Hoare vs Milner: comparing synchronizations in a graphical framework with mobility ⋆

Ivan Lanese [1] Ugo Montanari [2]

Computer Science Department, University of Pisa, Pisa, Italy

Abstract

We compare the expressive power of *Hoare* (i.e., CSP style) and *Milner* (i.e., CCS style) *synchronizations* for defining *graph transformations* in a framework where edges can perform actions on adjacent nodes to synchronize their evolutions. Furthermore, nodes can be communicated and merged. We show that the expressive powers of the two synchronization models are different, but no one is greater than the other. Finally, we show that in many interesting cases the behaviour of a synchronization model can be mimicked by the other one using suitable translations for the rewritten graphs.

Key words: graph transformations, Hoare synchronization, Milner synchronization, Synchronized Hyperedge Replacement, mobility, expressiveness.

1 Introduction

A fundamental aspect of many modern distributed systems is *synchronization*, i.e., how different components of the system can coordinate their behaviour in order to reach a common goal. Clearly, synchronization can be performed in different ways. This has emerged since the beginning of computational models for interacting systems: while CCS [11] used the so called *Milner synchronization*, where two processes interact by performing complementary actions, CSP [6] used *Hoare synchronization* where all the processes must synchronize by performing the same action.

We are interested in comparing these two synchronization models, but in a setting which is more complex than the original one. We work in the framework of *Synchronized Hyperedge Replacement* (SHR) [2,5,3], a *graph transformation* formalism aimed at representing distributed interacting systems. In

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

[1] Email: lanese@di.unipi.it

[2] Email: ugo@di.unipi.it

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

particular, we model system components as hyperedges and communication channels as shared nodes. System evolution is specified by *productions*, i.e., rules that describe the evolution of single hyperedges. Productions are synchronized by performing *actions* on adjacent nodes, and a set of productions can be executed concurrently only if the actions performed on each node are compatible. Compatible here means that they must synchronize using a given synchronization model. While SHR can be used with any synchronization model [9], here we are interested in comparing the Hoare and the Milner models. In addition, we consider *mobility* of nodes: references to nodes can be sent together with actions, and when actions are synchronized corresponding nodes are merged. As far as Milner synchronization model is concerned, this is the style of mobility used in Fusion Calculus [12] as pointed out in [7].

We will compare these two synchronization models from the point of view of which classes of reconfigurations they can specify, in three important cases: (i) one-step reconfigurations, (ii) reconfigurations specified by maximal (i.e., where no transition is possible from the final graph) computations and (iii) reconfigurations specified by any possible computation.

We will prove the following original results:

(i) the expressiveness of Hoare and Milner synchronization models are incomparable for all the above defined classes of reconfigurations;

(ii) the expressiveness of Milner synchronization is greater than the one of Hoare synchronization for graphs with no interface to the environment where each node is shared by exactly two edges, since Milner synchronization is asymmetric;

(iii) Hoare synchronization can be implemented using Milner synchronization and a suitable translation for graphs;

(iv) the encoding approach used in proving (iii) can not be used in the opposite direction, since it would require to force interleaving in a distributed structure.

Structure of the paper.
§ 2 introduces Synchronized Hyperedge Replacement and the Hoare and Milner synchronization models. In § 3 we define the formal setting for comparing the models. The comparison is carried out in § 4. The case of closed graphs with nodes shared by exactly two edges is analyzed in § 5. § 6 deals with the problem of implementing one model using the other one. Finally, conclusions and traces for future work are presented in § 7.

2 Synchronized Hyperedge Replacement

In this section we present Synchronized Hyperedge Replacement (SHR) [2] and, in particular, *the Hoare and the Milner synchronization models*, but first we introduce some mathematical notation.

Mathematical notation. Given a syntactic structure t (e.g., a term, a set of terms, an equation), we denote with $t\sigma$ the application of substitution σ to t (in a capture-avoiding way if t contains binders). The operator $|-|$ computes the number of elements in a vector or in a set. Given a set S, $\wp(S)$ is its powerset and S^* is the set of strings on alphabet S. Given a function f, we denote with $\mathrm{dom}(f)$ its domain and with $f|_S$ its restriction to the new domain S. Finally, when we use set operators on functions and substitutions, we refer to their representation as sets of pairs.

SHR [2] is an approach to (hyper)graph transformation that defines *global transitions* using *local productions*. Productions define how a single (hyper)edge can be rewritten and the *conditions* that this rewriting imposes on adjacent nodes. Thus the global transition is obtained by applying in parallel different productions whose conditions are compatible. What exactly compatible means depends on which synchronization model is used. In this work we will use both the Hoare and the Milner synchronization models. The former requires that all the edges connected to a node execute the same action on it. The latter requires two edges to interact by performing complementary actions while the others stay idle. For a general definition of synchronization models see [9].

We use the extension of SHR with *mobility* [5,3], that allows edges to send node references together with actions, and nodes whose references are matched during synchronization are unified.

We will give a formal description of SHR as labelled transition system, but first of all we need an algebraic representation for graphs.

An edge is an atomic item with a label and with as many ordered tentacles as the rank $\mathrm{rank}(L)$ of its label L. A graph is composed by a set of nodes and a set of such edges, and each edge is connected, by its tentacles, to its attachment nodes. A graph is connected to its environment by an interface which is a subset of its nodes. Nodes in the interface are called free nodes, while other nodes are called bound (or restricted). We will consider graphs up to isomorphisms that preserve free nodes, labels of edges, and connections between edges and nodes. We denote with *Graphs* the set of such graphs.

Now, we present a definition of graphs as syntactic judgements, where nodes correspond to names, free nodes to free names and edges to basic terms of the form $L(x_1, \ldots, x_n)$, where the x_i are arbitrary names and $\mathrm{rank}(L) = n$. Also, *nil* represents the graph with no edges, $|$ is the parallel composition of graphs (merging nodes with the same name) and νy is a declaration of a bound node y.

Definition 2.1 (Graphs as judgements) *Let $\mathcal{N}$ be a fixed infinite set of names and LE a ranked alphabet of labels. A* judgement *is of the form $\Gamma \vdash G$ where:*

(i) *$\Gamma \subseteq \mathcal{N}$ is a finite set of names (the free nodes of the graph);*

(ii) *G is a term generated by the grammar*

(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) $\nu x\ (G_1\|G_2) \equiv (\nu x\ G_1)\|G_2$ if $x \notin \mathrm{fn}(G_2)$

Table 1
Structural congruence for graph terms.

$G ::= L(\boldsymbol{x}) \mid G|G \mid \nu y\ G \mid nil$
where $\boldsymbol{x}$ *is a vector of names,* L *is an edge label with* $\mathrm{rank}(L) = |\boldsymbol{x}|$ *and* y *is a name.*

We define the restriction operator ν *as a binder. We denote with* fn *the function that given a term* G *returns the set* $\mathrm{fn}(G)$ *of free names in* G*. We demand that* $\mathrm{fn}(G) \subseteq \Gamma$*.*

When defining the interfaces, we use the notation Γ, x to denote the set obtained by adding x to Γ, assuming $x \notin \Gamma$ and Γ_1, Γ_2 to denote the union of Γ_1 and Γ_2, assuming $\Gamma_1 \cap \Gamma_2 = \emptyset$.

Graph terms are considered up to the axioms of structural congruence in Table 1. As far as judgements are concerned, we define $\Gamma \vdash G \equiv \Gamma' \vdash G'$ iff $\Gamma = \Gamma'$ and $G \equiv G'$.

Axioms (AG1), (AG2) and (AG3) define respectively the associativity, commutativity and identity over *nil* for operation $|$. Axioms (AG4) and (AG5) state that nodes of a graph can be restricted only once and in any order. Axiom (AG6) defines α-conversion of a graph w.r.t its bound names. Axiom (AG7) defines the interaction between restriction and parallel composition.

Note that function fn is well-defined on equivalence classes.

Judgements up to structural axioms are isomorphic to graphs up to isomorphisms. For a formal statement of the correspondence see [4].

We present now the steps of an SHR computation.

Definition 2.2 (SHR transition) *Let Act be a set of actions, and given* $a \in Act$ *let* $\mathrm{ar}(a)$ *be its arity. A* SHR transition *is of the form:*

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

where $\Gamma \vdash G$ *and* $\Phi \vdash G'$ *are judgements for graphs,* $\Lambda : \Gamma \to (Act \times \mathcal{N}^*)$ *is a total function and* $\pi : \Gamma \to \Gamma$ *is an idempotent substitution. Function* Λ *assigns to each node* x *the action* $a \in Act$ *and the vector* $\boldsymbol{y}$ *of node references exposed on* x *by the transition (in a more message-passing view, we say that node references are sent to* x*). If* $\Lambda(x) = (a, \boldsymbol{y})$ *then we define* $\mathrm{act}_\Lambda(x) = a$ *and* $\mathrm{n}_\Lambda(x) = \boldsymbol{y}$*. We require that* $\mathrm{ar}(\mathrm{act}_\Lambda(x)) = |\,\mathrm{n}_\Lambda(x)|$*.*

We define:

- $\mathrm{n}(\Lambda) = \{z | \exists x.z \in \mathrm{n}_\Lambda(x)\}$ *set of exposed names;*

- $\Gamma_\Lambda = \mathrm{n}(\Lambda) \setminus \Gamma$ *set of exposed fresh names.*

Substitution 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 exposed. Furthermore we require $\Phi = \Gamma\pi \cup \Gamma_\Lambda$, namely free nodes are never erased ($\supseteq$) and new nodes are bound unless exposed ($\subseteq$).

Note that the set of free names Φ of the resulting graph is fully determined by Λ and π (since $\Gamma = \mathrm{dom}(\Lambda)$). When writing Λ as set of pairs we write the triple $(x, a, \boldsymbol{y})$ for the pair $(x, (a, \boldsymbol{y}))$.

SHR transitions are derived from basic productions using suitable sets of inference rules.

Definition 2.3 (Production)
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 all x_i, $i = 1, \ldots, n$ are distinct.

We suppose to have for each edge label L of arity n a special idle production $x_1, \ldots, x_n \vdash L(x_1, \ldots, x_n) \xrightarrow{\Lambda_\epsilon, id} x_1, \ldots, x_n \vdash L(x_1, \ldots, x_n)$ where $\Lambda_\epsilon(x_i) = (\epsilon, \langle\rangle)$ for each i (ϵ is a special "idle" action with $\mathrm{ar}(\epsilon) = 0$). Idle productions are included in all sets of productions, which are also closed w.r.t. α-conversion of names in $\{x_1, \ldots, x_n\} \cup \Phi$.

We present now the set of inference rules for Hoare synchronization. The intuitive idea of Hoare synchronization is that all the edges connected to a node must expose the same action on that node.

Definition 2.4 (Rules for Hoare synchronization)

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

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

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

where $\sigma : \Gamma \to \Gamma$ is an idempotent substitution and:

(i) $\forall x, y \in \Gamma. x\sigma = y\sigma \Rightarrow \mathrm{act}_\Lambda(x) = \mathrm{act}_\Lambda(y)$

(ii) $\rho = \mathrm{mgu}(\{(\mathrm{n}_\Lambda(x))\sigma = (\mathrm{n}_\Lambda(y))\sigma | x\sigma = y\sigma\} \cup \{x\sigma = y\sigma | x\pi = y\pi\})$ *where we choose names in $\Gamma\sigma$ as representatives whenever possible*

(iii) $\forall z \in \Gamma. \Lambda'(z\sigma) = (\Lambda(z))\sigma\rho$

(iv) $\pi' = \rho|_{\Gamma\sigma}$

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

where:

(v) $(\exists y \in \Gamma . x\pi = y\pi) \Rightarrow x\pi \neq x$

(vi) $Z = (\{x\} \cup \mathrm{n}(\Lambda)) \setminus \mathrm{n}(\Lambda|_\Gamma)$

$$(new)\quad \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2}{\Gamma, x \vdash G_1 \xrightarrow{\Lambda\cup\{(x,a,\boldsymbol{y})\},\pi} \Phi' \vdash G_2}$$

where $x \notin \Gamma \cup \Phi$ and $\boldsymbol{y} \cap (\Gamma \cup \Phi \cup \{x\}) = \emptyset$.

A transition is obtained by composing productions, which are first applied on disconnected edges. Composition is performed by merging nodes and thus connecting the edges. Finally, nodes can be bound. In particular, rule (par) deals with the composition of transitions which have disjoint sets of nodes and rule (merge) allows to merge nodes (note that σ is a projection into representatives of equivalence classes). Condition (i) requires that we have the same action on merged nodes. Condition (ii) defines the most general unifier ρ of the union of two sets of equations: the first set identifies (the representatives of) the tuples associated to nodes merged by σ, while the second set of equations adds previous merges traced by π. Thus ρ is the merge resulting from both π and σ. Note that (iii) Λ is updated with these merges and that (iv) π' is ρ restricted to the nodes of the graph which is the source of the transition. Rule (res) binds node x, guaranteeing that x is not a representative if it belongs to a non trivial equivalence class and binding also all the nodes that were extruded on node x in the starting transition. Rule (new) allows adding to the source graph an isolated node where arbitrary actions (with fresh names) are performed.

We write $\mathcal{P} \Vdash_H \Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2$ if $\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2$ can be obtained from the productions in $\mathcal{P}$ using Hoare inference rules.

A similar set of rules can be defined also for Milner synchronization.

Definition 2.5 (Rules for Milner synchronization)

$$(par)\quad \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2 \qquad \Gamma' \vdash G_1' \xrightarrow{\Lambda',\pi'} \Phi' \vdash G_2'}{\Gamma, \Gamma' \vdash G_1|G_1' \xrightarrow{\Lambda\cup\Lambda',\pi\cup\pi'} \Phi, \Phi' \vdash G_2|G_2'}$$

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

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

where $\sigma : \Gamma \to \Gamma$ *is an idempotent substitution and:*

(i) $\forall x, y \in \Gamma. x\sigma = y\sigma \wedge \mathrm{act}_\Lambda(x) \neq \epsilon \wedge \mathrm{act}_\Lambda(y) \neq \epsilon \wedge x \neq y \Rightarrow$
$(\forall z \in \mathcal{N} \setminus \{x, y\}. z\sigma = x\sigma \Rightarrow \mathrm{act}_\Lambda(z) = \epsilon) \wedge$
$\mathrm{act}_\Lambda(x) = a \wedge \mathrm{act}_\Lambda(y) = \overline{a} \wedge a \neq \tau$

(ii) $\rho = \mathrm{mgu}(\{(\mathrm{n}_\Lambda(x))\sigma = (\mathrm{n}_\Lambda(y))\sigma | x\sigma = y\sigma\} \cup \{x\sigma = y\sigma | x\pi = y\pi\})$ *where we choose names in* $\Gamma\sigma$ *as representatives whenever possible*

(iii) $\Lambda'(z) = \begin{cases} (\tau, \langle\rangle) & \textit{if } x\sigma = y\sigma = z \wedge x \neq y \wedge \mathrm{act}_\Lambda(x), \mathrm{act}_\Lambda(y) \neq \epsilon \\ (\Lambda(x))\sigma\rho & \textit{if } x\sigma = z \wedge \mathrm{act}_\Lambda(x) \neq \epsilon \\ (\epsilon, \langle\rangle) & \textit{otherwise} \end{cases}$

(iv) $\pi' = \rho|_{\Gamma\sigma}$

(v) $U = (\Phi\sigma\rho) \setminus \Phi'$

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

where:

(vi) $(\exists y \in \Gamma. x\pi = y\pi) \Rightarrow x\pi \neq x$

(vii) $\mathrm{act}_\Lambda(x) = \epsilon \vee \mathrm{act}_\Lambda(x) = \tau$

(viii) $Z = \{x\}$ *if* $x \notin \mathrm{n}(\Lambda|_\Gamma), Z = \emptyset$ *otherwise*

$$\textit{(new)} \quad \frac{\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2}{\Gamma, x \vdash G_1 \xrightarrow{\Lambda\cup\{(x,\epsilon,\langle\rangle)\},\pi} \Phi, x \vdash G_2}$$

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

Rules for Milner synchronization suppose that actions can be normal actions a (representing input) or coactions $\overline{a}$ (representing "output"). We also assume $\overline{\overline{a}} = a$. Furthermore we have the two special actions ϵ and τ (completed synchronization) of arity 0.

Rules are similar to the ones for Hoare synchronization. The main differences are that in rule (merge) during action synchronization (i) we require to have (at most) two complementary non ϵ actions, and their composition is τ. Thus we may have to reintroduce restrictions (v) if some nodes were extruded by the synchronized actions. In rule (res), just nodes x where ϵ or τ actions are performed can be restricted, and since these actions have arity 0 only node x may have to be restricted in the final graph. Finally, in rule (new) only action ϵ is allowed on the newly created node.

We write $\mathcal{P} \Vdash_M \Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2$ if $\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2$ can be obtained from the productions in $\mathcal{P}$ using Milner inference rules. We drop the subscript M or H from $\Vdash$ when we refer to an unspecified synchronization model.

A *SHR computation* is a sequence of SHR transitions such that for each i the final graph of transition i is the starting graph of transition $i+1$. A SHR computation is called *trivial* if the starting graph is equal to the final graph.

3 Expressiveness measures

We want to study the expressiveness of the Hoare and Milner synchronization models in the SHR framework. Different measures of expressiveness can be useful, according to which is the intended use of the model. In our case, we are mainly interested in using graph transformation to express reconfigurations of the topology of distributed systems, thus the main point is which is the class of reconfigurations that can be expressed by a set of productions together with a synchronization model.

Formally, we define *reconfigurations* as functions $r : Graphs \to \wp(Graphs)$.

Intuitively, the *behaviour* of a set of productions $\mathcal{P}$ on a graph G w.r.t. a synchronization model S is the set of graphs that are the results of "suitable" computations starting from G. The choice of which computations are "suitable" determines the observable behaviour of the system.

Definition 3.1 (Behaviour function)
The function C-$\mathrm{behav}^S(\mathcal{P})(G)$ is the function that computes the set of graphs reachable from graph G using computations in the class C obtained from the productions in $\mathcal{P}$ using synchronization model S.

Thus we can say that the C_1-expressiveness of synchronization model S_1 is greater than the C_2-expressiveness of a synchronization model S_2, written as $(S_1, C_1) \geq (S_2, C_2)$ if there exists a function f from sets of productions to sets of productions such that for each set of productions $\mathcal{P}$ and for all graphs G we have that C_2-$\mathrm{behav}^{S_2}(\mathcal{P})(G) = C_1$-$\mathrm{behav}^{S_1}(f(\mathcal{P}))(G)$.

We will consider three different choices for C:

1 one-step computations;

max maximal computations (i.e., computations whose final state does not allow further non trivial transitions);

all all possible computations.

If a synchronization model S_2 is not as expressive as a synchronization model S_1, we can try to simulate reconfigurations of S_1 using reconfigurations of S_2 by translating the graph G (this will be done formally in § 5 and § 6).

4 The expressiveness of Hoare and Milner synchronizations are not comparable

In this section we show that the expressive power of Hoare and Milner synchronization models are different, but no one is greater than the other, independently of the class of computations used.

We first need an auxiliary definition.

Definition 4.1 (Monotonicity) *We define the following partial order on transitions: $\Gamma \cup \Gamma' \vdash G_1|G' \xrightarrow{\Lambda',\pi} \Phi \cup \Gamma'\pi \vdash G_2|G'\pi$ is greater than $\Gamma \vdash G_1 \xrightarrow{\Lambda,\pi}$*

$\Phi \vdash G_2$ iff $\Gamma' \vdash G'$ is a graph and $\Lambda = \Lambda'|_\Gamma$. A SHR system is monotone iff the set of derivable transitions is upward-closed (i.e., for each set of productions $\mathcal{P}$, if a transition is derivable, then all the greater transitions are derivable too).

Intuitively, in a monotone system we can always add to the graph an additional part which stays idle.

Proposition 4.2 *Milner SHR is monotone.*

Intuitively, this happens because Milner synchronization involves exactly two participants. The same is not true for Hoare synchronization, because there is a universal quantification on the participants connected to the node where the synchronization is performed.

Since the monotonicity property can be extended from transitions to general computations, each set of Milner computations must be upward-closed too. Using that, we can prove the following theorem.

Theorem 4.3 *$(Milner, C_1) \not\geq (Hoare, C_2)$ for each $C_1 \in \{1, all\}$ and each $C_2 \in \{1, max, all\}$.*

Proof. Let us consider the set of productions $\mathcal{P}$ generated by the only production:

$$x \vdash d(x) \xrightarrow{(x,a,\langle\rangle)} x \vdash d'(x)$$

For $C_2 \in \{1, all\}$ we have that:
$C_2\text{-}\,\mathrm{behav}^H(\mathcal{P})(x \vdash d(x)) = \{x \vdash d(x), x \vdash d'(x)\}$
$C_2\text{-}\,\mathrm{behav}^H(\mathcal{P})(x \vdash d(x)|d(x)) = \{x \vdash d(x)|d(x), x \vdash d'(x)|d'(x)\}$
while for $C_2 = max$ the behaviours do not contain the trivial reconfigurations (which are not maximal). Since, for each choice of C_2, behaviours are not monotone, the thesis follows from Proposition 4.2. □

The case of max-expressiveness requires a bit more work.

Theorem 4.4 *$(Milner, max) \not\geq (Hoare, C)$ for each $C \in \{1, max, all\}$.*

Proof. Let us consider the set of productions $\mathcal{P}$ generated by the only production:

$$x \vdash d(x) \xrightarrow{(x,a,\langle\rangle)} x \vdash nil$$

For each $C \in \{1, all\}$ we have that:
$C\text{-}\,\mathrm{behav}^H(\mathcal{P})(x \vdash d(x)) = \{x \vdash d(x), x \vdash nil\}$
$C\text{-}\,\mathrm{behav}^H(\mathcal{P})(x \vdash d(x)|d'(x)) = \{x \vdash d(x)|d'(x)\}$
while for $C = max$ we have not the trivial reconfiguration in the first case. Suppose that we can obtain this behaviour with Milner SHR. By monotonicity from the first case we have a transition from $x \vdash d(x)|d'(x)$ to $x \vdash d'(x)$. If $x \vdash d'(x)$ can not be rewritten then we have a contradiction, since it is not in the behaviour. Otherwise by monotonicity also $x \vdash d(x)|d'(x)$ can be rewritten and so it can not be in the behaviour for maximal computations, as it is. □

Now we consider the inverse problem, that is we prove that the expressiveness of Milner synchronization model can not be reached by Hoare SHR.

Notice that, if all nodes are free, Milner synchronization can not force productions to be executed together, i.e., each production can always be applied in isolation. Hence, restriction is fundamental for constraining the behaviour of components using Milner synchronization (and this does not surprise, since even in CCS restriction is necessary to reach Turing equivalence).

Notice that instead in Hoare SHR restriction just performs hiding of part of the observation, i.e., no transition can be forbidden by restriction. More formally, the following proposition holds.

Proposition 4.5 *Given a set of productions $\mathcal{P}$, if $\mathcal{P} \Vdash_H \Gamma, x \vdash G_1 \xrightarrow{\Lambda,\pi} \Phi \vdash G_2$ then $\mathcal{P} \Vdash_H \Gamma \vdash \nu x\ G_1 \xrightarrow{\Lambda',\pi'} \Phi' \vdash \nu Z\ G_2$ where Λ', π' and Φ' are subsets of Λ, π and Φ respectively, and $Z = \Phi \setminus \Phi'$.*

Theorem 4.6 *$(Hoare, C_1) \not\geq (Milner, C_2)$ for each $C_1, C_2 \in \{1, max, all\}$.*

Proof. Let us consider the set of productions $\mathcal{P}$ generated by the only production:

$$x \vdash d(x) \xrightarrow{(x,a,\langle\rangle)} x \vdash d'(x)$$

For each $C_2 \in \{1, all\}$ we have that:
$C_2\text{-}\,\mathrm{behav}^M(\mathcal{P})(x \vdash d(x)) = \{x \vdash d(x), x \vdash d'(x)\}$
$C_2\text{-}\,\mathrm{behav}^M(\mathcal{P})(\vdash \nu x\ d(x)) = \{\vdash \nu x\ d(x)\}$
while for $C_2 = max$ we have not the trivial reconfiguration in the first case. Since this behaviour does not satisfy Proposition 4.5, it can not be obtained by Hoare SHR (with any class of computations). □

5 Reconciling Hoare and Milner synchronizations

Until now we have shown that the expressiveness of Hoare and Milner SHR are quite different. We consider now a case where they become closer, i.e., when we consider closed graphs (i.e., graphs where all nodes are restricted) where each node is attached to exactly two tentacles. We call these graphs *closed 2-shared graphs*. Even if this case is quite simple, it shows some interesting features of the two synchronization models and it is a first step towards the more general results of next section.

The reconfigurations have to preserve the two invariants above. The invariant of having closed graphs is preserved automatically since new nodes are bound by default, and an extrusion can happen only if there is a free node on which it is performed.

The second condition is not preserved in general, but it can be enforced by constraining the allowed productions. Let us consider the application of a single production: when the rewritten edge is removed, all the nodes attached to it have one attached tentacle missing (two if the edge was connected two times to the same node). Thus when inserting the new graph, the same number

of connections to those nodes must be provided, and two connections must be provided for each new node. Notice also that when merges are performed, two nodes with one connection each are merged into one with two connections, thus occurrences of nodes in Λ or in π count as new connections for that node.

Definition 5.1 *A production P is* connection-preserving *if for each node x the number of occurrences of x in the right hand side, plus the ones in Λ, plus the number of nodes that are merged with x by π equals the number of occurrences of x in the left hand side (that is, 1) if x occurs there, and it is 2 for new nodes.*

Proposition 5.2 *Let $\mathcal{P}$ be a set of connection-preserving productions and G a closed 2-shared graph. If $\mathcal{P} \Vdash G \xrightarrow{\Lambda,\pi} G'$ then G' is a closed 2-shared graph.*

Proof. By rule induction on the derivation. □

Thus from now on we consider only connection-preserving productions. We will show later that this kind of productions is expressive enough to simulate general Hoare transitions (via a translation of graphs).

Theorem 5.3 *For closed 2-shared graphs, $(Milner, C) \geq (Hoare, C)$ for each $C \in \{1, max, all\}$.*

Proof. The set of productions for Milner model can be obtained by replicating each Hoare production with all possible "orientations" of actions, i.e., any action a must be substituted by either a or $\overline{a}$ and a production is needed for each possible combination of choices. □

This proves that for closed 2-shared graphs, Hoare synchronization is equal to Milner synchronization where the distinction between actions and coactions is dropped. In that case, Milner synchronization is strictly more expressive than Hoare synchronization and the additional expressiveness is given exactly by the asymmetry, as shown by the following proposition (whether this result holds for *max* expressiveness is an open problem).

Proposition 5.4 *For closed 2-shared graphs, $(Hoare, C_1) \not\geq (Milner, C_2)$ for each $C_1, C_2 \in \{1, all\}$.*

Proof. Let us consider the graph $\vdash \nu x\ d(x)|d(x)$. This graph is symmetric. Using Hoare synchronization, for any choice of production for the left edge, the same production can always be applied in the same step also to the right edge, and the result is again a symmetric graph. Thus for each choice of productions, if a transition exists, then also a transition that preserves the symmetry exists.

Using Milner synchronization and the set of productions generated by:

$x \vdash d(x) \xrightarrow{(x,a,\langle\rangle)} x \vdash c(x)$

$x \vdash d(x) \xrightarrow{(x,\overline{a},\langle\rangle)} x \vdash d(x)$

we have just one non trivial allowed transition, with final graph $\vdash \nu x\ c(x)|d(x)$.

Notice that this is also the result of all the allowed non trivial computations. Since no symmetric graph is obtained, this reconfiguration can not be performed using Hoare SHR. □

Thus we will consider a different form of simulation, that uses a translation for graphs. In particular, we define two functions $[\![-]\!], [\![-]\!]^{-1} : \mathit{Graphs} \to \mathit{Graphs}$ such that for each graph G we have $[\![[\![G]\!]]\!]^{-1} = G$ (but we may have $[\![[\![G]\!]^{-1}]\!] \neq G$).

We say that (C_1, S_1) can simulate (C_2, S_2) iff we have $C_2\text{-}\,\mathrm{behav}^{S_2}(\mathcal{P})(G) = [\![C_1\text{-}\,\mathrm{behav}^{S_1}(f(\mathcal{P}))([\![G]\!])]\!]^{-1}$, i.e., the result of a (C_2, S_2) reconfiguration can be obtained by translating the graph, reconfiguring it using (C_1, S_1) and translating it back again.

We use a translation based on the concept of amoeboid [8]: each node shared by n tentacles is translated into a graph called amoeboid with n external nodes. The inverse translation $[\![-]\!]^{-1}$ just removes the amoeboids and reinserts the nodes they stand for. In our case an amoeboid (which connects two nodes) for $[\![-]\!]$ is simply an edge $L(x, y)$, where L is a special label with productions of the following form for each action a of arity k:

$$x, y \vdash L(x, y) \xrightarrow{(x,a,\langle x_1,\ldots,x_k\rangle)(y,\overline{a},\langle y_1,\ldots,y_k\rangle)} x, y, x_1, \ldots, x_k, y_1, \ldots, y_k \vdash L(x, y) \mid \coprod_{i=1,\ldots,k} L(x_i, y_i)$$

and where $\coprod_{i\in I} G_i$ is the parallel composition of graphs G_i for each $i \in I$. As far as $[\![-]\!]^{-1}$ is concerned, an amoeboid is any chain of such edges. The translation f of productions just drops π and connects each pair of nodes merged by π using an L edge.

The following theorem holds.

Theorem 5.5 *For each set generated by connection-preserving productions $\mathcal{P}$, each closed 2-shared graph G, and each class of computations $C \in \{1, max, all\}$ we have $C\text{-}\,\mathrm{behav}^{M}(\mathcal{P})(G) = [\![C\text{-}\,\mathrm{behav}^{H}(f(\mathcal{P}))([\![G]\!])]\!]^{-1}$.*

Proof. The result holds because the L edge allows on its nodes complementary synchronizations, and amoeboids to be merged are instead connected using L edges. The tricky part is that the chains of L edges that are created (and that are translated into nodes by the inverse translation) have always odd length, and this is exactly the condition required to have complementary actions on the two ends of the chain. Notice also that the productions for L edges are connection-preserving. □

6 Dealing with general closed graphs

We want now to go back to the general case, at least as far as the number of tentacles attached to each node is concerned. In particular, we will show that

by using a different kind of amoeboids, the general case can be reduced to the 2-shared one.

As far as Hoare synchronization is concerned, we want to use amoeboids that perform the broadcast of the action.

Those amoeboids are composed by edges H (for Hoare) of arity 3 and edges C (for closing) of arity 1 to deal with nodes with less than 3 attached tentacles. These edges have for each action (we consider as an example an action a of arity 2) productions of the form:

$$x, y, z \vdash H(x, y, z) \xrightarrow{(x,a,\langle x_1,x_2\rangle)(y,a,\langle y_1,y_2\rangle)(z,a,\langle z_1,z_2\rangle)} x, y, z, x_1, y_1, z_1, x_2, y_2, z_2 \vdash H(x, y, z)|H(x_1, y_1, z_1)|H(x_2, y_2, z_2)$$

$$x \vdash C(x) \xrightarrow{(x,a,\langle x_1,x_2\rangle)} x, x_1, x_2 \vdash C(x)|C(x_1)|C(x_2)$$

Such an amoeboid imposes the same action to be executed on each node and it creates a copy of itself for each set of corresponding names, that are in this way connected in the resulting graph.

An amoeboid used to connect a set of nodes S is any connected graph composed by H and C edges whose nodes in S are attached to just one tentacle while whose other nodes are 2-shared. Thus for each graph G, $[\![G]\!]$ is a 2 shared graph.

Analogously we have to translate productions in order to make them connection-preserving. This can be done by splitting nodes that are used too many times and connecting the different copies using H edges, while nodes that are used too few times must be closed using C edges. Also, π is dropped and the nodes to be merged are connected using amoeboids.

Example 6.1 Let us consider the following production, which is used in [5] to specify a reconfiguration from a ring graph to a star one:

$$x, y \vdash r(x, y) \xrightarrow{(x,r,\langle w\rangle)(y,r,\langle w\rangle)} x, y, w \vdash s(y, w)$$

In this production the name x is not used in the right hand side, whereas the name w is used 3 times (two times in Λ and one by edge s) while it does not occur in the left hand side. We can translate the production into:

$$x, y \vdash r(x, y) \xrightarrow{(x,r,\langle w_1\rangle)(y,r,\langle w_2\rangle)} x, y, w_1, w_2, w_3 \vdash C(x), s(y, w_3), H(w_1, w_2, w_3)$$

which is a connection-preserving production such that the inverse translation of the right hand side is the right hand side of the starting production (up to renaming of nodes).

By using for functions $[\![-]\!]$ and $[\![-]\!]^{-1}$ the new amoeboids, we have the following result.

Theorem 6.2 *For each set of productions $\mathcal{P}$, each closed graph G, and each class of computations $C \in \{1, max, all\}$ we have that $C\text{-}\mathrm{behav}^H(\mathcal{P})(G) = [\![C\text{-}\mathrm{behav}^H(f(\mathcal{P}))([\![G]\!])]\!]^{-1}$ where f performs the above described translation of productions.*

This result can be composed with Theorem 5.3 to get a translation from Hoare synchronization to Milner synchronization for any closed graph. To deal with general graphs, one just needs to trace which nodes are free. This can be done by adding to each amoeboid representing a free node an edge $ENV(x)$ representing a connection with the environment. Such an edge must allow any action and it must attach a copy of itself to each node it receives (to simulate the fact that a node sent on a free node is extruded), like the C edge does. Note that in this way we may get amoeboids with many connections to the environment. We can add productions to delete them if they are redundant, but there is no way to force these reconfigurations to be executed before the normal transitions.

Now we want to apply the same approach to Milner synchronization. Milner amoeboids are essentially routers that create a path from an action to the corresponding coaction.

We start by introducing an M (for Milner) edge of arity 3. We want the edge to perform complementary actions on any pair of its three attachment nodes. Thus we have a production of the form:

$$x, y, z \vdash M(x, y, z) \xrightarrow{(x,a,\langle s_1,s_2\rangle)(y,\overline{a},\langle s_1,s_2\rangle)(z,\epsilon,\langle\rangle)} x, y, z \vdash M(x, y, z)$$

Note that in Milner synchronization we always merge pairs of nodes, thus it is not necessary to replicate the amoeboid. We also have to use a different kind of edge for dealing with nodes shared by less than 3 tentacles, which we denote by I (for inactive). This edge has only the idle production. This guarantees that actions and coactions are performed only by edges from the original graph.

For productions we use the same kind of translation that we have used in the Hoare case, with the new edges for amoeboids.

However w.r.t. Hoare model we have here an additional problem: many independent synchronizations may be allowed inside an amoeboid during one transition, but this is not allowed in standard Milner synchronization. In particular, this occurs when the pairs of interacting nodes are connected by disjoint paths inside the amoeboid. Also, cycles in the amoeboid may cause new isolated nodes to be created, but these can be discarded by $[\![-]\!]^{-1}$.

Using the new definition for the translation functions, we have the following partial correctness result.

Theorem 6.3 *For each set of productions $\mathcal{P}$, each closed graph G, and each class of computations $C \in \{1, max, all\}$ we have that $C\text{-}\mathrm{behav}^M(\mathcal{P})(G) \subseteq [\![C\text{-}\mathrm{behav}^M(f(\mathcal{P}))([\![G]\!])]\!]^{-1}$ where f performs the usual translation of produc-*

tions into connection-preserving ones.

The other inclusion holds, e.g., for amoeboids connecting at most 3 nodes, since in that case we can have at most one synchronization. Notice that this theorem can be composed with Theorem 5.5 to have an implementation of Milner synchronization using Hoare synchronization. The composed translation has been used in [8] to map Fusion Calculus into logic programming.

We show now that the problem of guaranteeing interleaving inside amoeboids of the above seen kind can not be solved.

Theorem 6.4 *Let $\mathcal{G} \subseteq$ Graphs contain for each n at least a graph with n nodes in its interface and let it be closed w.r.t. composition of graphs by joining them via a node in the interface. Then the maximum k such that all $G \in \mathcal{G}$ allow only transitions where at most k actions on the interface are not ϵ, if it exists is* 0.

Proof. Suppose that such a k exists and it is not 0 and take a graph with more than k nodes in its interface, and a transition where k of the actions are not ϵ. Take a node where ϵ action is executed. By connecting two such graphs by merging these two nodes, we get a graph which allows at least $2k$ non ϵ actions on its interface. This gives a contradiction. □

This proves that we can not have a set of amoeboids for Milner synchronization (since this requires $k = 2$), since the closure property is needed to model mobility. Notice in fact that if we want to model reconfigurations without mobility we can use, e.g., amoeboids with a tree structure whose leaves are the interface and whose roots check that the resulting action is a τ. Using mobility, the tree shape can not be preserved.

Also in that case, free nodes can be managed using edges standing for connections to the environment.

7 Conclusion and future works

We have analyzed the expressive power of Hoare and Milner synchronizations in the SHR setting, proving that they are incomparable and that implementing one synchronization with the other is not a trivial task. Also, for Milner synchronization no fully satisfactory simulation can be obtained using the concept of amoeboid. Notice that no counterexample (but the last one) uses mobility, thus we have proved that the expressiveness is incomparable without mobility, and that adding mobility does not help to bridge the gap.

These results justify the idea of having different synchronization models available in the same framework in order to be able to use all of them without complex translations. Such an approach was used in process calculus ACP [1], and has been extended to deal with graph transformations and mobility in [9,10].

As future work we want to carry out a similar comparison among generic synchronization models as defined in [9]. Another issue is to consider not only the allowed reconfigurations, but also the labels of the transitions. Finally, the possibility of using maximal expressivity to break symmetry in Hoare synchronization must be further investigated (see discussion before Proposition 5.4).

References

[1] J.A. Bergstra, and J.W. Klop. The algebra of recursively defined processes and the algebra of regular processes. In *Proc. of ICALP'84*, volume 172 of *LNCS*, pages 82–94. Springer, 1984.

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

[3] G. Ferrari, U. Montanari, and E. Tuosto. A LTS semantics of ambients via graph synchronization with mobility. In *Proc. of ICTCS'01*, volume 2202 of *LNCS*, pages 1–16. Springer, 2001.

[4] D. Hirsch. Graph transformation models for software architecture styles. PhD thesis, Departamento de Computación, Facultad de Ciencias Exactas y Naturales, U.B.A., 2003.

[5] D. Hirsch and U. Montanari. Synchronized hyperedge replacement with name mobility. In *Proc. of CONCUR'01*, volume 2154 of *LNCS*. Springer, 2001.

[6] C.A.R. Hoare. *CSP – Communicating Sequential Processes*. International Series in Computer Science. Prentice-Hall, 1985.

[7] I. Lanese and U. Montanari. A graphical fusion calculus. In *Proc. of the Workshop of the COMETA Project on Computational Metamodels*, volume 104 of *ENTCS*, pages 199–215. Elsevier, 2004.

[8] I. Lanese and U. Montanari. Mapping fusion and synchronized hyperedge replacement into logic programming. *Theory and Practice of Logic Programming, Special Issue on Multiparadigm Languages and Constraint Programming*, 2004. To appear.

[9] I. Lanese and U. Montanari. Synchronization algebras with mobility for graph transformations. In *Proc. of FGUC'04 – Foundations of Global Ubiquitous Computing*, ENTCS, 2004. To appear.

[10] I. Lanese and E. Tuosto. Synchronized hyperedge replacement for heterogeneous systems. In *Proc. of COORDINATION'05*, volume 3454 of *LNCS*, pages 220–235. Springer, 2005.

[11] R. Milner. A calculus of communicating systems. In volume 92 of *LNCS*. Springer, 1989.

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