify atomic reconfigurations of large systems, can be implemented using binary synchronizations combined with a transactional mechanism. To this aim we show a mapping from SHR, a graph transformation framework allowing multiparty synchronizations, to a generalization of Fusion Calculus featuring a transactional mechanism inspired by the Zero-Safe Petri nets. To complete the correspondence between the two formalisms we also present a mapping in the opposite direction.

1 Introduction

A key aspect when specifying complex software systems is to find the right level of abstraction, and in particular the kind of primitives to be used. On the one side, one would expect to rely on very powerful primitives that enable for a simple and compact specification of complex behaviours. On the other side, one would prefer a small number of easily implementable primitives to shorten the gap between modeling and implementation.

In this paper we concentrate on synchronous multiparty synchronizations, which are a powerful tool for defining complex reconfigurations that allow the propagation of synchronizations through a chain of pairwise synchronized components. As an instance of this approach we have chosen Synchronized Hyperedge Replacement (SHR) [1,2,3], a graph transformation framework in which complex transformations are specified in terms of simple rules that define the behavior of single edges (or, more precisely, hyperedges). Although this makes SHR specifications quite compact, relying on a synchronous multiparty synchronization mechanism has always been considered as a drawback, mainly from the implementation point of view. We address this problem by showing how synchronous

* Research partially funded by EU Integrated Project Sensoria, contract n. 016004.
** We decided to work on highlighting the relationships among three different computational models (namely, the Fusion Calculus, Synchronized Hyperedge Replacement and Zero-Safe nets) since looking at problems from different perspectives is one of the lessons we have learnt from Ugo.

P. Degano et al. (Eds.): Montanari Festschrift, LNCS 5065, pp. 76–95, 2008.

Prefixes:

$$\alpha ::= \quad u\vec{x} \quad \text{(Input)} \quad | \quad \overline{u}\vec{x} \quad \text{(Output)} \quad | \quad \phi \quad \text{(Fusion)}$$

Agents:

$$S ::= \sum_i \alpha_i.P_i \text{ (Guarded sum)}$$

$$P ::= \quad S \quad \text{(Sequential Agent)} \quad \Big| \quad P_1|P_2 \text{ (Composition)}$$
$$(x)P \text{ (Scope)} \qquad C[\vec{v}] \text{ (Constant application)}$$

Fig. 1. Fusion Calculus syntax

multiparty synchronizations can be implemented by using normal binary synchronizations combined with a transactional mechanism. To this end we consider Fusion Calculus [4], a variant of π-calculus [5] whose close relation with SHR has been first shown in [6], and extend it with a transactional mechanism inspired by Zero-Safe Petri nets [7]. We call the resulting calculus Zero-Safe Fusion. While it is clear from [6] that Fusion Calculus is less expressive than SHR, we show that Zero-Safe Fusion is able to model the full SHR. We also show how to extend the mapping from Fusion to SHR in [6] to deal with Zero-Safe Fusion, thus proving that the two frameworks have the same expressive power.

Structure of the paper: Sections 2 and 3 present the background on Fusion Calculus and SHR respectively. Section 4 introduces Zero-Safe Fusion Calculus. Then, the mappings from SHR to Zero-Safe Fusion is presented in Section 5, while Section 6 discusses the assumptions made to simplify the mapping. Section 7 shows the inverse mapping. Finally, Section 8 draws some conclusions. For space limitation, we omit here the proofs of main results and we refer the interested reader to the full version of this paper [8].

2 Fusion Calculus

Fusion [4,9] has been proposed as a calculus for modeling distributed and mobile systems, and it is based on binary synchronization and name fusion. Interestingly, it both simplifies and generalizes the π-calculus [5]. This paper will consider Fusion Calculus with guarded summation, constant definitions to model infinite behaviors, and without match and mismatch operators. This section reports on its original syntax and structural congruence, while we refer to [9] for the original presentation of its operational semantics. Section 4 defines the operational semantics of a generalization of Fusion Calculus that accounts for transactions, which subsumes the original calculus.

We rely on an infinite set $\mathcal{N}$ of channel names ranged over by $u, v, \ldots, z$. We write $\vec{x}$ for indicating a vector of names, and ϕ for denoting an equivalence relation over $\mathcal{N}$ that can be represented by a finite set of equalities (that is always considered closed under reflexivity, symmetry and transitivity).

Definition 1 (Fusion syntax). *The sets of Fusion prefixes, ranged over by α, and Fusion agents, ranged over by $P, Q, \ldots$ are defined in Figure 1. Constants, ranged over by C are defined by equalities like $C[\vec{x}] \triangleq P$. We require the occurrence of constants to be prefix-guarded inside constant definitions.*

Note that we syntactically distinguish sequential agents S, i.e., agents whose topmost operator is a summation, from general agents P. We define the inactive process 0 as the empty summation, and we will omit trailing occurrences of 0. We use $+$ to denote binary summation.

The scope operator acts as a binder for names, thus x is bound in $(x)P$. Given an agent P, $\mathrm{fn}(P)$ denotes the set of all names occurring free in P. For constant definitions $C[\vec{x}] \triangleq P$ we require $\mathrm{fn}(P) \subseteq \vec{x}$. Processes are equivalence classes of agents defined by the structural congruence introduced below.

Definition 2. *The structural congruence $\equiv$ between agents is the least congruence satisfying the α-conversion law, the abelian monoid laws for summation and composition (associativity, commutativity and 0 as identity), the scope laws $(x)0 \equiv 0$, $(x)(y)P \equiv (y)(x)P$, the scope extension law $P|(z)Q \equiv (z)(P|Q)$ when $z \notin \mathrm{fn}(P)$, and the law for constants $C[\vec{v}] \equiv P\{\vec{v}/\vec{x}\}$ if $C[\vec{x}] \triangleq P$.*

Note that fn can be trivially extended from agents to processes.

3 Synchronized Hyperedge Replacement

Synchronized Hyperedge Replacement (SHR) is a graph transformation framework in which the evolution of a whole graph is defined in terms of synchronizing rules, called productions. Each production describes the evolution of a single edge. SHR has been introduced in [10], and several variants studying alternative mechanisms for handling synchronization and mobility have been proposed in the literature [11,12,13]. In this paper we concentrate on Milner SHR [11] as presented in [12]: the synchronization model has been inspired by Milner's π-calculus [5], and the mobility model follows the Fusion style.

Our graphs (or, more precisely, hypergraphs) are composed by edges and nodes. Each edge has a label L and it is attached to n nodes, where n is the rank $\mathrm{rank}(L)$ of its label. A graph is connected to its environment by an interface, which is a subset of the nodes of the graph. Nodes in the interface are free, while the others are bound. We give an algebraic definition of graphs.

Definition 3 (Graphs). *Let $\mathcal{N}$ be a fixed infinite set of nodes and LE a ranked set of labels. A* graph *has the form $\Gamma \vdash G$ where:*

1. *$\Gamma \subseteq \mathcal{N}$ is a finite set of nodes (the free nodes of the graph);*
2. *G is a* graph term *generated by the grammar*
   $G ::= L(\vec{x}) \mid G_1|G_2 \mid \nu y\, G \mid \mathrm{nil}$
   where $\vec{x} \in \mathcal{N}^*$ is a vector of nodes, $L \in LE$ is an edge label with $\mathrm{rank}(L) = |\vec{x}|$ *and $y \in \mathcal{N}$ is a node.*

Table 1. Structural congruence for graphs

(AG1) $(G_1\|G_2)\|G_3 \equiv G_1\|(G_2\|G_3)$ (AG2) $G_1\|G_2 \equiv G_2\|G_1$ (AG3) $G\|nil \equiv G$
(AG4) $\nu x\ \nu y\ G \equiv \nu y\ \nu x\ G$ (AG5) $\nu x\ G \equiv G$ if $x \notin \mathrm{fn}(G)$
(AG6) $\nu x\ G \equiv \nu y\ G\{y/x\}$ if $y \notin \mathrm{fn}(G)$
(AG7) $\nu x\ (G_1\|G_2) \equiv G_1\|\nu x\ G_2$ if $x \notin \mathrm{fn}(G_1)$

The restriction operator ν *is a binder, and we denote with* $\mathrm{fn}(G)$ *the set of free nodes in* G. *We demand that* $\mathrm{fn}(G) \subseteq \Gamma$.

Graph terms and graphs are considered up to the structural congruence $\equiv$ given by the axioms in Table 1.

Example 1. A ring-shaped graph composed by three edges with label R of rank 2 can be written as:

$$x \vdash \nu y, z\ R(x,y)|R(y,z)|R(z,x)$$

Notice that here nodes y and z are bound, while x is free.

Graphs interact by performing actions on nodes, thus we assume a set of actions Act. Since we are interested in Milner synchronization, we assume that $Act = In \uplus Out \uplus \{\tau\}$ where In is a set of input actions, Out a set of output actions, and τ a special action denoting a complete (internal) synchronization. Each action a has an arity $\mathrm{arity}(a)$, denoting the number of parameters. We assume a bijection $\bar{\cdot}$ between In and Out (and we use the same notation to denote the inverse) such that $\mathrm{arity}(\overline{a}) = \mathrm{arity}(a)$. We also assume $\mathrm{arity}(\tau) = 0$. Then, the transitions describing the evolution of a graph are as follows.

Definition 4 (SHR transitions). *A* SHR transition *is a relation of the form:*

$$\Gamma \vdash G \xrightarrow{\Lambda,\pi} \Phi \vdash G'$$

where $\Gamma \vdash G$ *and* $\Phi \vdash G'$ *are graphs,* $\Lambda : \Gamma \to (Act \times \mathcal{N}^*)$ *is a partial function and* $\pi : \Gamma \to \Gamma$ *is an idempotent renaming. Function* Λ *assigns to each node* x *the action* a *and the vector* $\vec{y}$ *of node references communicated to* x. *If* $\Lambda(x) = (a, \vec{y})$ *then we define* $\mathrm{act}_\Lambda(x) = a$ *and* $\mathrm{n}_\Lambda(x) = \vec{y}$. *We require that* $\mathrm{arity}(\mathrm{act}_\Lambda(x)) = |\,\mathrm{n}_\Lambda(x)|$.

We define:

- $\mathrm{n}(\Lambda) = \{z | \exists x. z \in \mathrm{n}_\Lambda(x)\}$ *set of communicated nodes;*
- $\Gamma_\Lambda = \mathrm{n}(\Lambda) \setminus \Gamma$ *set of communicated fresh nodes.*

Renaming π *allows to merge nodes. Since* π *is idempotent, it maps every node into a standard representative of its equivalence class. We require that* $\forall x \in \mathrm{n}(\Lambda). x\pi = x$, *i.e., only references to representatives can be communicated. Furthermore, we require* $\Phi = \Gamma\pi \cup \Gamma_\Lambda$, *namely free nodes can be merged but never erased* ($\supseteq$) *and new nodes are bound unless communicated* ($\subseteq$).

We will not write π when it is the identity.

Example 2 (Ring to star). The transition modeling the classical [1] ring-to-star reconfiguration for the ring graph in Example 1 is:

$$x \vdash \nu y, z\ R(x,y)|R(y,z)|R(z,x) \xrightarrow{(x,\tau,\langle\rangle)} x \vdash \nu y, z, w\ S(w,x)|S(w,y)|S(w,z)$$

The graph on the right-hand-side is composed by edges with label S of arity 2. Notice that actions are only observed on the free nodes, i.e. on x.

We will use this example as running example throughout the paper.

Definition 5 (Productions). *A* production *is an SHR transition of the form:*

$$x_1, \ldots, x_n \vdash L(x_1, \ldots, x_n) \xrightarrow{\Lambda,\pi} \Phi \vdash G$$

where $x_1, \ldots, x_n$ are all distinct.

For each edge label L of rank n we assume an idle production $x_1, \ldots, x_n \vdash L(x_1, \ldots, x_n) \xrightarrow{\Lambda_\perp, id} x_1, \ldots, x_n \vdash L(x_1, \ldots, x_n)$ where $\Lambda_\perp$ is always undefined. Idle productions are included in all sets of productions.

Given a set $Prod$ of productions, valid SHR transitions are derived from the productions in $Prod$ via the inference rules below.

Definition 6 (Inference rules for Milner SHR)

$$\textit{(prod-M)} \quad \frac{x_1, \ldots, x_n \vdash L(x_1, \ldots, x_n) \xrightarrow{\Lambda,\pi} \Phi \vdash G \in Prod}{\left(x_1, \ldots, x_n \vdash L(x_1, \ldots, x_n) \xrightarrow{\Lambda,\pi} \Phi \vdash G\right)\sigma}$$

where σ is a bijective renaming and it is applied to all the parts of the production.

$$\textit{(par-M)} \quad \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2 \qquad \Gamma' \vdash G'_1 \xrightarrow{\Lambda',\pi'} \Phi' \vdash G'_2}{\Gamma, \Gamma' \vdash G_1|G'_1 \xrightarrow{\Lambda\cup\Lambda',\pi\cup\pi'} \Phi, \Phi' \vdash G_2|G'_2}$$

where $(\Gamma \cup \Phi) \cap (\Gamma' \cup \Phi') = \emptyset$.

$$\textit{(merge-M)} \quad \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2}{\Gamma\sigma \vdash G_1\sigma \xrightarrow{\Lambda',\pi'} \Phi' \vdash \nu U\ G_2\sigma\rho}$$

where $\sigma = \{x/y\}$ is a renaming and:

1.a $x, y \in \mathrm{dom}(\Lambda) \Rightarrow \mathrm{act}_\Lambda(x) = a \wedge \mathrm{act}_\Lambda(y) = \overline{a} \wedge a \neq \tau$
1.b $\rho = \mathrm{mgu}(\{(\mathrm{n}_\Lambda(x))\sigma = (\mathrm{n}_\Lambda(y))\sigma\} \cup \{z\sigma = w\sigma | z\pi = w\pi\})$ *where ρ maps names to representatives in $\Gamma\sigma$ whenever possible*
1.c $\Lambda'(z) = \begin{cases} (\tau, \langle\rangle) & \textit{if } z = x \wedge x, y \in \mathrm{dom}(\Lambda) \\ (\Lambda(w))\sigma\rho & \textit{if } \exists w \in \mathrm{dom}(\Lambda). w\sigma = z \\ \textit{undefined} & \textit{otherwise} \end{cases}$
1.d $\pi' = \rho|_{\Gamma\sigma}$

1.e $U = (\Phi\sigma\rho) \setminus \Phi'$

$$\textit{(res-M)} \quad \frac{\Gamma, x \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2}{\Gamma \vdash \nu x\ G_1 \xrightarrow{\Lambda|_\Gamma,\pi|_\Gamma} \Phi' \vdash \nu Z\ G_2}$$

where:

2.a $(\exists y \in \Gamma. x\pi = y\pi) \Rightarrow x\pi \neq x$
2.b $\mathrm{act}_\Lambda(x)$ *undefined* $\vee\, \mathrm{act}_\Lambda(x) = \tau$
2.c $Z = \{x\}$ *if* $x \notin \mathrm{n}(\Lambda|_\Gamma), Z = \emptyset$ *otherwise*

$$\textit{(new-M)} \quad \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2}{\Gamma, x \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi, x \vdash G_2}$$

where $x \notin \Gamma \cup \Phi$.

Rule (prod-M) says that productions are the basic transitions, and it allows them to be bijectively renamed. Rule (par-M) accounts for the parallel evolution of two disjoint components of a graph. Rule (merge-M) deals with synchronization and allows to merge two nodes x and y into a unique node x (represented by the substitution $\sigma = \{x/y\}$) provided that actions performed on x and y are complementary (if they are both defined). The result of such an interaction is represented by the action τ over the node x (see the definition of Λ' in side-condition *1.c*). The effect of merging two nodes is also reflected by the renaming π'. Note that in addition to the original renaming π, π' should reflect that (i) x and y have been merged (second operand of the set union expression in side-condition *1.b*) and (ii) the names communicated by the action performed on x are fused with the names communicated by the action on y (first operand of the set union expression in side-condition *1.b*). Then, the obtained ρ and π' are applied to the names in the transition label Λ, to the obtained graph G_2, and to the set Φ of free names of G_2. Moreover, νU binds the names that occur free in G_2 but are not in Φ'. This is analogous to the rule *close* of π-calculus. Rule (res-M) allows to bind a node x whenever no action is performed on it or a τ action is performed. Finally, rule (new-M) allows to introduce a free isolated node x to both the left- and right-hand-side graphs of the transition.

Example 3. The ring-to-star reconfiguration transition in Example 2 can be derived from the production:

$$x, y \vdash R(x, y) \xrightarrow{(x, in_1, \langle w\rangle),(y, out_1, \langle w\rangle)} x, y, w \vdash S(w, x)$$

using the inference rules assuming $\overline{in_1} = out_1$. For each node, we will have a synchronization between the actions in_1 and out_1, which will also merge the nodes w belonging to different productions (actually, nodes w are renamed by (prod-M) in order to make rule (par-M) applicable).

The following lemma shows that derivations of SHR transitions can be normalized to some extent.

Lemma 1. *Suppose to have a derivation for an SHR transition. Then the same transition can be derived using a normalized derivation where no rule different from (res-M) is applied after any application of rule (res-M), i.e. all applications of (res-M) are at the end of the derivation.*

Proof. This is a consequence of the fact that structurally equivalent graphs have the same transitions, see e.g. [14].

4 Zero-Safe Fusion Calculus

In this section we introduce *Zero-Safe Fusion* (ZS Fusion), which extends Fusion Calculus [4,9] with a transaction mechanism inspired by the Zero-Safe Petri nets [7]. At the syntactic level we introduce a new form of prefixes, called *transactional prefixes*, which are composed by several standard prefixes that are executed as a unique action. Transactional prefixes are a general version of the join prefixes introduced by the Join calculus [15] and adopted by the *General Rendezvous Calculus* (GRC) [16]. The main difference is that our proposal allows for output and fusion actions inside join prefixes, while Join and GRC do not. Moreover, all actions occurring in a transactional prefix can be executed concurrently. This is the main difference of ZS Fusion with respect to other proposals appeared in the literature [17,18], in which transactional prefixes are sequences of basic actions.

4.1 Syntax

ZS Fusion agents are analogous to Fusion agents introduced by Definition 1, while sequential agents have the following form:

$$S ::= \sum_i \beta_i.P_i \quad \text{(Guarded sum)} \qquad \text{where } \beta \text{ is} \quad \beta ::= \wedge_i \alpha_i \quad \text{(Transaction)}$$

The prefix $\wedge_i \alpha_i$ stands for the transactional execution of several standard prefixes α_i. Normal prefixes are recovered as transactional prefixes containing just one component. Thus, we use $\alpha_1.P$ as a shorthand for $\wedge_{i\in\{1\}}\alpha_i.P$. We also use $\wedge.P$ as a shorthand for $\wedge_{i\in\emptyset}\alpha_i.P$.

In addition to the static syntax, we use the production below as run-time syntax to provide the operational semantics of the calculus.

$$S ::= \langle\beta\rangle.P \qquad \text{(Partial transaction)}$$

where the prefix $\langle\beta\rangle$ denotes a started transaction that has to execute β to complete. We write $\langle\rangle.P$ as a shorthand for $\langle\wedge_{i\in\emptyset}\alpha_i\rangle.P$.

Processes are agents up-to the structural congruence in Definition 2.

A sequential process S is *stable* if it has the form $\sum_i \beta_i.P_i$ or $\langle\rangle.P$, i.e., all partial transactional prefixes are empty. A sequential process S is *static* if it has the form $\sum_i \beta_i.P_i$. A process P is stable (resp. static) when all its sequential subprocesses are stable (resp. static).

Given a stable process P, $\widetilde{P}$ denotes the static process obtained from P by removing all prefixes of the form $\langle\rangle$. Formally,

$$\widetilde{\langle\rangle.P} = P \qquad \widetilde{P_1|P_2} = \widetilde{P_1}|\widetilde{P_2} \qquad \widetilde{(x)P} = (x)\widetilde{P} \qquad \widetilde{P} = P \textit{ otherwise}$$

4.2 Operational Semantics

We define the operational semantics of the ZS Fusion Calculus in two steps: first, we define a small-step behavior, and then we provide the transactional behavior of the calculus in terms of the small-steps.

Transactions require a few constraints to be satisfied:

1. the starting and final processes of each transaction must be static: this ensures that each transactional prefix has been completely executed;
2. there can be at most one communication/synchronization on each channel during each transaction: intuitively this means that channels are resources that cannot be shared, thus during each transaction only one process or a pair of interacting processes can use them;
3. fusions produced inside a transaction should be applied only when the transaction ends: on one side, this corresponds to the fact that tokens produced by a Zero-Safe transaction are available only at the end of the transaction and, on the other side, this ensures that the order in which elementary actions are executed inside a transaction is irrelevant.

In order to satisfy conditions 2 and 3, we define the small-step semantics only for processes without restriction, while we take restrictions into account when computing transactions. This is necessary since restrictions force the application of fusions, which is against condition 3. Notice also that α-conversion of bound names would make the checking of condition 2 more complex.

The small-step semantics exploits the labels below:

$$\gamma ::= u\vec{x} \mid \overline{u}\vec{x} \mid \{\vec{x} = \vec{y}\} \mid \gamma \wedge \gamma$$

We write $\emptyset$ for the empty substitution and we consider $\emptyset$ as the neutral element for $\wedge$.

The first three labels are standard Fusion labels for input, output and fusions, while the last one is the label for transactions. Given an action γ, $subj(\gamma)$ is the set of channels over which the actions in γ are taking place. Similarly $obj(\gamma)$ is the set of objects. They are defined as follows, where $\mathrm{Set}(\vec{x})$ is the set of names in $\vec{x}$.

$$\begin{array}{rlrl} subj(u\vec{x}) &= \{u\} & subj(\overline{u}\vec{x}) &= \{u\} \\ subj(\{\vec{x}=\vec{y}\}) &= \emptyset & subj(\gamma_1 \wedge \gamma_2) &= subj(\gamma_1) \cup subj(\gamma_2) \end{array}$$

$$\begin{array}{rlrl} obj(u\vec{x}) &= \mathrm{Set}(\vec{x}) & obj(\overline{u}\vec{x}) &= \mathrm{Set}(\vec{x}) \\ obj(\{\vec{x}=\vec{y}\}) &= \emptyset & obj(\gamma_1 \wedge \gamma_2) &= obj(\gamma_1) \cup obj(\gamma_2) \end{array}$$

(Start)
$$\mathit{subj}(\alpha_i) \vdash \alpha_1 \wedge \ldots \wedge \alpha_n.P \xrightarrow{\alpha_i} \langle \alpha_1 \wedge \ldots \wedge \alpha_{i-1} \wedge \alpha_{i+1} \wedge \ldots \wedge \alpha_n \rangle.P$$

(Pref)
$$\mathit{subj}(\alpha_i) \vdash \langle \alpha_1 \wedge \ldots \wedge \alpha_n \rangle.P \xrightarrow{\alpha_i} \langle \alpha_1 \wedge \ldots \wedge \alpha_{i-1} \wedge \alpha_{i+1} \wedge \ldots \wedge \alpha_n \rangle.P$$

(Empty)
$$\emptyset \vdash \wedge.P \xrightarrow{\emptyset} \langle\rangle.P$$

(Sum)
$$\frac{\Gamma \vdash P \xrightarrow{\gamma} P'}{\Gamma \vdash P + Q \xrightarrow{\gamma} P'}$$

(Par)
$$\frac{\Gamma \vdash P \xrightarrow{\gamma} P'}{\Gamma \vdash P|Q \xrightarrow{\gamma} P'|Q}$$

(Com)
$$\frac{\Gamma \vdash P \xrightarrow{u\vec{x}} P' \qquad \Gamma \vdash Q \xrightarrow{\overline{u}\vec{y}} Q' \qquad |\vec{x}| = |\vec{y}|}{\Gamma \vdash P|Q \xrightarrow{\{\vec{x}=\vec{y}\}} P'|Q'}$$

(Seq)
$$\frac{\Gamma \vdash P \xrightarrow{\gamma_1} P'' \qquad \Gamma' \vdash P'' \xrightarrow{\gamma_2} P' \qquad \Gamma \cap \Gamma' = \emptyset}{\Gamma \cup \Gamma' \vdash P \xrightarrow{\gamma_1 \wedge \gamma_2} P'}$$

Fig. 2. Small-step semantics for Zero-Safe Fusion

We also define the following auxiliary functions to work with labels:

$$\begin{array}{rlrl} \mathrm{subst}(u\vec{x}) &= \emptyset & \mathrm{subst}(\overline{u}\vec{x}) &= \emptyset \\ \mathrm{subst}(\{\vec{x}=\vec{y}\}) &= \{\vec{x}=\vec{y}\} & \mathrm{subst}(\gamma_1 \wedge \gamma_2) &= \mathrm{subst}(\gamma_1) \cup \mathrm{subst}(\gamma_2) \end{array}$$

$$\begin{array}{rlrl} \mathrm{comm}(u\vec{x}) &= \{u\vec{x}\} & \mathrm{comm}(\overline{u}\vec{x}) &= \{\overline{u}\vec{x}\} \\ \mathrm{comm}(\{\vec{x}=\vec{y}\}) &= \emptyset & \mathrm{comm}(\gamma_1 \wedge \gamma_2) &= \mathrm{comm}(\gamma_1) \cup \mathrm{comm}(\gamma_2) \end{array}$$

The small-step semantics is defined by the labeled transition system in Figure 2. Rules are quite standard. The key point is that the set Γ records channels involved in the action γ. Note that sequential composition is allowed only when the actions take place over different channels. Rule (Empty) allows to execute empty transactions.

Labels for transactional semantics have the form $(Y)S, \phi$ where Y is the set of extruded names, S is a set of input/output actions $x\vec{y}$ or $\overline{x}\vec{y}$ and ϕ is a fusion. We drop (Y) if Y is empty.

Given a fusion ϕ we define $\phi \setminus z$ as $(\phi \cap ((\mathcal{N} \setminus \{z\}) \times (\mathcal{N} \setminus \{z\}))) \cup \{(z, z)\}$. We extend to labels the notions of free and bound names (the names in a fusion are those names belonging to non-singleton equivalence classes).

Transactional semantics is defined by the labeled transition system in Figure 3. Rule (Cmt) states that a transactional step is a small-step among two stable processes. Rule (Scope) allows to bind a name z belonging to a non-trivial equivalence class induced by ϕ (and not occurring as a subject in S). Name z is renamed by an equivalent name x both in the label and in the resulting process. The remaining rules are the usual ones.

The following definition introduces a notion of decomposition of Fusion processes that will be used in the following sections.

Definition 7. *The standard decomposition of a process P is defined as:*

$$P = \hat{P}\sigma_P$$

(Cmt)
$$\frac{\Gamma \vdash P \xrightarrow{\gamma} P' \qquad P \text{ and } P' \text{ stable}}{P \xRightarrow{(\mathrm{comm}(\gamma),\mathrm{subst}(\gamma))} \widetilde{P'}}$$

(Scope)
$$\frac{P \xRightarrow{(Y)S,\phi} P', z\phi x, z \neq x, z \notin subj(S)}{(z)P \xRightarrow{(Y)S\{x/z\},\phi\backslash z} P'\{x/z\}}$$

(Pass)
$$\frac{P \xRightarrow{\lambda} P', z \notin \mathrm{fn}(\lambda)}{(z)P \xRightarrow{\lambda} (z)P'}$$

(Open)
$$\frac{P \xRightarrow{(Y)S,\phi} P', z \in obj(S) \setminus Y, z \notin subj(S), z \notin \mathrm{fn}(\phi)}{(z)P \xRightarrow{(Y \cup \{z\})S,\phi} P'}$$

(Struct)
$$\frac{P \xRightarrow{\lambda} P' \quad P \equiv Q \quad P' \equiv Q'}{Q \xRightarrow{\lambda} Q'}$$

Fig. 3. Transactions for Zero-Safe Fusion

where σ_P is the standard substitution and $\hat{P}$ is the standard process of P. This decomposition satisfies:

$$P = Q\sigma \text{ implies } \hat{P} = \hat{Q} \wedge \sigma_P = \sigma_Q \sigma$$

We denote with fnarray(P) the array of the free name occurrences in P, ordered according to some fixed order dictated by the structure of (a suitable normal form w.r.t $\equiv$ of) P. In particular $\sigma_P = \{fnarray(P)/fnarray(\hat{P})\}$.

The following propositions about small-steps will be used for studying properties of transactions in ZS Fusion.

Proposition 1. *If $\Gamma \vdash P \xrightarrow{\gamma_1 \wedge \gamma_2} P''$ then there exists P' such that $\Gamma' \vdash P \xrightarrow{\gamma_1} P'$ and $\Gamma'' \vdash P' \xrightarrow{\gamma_2} P''$ with $\Gamma' \cup \Gamma'' = \Gamma$.*

Proposition 2. *If $\Gamma \vdash P \xrightarrow{\gamma_1 \wedge \gamma_2} P'$ then $\Gamma \vdash P \xrightarrow{\gamma_2 \wedge \gamma_1} P'$.*

Proposition 3. *If $\Gamma \vdash P \xrightarrow{\gamma_1 \wedge (\gamma_2 \wedge \gamma_3)} P'$ then $\Gamma \vdash P \xrightarrow{(\gamma_1 \wedge \gamma_2) \wedge \gamma_3} P'$.*

Because of the propositions above from now on we will consider labels up to associativity and commutativity of $\wedge$.

Proposition 4. *Given a process $P = \langle\beta\rangle.Q$ or $P = \beta.Q$. If $\Gamma \vdash P \xrightarrow{\gamma} P'$ with P' stable, then*

1. *$\gamma = \beta$ and $P' = \langle\rangle.Q$;*
2. *$\Gamma = \{subj(\alpha) | \alpha \in comm(\gamma)\}$.*

Proposition 5. *If $\Gamma \vdash P \xrightarrow{\gamma} Q$, then*

1. *$subj(\alpha_i) \neq subj(\alpha_j)$ for all $\alpha_i, \alpha_j \in comm(\gamma)$ with $i \neq j$.*
2. *$\{subj(\alpha) | \alpha \in comm(\gamma)\} \subseteq \Gamma$.*

Notice that in the proposition above we may have $\{subj(\alpha)|\alpha \in comm(\gamma)\} \subsetneq \Gamma$ since names on which a synchronization is performed are in the right-hand-side but not in the left-hand-side.

In order to state the proposition below we introduce the following notation: $\Gamma \vdash P \xrightarrow{\gamma}_\epsilon P'$ stands for either $\Gamma \vdash P \xrightarrow{\gamma} P'$ or $P = P'$, with empty Γ and γ.

Proposition 6. *If $\Gamma \vdash P_1|P_2 \xrightarrow{\gamma}_\epsilon Q$ with Q stable, then $\Gamma_1 \vdash P_1 \xrightarrow{\gamma_1}_\epsilon Q_1$ and $\Gamma_2 \vdash P_2 \xrightarrow{\gamma_2}_\epsilon Q_2$ with:*

1. *$\Gamma = \Gamma_1 \cup \Gamma_2$, $\Gamma_1 \cap \Gamma_2 = subj(\gamma_1) \cap subj(\gamma_2)$;*
2. *$\gamma = \gamma_1|_D \wedge \gamma_2|_D \wedge F_{\gamma_1|\gamma_2}$ where $D = \Gamma \setminus (subj(\gamma_1) \cap subj(\gamma_2))$, $\gamma|_D$ denotes the conjunction of all actions in γ whose subjects are in D, and $F_{\gamma_1|\gamma_2}$ denotes the set of fusions $\vec{x} = \vec{y}$ such that $a\vec{x} \in \gamma_1$ and $\overline{a}\vec{y} \in \gamma_2$;*
3. *$Q \equiv Q_1|Q_2$.*

Proof. If follows by induction on the derivation $\Gamma \vdash P_1|P_2 \xrightarrow{\gamma} Q$.

The following proposition shows that ZS Fusion transactions are fully determined by their set of consumed prefixes. We define prefixes β_i consumed by a transaction $P \stackrel{\lambda}{\Longrightarrow} P'$ as the ones occurring in P but not in P'. To this end, one needs to distinguish different occurrences of the same prefix. We will not formalize this idea, but, roughly, given a process one can assign unique tags to each prefix, and make transitions to preserve the tag of non-consumed prefixes.

Proposition 7. *Let P be a process. If $P \stackrel{\lambda}{\Longrightarrow} Q$ and $P \stackrel{\lambda'}{\Longrightarrow} Q'$ are two transactions consuming the same prefixes (i.e., prefixes with the same tags), then $\lambda = \lambda'$ and $Q \equiv Q'$.*

The following lemma characterizes sets of compatible prefixes, i.e., prefixes that can be consumed together in a transaction.

Lemma 2. *Let P be a process, and consider a transaction that consumes a set of prefixes $S = \{\beta_i\}_{i \in I}$ in P. Then for each name x in $subj(S)$ there are two possibilities:*

- *x is not bound in P, and it is the subject of exactly one action in S;*
- *x is the subject of two complementary actions in S.*

5 From SHR to Zero-Safe Fusion

This section presents a translation from SHR to ZS Fusion Calculus, i.e., given a graph $\Gamma \vdash G$ and a set of productions *Prod* describing its behavior, we translate them into a process $[\![\Gamma \vdash G]\!]_{g2p}$ together with a set of constant definitions $[\![Prod]\!]_{p2c}$. We also provide an operational correspondence result relating the original SHR model and its encoding into ZS Fusion.

Table 2. Translations

Translation	From	To	Section
$[\![\bullet]\!]_{g2p}$	Graphs	Processes	5
$[\![\bullet]\!]_{p2c}$	Production sets	Constant definitions	5
$[\![\bullet]\!]_{a2p}$	SHR actions	Prefixes/labels	5
$[\![\bullet]\!]_{p2p}$	Productions	Processes	5
$[\![\bullet]\!]_{a2g}$	Agents	Graph terms	7
$[\![\bullet]\!]_{p2a}$	Prefixes	SHR actions	7

In order to define the mapping above and the inverse mapping presented in Section 7, we exploit different translation functions, denoted by $[\![\bullet]\!]_\pi$, distinguished by their subscript π. To make the presentation clearer we summarize the used translation functions in Table 2.

In this section we only consider SHR models whose set Act of SHR actions is such that In contains at most one action for each arity. Consequently, we use in_n to denote the input action of arity n, and out_n for the corresponding output action. In addition, we consider only productions $\Gamma \vdash G \xrightarrow{\Lambda,\pi} \Phi \vdash G'$ where $\pi = id$ and action τ is never used in Λ. We also assume that each SHR edge is attached to any chosen node at most once. Section 6 shows that these assumptions are not restrictive.

We assume two bijective mappings: one between SHR nodes and ZS Fusion names, and the other between SHR edge labels and Fusion constant names. For simplicity, we will not write these mappings explicitly.

Definition 8 (Translation from graphs to processes). *We define the translation $[\![\Gamma \vdash G]\!]_{g2p}$ of a graph $\Gamma \vdash G$ by induction on the structure of G (we drop Γ because it is not relevant):*

$$[\![L(\vec{x})]\!]_{g2p} = L[\vec{x}] \qquad [\![G_1|G_2]\!]_{g2p} = [\![G_1]\!]_{g2p}|[\![G_2]\!]_{g2p}$$
$$[\![\nu y\ G]\!]_{g2p} = (y)[\![G]\!]_{g2p} \qquad [\![\mathrm{nil}]\!]_{g2p} = 0$$

Lemma 3. *If $G \equiv G'$ then $[\![G]\!]_{g2p} \equiv [\![G']\!]_{g2p}$.*

We also need a translation from SHR actions to ZS Fusion prefixes. We overload the notation using the same translation function also from SHR synchronization functions Λ to ZS Fusion labels, since it is a straightforward generalization of the previous one.

Definition 9 (Translation for actions). *Basic actions are translated as follows:*
$[\![(x, in_n, \vec{y})]\!]_{a2p} = x\vec{y}$
$[\![(x, out_n, \vec{y})]\!]_{a2p} = \overline{x}\vec{y}$

The translation of a synchronization function Λ is the set S obtained by translating all the basic actions in Λ but the ones of the form $(x, \tau, \langle\rangle)$, which are deleted. With $[\![\Lambda]\!]_{a2p}$ we also denote $\bigwedge_{\alpha\in S} \alpha$.

As already mentioned, constant definitions are obtained from productions. In particular, we produce a constant definition for each edge label L.

Definition 10 (Translation from productions to constants). *Let L be a label and let $Prod_L$ be the set of non-idle productions for L. We can assume that all the productions in $Prod_L$ have the same left-hand-side $L(x_1, \ldots, x_n)$ (this can be guaranteed up to α-conversion). Then, the translation of $Prod_L$ gives the following definition:*

$$L[x_1, \ldots, x_n] \triangleq \sum_{Pr \in Prod_L} [\![Pr]\!]_{p2p}$$

where the translation of a single production is:

$$[\![\Gamma \vdash L(x_1, \ldots, x_n) \xrightarrow{\Lambda} \Gamma' \vdash G]\!]_{p2p} = (Y')(\bigwedge_{l \in \Lambda} [\![l]\!]_{a2p}).[\![\Gamma' \vdash G]\!]_{g2p}$$

where $Y' = \Gamma' \setminus \Gamma$. To be precise the translation above does not produce ZS Fusion processes, since one can have restrictions just below sums, but one can easily move them outside to have guarded sums.

Example 4. The production

$$x, y \vdash R(x, y) \xrightarrow{(x, in_1, \langle w \rangle), (y, out_1, \langle w \rangle)} x, y, w \vdash S(w, x)$$

from Example 3 gives rise to the following constant definition:

$$R[x, y] \triangleq \nu w\ (xw \wedge \overline{y}w).S[w, x]$$

If we consider also a production:

$$x, y \vdash R(x, y) \rightarrow x, y \vdash \nu z\ R(x, z) | R(z, y)$$

then the constant definition becomes:

$$R[x, y] \triangleq \nu w, z\ (xw \wedge \overline{y}w).S[w, x] + \wedge.(R(x, z) | R(z, y))$$

We describe now the relation between the possible evolutions of a graph $\Gamma \vdash G$ and the computations of $[\![\Gamma \vdash G]\!]_{g2p}$. Roughly, there is a bijective correspondence between transitions of $\Gamma \vdash G$ and transactions of $[\![\Gamma \vdash G]\!]_{g2p}$. Nevertheless, the two models handle fusions differently: transactions in ZS Fusion apply the computed fusion ϕ to restricted names only, and propagate the remaining part, while SHR transitions apply the substitution π (i.e., an mgu of ϕ) immediately to both the obtained graph and the objects of performed actions. We introduce the notation $S \triangleleft \pi$ to indicate that the substitution π is applied only to the objects of the actions in S. We also write open(P) to denote the set of processes obtained from P by removing all restrictions (this is a set since bound names are α-convertible; we assume however that distinct names are kept distinct).

We start the proof of operational correspondence by stating the following lemma that deals with small-step transitions.

Lemma 4. *For each graph $\Gamma \vdash G$ without restriction operator and each set of productions Prod, if there is an SHR transition $\Gamma \vdash G \xrightarrow{\Lambda,\pi} \Phi \vdash G'$ then for each $P' \in \mathrm{open}([\![G]\!]_{g2p})$ there is a small-step $P' \xrightarrow{\gamma} P$ such that π is an mgu of* $\mathrm{subst}(\gamma)$, $[\![\Lambda]\!]_{a2p} = \mathrm{comm}(\gamma) \lhd \pi$ *and* $\widetilde{P}\pi \in \mathrm{open}([\![G']\!]_{g2p})$.

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

Theorem 1. *For each graph $\Gamma \vdash G$ and each set of productions Prod, there is an SHR transition $\Gamma \vdash G \xrightarrow{\Lambda,\pi} \Phi \vdash G'$ iff there is a ZS Fusion transaction $[\![G]\!]_{g2p} \xRightarrow{(Y)S,\phi} P$ such that π is an mgu of ϕ, $S \lhd \pi = [\![\Lambda]\!]_{a2p}$, $[\![G']\!]_{g2p} = P\pi$ and $Y = (\Phi \setminus \Gamma)$.*

Proof. The *only if* part is by induction on the derivation of the SHR transition exploiting Lemma 4. The *if* part exploits Lemma 2 and Lemma 7.

Example 5 (Ring to star). We can now apply the translation to the ring-to-star reconfiguration in Example 2. We remember that the final transition is:

$$x \vdash \nu y, z\ R(x,y)|R(y,z)|R(z,x) \xrightarrow{(x,\tau,\langle\rangle)} x \vdash \nu x, y, w\ S(w,x)|S(w,y)|S(w,z)$$

In the ZS Fusion setting we have:

$$[\![x \vdash \nu y, z\ R(x,y)|R(y,z)|R(z,x)]\!]_{g2p} = (yz)\ R[x,y]|R[y,z]|R[z,x]$$

We also have the following constant definition:

$$R[x,y] \triangleq \nu w\ (xw \wedge \overline{y}w).S[w,x]$$

Thus, the starting process is structural congruent to:

$$(w_1 w_2 w_3 yz)\ (xw_1 \wedge \overline{y}w_1).S[w_1,x]|(yw_2 \wedge \overline{z}w_2).S[w_2,y]|(zw_3 \wedge \overline{x}w_3).S[w_3,z]$$

The only small step that can be lifted to a transaction is:

$$(xw_1 \wedge \overline{y}w_1).S[w_1,x]|(yw_2 \wedge \overline{z}w_2).S[w_2,y]|(zw_3 \wedge \overline{x}w_3).S[w_3,z] \xrightarrow{\{w_1=w_2=w_3\}} \langle\rangle.S[w_1,x]|\langle\rangle.S[w_2,y]|\langle\rangle.S[w_3,z]$$

By applying rule (Cmt) and rule (Pass) twice, we derive:

$$(yz)(xw_1 \wedge \overline{y}w_1).S[w_1,x]|(yw_2 \wedge \overline{z}w_2).S[w_2,y]|(zw_3 \wedge \overline{x}w_3).S[w_3,z] \xRightarrow{\{w_1=w_2=w_3\}} (yz)S[w_1,x]|S[w_2,y]|S[w_3,z]$$

Finally, we can apply rule (Scope) twice and rule (Pass) for deriving:

$$(w_1 w_2 w_3 yz)(xw_1 \wedge \overline{y}w_1).S[w_1,x]|(yw_2 \wedge \overline{z}w_2).S[w_2,y]|(zw_3 \wedge \overline{x}w_3).S[w_3,z] \xRightarrow{\emptyset} (w_1 yz)S[w_1,x]|S[w_1,y]|S[w_1,z]$$

which is the desired transaction.

6 About Assumptions

This section is devoted to discuss the assumptions made in the previous section, in particular:

1. there is at most one SHR input action for each arity;
2. each edge is attached to any chosen node at most once;
3. in all SHR productions, the substitution π is the identity and action τ is never used.

The discussions below, and the first one in particular, besides justifying the assumptions made to simplify the mapping, highlight some properties of SHR.

6.1 Simulating Multiple Actions

SHR, contrary to common nominal calculi that have prefixes of the form $x\vec{y}$, featuring only the subject and the objects, has prefixes of the form $xa\vec{y}$ including also an action a. We show below that while having actions is very handy from the specification point of view since it allows for much more compact specifications, it does not change the expressive power. Notice that the simulation of many actions exploits multiparty synchronizations, and could not be done in such a way e.g. in Fusion Calculus. Notice also that prefixes of different arities behave in different ways also in Fusion Calculus, thus having no actions actually corresponds in SHR to have at most one action for each arity.

To show that the introduction of multiple actions does not change the expressive power, we show that any Milner SHR model can be mapped into a Milner SHR model where any two input actions have different arities preserving the original behavior.

The mapping works as follows. Suppose we have n input actions denoted by a_i with $i \in \{1, \dots, n\}$, and their corresponding output actions. We consider a mapping that translates any node x into a tuple of $n+1$ nodes written $x, x_1, \dots, x_n$. For instance, if we have two input actions a_1 and a_2 with arity 1 then the graph $x, y \vdash L(x, y)$ is translated into $x, x_1, x_2, y, y_1, y_2 \vdash L(x, x_1, x_2, y, y_1, y_2)$. Then, performing an action a_n over the node x is translated as the execution of the unique action a on both x_n and x, where x is used for mutual exclusion. The same parameters, that are the translation of the original ones, are sent on both channels.

We denote the translation function defined above by $[\![\bullet]\!]_{n21}$, and we consider its trivial extension to SHR productions.

Example 6. The transition $x, y \vdash L(x, y) \xrightarrow{(x,a_1,\langle y\rangle)} x, y \vdash L(x, x)$ is translated into:

$$x, x_1, x_2, y, y_1, y_2 \vdash L(x, x_1, x_2, y, y_1, y_2) \xrightarrow{(x,a,\langle y,y_1,y_2\rangle),(x_1,a,\langle y,y_1,y_2\rangle)} x, x_1, x_2, y, y_1, y_2 \vdash L(x, x_1, x_2, x, x_1, x_2)$$

The following theorem states the behavioral correspondence between the two models.

Theorem 2. *The transition $\Gamma \vdash G \xrightarrow{\Lambda,\pi} \Phi \vdash G'$ can be derived using productions in Prod iff $[\![\Gamma \vdash G]\!]_{n21} \xrightarrow{[\![\Lambda]\!]_{n21},[\![\pi]\!]_{n21}} [\![\Phi \vdash G']\!]_{n21}$ can be derived using productions in $[\![Prod]\!]_{n21}$.*

Proof. By induction on the derivation $\Gamma \vdash G \xrightarrow{\Lambda,\pi} \Phi \vdash G'$.

6.2 Each Edge is Attached to Any Chosen Node at Most Once

This is the most important assumption that we have made. Notice that such a restriction is not preserved by general SHR transitions. Nevertheless, we require this condition in order to simplify the definition of the encoding presented in Section 5. This restriction avoids the following mismatch: actions belonging to the same prefix in a ZS Fusion process are executed sequentially and, hence, they cannot interact with each other, while two actions in the same SHR production can interact. For instance, the production $x, y \vdash L(x,y) \xrightarrow{(x,a,\langle\rangle),(y,\overline{a},\langle\rangle)} x, y \vdash L(x,y)$ may be applied to the edge $L(x,x)$ for deriving the transition $x \vdash L(x,x) \xrightarrow{(x,\tau,\langle\rangle)} x \vdash L(x,x)$.

Nevertheless, we can model the above system by avoiding non linear attachments as follows. We represent the edge $L(x,y)$ by using two different edges $L_1(x,w)$ and $L_2(y,w)$, whose corresponding productions are:

$$x, w \vdash L_1(x,w) \xrightarrow{(x,a,\langle\rangle),(w,b,\langle\rangle)} x, y \vdash L_1(x,w)$$
$$y, w \vdash L_2(y,w) \xrightarrow{(y,\overline{a},\langle\rangle),(w,\overline{b},\langle\rangle)} x, y \vdash L_2(y,w)$$

Note that we use a new auxiliary action b for synchronizing the rewritings of L_1 and L_2. Then, the initial graph $x, y \vdash L(x,y)$ can be simulated by the graph $x, y \vdash \nu w\ (L_1(x,w)|L_2(y,w))$. Consequently, we can simulate the transition $x \vdash L(x,x) \xrightarrow{(x,\tau,\langle\rangle)} x \vdash L(x,x)$ with the following one:

$$x \vdash \nu w\ (L_1(x,w)|L_2(x,w)) \xrightarrow{(x,\tau,\langle\rangle)} x \vdash \nu w\ (L_1(x,w)|L_2(x,w))$$

6.3 No Non-trivial Renamings Nor τ Actions in SHR Productions

Requiring substitutions appearing in SHR productions to be the identity does not affect the expressive power, since it is almost folklore that in SHR arbitrary idempotent substitutions can be generated by adding a synchronization on a private node z. Analogously, a τ action on a node can be generated by performing two complementary actions on the node itself. Thus one can use a simple encoding to map SHR models with arbitrary substitutions and τ actions onto models without them.

7 From Zero-Safe Fusion to SHR

We extend the approach followed in [6] for translating Fusion processes into SHR graphs. Basically, every sequential agent is encoded as a graph containing exactly one edge, whose label corresponds to the sequential agent renamed so to use only standard names. To this end, we exploit the notion of standard decomposition introduced by Definition 7.

Definition 11 (Translation of ZS Fusion agents). *The translation* $[\![\_]\!]_{a2g}$ *from Fusion agents to graph terms is defined as follows:*

- $[\![0]\!]_{a2g} = nil$
- $[\![S]\!]_{a2g} = L_{\hat{P}}(fnarray(P))$*, where* P *is the process corresponding to the sequential agent* S*.*
- $[\![P_1|P_2]\!]_{a2g} = [\![P_1]\!]_{a2g}|[\![P_2]\!]_{a2g}$
- $[\![(x)P]\!]_{a2g} = \nu x\ [\![P]\!]_{a2g}$
- $[\![C[\vec{v}]]\!]_{a2g} = [\![P\{\vec{v}/\vec{x}\}]\!]_{a2g}$*, if* $C[\vec{x}] \triangleq P$

Lemma 5. $P_1 \equiv P_2 \Rightarrow [\![P_1]\!]_{a2g} \equiv [\![P_2]\!]_{a2g}$.

Note that a sequential agent $\sum_i \beta_i.P_i$ is translated as a graph $[\![\sum_i \beta_i.P_i]\!]_{a2g}$, containing just one edge. Then, we define the set of SHR productions corresponding to a ZS Fusion process by defining the behavior of each edge. The following auxiliary definition provides a mapping for communication actions.

Definition 12 (Translation of communication actions). *Communication actions are translated as follows:*

- $[\![u\vec{x}]\!]_{p2a} = (in_n, \vec{x})$ *where* $n = |\vec{x}|$
- $[\![\overline{u}\vec{x}]\!]_{p2a} = (out_n, \vec{x})$ *where* $n = |\vec{x}|$

The productions corresponding to a particular sequential agent are defined as follows.

Definition 13 (Productions for a ZS Fusion agent). *Let* $S = \sum_i \beta_i.P_i$ *be a sequential agent and* $\Gamma = fn(S)$ *the set of free names in* S*. The set of productions* $\mathcal{P}_S$ *associated with* S *contains the following rules (one for each possible* i*).*

$$\Gamma \vdash [\![\sum_i \beta_i.P_i]\!]_{a2g} \xrightarrow{\Lambda_{\beta_i} \lhd \pi_{\beta_i}, \pi_{\beta_i}} \Gamma\pi_{\beta_i} \vdash [\![P_i\pi_{\beta_i}]\!]_{a2g}$$

where

$$\Lambda_{\beta_i}(x) = \begin{cases} [\![\alpha]\!]_{p2a} & \text{if } \alpha \in comm(\beta_i) \text{ and } subj(\alpha) = x \\ undefined & otherwise \end{cases}$$

and π_{β_i} *is an mgu of* $subst(\beta_i)$*.*

Lemma 6. *Let* P *be a ZS Fusion process and* σ *a renaming. Then* $[\![P]\!]_{a2g}\sigma = [\![P\sigma]\!]_{a2g}$*.*

The remaining of this section is devoted to show the completeness and soundness of the encoding. The following lemma states the completeness of the encoding for small-steps.

Lemma 7. *Let P be a static ZS Fusion process. If $\Delta \vdash P \xrightarrow{\gamma_1} P''$ and $\Delta' \vdash P'' \xrightarrow{\gamma_2}_\epsilon P'$ where P' is stable and $\Delta \cap \Delta' = \emptyset$, then for each $\Gamma \supseteq \mathrm{fn}(P)$ and for each mgu π of subst$(\gamma_1 \wedge \gamma_2)$:*

$$\Gamma \vdash [\![P]\!]_{a2g} \xrightarrow{\Lambda \lhd \pi, \pi} \Gamma\pi \vdash [\![\widetilde{P'}\pi]\!]_{a2g}$$

where

$$\Lambda(x) = \begin{cases} [\![\alpha]\!]_{p2a} & \text{if } \alpha \in comm(\gamma_1 \wedge \gamma_2) \text{ and } subj(\alpha) = x \\ (\tau, \langle\rangle) & \text{if } x \in \Delta \cup \Delta' \text{ and } \nexists \alpha \in comm(\gamma_1 \wedge \gamma_2) \text{ with } subj(\alpha) = x \\ undefined & otherwise \end{cases}$$

Proof. By induction on the derivation $\Delta \vdash P \xrightarrow{\gamma_1} P''$.

The completeness of the encoding considering ZS transactions and SHR transitions is stated by the following theorem.

Theorem 3. *Let P be a static ZS Fusion process. If $P \xRightarrow{(Y)S,\phi} P'$, then for each $\Gamma \supseteq \mathrm{fn}(P)$ and for each π mgu of ϕ:*

$$\Gamma \vdash [\![P]\!]_{a2g} \xrightarrow{\Lambda \lhd \pi, \pi} \Gamma\pi \vdash [\![\widetilde{P'}\pi]\!]_{a2g}$$

where

$$\Lambda(x) = \begin{cases} [\![\alpha]\!]_{p2a} & \text{if } \alpha \in S \text{ and } subj(\alpha) = x \\ undefined \text{ or } (\tau, \langle\rangle) & otherwise \end{cases}$$

Proof. By induction on the derivation $P \xRightarrow{(Y)S,\phi} P'$, using Lemma 7.

The following results show the soundness of the encoding.

Lemma 8. *Let P be a static ZS Fusion process without restrictions. If $\Gamma \vdash [\![P]\!]_{a2g} \xrightarrow{\Lambda, \pi} \Phi \vdash G$ with $fn(P) \subseteq \Gamma$, then $\Delta \vdash P \xrightarrow{\gamma} P'$, with $G = [\![\widetilde{P'}\pi]\!]_{a2g}$, $\Phi = \Gamma\pi$, π is an mgu of* subst(γ), *and $\Lambda = \Lambda' \lhd \pi$ where*

$$\Lambda'(x) = \begin{cases} [\![\alpha]\!]_{p2a} & \text{if } \alpha \in \mathrm{comm}(\gamma) \text{ and } subj(\alpha) = x \\ (\tau, \langle\rangle) & \text{if } x \in \Delta \text{ and } \nexists \alpha \in comm(\gamma) \text{ with } subj(\alpha) = x \\ undefined & otherwise \end{cases}$$

Proof. By induction on the derivation $\Gamma \vdash [\![P]\!]_{a2g} \xrightarrow{\Lambda, \pi} \Phi \vdash G$.

Theorem 4. *Let P be a static ZS Fusion process. If $\Gamma \vdash [\![P]\!]_{a2g} \xrightarrow{\Lambda,\pi} \Phi \vdash G$ with $fn(P) \subseteq \Gamma$, then $P \xRightarrow{(Y)S,\phi} P'$, with $G = [\![\widetilde{P'}\pi]\!]_{a2g}$, $\Phi = (\Gamma\pi \cup \{Y\})$, π is an mgu of ϕ, and $\Lambda = \Lambda' \lhd \pi$ where*

$$\Lambda'(x) = \begin{cases} [\![\alpha]\!]_{p2a} & \text{if } \alpha \in S \text{ and } subj(\alpha) = x \\ \text{undefined or } (\tau, \langle\rangle) & \text{otherwise} \end{cases}$$

Proof. By induction on $\Gamma \vdash [\![P]\!]_{a2g} \xrightarrow{\Lambda,\pi} \Phi \vdash G$, using Lemma 8.

8 Conclusion

We have shown the strict relationship between SHR and ZS Fusion Calculus, in particular highlighting how synchronous multiparty synchronizations can be implemented using binary synchronizations and a transactional mechanism. This can be used as a starting point to provide a distributed implementation of SHR. We leave for future work the analysis of the relationships between SHR and ZS Fusion from an observational point of view, instead of the operational one considered here. Actually, there are some differences between the observations that can be done on SHR and on ZS Fusion, e.g. because of the different treatment of substitutions and because of τ actions, thus the observational relation between the two models is not straightforward.

Other possible lines for future work are the analysis of the expressive power and usability of ZS Fusion calculus, and its comparison with other transactional frameworks such as, e.g., CJoin [19], πt [20], and Webπ [21]. The transactional mechanism of ZS Fusion is quite minimal and more in the ACID transactions style, thus it would be interesting to compare it with the calculi above that exploit the compensation approach.

References

1. Hirsch, D., Inverardi, P., Montanari, U.: Reconfiguration of software architecture styles with name mobility. In: Porto, A., Roman, G.-C. (eds.) COORDINATION 2000. LNCS, vol. 1906, pp. 148–163. Springer, Heidelberg (2000)
2. Hirsch, D.: Graph transformation models for software architecture styles. PhD thesis, Departamento de Computación, Facultad de Ciencias Exactas y Naturales, U.B.A (2003)
3. Lanese, I., Montanari, U.: Hoare vs Milner: Comparing synchronizations in a graphical framework with mobility. In: Proc. of GT-VC 2005. Elect.Notes in Th.Comput.Sci., vol. 154(2), pp. 55–72. Elsevier Science, Amsterdam (2005)
4. Parrow, J., Victor, B.: The fusion calculus: Expressiveness and symmetry in mobile processes. In: Proc. of LICS 1998. IEEE Computer Society Press, Los Alamitos (1998)
5. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes. Inform.and Comput. 100, 1–77 (1992)

6. Lanese, I., Montanari, U.: A graphical fusion calculus. In: Proc. of the Workshop of the COMETA Project. Elect.Notes in Th.Comput.Sci., vol. 104, pp. 199–215. Elsevier Science, Amsterdam (2004)
7. Bruni, R., Montanari, U.: Zero-safe nets: Comparing the collective and individual token approaches. Inform.and Comput. 156(1-2), 46–89 (2000)
8. Lanese, I., Melgratti, H.: Synchronous multiparty synchronizations and transactions, `http://cs.unibo.it/~lanese/work/ugo65-TR.pdf`
9. Victor, B.: The fusion calculus: Expressiveness and symmetry in mobile processes. PhD thesis, Dept.of Computer Systems, Uppsala University, Sweden (1998)
10. Castellani, I., Montanari, U.: Graph grammars for distributed systems. In: Ehrig, H., Nagl, M., Rozenberg, G. (eds.) Graph Grammars 1982. LNCS, vol. 153, pp. 20–38. Springer, Heidelberg (1983)
11. Hirsch, D., Montanari, U.: Synchronized hyperedge replacement with name mobility. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, pp. 121–136. Springer, Heidelberg (2001)
12. Ferrari, G., Montanari, U., Tuosto, E.: A LTS semantics of ambients via graph synchronization with mobility. In: Restivo, A., Ronchi Della Rocca, S., Roversi, L. (eds.) ICTCS 2001. LNCS, vol. 2202, pp. 1–16. Springer, Heidelberg (2001)
13. Lanese, I., Montanari, U.: Synchronization algebras with mobility for graph transformations. In: Proc. of FGUC 2004. Elect.Notes in Th.Comput.Sci., vol. 138, pp. 43–60. Elsevier, Amsterdam (2004)
14. Lanese, I.: Synchronization strategies for global computing models. PhD thesis, Computer Science Department, University of Pisa, Pisa, Italy (2006)
15. Fournet, C., Gonthier, G.: The reflexive CHAM and the join-calculus. In: Proc. of POPL 1996, pp. 372–385. ACM Press, New York (1996)
16. Bocchi, L., Wischik, L.: A process calculus of atomic commit. In: Proc. of WS-FM 2004. Elect.Notes in Th.Comput.Sci., vol. 105, pp. 119–132. Elsevier, Amsterdam (2004)
17. Bruni, R., Montanari, U.: Concurrent models for linda with transactions. Math.Struct.in Comput.Sci. 14(3), 421–468 (2004)
18. Gorrieri, R., Marchetti, S., Montanari, U.: A2CCS: Atomic actions for CCS. Theoret.Comput.Sci. 72(2&3), 203–223 (1990)
19. Bruni, R., Melgratti, H., Montanari, U.: Nested commits for mobile calculi: extending Join. In: Proc. of IFIP-TCS 2004, pp. 569–582. Kluwer Academic, Dordrecht (2004)
20. Bocchi, L., Laneve, C., Zavattaro, G.: A calculus for long-running transactions. In: Najm, E., Nestmann, U., Stevens, P. (eds.) FMOODS 2003. LNCS, vol. 2884, pp. 124–138. Springer, Heidelberg (2003)
21. Laneve, C., Zavattaro, G.: Foundations of web transactions. In: Sassone, V. (ed.) FOSSACS 2005. LNCS, vol. 3441, pp. 282–298. Springer, Heidelberg (2005)