

# The Verification and Control of Interacting Similar Discrete-Event Systems\*

Kurt Rohloff<sup>†</sup>

BBN Technologies

10 Moulton St., Cambridge, MA 02138, USA

krohloff@bbn.com; <http://www.dist-systems.bbn.com/people/krohloff>

Stéphane Lafortune

Department of Electrical Engineering and Computer Science

The University of Michigan

1301 Beal Ave., Ann Arbor, MI 48109-2122, USA

stephane@eecs.umich.edu; [www.eecs.umich.edu/umdes](http://www.eecs.umich.edu/umdes)

October 28, 2005

## Abstract

This paper explore issues related to the control and verification of similar module systems in the discrete-event systems framework. Similar module systems are distributed systems comprised of subsystem modules that exhibit isomorphic local behavior coordinated on global event occurrences. When given a global model of these systems, it is shown how to decompose the global model into the component subsystems in polynomial time. It is also shown how to perform various verification tasks for these interacting systems while mitigating common state explosion difficulties by taking advantage of the special similar module system structure. Control properties of the similar module systems are also discussed. It is assumed that the local modules are supervised by exactly one local controller and the controllers enforce the same local control policy. Necessary and sufficient conditions for achieving local and global control specifications in this setting are identified.

## 1 Introduction

Many significant, real-world systems are comprised of concurrent processes that are most conveniently modeled in a distributed manner. The current standard methods to solve many important verification and control problems for these distributed systems typically involve converting the distributed system models into equivalent monolithic system models. Unfortunately, these monolithic system conversions are generally computationally expensive. This has prompted researchers

---

\*This research was supported in part by NSF grants CCR-0082784 and CCR-0325571.

<sup>†</sup>The majority of the work presented in this paper was performed while the first author was at the University of Michigan.

to investigate solution methods for distributed system problems that avoid these procedures. However, it has been found that many of these important verification and control problems for distributed systems are still computationally intractable despite the avoidance of the monolithic conversions [8, 9, 14, 26, 28, 30].

This paper expands on prior investigations into the control of distributed systems by exploring the properties of these systems that can be modeled as a set of interacting discrete-event system modules where the system module behaviors are identical with respect to a relabeling of local behavior. This class of distributed systems is referred to as the class of *similar module systems* and is formally defined below. For these systems, the behaviors of the modules are represented as languages over a set of events partitioned into private and global events generated by finite-state automata. All modules must coordinate their behavior on the occurrence of global events, while private events represent behavior that is relevant to only one module.

The similar module system model used in this paper is designed to be as general as possible in order for it to have wide applicability, yet still have computationally reasonable solution methods for otherwise difficult verification and control problems. Consequently, many important real-world processes are included in the class of similar module systems that can be modeled in the framework discussed in this paper. Aspects of the behavior of systems such as communication networks where the nodes are all running the same communication protocol, distributed sensor networks and platoons of similar unmanned aerial vehicles can all be effectively represented by similar module systems for the purpose of verification and control with respect to logical specifications. A number of biological processes can also be effectively represented by similar module systems [1, 21].

The work in this paper is presented using the framework of discrete-event systems and supervisory control theory surveyed in the seminal work [25]. The interested reader may consult the text [2] for a general introduction to this framework. The finite automaton modeling formalism used in this paper is generally considered to be the simplest one for discrete-event systems that is expressive enough to model the behavior of real-world systems in a reasonable manner. Furthermore, as the behavior of the similar module systems is coordinated on the occurrence of global events, an equivalent monolithic global system model of a set of interacting similar module systems is constructed by composing the similar module systems using the parallel composition operation. The parallel composition operation is currently the standard method to model the coordination of behavior of distributed discrete-event systems.

Because similar module systems exhibit behavior that is distinctly local and global, specifications on the behavior of similar modular systems can be made at both the global and local levels. Global specifications describe the desired behavior of a system when the modules in the similar module system interact. Local specifications describe the desired locally relevant behavior of a system module when it interacts with other modules. The global and local behavioral specifications used in the paper are specified as the languages marked by finite-state automata.

Properties related to the control of similar modular systems for both local and global specifications given as regular languages are explored when the desired behavior of the controlled similar module system should match the given specification. It is assumed that when the modules are controlled, they are controlled locally with exactly one controller per module such that the controllers make local observations of the behavior of a module and enforce local control actions. As with the system modules, the controllers are similar in that all controllers enforce the same control policy at each of their respective subsystems with respect to a renaming of the relevant local events. It is also assumed that there is no communication between controllers besides the implicit communication due to the occurrence of observable global events.

Beyond the introduction of the similar module systems, the main contributions of this paper include:

- A polynomial time decomposition operation for converting a monolithic similar module system model into a more efficiently encoded distributed similar module system model.
- A quotient machine construction for more efficiently testing state reachability and other important properties of composed similar module systems that avoids expensive parallel composition operations.
- A polynomial time method for verifying local behavior specifications and a set of necessary and sufficient conditions for achieving local control objectives in similar module systems.
- A set of necessary and sufficient conditions for achieving global control objectives in similar module systems that imply a polynomial time method for synthesizing similar module systems that satisfy global control specifications.

The work in this paper generalizes and extends that in [27] which contains preliminary versions of some of the results presented herein. The extensions in this paper beyond those shown in [27] include the improved method for performing the verification of important system properties such as state reachability in similar module systems and the polynomial time decomposition operation for composed similar module systems.

The next section gives a survey of the literature relevant to the work presented in this paper. Section 3 introduces the system model along with the notation and basic system properties used in this paper. The quotient automaton is introduced in Section 4 for decreasing the computational difficulty of testing state reachability, global non-blockingness and deadlock-freeness in similar module systems. In Section 5 the properties related to the verification of similar module systems for specifications made at local sites are discussed. Section 6 shows the computationally efficient method of decomposing global models of similar module systems into distributed models. The control formalisms used in this paper are presented Section 7. Section 8 discusses properties related to the control of similar module systems for specifications made at the local sites, and the control of similar module systems for the satisfaction of global specifications is discussed in Section 9. This paper concludes with a discussion of the results and possible avenues for future work in Section 10.

## 2 Related Literature Survey

The supervision of modular discrete-event systems is currently receiving much attention in the control community; see, e.g., [3, 12, 13, 15–17, 19, 20, 22, 23, 31–33]. Some of the earlier results related to modular supervision are shown in [17, 19, 23, 31, 33]. Properties of modular discrete-event systems when the modules have disjoint event sets are investigated in [22, 23]. Various local specification and concurrent supervision problems, respectively, are investigated in [12, 13]. Properties related to the supervision of modular systems using specific architectures are explored in [15, 16]. A form of modular control where each controller has a different objective is discussed in [3] and situations when local non-blocking behavior implies global non-blocking behavior are discussed in [20].

Excluding [7, 27], there has been little work in the supervisory control community that exploits system symmetry when controlling discrete-event systems. Group-theoretic methods are

used in [7] to define classes of states and transitions in monolithic system that are equivalent under predefined group-theoretic permutation operations. These classes of equivalent states and transitions are used to construct a quotient automaton that has a state space smaller than that of the original system which is used to perform various centralized controller synthesis operations for the original system. Similar module systems are briefly discussed in [7] and the predefined quotient structure is extended to this setting. However, centralized control is discussed exclusively in [7]. A similar quotient automaton with predefined state equivalence classes is also discussed in [27] for similar module systems which is likewise based on state permutation equivalences.

A type of quotient automaton is presented below to test properties such as state reachability, non-blockingness and deadlock freeness that uses a much more efficient predefined state aggregation method for defining state equivalence classes than seen in [7, 27]. This results in a quotient automaton with a much smaller state space than the previously discussed quotient automata with predefined state equivalence classes. Furthermore, necessary and sufficient conditions are discussed for the existence of decentralized controllers for these systems using both local and global specifications. This control setting is not discussed in [7].

The work in [7, 27] and this paper is inspired by an extensive literature from the formal methods community based on the uses of symmetry for performing model checking over large symmetric systems. Most of the current work in the formal methods community related to symmetry is based on methods presented in [5, 6]. Some of this work is also summarized in the text [4]. Due to the natural similarity of the supervisory control theory models used in this paper and the Kripke structure models used in the formal methods literature, results related to the verification of symmetric systems as presented in this literature are briefly discussed.

Group-theoretic methods and the quotient automaton structures for Kripke structures are presented in [5, 6]. These references present how various temporal logic properties can hold in the original system if and only if those properties hold in the quotient automaton. Defining effective state equivalence classes is computationally at least as difficult as the graph isomorphism problem [10]. This has induced several authors to attempt to design distributed systems with special architectures so that symmetry is guaranteed to occur *a priori*, as in [6, 7, 27] and this paper. Even when used with distributed systems, the symmetry reductions due to the construction of the quotient automata presented in [5, 6] do not alleviate the state-explosion problem in the worst case. However, several real-world examples are presented in [11] where the quotient models allow for 80% to 90% reductions in computation time and space for several important temporal logic verification problems.

There has also been work in the formal methods community on the use of partial order reductions for the verification of concurrent systems [4]. Although the reduction methods appear similar, partial order reduction methods are fundamentally different from the symmetry reduction methods discussed herein.

### 3 Similar Module Systems

As was stated in the introduction, the similar module systems are modeled as a set of finite automata  $\{G_1, \dots, G_n\}$  that interact on the occurrence of common global events. Furthermore, all of the modules in  $\{G_1, \dots, G_n\}$  are exact copies of one another with respect to a renaming of private events.

Let the automaton  $G_i = (X, x_0, \Sigma_i, \delta_i, X_m)$  be the  $i$ th module in the system, where  $i \in \{1, \dots, n\}$ . The module event set  $\Sigma_i$  is partitioned into the distinct subsets  $\Sigma_g$  and  $\Sigma_{pi}$ , the (unique) global event set and the private event set for module  $i$ , respectively. The private event

sets  $\Sigma_{p1}, \dots, \Sigma_{pn}$  are copies of one another for  $G_1, \dots, G_n$ , respectively, such that for all  $i, j, i \neq j$ ,  $\Sigma_{pi} \cap \Sigma_{pj} = \emptyset$ . It is assumed that  $\Sigma_g \cap \Sigma_{pi} = \emptyset$  and  $\Sigma$  denotes  $\Sigma_1 \cup \dots \cup \Sigma_n$ .

For any local event  $\sigma_i \in \Sigma_{pi}$  there are corresponding local events  $\Psi_{i1}(\sigma_i) \in \Sigma_{p1}, \dots, \Psi_{in}(\sigma_i) \in \Sigma_{pn}$  in the other private event sets. The one-to-one mapping  $\Psi_{ij}(\cdot)$  is extended for notational simplicity in this setting to  $\Psi_{ij} : \Sigma \rightarrow \Sigma$  so that the private event set of module  $i$  is mapped to the  $j$ th private event set, the private event set of module  $j$  is mapped to the  $i$ th private event set and all other events are mapped to themselves. The function  $\Psi_{ij}(\cdot)$  is extended in the usual manner to map between strings of events and languages. Note that  $\Psi_{ii}(\cdot)$  is the identity function and  $\Psi_{ii}(\cdot) = \Psi_{ij}(\Psi_{ij}(\cdot))$ . As an example of the operation of the  $\Psi_{ij}(\cdot)$  function, suppose there are system event sets  $\Sigma_i = \{a_i, b_i, d, g\}$  and  $\Sigma_j = \{a_j, b_j, d, g\}$ . In this case  $\Sigma_g = \{d, g\}$ ,  $\Sigma_{p1} = \{a_i, b_i\}$  and  $\Sigma_{p2} = \{a_j, b_j\}$ . Then,  $\Psi_{ij}(dga_i a_j b_j g b_i) = dga_j a_i b_i g b_j$ .

With the above framework, similar module systems can now be formally defined.

**Definition 1** *The set of modules  $\{G_1, \dots, G_n\}$  as described above is a similar module system if for all  $i, j \in \{1, \dots, n\}$ , for all  $x \in X$  and  $\gamma \in \Sigma_i$ ,  $\delta_i(x, \gamma) = \delta_j(x, \Psi_{ij}(\gamma))$  if it is defined.*

Example 1 shows a simple example of the systems discussed in this chapter.

**Example 1** *Consider the 2 module similar module system composed of  $G_1$  and  $G_2$  seen in Figure 1. The relevant event sets are:  $\Sigma_g = \{\gamma, \lambda\}$ ,  $\Sigma_{p1} = \{\alpha_1, \beta_1\}$  and  $\Sigma_{p2} = \{\alpha_2, \beta_2\}$ . Note that  $G_1$  and  $G_2$  are both locally non-blocking and deadlock free.*

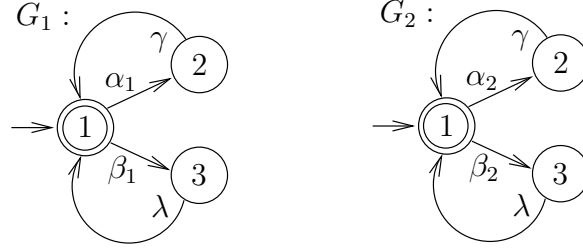


Figure 1: The automata  $G_1$  and  $G_2$ .

The natural projection  $P_i : \Sigma^* \rightarrow \Sigma_i^*$  is defined such that  $P_i(s)$  is the string  $s$  with events in  $\Sigma \setminus \Sigma_i$  erased. Similarly, let  $P_{pi} : \Sigma^* \rightarrow \Sigma_{pi}^*$  be the natural projection that erases events in  $\Sigma \setminus \Sigma_{pi}$  and let  $P_g : \Sigma^* \rightarrow \Sigma_g^*$  be the natural projection that erases events in  $\Sigma \setminus \Sigma_g$ .

The automata  $\{G_1, \dots, G_n\}$  model the private behaviors of the modules of the similar module system, but the parallel composition  $G_1 \parallel \dots \parallel G_n$  is used to model the global behavior of all of the interacting subsystems. The behavior of the composed automaton  $G_1 \parallel \dots \parallel G_n$  is equivalent to the concurrent behavior of the modules  $\{G_1, \dots, G_n\}$  where the local behaviors of  $\{G_1, \dots, G_n\}$  are coordinated on the occurrence of common global events in  $\Sigma_g$ .

To review the parallel composition operation, suppose the automata  $G_1$  and  $G_2$  are given. The parallel composition of  $G_1$  and  $G_2$  denoted by  $G_1 \parallel G_2$  is defined as follows:

$$G_1 \parallel G_2 := ((X \times X), (x_o, x_o), \Sigma_1 \cup \Sigma_2, \delta^{G_1 \parallel G_2}, (X_m \times X_m))$$

where

$$\delta^{G_1 \parallel G_2}((x^{G_1}, x^{G_2}), \sigma) = \left\{ \begin{array}{ll} (\delta_1(x^{G_1}, \sigma), \delta_2(x^{G_2}, \sigma)) & \text{if } \delta_1(x^{G_1}, \sigma)! \wedge \delta_2(x^{G_2}, \sigma)! \\ (\delta_1(x^{G_1}, \sigma), x^{G_2}) & \text{if } \delta_1(x^{G_1}, \sigma)! \wedge (\sigma \notin \Sigma_2) \\ (x^{G_1}, \delta_2(x^{G_2}, \sigma)) & \text{if } \delta_2(x^{G_2}, \sigma)! \wedge (\sigma \notin \Sigma_1) \\ \text{undefined} & \text{otherwise} \end{array} \right\} \quad (1)$$

Note that the unary operator ! is used such that  $f(\alpha)!$  returns true if  $f(\cdot)$  is defined for input  $\alpha$ , false otherwise. The parallel composition  $G_1\|G_2$  is also defined in [2] for the case where  $G_1$  and  $G_2$  are not similar module systems.

Example 2 shows the automaton  $G_1\|G_2$  constructed from the automata in Example 1.

**Example 2** Consider the composed system  $G_1\|G_2$  as seen in Figure 2.

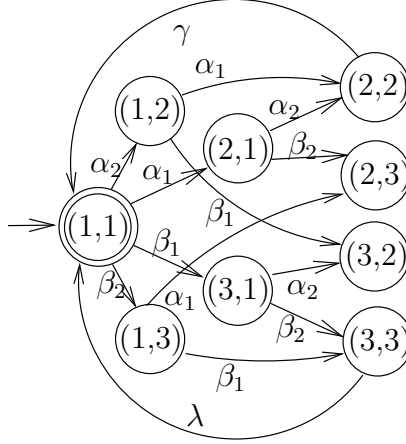


Figure 2: The automaton  $G_1\|G_2$ .

The automaton  $G_1\|G_2$  has a large degree of state-based symmetry due to the similarity of the component automata  $G_1$  and  $G_2$ . Consider for instance the states  $(1,2)$  and  $(2,1)$ ; these states are reached by the occurrence of  $\alpha_2$  and  $\alpha_1$  events respectively.

Now consider the states  $(2,3)$  and  $(3,2)$ . The state  $(2,3)$  can be reached from  $(1,1)$  on the occurrence of either  $\alpha_1\beta_2$  or  $\beta_2\alpha_1$ . The state  $(3,2)$  can be reached from  $(1,1)$  on the occurrence of either  $\alpha_2\beta_1$  or  $\beta_1\alpha_2$ . Notice that if the subscript labels of the events in the strings leading to state  $(2,3)$  are swapped then the resulting string will lead to state  $(3,2)$ .

This is a symptom of properties inherent to similar module systems such that many properties of state  $(2,3)$  are also held in a sense by state  $(3,2)$ . This is because the swapping of event subscript labels is equivalent to reordering the parallel composition of the component automata by swapping state locations. This operation is valid because the parallel composition operation is commutative and therefore the order of parallel composition is arbitrary. The various similar modules are identical with respect to a renaming of private events.

For  $G_1\|G_2$  there are six classes of states that could be considered equivalent with respect to a reordering of the component states. The six sets of equivalent state classes are

$$\{(1,1)\}, \{(1,2), (2,1)\}, \{(1,3), (3,1)\}, \{(2,2)\}, \{(2,3), (3,2)\}, \{(3,3)\}.$$

It is now shown that  $P_i(\mathcal{L}_m(G_1\|\dots\|G_n))$ , the local behavior relevant to module  $i$  of the interacting modules, is language equivalent to the local behavior of module  $i$  when the modules operate in isolation (i.e.,  $\mathcal{L}_m(G_i)$ ).

**Theorem 1** For a similar module system  $\{G_1, \dots, G_n\}$  as introduced above with respective local projection operations  $\{P_1, \dots, P_n\}$  and for  $i \in \{1, \dots, n\}$ ,

$$P_i(\mathcal{L}_m(G_1\|\dots\|G_n)) = \mathcal{L}_m(G_i).$$

**Proof:** It is known that

$$\mathcal{L}_m(G_1 \parallel \cdots \parallel G_n) = [P_1^{-1}(\mathcal{L}_m(G_1)) \cap \cdots \cap P_n^{-1}(\mathcal{L}_m(G_n))],$$

so it is sufficient to show that  $P_i [P_1^{-1}(\mathcal{L}_m(G_1)) \cap \cdots \cap P_n^{-1}(\mathcal{L}_m(G_n))] = \mathcal{L}_m(G_i)$ .

This proof is composed of two parts.

It is known that  $P_1^{-1}(\mathcal{L}_m(G_1)) \cap \cdots \cap P_n^{-1}(\mathcal{L}_m(G_n)) \subseteq P_i^{-1}(\mathcal{L}_m(G_i))$ .

$\Rightarrow P_i [P_1^{-1}(\mathcal{L}_m(G_1)) \cap \cdots \cap P_n^{-1}(\mathcal{L}_m(G_n))] \subseteq P_i [P_i^{-1}(\mathcal{L}_m(G_i))]$ .

For any language  $K$  it is known that  $P_i [P_i^{-1}(K)] = K$ , so

$P_i [P_1^{-1}(\mathcal{L}_m(G_1)) \cap \cdots \cap P_n^{-1}(\mathcal{L}_m(G_n))] \subseteq \mathcal{L}_m(G_i)$

$\Rightarrow P_i(\mathcal{L}_m(G_1 \parallel \cdots \parallel G_n)) \subseteq \mathcal{L}_m(G_i)$ .

The other direction is proved by showing

$(t_i \in \mathcal{L}_m(G_i)) \Rightarrow (t_i \in P_i [\cap_{j=1}^n P_j^{-1}(\mathcal{L}_m(G_j))])$ .

Let  $t_i \in \mathcal{L}_m(G_i)$ . This implies that  $t_i \in P_i(P_i^{-1}(\mathcal{L}_m(G_i)))$ . Let the string  $t_{\{1, \dots, n\}}$  be a copy of  $t_i$  except that the occurrence of all private events  $\sigma_{pi} \in \Sigma_{pi}$  in  $t_i$  are replaced with the string of events  $\sigma_{p1} \Psi_{12}(\sigma_{p1}), \dots, \Psi_{1n}(\sigma_{p1})$ . Due to the construction of  $\{G_1, \dots, G_n\}$ ,  $t_{\{1, \dots, n\}} \in P_j^{-1}(\mathcal{L}_m(G_j))$  for all  $j \in \{1, \dots, n\}$ . Thus,

$\forall j \in \{1, \dots, n\} \ t_{\{1, \dots, n\}} \in P_j^{-1}(\mathcal{L}_m(G_j))$

$\Rightarrow t_{\{1, \dots, n\}} \in (\cap_{j=1}^n P_j^{-1}(\mathcal{L}_m(G_j)))$ .

$\Rightarrow t_i \in P_i [\cap_{j=1}^n P_j^{-1}(\mathcal{L}_m(G_j))]$ .

Therefore,  $\mathcal{L}_m(G_i) \subseteq P_i [\cap_{j=1}^n P_j^{-1}(\mathcal{L}_m(G_j))]$  which implies

$$\mathcal{L}_m(G_i) \subseteq P_i(\mathcal{L}_m(G_1 \parallel \cdots \parallel G_n)).$$

■

It is a simple matter to extend the result in Theorem 1 to the case of languages generated instead of languages marked, i.e.,

$$P_i [P_1^{-1}(\mathcal{L}(G_1)) \cap \cdots \cap P_n^{-1}(\mathcal{L}(G_n))] = \mathcal{L}(G_i).$$

Some basic properties of distributed discrete-event systems are now presented.

**Definition 2** [31] *A language  $K \subseteq \Sigma^*$  is separable with respect to the projections  $\{P_1, \dots, P_n\}$  if  $K = P_1^{-1}(P_1(K)) \cap \cdots \cap P_n^{-1}(P_n(K))$ .*

According to the above definition, a language is separable if with distributed local knowledge of  $K$  at all modules, i.e.,  $P_1(K), \dots, P_n(K)$ , the language  $K$  can be recovered exactly through combining all local knowledge:  $P_1^{-1}(P_1(K)) \cap \cdots \cap P_n^{-1}(P_n(K))$ . Note that in [29] a concept similar to separability called decomposability is defined where a language  $K \subseteq \Sigma^*$  is decomposable with respect to the set of projections  $\{P_1, \dots, P_n\}$  and the automaton  $G$  if  $K = P_1^{-1}(P_1(K)) \cap \cdots \cap P_n^{-1}(P_n(K)) \cap \mathcal{L}(G)$ .

In general the current known methods to test a language for separability take exponential time. These methods rely on computing automata  $\{B_1, \dots, B_n\}$  such that for all  $i \in \{1, \dots, n\}$ ,  $\mathcal{L}(B_i) = P_i^{-1}(P_i(K))$  and then testing to see if  $K = \mathcal{L}(B_1 \parallel \cdots \parallel B_n)$ .

**Definition 3** [33] *The languages in the set  $\{L_1, \dots, L_n\}$  are called non-conflicting if*

$$\overline{L_1 \cap \cdots \cap L_n} = \overline{L_1} \cap \cdots \cap \overline{L_n}.$$

It is known that the parallel composition of a set of non-blocking automata need not be non-blocking unless the respective languages marked by the automata are non-conflicting.

**Definition 4** A language  $K \subseteq \Sigma^*$  is symmetric with respect to the mappings  $\Psi_{11}, \dots, \Psi_{1n}$  if  $\forall i \in \{1, \dots, n\} \Psi_{1i}(K) = K$ .

A language marked by a deterministic automaton  $B$  can be generally tested for symmetry in linear time. To do this, for all  $i \in \{1, \dots, n\}$ , construct  $B_{1i}$  such that  $\mathcal{L}(B_{1i}) = \Psi_{1i}(\mathcal{L}(B))$ . This can be done by replacing the event labels on the state transitions in  $B$  according to the  $\Psi_{1i}(\cdot)$  function and can be performed in linear time. Then, if for all  $i \in \{1, \dots, n\}$ ,  $\mathcal{L}(B_{1i}) = \mathcal{L}(B)$ , then by definition  $\mathcal{L}(B)$  is symmetric. Note that the language equivalence of deterministic automata can be tested in linear time using standard methods.

The symmetry definition is used to convey the intuition that  $K$  is identical with respect to a relabeling of private events for the similar module system  $\{G_1, \dots, G_n\}$ . This prompts the following lemmas whose proofs are straightforward and therefore omitted.

**Lemma 1** Given a set  $\Sigma_i \subseteq \Sigma$ , a language  $K \subseteq \Sigma^*$  with the natural projection operation  $P_i : \Sigma^* \rightarrow \Sigma_i^*$  and corresponding inverse projection  $P_i^{-1} : \Sigma_i^* \rightarrow \Sigma^*$ ,

1.  $P_i(\overline{K}) = \overline{P_i(K)}$ .
2.  $P_i^{-1}(\overline{K}) = \overline{P_i^{-1}(K)}$ .

**Lemma 2** Given a language  $K \subseteq \Sigma^*$ , a set of projection operations  $\{P_1, \dots, P_n\}$  and translation mappings  $\{\Psi_{11}, \dots, \Psi_{1n}\}$ ,

$$\forall i \in \{1, \dots, n\} (K = \Psi_{1i}(K)) \Rightarrow (\Psi_{1i}(P_1(K)) = P_i(K)).$$

The next result relates the  $\Psi_{ij}(\cdot)$  operator with the prefix-closure of languages.

**Lemma 3** Given a language  $K$  and the  $\Psi_{ij}(\cdot)$  operator introduced above,

$$\Psi_{ij}(\overline{K}) = \overline{\Psi_{ij}(K)}.$$

**Proof:** Suppose  $s \in \overline{\Psi_{ij}(K)}$ . Then there exists a string  $t$  such that  $st \in \Psi_{ij}(K)$ .

$$\begin{aligned} &\Rightarrow \Psi_{ij}(st) \in K \\ &\Rightarrow \Psi_{ij}(s)\Psi_{ij}(t) \in K \\ &\Rightarrow \Psi_{ij}(s) \in \overline{K} \\ &\Rightarrow s \in \Psi_{ij}(\overline{K}) \end{aligned}$$

The reverse direction of this proof follows from performing the steps above in the opposite order. ■

Lemma 3 can be used to show the following theorem about the symmetry of prefix-closed languages.

**Theorem 2** Let  $K$  be symmetric with respect to  $\{\Psi_{11}, \dots, \Psi_{1n}\}$ . Then,  $\overline{K}$  is symmetric with respect to  $\{\Psi_{11}, \dots, \Psi_{1n}\}$ .

**Proof:** Because  $K$  is symmetric with respect to  $\{\Psi_{11}, \dots, \Psi_{1n}\}$ , then for all  $i, j \in \{1, \dots, n\}$ ,  $\Psi_{ij}(K) = K$ . Hence, by Lemma 3,  $\forall i, j \in \{1, \dots, n\}$   $\Psi_{ij}(\overline{K}) = \overline{K}$  and therefore  $\overline{K}$  is symmetric with respect to  $\{\Psi_{11}, \dots, \Psi_{1n}\}$ . ■

Even though a language  $K$  may be symmetric with respect to  $\{\Psi_{11}, \dots, \Psi_{1n}\}$  and separable with respect to the sets of projections  $\{P_1, \dots, P_n\}$ , it is possible that the sets of languages  $\{P_1^{-1}(P_1(K)), \dots, P_n^{-1}(P_n(K))\}$  are conflicting. Consider the following example.

**Example 3** Consider the automaton  $G$  that is the trimmed version of  $G_1 \parallel G_2$  from Example 1 with the states renamed for convenience. This automaton can be seen in Figure 3.

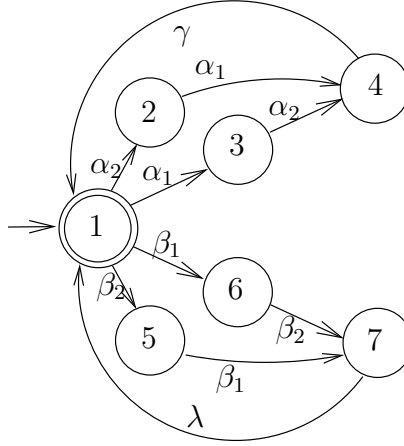


Figure 3: The automaton  $G = \text{Trim}(G_1 \parallel G_2)$ .

Note that  $\mathcal{L}_m(G) = ((\alpha_1\alpha_2 + \alpha_2\alpha_1)\gamma + (\beta_1\beta_2 + \beta_2\beta_1)\lambda)^*$ . By inspection, this language is symmetric with respect to  $\{\Psi_{11}, \dots, \Psi_{1n}\}$  and separable with respect to  $\{P_1, \dots, P_n\}$ .

Now consider the  $G_1$  and  $G_2$  automata seen in Figure 1. Note that  $P_1(\mathcal{L}(G)) = \mathcal{L}(G_1)$  and  $P_2(\mathcal{L}(G)) = \mathcal{L}(G_2)$ . As seen in Figure 2, the parallel composition of  $G_1$  and  $G_2$  is blocking. Therefore  $\{P_1^{-1}(P_1(\mathcal{L}_m(G_1))), \dots, P_n^{-1}(P_n(\mathcal{L}_m(G_2)))\}$  are conflicting.

Finally, note that even though a language  $K$  may be symmetric with respect to  $\{\Psi_{11}, \dots, \Psi_{1n}\}$  and  $\{P_1^{-1}(P_1(K)), \dots, P_n^{-1}(P_n(K))\}$  are non-conflicting, it is possible that  $K$  is not separable with respect to  $\{P_1, \dots, P_n\}$ . Consider the language  $K = \{\epsilon, \alpha_1, \alpha_2\}$ .  $K$  is symmetric with respect to the mapping induced by the event sets  $\{\alpha_1\}$  and  $\{\alpha_2\}$ . Because

$$P_1^{-1}(P_1(K)) = \{\alpha_2^*, \alpha_2^*\alpha_1\alpha_2^*\} = \overline{P_1^{-1}(P_1(K))}$$

and

$$P_2^{-1}(P_2(K)) = \{\alpha_1^*, \alpha_1^*\alpha_2\alpha_1^*\} = \overline{P_2^{-1}(P_2(K))},$$

it holds that

$$P_1^{-1}(P_1(K)) \cap P_2^{-1}(P_2(K)) = \overline{P_1^{-1}(P_1(K))} \cap \overline{P_2^{-1}(P_2(K))} = \{\epsilon, \alpha_1, \alpha_2, \alpha_1\alpha_2, \alpha_2\alpha_1\}.$$

Consequently,  $\overline{P_1^{-1}(P_1(K))} \cap \overline{P_2^{-1}(P_2(K))} = \overline{P_1^{-1}(P_1(K)) \cap P_2^{-1}(P_2(K))}$ , so  $P_1^{-1}(P_1(K))$  and  $P_2^{-1}(P_2(K))$  are non-conflicting. However,  $K = \{\epsilon, \alpha_1, \alpha_2\}$  is not separable with respect to  $\{\{\alpha_1\}, \{\alpha_2\}\}$  because  $\{\alpha_1\alpha_2, \alpha_2\alpha_1\} \subset P_1^{-1}(P_1(K)) \cap P_2^{-1}(P_2(K))$ , but neither of the strings  $\alpha_1\alpha_2$  or  $\alpha_2\alpha_1$  are in  $K$ .

## 4 Quotient Automata for Verification

Given a set of automata  $\{G_1, \dots, G_n\}$  such that the size of their respective state spaces is bounded by  $k$ , the composed automaton  $G_1 \parallel \dots \parallel G_n$  has  $k^n$  reachable states in the worst case. Therefore, as  $n$  grows, the size of the state space of  $G_1 \parallel \dots \parallel G_n$  can become unbearably large. This would consequently cause many verification and control procedures that require an enumeration over all of the reachable states of  $G_1 \parallel \dots \parallel G_n$ , such as those discussed in [26, 28], to take an unbearable amount of time if  $n$  is large.

However, for many types of symmetric systems a quotient automaton construction can be used to greatly decrease computation time when testing if some important properties hold at various system states [4, 27]. The underlying idea behind a quotient automaton construction is that the system states,  $X^n$  for  $G_1 \parallel \dots \parallel G_n$ , are partitioned into equivalence classes  $\{X_1, \dots, X_z\} \subseteq 2^{(X^n)}$  and each state equivalence class  $X_j$  is assigned an unique representative state  $x'_j$  from a set of representative states  $\{x'_1, \dots, x'_z\}$ . The set of representative states  $\{x'_1, \dots, x'_z\}$  forms the state space of the quotient automaton and the transition structure is defined such that if there is a transition between two states  $x_i \in X_i$  and  $x_j \in X_j$ , then there is a transition between the corresponding representative states  $x'_i$  and  $x'_j$  in the quotient automaton. Depending on how the states of  $G_1 \parallel \dots \parallel G_n$  are partitioned into  $\{X_1, \dots, X_z\}$  and how the labelling of transitions in the quotient automaton are assigned, it could be necessary and sufficient for various system properties to hold in the original system if the properties hold in the quotient automaton.

In general, defining the most efficient state partition for a quotient automaton to verify important system properties such as state reachability is computationally difficult and at least as difficult as the graph isomorphism problem [10]. However, for special systems such as the similar module systems discussed in this paper, there has been research (such as in [7, 27]) into using predefined efficient state partitionings for the construction of quotient automata constructions that are used to test important system properties in the composed system  $G_1 \parallel \dots \parallel G_n$ . This predefinition of the quotient structure avoids the difficult computations associated with finding the most efficient state partitioning, but still allows for efficient state partitionings that are generally useful in practice.

This section presents a predefined state partitioning of the system states in the composed similar module system  $G_1 \parallel \dots \parallel G_n$  for the construction of a nondeterministic quotient automaton  $\tilde{G}$ . This partitioning uses sets of states with the same component states as the state equivalence classes used in the quotient automaton construction. The  $\tilde{G}$  automaton constructed in this manner can be used to test state reachability properties in the composed similar module system  $G_1 \parallel \dots \parallel G_n$  it was constructed from. A simple tutorial example is shown that demonstrates how the states of the composed similar module system can be partitioned into sets of “equivalent” states.

**Example 4** *Reconsider the system first introduced in Example 1 with a third component  $G_3$  similar to  $G_1$  and  $G_2$  in Figure 1. The automaton  $G_1 \parallel G_2 \parallel G_3$  can be seen in Figure 4.*

*State (1, 1, 2) in  $G_1 \parallel G_2 \parallel G_3$  is equivalent to (2, 1, 1), (1, 2, 2), (2, 1, 2), etc... with respect to the sets of module states these composed states contain. By inspection, these states are also equivalent with respect to some important system properties such as deadlock, blocking and reachability. However, not all of these states are reachable from the initial state in the same number of transitions.*

The intuition behind Example 4 is that state orderings and duplicated component states do not matter when testing several important global properties. A quotient automaton  $\tilde{G}$  for testing these properties in similar module systems is now formally introduced.

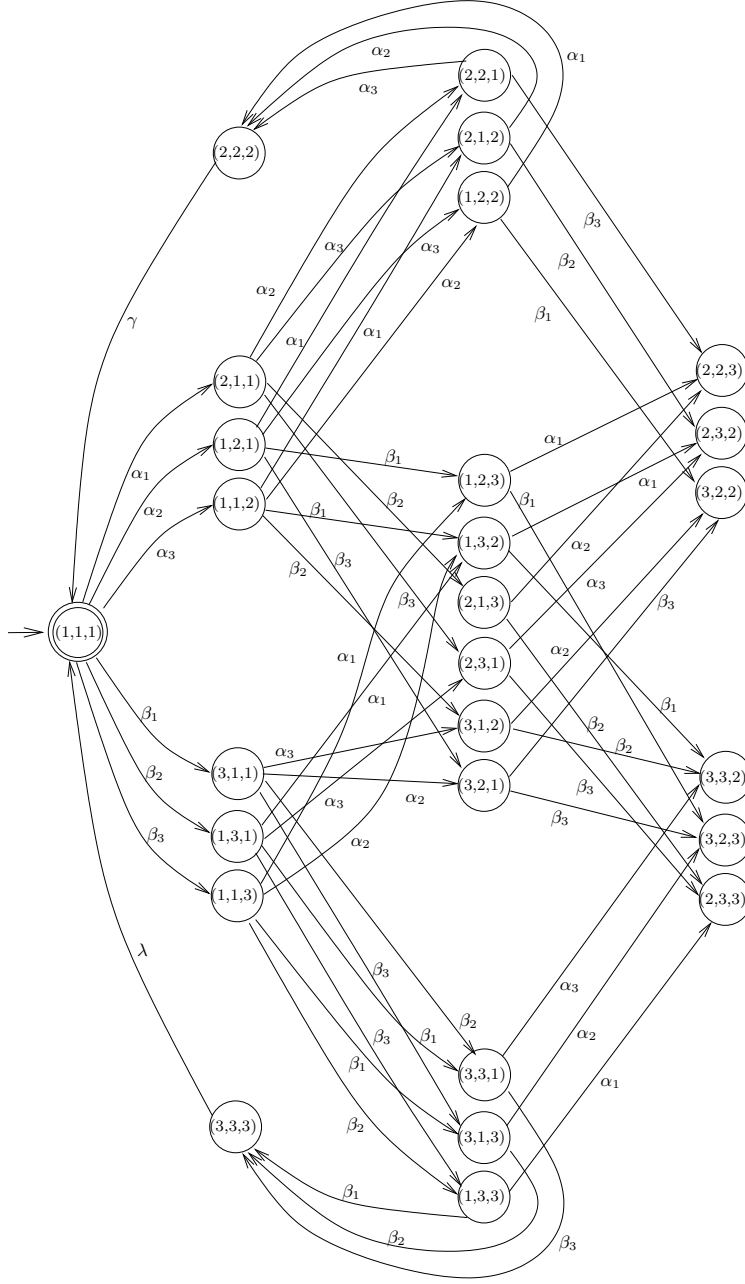


Figure 4: The automaton  $G_1||G_2||G_3$ .

#### 4.1 The Construction of $\tilde{G}$

Define  $G^\parallel = (X^\parallel, x_0^\parallel, \Sigma, \delta^\parallel, X_m^\parallel) = G_1||\dots||G_n$ . A state  $x^\parallel \in X^\parallel$  from the composed automaton is an  $n$ -tuple of states in  $X$ . Let  $\tilde{x}$  represent the set of module states that compose the  $n$ -tuple  $x^\parallel$ . For example, the set  $\{1, 2, 6\}$  is the set of module states that compose the states  $(1, 2, 6, 2)$  and  $(1, 6, 6, 2)$  in  $X^\parallel$ . The function  $Comp : X^\parallel \rightarrow 2^X$  is defined such that  $Comp(x^\parallel) = \tilde{x}$  and the function  $Comp^{-1}(\tilde{x})$  returns the set of states that have exactly the states in  $\tilde{x}$  in their  $n$ -tuple. As an example of the operation of the  $Comp(\cdot)$  and  $Comp^{-1}(\cdot)$  functions for the system in Example 4,  $Comp((2, 1, 2)) = \{1, 2\}$ ,  $Comp((2, 1, 3)) = \{1, 2, 3\}$ ,  $Comp^{-1}(\{1, 2\}) = \{(1, 2, 2), (2, 1, 2), (2, 2, 1), (1, 1, 2), (1, 2, 1), (2, 1, 1)\}$ . Note that [6, 7, 27] uses state component

permutation operators to define the state equivalence classes. It is discussed below that the state equivalence classes defined using the  $Comp(\cdot)$  operator are generally much more efficient than those using permutation operators.

The set of state equivalence classes defined by the  $Comp(\cdot)$  function is used as the state space of  $\tilde{G}$ . Let  $\tilde{G} = (\tilde{X}, \{x_0\}, \Sigma_1, \tilde{\delta}, \tilde{X}_m)$  where  $\tilde{X} = 2^X$ , the power set of the component automaton  $G_i$  states, and let  $\tilde{X}_m$ , the set of marked states, be the states of  $\tilde{G}$  such that all component states in the set are marked, i.e.,  $\tilde{X}_m = 2^{X_m}$ . The possibly nondeterministic transition operator  $\tilde{\delta}(\cdot, \cdot)$  is defined such that if there exists two states  $x^\parallel \in Comp^{-1}(\tilde{x})$  and  $y^\parallel \in Comp^{-1}(\tilde{y})$  and an event  $\sigma_i \in \Sigma_i$  such that  $\delta^\parallel(x^\parallel, \sigma_i) = y^\parallel$ , then there is a corresponding transition in  $\tilde{G}$  from  $\tilde{x}$  to  $\tilde{y}$  labeled by  $\Psi_{i1}(\sigma_i)$ . A formal construction algorithm for  $\tilde{G}$  is now given.

**Algorithm 1** *Quotient Construction Construction Algorithm.*

*Input:*

$$\{G_1, \dots, G_n\} \text{ such that } \forall i \in \{1, \dots, n\}, \\ G_i = (X^i, x_0^i, \Sigma_i, \delta_i, X_m^i).$$

*Output:*

$$\tilde{G} = (\tilde{X}, \tilde{x}_0, \Sigma, \tilde{\delta}, \tilde{X}_m). \\ (\text{Note that } \tilde{\delta} \subseteq (\tilde{X} \times \Sigma \times \tilde{X})).$$

*Assumptions:*

$$\{G_1, \dots, G_n\} \text{ is a similar module system with respect to } \{\Psi_{11}, \dots, \Psi_{1n}\} \\ S \text{ is a stack with the normal push and pop operations.}$$

*Initialize:*

$$\tilde{X} := \{\{x_0\}\}; \\ \tilde{x}_0 := \{x_0\}; \\ \tilde{\delta} := \emptyset; \\ S := [\tilde{x}_0]; \\ \text{If } \tilde{x}_0 \subseteq X_m \\ \quad \text{Then } \tilde{X}_m := \tilde{x}_0 \\ \quad \text{Else } \tilde{X}_m := \emptyset;$$

*Repeat:*

$$\{ \\ \tilde{x}_s := pop(S) \\ \text{Do the following for all } \sigma \in \Sigma: \\ \{ \\ \text{If } (\sigma \in \Sigma_g) \wedge (\forall x \in \tilde{x}, \delta_1(x, \sigma)!) \\ \quad \text{Then} \\ \quad \{ \\ \quad \tilde{x}_c := \{\delta_1(x, \sigma) | x \in \tilde{x}\}; \\ \quad \tilde{\delta} := \tilde{\delta} \cup (\tilde{x}_s, \sigma, \tilde{x}_c); \\ \quad \text{If } \tilde{x}_c \notin \tilde{X} \\ \quad \quad \text{Then} \\ \quad \quad \{$$

```

         $\tilde{X} := \tilde{X} \cup \tilde{x}_c;$ 
         $push(S, \tilde{x}_c);$ 
         $If \tilde{x}_c \subseteq X_m$ 
             $Then \tilde{X}_m := \tilde{X}_m \cup \{\tilde{x}_c\}$ 
        }
    }
}
If  $(\sigma \in \Sigma_{p1}) \wedge (|\tilde{x}| < n) \wedge (\delta_1(x, \sigma)!) \wedge (x \in \tilde{x})$ 
    Then
        {
         $\tilde{x}_c := \tilde{x} \cup \{\delta_1(x, \sigma)\};$ 
         $\tilde{\delta} := \tilde{\delta} \cup (\tilde{x}_s, \sigma, \tilde{x}_c);$ 
         $If \tilde{x}_c \notin \tilde{X}$ 
            Then
                {
                 $\tilde{X} := \tilde{X} \cup \tilde{x}_c;$ 
                 $push(S, \tilde{x}_c);$ 
                 $If \tilde{x}_c \subseteq X_m$ 
                     $Then \tilde{X}_m := \tilde{X}_m \cup \{\tilde{x}_c\}$ 
                }
            }
        }
    }
}
If  $(\sigma \in \Sigma_{p1}) \wedge (|\tilde{x}| > 1) \wedge (\delta_1(x, \sigma)!) \wedge (x \in \tilde{x})$ 
    Then
        {
         $\tilde{x}_c := [\tilde{x} \setminus \{x\}] \cup \{\delta_1(x, \sigma)\};$ 
         $\tilde{\delta} := \tilde{\delta} \cup (\tilde{x}_s, \sigma, \tilde{x}_c);$ 
         $If \tilde{x}_c \notin \tilde{X}$ 
            Then
                {
                 $\tilde{X} := \tilde{X} \cup \tilde{x}_c;$ 
                 $push(S, \tilde{x}_c);$ 
                 $If \tilde{x}_c \subseteq X_m$ 
                     $Then \tilde{X}_m := \tilde{X}_m \cup \{\tilde{x}_c\}$ 
                }
            }
        }
    }
}
}
Until  $S$  is empty;
Return  $\tilde{G} = (\tilde{X}, \tilde{x}_0, \Sigma, \tilde{\delta}, \tilde{X}_m)$ .

```

Note that even if  $G^{\parallel}$  is a deterministic automaton, the corresponding  $\tilde{G}$  automaton maybe nondeterministic. That is, at a state  $\tilde{x} \in \tilde{X}$  there may be several outgoing transitions labelled by an event  $\sigma$ . In this case the state transition function  $\tilde{\delta}$  is defined such that  $\tilde{\delta} : \tilde{X} \times \Sigma_1 \rightarrow 2^{\tilde{X}}$

where for a state  $\tilde{x} \in \tilde{X}$  and an event  $\sigma \in \Sigma_1$ ,  $\tilde{\delta}(\tilde{x}, \sigma) \subseteq \tilde{X}$  represents the set of states in  $\tilde{G}$  that lead from  $\tilde{x}$  by transitions labelled by  $\sigma$ . Therefore, the state transition function  $\tilde{\delta}$  is defined such that

$$\tilde{\delta}(\tilde{x}, \sigma) = \left\{ \begin{array}{ll} \{\delta_1(x, \sigma) | x \in \tilde{x}\} & \text{if } (\sigma \in \Sigma_g) \wedge (\forall x \in \tilde{x}, \delta_1(x, \sigma)!) \\ \tilde{x} \cup \{\delta_1(x, \sigma)\} & \text{if } (\sigma \in \Sigma_{p1}) \wedge (|\tilde{x}| < n) \wedge (\delta_1(x, \sigma)!) \wedge (x \in \tilde{x}) \\ [\tilde{x} \setminus \{x\}] \cup \{\delta_1(x, \sigma)\} & \text{if } (\sigma \in \Sigma_{p1}) \wedge (|\tilde{x}| > 1) \wedge (\delta_1(x, \sigma)!) \wedge (x \in \tilde{x}) \end{array} \right\} \quad (2)$$

The state transition definition is also extended in the usual manner to allow for strings of transitions labeled by strings of events in  $\Sigma_1^*$  instead of single transitions labeled by a single event.

The intuition behind the definition of  $\tilde{\delta}(\cdot, \cdot)$  is now shown in three parts for the three cases in which this transition operator is defined.

The first case of the  $\tilde{\delta}(\cdot, \cdot)$  definition corresponds to the occurrence of a global event  $\sigma \in \Sigma_g$  such that all elements of  $\tilde{x}$  are updated simultaneously on the occurrence of  $\sigma$ . All elements of  $\tilde{x}$  need to be updated on the occurrence of  $\sigma$  because all states in  $Comp^{-1}(\tilde{x})$  from  $G^{\parallel}$  are updated on the occurrence of  $\sigma$  according to the transition rules of  $\delta^{\parallel}(\cdot, \cdot)$ . Therefore  $\delta_1(x, \sigma)$  is defined for all  $x \in \tilde{x}$ . This implies that  $Comp(\delta^{\parallel}(x^{\parallel}, \sigma)) \in \tilde{\delta}(\tilde{x}, \sigma)$  and in this case  $\{\delta_1(x, \sigma) | x \in \tilde{x}\} \in \tilde{\delta}(\tilde{x}, \sigma)$  if  $\sigma \in \Sigma_g$  and  $\delta_1(x, \sigma)$  is defined for all  $x \in \tilde{x}$ .

The second case of the definition corresponds to the situation where there exists some composed state  $x^{\parallel} \in Comp^{-1}(\tilde{x})$  such that there exists  $i, j \in \{1, \dots, n\}$  such that  $i \neq j$  and  $x^{\parallel^i} = x^{\parallel^j}$ . In other words,  $|\tilde{x}| < n$  as required for this case in the definition of  $\tilde{\delta}(\cdot, \cdot)$ . Therefore, at  $x^{\parallel}$ , if an event  $\sigma_{pi} \in \Sigma_{pi}$  were to occur such that  $\delta_i(x^{\parallel^i}, \sigma_{pi})!$ , then the resulting state  $\delta^{\parallel}(x^{\parallel}, \sigma_{pi})$  exists and

$$Comp(\delta(x^{\parallel}, \sigma_{pi})) = Comp(x^{\parallel}) \cup \{\delta_i(x^{\parallel^i}, \sigma_{pi})\}$$

as a replicated state in  $x^{\parallel}$  would update on the occurrence of  $\sigma_{pi}$ . Therefore,

$$Comp(x^{\parallel}) \cup \{\delta_i(x^{\parallel^i}, \sigma_{pi})\} \in \tilde{\delta}(x^{\parallel}, \Psi_{i1}(\sigma_{pi})),$$

and for this case, there is some  $x \in \tilde{x}$ ,  $\sigma \in \Sigma_{p1}$  such that  $\tilde{x} \cup \{\delta_1(x, \sigma)\} \in \tilde{\delta}(\tilde{x}, \sigma)$  as specified in the definition of  $\tilde{\delta}(\cdot, \cdot)$ .

The third case of the definition corresponds to the situation where there exists some state  $x^{\parallel} \in Comp^{-1}(\tilde{x})$  such that there exists  $i \in \{1, \dots, n\}$  and for any  $j \in (\{1, \dots, n\} \setminus \{i\})$ ,  $x^{\parallel^i} \neq x^{\parallel^j}$ . Thus,  $|\tilde{x}| > 1$ , as required for this case of the definition of  $\tilde{\delta}(\cdot, \cdot)$ . Therefore, at  $x^{\parallel}$ , if an event  $\sigma_{pi} \in \Sigma_{pi}$  were to occur such that  $\delta_i(x^{\parallel^i}, \sigma_{pi})!$ , then the resulting state  $\delta(x^{\parallel}, \sigma_{pi})$  would be in

$$\left( \left[ Comp(x^{\parallel}) \setminus \{x^{\parallel^i}\} \right] \cup \{\delta_i(x^{\parallel^i}, \sigma_{pi})\} \right) \in \tilde{\delta}(Comp(\tilde{x}), \Psi_{i1}(\sigma_{pi})),$$

and for this case there is some  $x \in \tilde{x}$  and  $\sigma_{pi} \in \Sigma_{pi}$  such that

$$[\tilde{x} \setminus x] \cup \{\delta_1(x, \sigma)\} \in \tilde{\delta}(\tilde{x}, \sigma)$$

as specified in the definition of  $\tilde{\delta}(\cdot, \cdot)$ . For this case a module state that is not replicated in  $x^{\parallel}$  is updated on the occurrence of a private event.

Also, the notation is sometimes used that for two states  $\tilde{x}, \tilde{y} \in \tilde{X}$  and an event  $\sigma \in \Sigma_1$ ,  $\tilde{x} \xrightarrow{\sigma}_{\tilde{G}} \tilde{y}$  denotes that there is a transition in  $\tilde{G}$  labeled by  $\sigma$  from  $\tilde{x}$  to  $\tilde{y}$ . Similarly, for two states  $x^{\parallel}, y^{\parallel} \in X^{\parallel}$  and an event  $\sigma \in \Sigma$ ,  $x^{\parallel} \xrightarrow{\sigma}_{G^{\parallel}} y^{\parallel}$  denotes that there is a transition in  $G^{\parallel}$  labeled by  $\sigma$  from  $x^{\parallel}$  to  $y^{\parallel}$ .

The similar module system introduced in Example 4 is now used to demonstrate the construction of a simple reduced state space composed automaton  $\tilde{G}$ .

**Example 5** Consider the similar module system comprised of the automata  $G_1, G_2, G_3$  discussed in Example 4 above. The set of state equivalence classes in the composed automaton  $G_1 \parallel G_2 \parallel G_3$  with respect to the  $\text{Comp}(\cdot)$  operation is  $\{\{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$ . The quotient automaton  $\tilde{G}$  constructed from  $G_1, G_2, G_3$  can be seen in Figure 5.

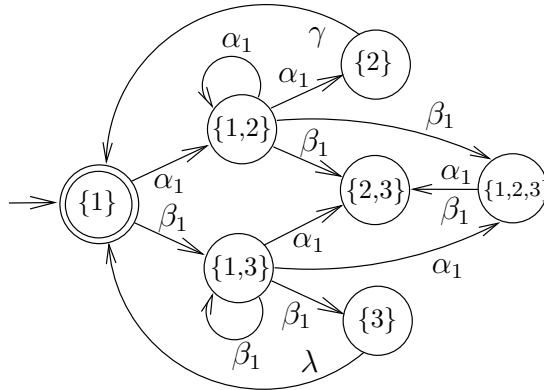


Figure 5: The automaton  $\tilde{G}$  constructed from  $G_1, G_2, G_3$ .

The three different cases for transitions in the definition of  $\tilde{\delta}(\cdot, \cdot)$  are all exhibited in this example of  $\tilde{G}$ .

Consider the  $\{2\} \xrightarrow{\gamma}_{\tilde{G}} \{1\}$  transition in  $\tilde{G}$ . This transition is in the first case of the  $\tilde{\delta}(\cdot, \cdot)$  definition and in the original  $G_1 \parallel G_2 \parallel G_3$  automaton, it corresponds to the  $(2, 2, 2) \xrightarrow{\gamma}_{G_1 \parallel G_2 \parallel G_3} (1, 1, 1)$  transition. That is, a global event  $\gamma$  occurs and all states of  $(2, 2, 2)$  are updated simultaneously.

Now consider the  $\{1, 2\} \xrightarrow{\alpha_1}_{\tilde{G}} \{1, 2\}$  transition in  $\tilde{G}$ . This transition corresponds to several transitions in the  $G_1 \parallel G_2 \parallel G_3$  automaton, including the transition  $(1, 1, 2) \xrightarrow{\alpha_2}_{G_1 \parallel G_2 \parallel G_3} (1, 2, 2)$ . For this transition, module state 1 is replicated and the second module state of  $(1, 1, 2)$  is updated on the occurrence of  $\alpha_2$ , but the first module state remains 1. The  $\{1, 2\} \xrightarrow{\alpha_1}_{\tilde{G}} \{1, 2\}$  transition is in the second case for transitions in the definition of  $\tilde{\delta}(\cdot, \cdot)$ .

The  $\{1, 2\} \xrightarrow{\alpha_1}_{\tilde{G}} \{2\}$  transition corresponds to the third case for in the definition of  $\tilde{\delta}(\cdot, \cdot)$ . In the original  $G_1 \parallel G_2 \parallel G_3$  automaton, this transition corresponds to the  $(2, 2, 1) \xrightarrow{\alpha_3}_{G_1 \parallel G_2 \parallel G_3} (2, 2, 2)$  transition among others. In this case, there is only one module in state 1 and on the occurrence of  $\alpha_3$  this module updates to state 2 and the other modules remains unaltered.

Also notice that  $\tilde{G}$  there is a transition  $\{1\} \xrightarrow{\alpha_1}_{\tilde{G}} \{1, 2\}$ . This corresponds to the second case of transitions in the definition of  $\tilde{\delta}(\cdot, \cdot)$ . It is true that  $\{2, 2, 1\} \in \text{Comp}^{-1}(\{1, 2\})$  but in  $G_1 \parallel G_2 \parallel G_3$  there is no event  $\sigma$  such that there is a transition  $(1, 1, 1) \xrightarrow{\sigma}_{G_1 \parallel G_2 \parallel G_3} (2, 2, 1)$ . Therefore, state reachability in  $\tilde{G}$  does not exactly imply state reachability in  $G_1 \parallel G_2 \parallel G_3$  for the same number of transitions. However, there is a string  $\alpha_1 \alpha_2$  such that  $(1, 1, 1) \xrightarrow{\alpha_1 \alpha_2}_{G_1 \parallel G_2 \parallel G_3} (2, 2, 1)$ . It is shown below that state reachability in  $\tilde{G}$  is shown to imply a form of reachability in  $G_1 \parallel \dots \parallel G_n$  and vice-versa.

## 4.2 Verifying System Properties Using $\tilde{G}$

Fundamental reachability results related to the  $\tilde{G}$  and  $G^\parallel$  constructions for similar module systems are now shown.

**Lemma 4** Suppose a similar module system  $\{G_1, \dots, G_n\}$  is given that is used to construct  $G^\parallel$  and  $\tilde{G}$ . For two states of  $G^\parallel$ ,  $x_1^\parallel, x_2^\parallel \in X^\parallel$  such that  $\text{Comp}(x_1^\parallel) = \tilde{x}_1$  and  $\text{Comp}(x_2^\parallel) = \tilde{x}_2$  and a

string of transitions labeled by  $t^\parallel \in \Sigma^*$  such that  $x_1^\parallel \xrightarrow{t^\parallel}_{G^\parallel} = x_2^\parallel$ , there exists a string of transitions labeled by  $\tilde{t} \in \Sigma_1^*$  such that according to the transition rules of  $\tilde{G}$ ,  $\tilde{x}_1 \xrightarrow{\tilde{t}}_{\tilde{G}} \tilde{x}_2$ .

**Proof:** This lemma is shown using a proof by generalized induction on the length of  $t^\parallel$ . For the base of induction, suppose  $|t^\parallel| = 1$ . The base of induction is shown in three cases according to the three cases in the definition of  $\tilde{\delta}(\cdot, \cdot)$ .

For the first case, suppose that  $t^\parallel = \sigma_g \in \Sigma_g$ . It is assumed that  $\delta^\parallel(x_1^\parallel, \sigma_g) = x_2^\parallel$ , so  $\forall i \in \{1, \dots, n\}$ ,  $\delta_i(x_1^\parallel, \sigma_g) = x_2^\parallel$ . Consequently,  $\forall x \in \tilde{x}_1$

$$\exists i \in \{1, \dots, n\} \text{ such that } \delta_1(x, \sigma_g) = x_2^\parallel \in \tilde{x}_2$$

and  $\forall x \in \tilde{x}_2$

$$\exists i \in \{1, \dots, n\} \text{ such that } x_1^\parallel \in \tilde{x}_1 \text{ and } \delta_1(x_1^\parallel, \sigma_g) = x.$$

Therefore, by the first case of the definition of  $\tilde{\delta}(\cdot, \cdot)$ ,  $\tilde{x}_1 \xrightarrow{\sigma_g}_{\tilde{G}} \tilde{x}_2$ .

For the second case of the base of induction, suppose that  $t^\parallel = \sigma_{pi} \in \Sigma_{pi}$  and  $\exists j \in \{1, \dots, n\}$  such that  $j \neq i$  and  $x_i^\parallel = x_j^\parallel$ . Therefore, when module  $G_i$  is in state  $x_1^\parallel$ , the module can update on the occurrence of  $\sigma_{pi}$  to state  $x_2^\parallel$  and all other modules in  $G^\parallel$  do not update, i.e.,  $\forall k \neq i$ ,  $x_1^\parallel = x_2^\parallel$ .  $\delta^\parallel(x_1^\parallel, \sigma_{pi}) = x_2^\parallel$ , so  $\delta_i(x_1^\parallel, \sigma_{pi}) = x_2^\parallel$  and for  $k \in \{1, \dots, n\}$ ,  $i \neq k$   $x_1^\parallel = x_2^\parallel$ . Therefore, by the second case of the definition of  $\tilde{\delta}(\cdot, \cdot)$ ,  $\tilde{x}_1 \xrightarrow{\sigma_{pi}}_{G^\parallel} \tilde{x}_2$ .

For the third case of the base of induction, suppose that  $t^\parallel = \sigma_{pi} \in \Sigma_{pi}$  and  $\forall j \in \{1, \dots, n\}$  such that  $j \neq i$  and  $x_i^\parallel \neq x_j^\parallel$ . Therefore, when module  $G_i$  is in state  $x_1^\parallel$ , the module can update on the occurrence of  $\sigma_{pi}$  to state  $x_2^\parallel$  and all other modules in  $G^\parallel$  do not update, i.e.,  $\forall k \neq i$ ,  $x_1^\parallel = x_2^\parallel$ .  $\delta^\parallel(x_1^\parallel, \sigma_{pi}) = x_2^\parallel$ , so  $\delta_i(x_1^\parallel, \sigma_{pi}) = x_2^\parallel$  and for  $k \in \{1, \dots, n\}$ ,  $i \neq k$   $x_1^\parallel = x_2^\parallel$ . Therefore, by the second case of the definition of  $\tilde{\delta}(\cdot, \cdot)$ ,  $\tilde{x}_1 \xrightarrow{\sigma_{pi}}_{G^\parallel} \tilde{x}_2$ . This completes the base of induction.

For the induction hypothesis, suppose that there is a string of transitions labeled by  $t^\parallel \in \Sigma^n$  such that  $x_1^\parallel \xrightarrow{t^\parallel}_{G^\parallel} x_2^\parallel$ ,  $Comp(x_1^\parallel) = \tilde{x}_1$  and  $Comp(x_2^\parallel) = \tilde{x}_2$ . Then, there exists a string of transitions labeled by  $\tilde{t} \in \Sigma_1^*$  such that  $\tilde{x}_1 \xrightarrow{\tilde{t}}_{\tilde{G}} \tilde{x}_2$ .

For the induction step, suppose there is a string of transitions labeled by  $t^\parallel \in \Sigma^n$  and a transition labeled by  $\sigma^\parallel \in \Sigma$  such that  $x_1^\parallel \xrightarrow{t^\parallel \sigma^\parallel}_{G^\parallel} = x_3^\parallel$ ,  $Comp(x_1^\parallel) = \tilde{x}_1$  and  $Comp(x_3^\parallel) = \tilde{x}_3$ . Because  $x_1^\parallel \xrightarrow{t^\parallel}_{G^\parallel} = x_3^\parallel$ , there must be a state  $x_2^\parallel$  such that  $x_1^\parallel \xrightarrow{t^\parallel}_{G^\parallel} x_2^\parallel$  and  $x_2^\parallel \xrightarrow{\sigma^\parallel}_{G^\parallel} x_3^\parallel$  where  $Comp(x_2^\parallel) = \tilde{x}_2$ . Because of the induction hypothesis there is a string of transitions labeled by  $\tilde{t} \in \Sigma_1^*$  such that  $\tilde{x}_1 \xrightarrow{\tilde{t}}_{\tilde{G}} \tilde{x}_2$ . By the same argument as used in the proof of the base of induction above, there is a transition labeled by  $\tilde{\sigma} \in \Sigma_1$  such that  $\tilde{x}_2 \xrightarrow{\tilde{\sigma}}_{\tilde{G}} \tilde{x}_3$ . Therefore, there is a string of transitions labeled by  $\tilde{t}\tilde{\sigma} \in \Sigma_1^*$  such that  $\tilde{x}_1 \xrightarrow{\tilde{t}\tilde{\sigma}}_{\tilde{G}} \tilde{x}_3$ . ■

**Lemma 5** Suppose a similar module system  $\{G_1, \dots, G_n\}$  is given that is used to construct  $G^\parallel$  and  $\tilde{G}$  with two states of the quotient automaton  $\tilde{x}_1, \tilde{x}_2 \in 2^X$ , a state  $x_1^\parallel \in X^\parallel$  such that  $Comp(x_1^\parallel) = \tilde{x}_1$  and a string of transitions labeled by  $\tilde{t} \in \Sigma_1^*$  such that  $\tilde{x}_1 \xrightarrow{\tilde{t}}_{\tilde{G}} \tilde{x}_2$ . Then there exists a state  $x_2^\parallel \in X^\parallel$  and a string of transitions labeled by  $t^\parallel \in \Sigma^*$  such that  $Comp(x_2^\parallel) = \tilde{x}_2$  and  $x_1^\parallel \xrightarrow{t^\parallel}_{G^\parallel} x_2^\parallel$ .

**Proof:** The  $\tilde{G}$  automaton can be thought of as a partial simulation of the behavior of the corresponding  $G^{\parallel}$  automaton. For  $\tilde{x}_1 \xrightarrow{\tilde{t}}_{\tilde{G}} \tilde{x}_2$ , suppose that  $\tilde{t}' < \tilde{t}$  and  $\tilde{x}_1 \xrightarrow{\tilde{t}'}_{\tilde{G}} \tilde{x}$ . If  $\sigma_g \in \Sigma_g$  and  $\tilde{t}'\sigma_g < \tilde{t}$ , then for a  $x^{\parallel} \in \text{Comp}^{-1}(\tilde{x})$ , the transition at  $\tilde{x}$  labeled by  $\sigma_g$  corresponds to a transition in  $G^{\parallel}$  where all component states of  $x^{\parallel}$  update simultaneously. If  $\sigma_p \in \Sigma_{p1}$  and  $\tilde{t}'\sigma_p < \tilde{t}$ , then for  $x^{\parallel} \in \text{Comp}^{-1}(\tilde{x})$ , the transition at  $\tilde{x}$  labeled by  $\sigma_p$  corresponds in  $G^{\parallel}$  to exactly one component state  $x^{\parallel i}$  of  $x^{\parallel}$  for some  $i \in \{1, \dots, n\}$  that updates on the occurrence of an event  $\sigma_{pi} = \Psi_{1i}(\sigma_p)$ .

Therefore, if  $\tilde{x}_1 \xrightarrow{\tilde{t}}_{\tilde{G}} \tilde{x}_2$ , then for every component state  $x_1 \in \tilde{x}_1$ , there must be a string of transitions labeled by events in  $\Sigma_1^*$  that leads to a component state  $x_2 \in \tilde{x}_2$  according to the transition rules of  $G_1$  and for every component state  $x_2 \in \tilde{x}_2$ , there must be a string of transitions labeled by events in  $\Sigma_1^*$  from a component state in  $x_1 \in \tilde{x}_1$  to the component state  $x_2$  according to the transition rules of  $G_1$ . All of these strings are able to occur synchronously with respect to the occurrence of global events and are copies of the string  $\tilde{t}$  with some events from  $\Sigma_{p1}$  removed.

With this reasoning the string of events  $\tilde{t} \in \Sigma_1^*$  such that  $\tilde{x}_1 \xrightarrow{\tilde{t}}_{\tilde{G}} \tilde{x}_2$  can be split into a set of strings  $T = \{t_{x_1^1 x_2^1}, \dots, t_{x_1^m x_2^m}\} \subseteq \Sigma_1^*$  such that:

- For all  $i \in \{1, \dots, m\}$ ,  $t_{x_1^i x_2^i}$  is a copy of  $\tilde{t}$  with some  $\Sigma_{p1}$  events removed.
- $P_g(t_{x_1^1 x_2^1}) = \dots = P_g(t_{x_1^m x_2^m})$
- For every component state  $x_1^i \in \tilde{x}_1$ , there is at least one component state  $x_2^i \in \tilde{x}_2$  and a string of events  $t_{x_1^i x_2^i} \in T$  such that  $\delta^1(x_1^i, t_{x_1^i x_2^i}) = x_2^i$ .
- For every component state  $x_2^j \in \tilde{x}_2$ , there is at least one component state  $x_1^j \in \tilde{x}_1$  and a string of events  $t_{x_1^j x_2^j} \in T$  such that  $\delta^1(x_1^j, t_{x_1^j x_2^j}) = x_2^j$ .
- $m = \max\{|\tilde{x}_1|, |\tilde{x}_2|\} \leq n$ .

Consider the ordered list of  $n$  strings

$$L_t = [t_{x_1^1 x_2^1}, \dots, t_{x_1^m x_2^m}, t_{x_1^m x_2^m}, \dots, t_{x_1^m x_2^m}].$$

Corresponding to  $L_t$ , there are two ordered lists of pairs of states in  $X$

$$\begin{aligned} L_x^1 &= [x_1^1, \dots, x_1^m, x_1^m, \dots, x_1^m] \\ L_x^2 &= [x_2^1, \dots, x_2^m, x_2^m, \dots, x_2^m] \end{aligned}$$

such that if

$$\begin{aligned} x_{1\Psi}^{\parallel} &= (x_1^1, \dots, x_1^m, x_1^m, \dots, x_1^m) \\ x_{2\Psi}^{\parallel} &= (x_2^1, \dots, x_2^m, x_2^m, \dots, x_2^m), \end{aligned}$$

then

$$\begin{aligned} \text{Comp}(x_{1\Psi}^{\parallel}) &= \tilde{x}_1 \\ \text{Comp}(x_{2\Psi}^{\parallel}) &= \tilde{x}_2. \end{aligned}$$

Moreover, according to the transition rules of  $G_1$ ,  $\delta^1(x_{k,1}, t_k) = x_{k,2}$  where  $L_t(k) = t_k$  is the  $k$ th string in  $L_t$  and  $L_x^1(k) = x_{k,1}$  and  $L_x^2(k) = x_{k,2}$  are the  $k$ th states in  $L_x^1$  and  $L_x^2$  respectively.

Suppose that  $x_{1\Psi}^{\parallel} = x_1^{\parallel}$ . It is discussed below how this assumption can be done without loss of generality.

The list of strings  $L_t$  are now converted using the  $\Psi_{1i}(\cdot)$  operator to the list

$$L_t^{\Psi} = [\Psi_{11}(t_{x_1x_2}^1), \dots, \Psi_{1m}(t_{x_1x_2}^m), \Psi_{1(m+1)}(t_{x_1x_2}^m), \dots, \Psi_{1n}(t_{x_1x_2}^m)].$$

Because all of the strings in this converted list can still be synchronized on the occurrence of events in  $\Sigma_g$ , the intersection

$$( P_1^{-1}(\Psi_{11}(t_{x_1x_2}^1)) \cap \dots \cap P_m^{-1}(\Psi_{1m}(t_{x_1x_2}^m)) \cap P_{m+1}^{-1}(\Psi_{1(m+1)}(t_{x_1x_2}^m)) \cap \dots \cap P_n^{-1}(\Psi_{1n}(t_{x_1x_2}^m)) )$$

is nonempty. Let  $t_{\Psi}^{\parallel}$  be some string in this intersection. From the construction above for  $L_x^1$  and  $L_x^2$ , if  $x_{1\Psi}^{\parallel}$  is the  $n$ -tuple corresponding to the list of states in  $L_x^1$  and  $x_{2\Psi}^{\parallel}$  is the  $n$ -tuple corresponding to the list of states in  $L_x^2$ , then  $\delta^{\parallel}(x_{1\Psi}^{\parallel}, t_{\Psi}^{\parallel}) = x_{2\Psi}^{\parallel}$ . Let  $x_2^{\parallel}$  be a copy of  $x_{2\Psi}^{\parallel}$ . Therefore, there exists a state  $x_2^{\parallel} \in X^{\parallel}$  and a string of transitions labeled by  $t^{\parallel} \in \Sigma^*$  such that  $Comp(x_2^{\parallel}) = \tilde{x}_2$  and  $x_1^{\parallel} \xrightarrow{t^{\parallel}}_{G^{\parallel}} x_2^{\parallel}$ .

Notice that the lists  $L_t$ ,  $L_x^1$  and  $L_x^2$  could be constructed in an alternative manner by replicating the appropriate elements of these lists in the proper locations to force that  $x_{1\Psi}^{\parallel}$  be a copy of any arbitrary  $x_1^{\parallel}$ . ■

An example is now given to aid in the visualization of the constructions used in the proof of Lemma 5.

**Example 6** *It is now shown how to construct lists  $L_t$ ,  $L_x^1$ ,  $L_x^2$  and the states  $x_{1\Psi}^{\parallel}$ ,  $x_{2\Psi}^{\parallel}$  in Lemma 5 from given  $\tilde{x}_1$ ,  $\tilde{x}_2$ ,  $x_1^{\parallel}$  and  $\tilde{t}$  using the  $\tilde{G}$  automaton in Example 4. Suppose  $\tilde{x}_1 = \{1, 2\}$ ,  $\tilde{x}_2 = \{2, 3\}$ ,  $\tilde{t} = \alpha_1\gamma\beta_1\beta_1\alpha_1$  and  $x_1^{\parallel} = (1, 1, 2)$ .*

*From  $x_1^{\parallel} = (1, 1, 2)$ ,  $L_x^1$  is set to be  $[1, 1, 2]$ . Let  $L_t = [\alpha_1\gamma\alpha_1, \alpha_1\gamma\beta_1, \gamma\beta_1]$  and let  $L_x^2 = [2, 3, 3]$ . Therefore,  $x_{1\Psi}^{\parallel} = (1, 1, 2)$  and  $x_{2\Psi}^{\parallel} = (2, 3, 3)$ . The proper  $L_t$  and  $L_x^2$  can be found mechanically by searching over all possible candidate strings which is guaranteed to be finite.*

*Note that*

$$\begin{aligned} \delta_1(1, \alpha_1\gamma\alpha_1) &= 2, \\ \delta_1(1, \alpha_1\gamma\beta_1) &= 3, \\ \delta_1(2, \gamma\beta_1) &= 3. \end{aligned}$$

*Also note that  $L_t^{\Psi} = [\alpha_1\gamma\alpha_1, \alpha_2\gamma\beta_2, \gamma\beta_3]$  and it is possible for the string  $t^{\parallel}$  corresponding to  $\tilde{t}$  to be  $\alpha_1\alpha_2\gamma\alpha_1\beta_2\beta_3$ . Therefore,  $\delta^{\parallel}((1, 1, 2), \alpha_1\alpha_2\gamma\alpha_1\beta_2\beta_3) = (2, 3, 3)$ .*

The dual of Lemma 5 is now shown where  $x_2^{\parallel}$  is specified instead of  $x_1^{\parallel}$ .

**Lemma 6** *Suppose a similar module system  $\{G_1, \dots, G_n\}$  is given that is used to construct  $G^{\parallel}$  and  $\tilde{G}$  with two states of the quotient automaton  $\tilde{x}_1, \tilde{x}_2 \in 2^X$ , a state  $x_2^{\parallel} \in X^{\parallel}$  such that  $Comp(x_2^{\parallel}) = \tilde{x}_2$  and a string of transitions labeled by  $\tilde{t} \in \Sigma^*$  such that  $\tilde{x}_1 \xrightarrow{\tilde{t}}_{\tilde{G}} \tilde{x}_2$ . Then there exists a state  $x_1^{\parallel} \in X^{\parallel}$  and string of transitions labeled by  $t^{\parallel} \in \Sigma_1^*$  such that  $Comp(x_1^{\parallel}) = \tilde{x}_1$  and  $x_1^{\parallel} \xrightarrow{t^{\parallel}}_{G^{\parallel}} x_2^{\parallel}$ .*

**Proof:** This proof follows from the same construction as in Lemma 5 except that the lists  $L_t$ ,  $L_x^1$  and  $L_x^2$  could be constructed to correspond to any state  $x_2^\parallel$  such that  $x_2^\parallel \in \text{Comp}^{-1}(\tilde{x}_2)$  and  $x_2^\parallel$  is the  $n$ -tuple corresponding to the list of states in  $L_x^2$ . ■

**Theorem 3** *Suppose a similar module system  $\{G_1, \dots, G_n\}$  is given that is used to construct  $G^\parallel$  and  $\tilde{G}$ . A state  $x^\parallel \in X^\parallel$  deadlocks according to the transition rules of  $G^\parallel$  if and only if the state  $\text{Comp}(x^\parallel) = \tilde{x}$  deadlocks according to the transition rules of  $\tilde{G}$ .*

**Proof:** The proof of this theorem is demonstrated in two parts. First, it is shown that if  $x^\parallel \in X^\parallel$  deadlocks according to the transition rules of  $G^\parallel$ , then the state  $\text{Comp}(x^\parallel) = \tilde{x}$  deadlocks according to the transition rules of  $\tilde{G}$ .

If  $x^\parallel \in X^\parallel$  deadlocks according to the transition rules of  $G^\parallel$ , then there is no global event  $\sigma_g \in \Sigma_g$  that could synchronize a state transition in all component states of  $x^\parallel$ . Due to the first case of the state transition function  $\tilde{\delta}(\cdot, \cdot)$ , there is therefore no global event  $\sigma_g \in \Sigma_g$  that could synchronize a state transition in all component states of  $\tilde{x}$ . Also, there are no private events  $\sigma_{pi} \in \Sigma_{pi}$  that could occur at any of the component states of  $x^\parallel$ . There can therefore be no events  $\sigma_{p1} \in \Sigma_{p1}$  that could occur at any of the states of  $\tilde{x}$  according to the transition rules of  $G_1$ . Consequently there are no transitions driven by an event  $\sigma_{p1} \in \Sigma_{p1}$  that could occur at the state  $\tilde{x}$  according to the second and third cases of the transition rules of  $\tilde{G}$ . Therefore, the state  $\text{Comp}(x^\parallel) = \tilde{x}$  deadlocks according to the transition rules of  $\tilde{G}$ .

Now it is shown that if  $x^\parallel \in X^\parallel$  does not deadlock according to the transition rules of  $G^\parallel$ , then the state  $\text{Comp}(x^\parallel) = \tilde{x}$  does not deadlock according to the transition rules of  $\tilde{G}$ . If  $x^\parallel \in X^\parallel$  does not deadlock according to the transition rules of  $G^\parallel$ , then there is a state  $y^\parallel \in X^\parallel$  and an event  $\sigma \in \Sigma$  for a transition in  $G^\parallel$  such that  $x^\parallel \xrightarrow{\sigma}_{G^\parallel} y^\parallel$ . Therefore, by Lemma 4, there exists a string of transitions labeled by  $\tilde{t} \in \Sigma_1^*$  such that according to the transition rules of  $\tilde{G}$ ,  $\text{Comp}(x^\parallel) \xrightarrow{\tilde{t}}_{\tilde{G}} \text{Comp}(y^\parallel)$ . Therefore, the state  $\text{Comp}(x^\parallel) = \tilde{x}$  does not deadlock according to the transition rules of  $\tilde{G}$ . ■

**Theorem 4** *Suppose a similar module system  $\{G_1, \dots, G_n\}$  is given that is used to construct  $G^\parallel$  and  $\tilde{G}$ . A state  $x^\parallel \in X^\parallel$  is reachable according to the transition rules of  $G^\parallel$  if and only if the state  $\text{Comp}(x^\parallel) = \tilde{x}$  is reachable according to the transition rules of  $\tilde{G}$ .*

**Proof:** This theorem is demonstrated in two parts. For the first part of this proof, suppose that a state  $x^\parallel \in X^\parallel$  is reachable according to the transition rules of  $G^\parallel$ . Therefore, there is a string of transitions labeled by  $t^\parallel$  such that  $x_0^\parallel \xrightarrow{t^\parallel}_{G^\parallel} x^\parallel$ . Because of Lemma 4, there is a string of transitions labeled by  $\tilde{t}$  such that  $\tilde{x}_0 \xrightarrow{\tilde{t}}_{\tilde{G}} \tilde{x}$  where  $\text{Comp}(x_0^\parallel) = \tilde{x}_0$ .

For the second part of this proof, suppose that a state  $\tilde{x} \in \tilde{X}$  is reachable according to the transition rules of  $\tilde{G}$ . Therefore, there is a string of transitions labeled by  $\tilde{t}$  such that  $\tilde{x}_0 \xrightarrow{\tilde{t}}_{\tilde{G}} \tilde{x}$ . Using Lemma 6 there is a string of transitions labeled by  $t^\parallel$  such that for some  $x_0^{\parallel'} \in \text{Comp}^{-1}(\tilde{x}_0)$ ,  $x_0^{\parallel'} \xrightarrow{t^\parallel}_{G^\parallel} x^\parallel$ . Because  $\text{Comp}^{-1}(\tilde{x}_0) = \{x_0^{\parallel'}\}$ , there is a string of transitions labeled by  $t^\parallel$  such that  $x_0^{\parallel'} \xrightarrow{t^\parallel}_{G^\parallel} x^\parallel$  where  $\text{Comp}(x_0^{\parallel'}) = \tilde{x}_0$ . ■

**Theorem 5** *Suppose a similar module system  $\{G_1, \dots, G_n\}$  is given that is used to construct  $G^\parallel$  and  $\tilde{G}$ . A state  $x^\parallel \in X^\parallel$  is blocking according to the transition rules of  $G^\parallel$  if and only if the state  $\text{Comp}(x^\parallel) = \tilde{x}$  is blocking according to the transition rules of  $\tilde{G}$ .*

**Proof:** Suppose that the state  $x^\parallel \in X^\parallel$  is nonblocking according to the transition rules of  $G^\parallel$ . Therefore, there is a state  $x_m^\parallel \in X_m^\parallel$  and a string of transitions labeled by  $t^\parallel$  such that  $x^\parallel \xrightarrow{t^\parallel}_{G^\parallel} x_m^\parallel$ . Because of Lemma 4, there is a string of transitions labeled by  $\tilde{t}$  such that  $\tilde{x} \xrightarrow{\tilde{t}}_{\tilde{G}} \tilde{x}_m$  where  $Comp(x_m^\parallel) = \tilde{x}_m$ . Because  $Comp(x_m^\parallel) = \tilde{x}_m$ ,  $\tilde{x}_m$  must be marked.

Suppose that the state  $\tilde{x} \in \tilde{X}$  is nonblocking according to the transition rules of  $\tilde{G}$ . Therefore, there is a state  $\tilde{x}_m \in \tilde{X}_m$  and a string of transitions labeled by  $\tilde{t}$  such that  $\tilde{x} \xrightarrow{\tilde{t}}_{\tilde{G}} \tilde{x}_m$ . By Lemma 5 there is a string of transitions labeled by  $t^\parallel$  and a state  $x_2^\parallel$  such that  $x^\parallel \xrightarrow{t^\parallel}_{G^\parallel} x_2^\parallel$  and  $Comp(x_2^\parallel) = \tilde{x}_m$ . Because  $Comp(x_2^\parallel) = \tilde{x}_m$ , and  $\tilde{x}_m$  is marked, then  $x_2^\parallel$  must also be marked. Therefore  $x^\parallel$  is nonblocking.  $\blacksquare$

### 4.3 The Size of the State Space of $\tilde{G}$

It should be apparent that the  $\tilde{G}$  constructed from the similar module system  $\{G_1, \dots, G_n\}$  has a smaller state space than the  $G^\parallel$  automaton constructed from the same set of modules. The worst-case size of the  $\tilde{G}$  automaton state space is now quantified. Suppose that  $k = |X|$ , the size of the state space of the individual modules. Consider the following example that shows how the  $\tilde{G}$  construction can have a bounded number of states even if the number of modules is unbounded.

**Example 7** Consider the similar module system  $G_1, G_2$  as in Example 1 and the corresponding  $\tilde{G}_2$  constructed from these automata which can be seen in Figure 6. The automata  $G_1, G_2$  use  $X = \{1, 2, 3\}$  as their state space and  $\tilde{G}_2$  uses  $\{\{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}\}$  as its state space.

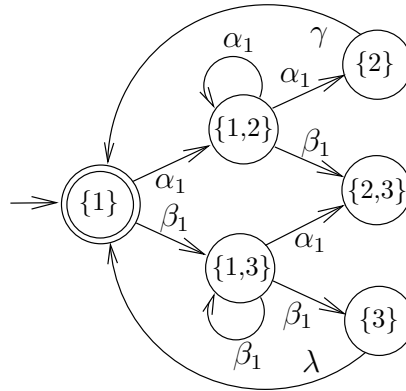


Figure 6: The automaton  $\tilde{G}_2$  constructed from  $G_1, G_2$ .

Now consider the set of similar modules except with three components  $G_1, G_2, G_3$  and corresponding  $\tilde{G}$  automaton discussed in Example 5. The automaton  $\tilde{G}$  can be seen in Figure 5. The automata  $G_1, G_2, G_3$  use  $X = \{1, 2, 3\}$  as their state space and  $\tilde{G}$  for this set of modules is  $2^X = \{\{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$  as its state space. Note that the largest cardinality subset of  $X$  is  $\{1, 2, 3\}$ .

Again consider the same set of similar modules except with four components  $G_1, G_2, G_3, G_4$  and the corresponding quotient automaton  $\tilde{G}_4$  which can be seen in Figure 7. The automata  $G_1, G_2, G_3, G_4$  use  $X = \{1, 2, 3\}$  as their state space and  $\tilde{G}_4$  uses

$$2^X = \{\{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$$

as its state space. The only difference between the  $\tilde{G}_4$  automaton and the  $\tilde{G}$  automaton for the three module system seen in Figure 5 is that there are added self-loop transitions at state  $\{1, 2, 3\}$  labeled by  $\alpha_1$  and  $\beta_1$  events.

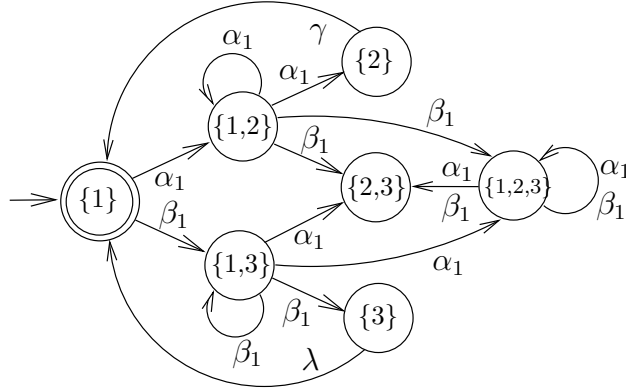


Figure 7: The automaton  $\tilde{G}_4$  constructed from  $G_1, G_2, G_3, G_4$ .

Now consider the same set of  $n$  similar modules  $\{G_1, \dots, G_n\}$  with  $n \geq 4$ . The automaton  $\tilde{G}_n$  constructed from this set of modules still uses  $2^X = \{\{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$  as its state space and is structurally equivalent to the  $\tilde{G}_4$  automaton. That is, all defined transitions in  $\tilde{G}_n$  are also defined in  $\tilde{G}_4$ . Therefore, the  $\tilde{G}_n$  construction is equivalent to the  $\tilde{G}_4$  construction no matter how many modules are used as long as  $n \geq 4$ . Furthermore, the size of the state space of  $\tilde{G}_n$  does not change for  $n \geq 3$ . This is due to the fact that  $|X| = 3$ , and for  $n \geq 3$ ,  $\tilde{X} = 2^X$ .

Given an  $n$ -module system, each state of  $\tilde{G}$  can have at most  $n$  component states and in each state of  $\tilde{G}$  no component states are repeated. Therefore the maximum number of component states in  $\tilde{G}$  is  $\max(n, k)$ . Consequently the worst-case size of  $\tilde{X}$  needs to be analyzed in two cases,  $n \geq k$  and  $n < k$ .

**Case 1** :  $n < k$

Each of the states of  $\tilde{G}$  has at most  $n$  component states. There are possibly  $\binom{k}{i}$  states with  $i$  component states. Therefore if the sum of the possible number of state sets for each number of components is taken from  $i$  to  $n$ , then the number of possible state sets for  $\tilde{G}$  for  $n < k$  is:

$$\sum_{i=1}^n \binom{k}{i}. \quad (3)$$

Note that for  $n < k$ , the number of possible state sets for  $\tilde{G}$  is bounded by  $2^k - 1$ .

**Case 2** :  $n \geq k$

For this case, any state set in the power set  $2^X$  is a possible state of  $\tilde{G}$  except for the empty set  $\emptyset$ . Therefore, the size of the state space of  $\tilde{G}$  is at most

$$|2^X \setminus \{\emptyset\}| = 2^k - 1. \quad (4)$$

This means that if  $n \geq k$  then the size of  $\tilde{G}$  is independent of the number of modules and testing state reachability, blocking and deadlock-freeness can be done without regard to the number of components.

Therefore if  $\tilde{X}_{\tilde{G}}$  is the set of reachable states in  $\tilde{G}$ , then

$$|\tilde{X}_{\tilde{G}}| \leq \begin{cases} \sum_{i=1}^n \binom{k}{i} & \text{if } n < k \\ 2^k - 1 & \text{if } n \geq k \end{cases} \quad (5)$$

Furthermore, no matter how many modules comprise a similar module system,  $|\tilde{X}_{\tilde{G}}| \leq 2^k - 1$ . This is much less than the worst-case number of reachable states in  $G^{\parallel}$ ,  $k^n$ . Therefore, the verification of such important system properties as state reachability and blocking can be performed with a large reduction in the number of composed states that need to be searched over. Furthermore, using the  $\tilde{G}$  construction, the difficulty of verifying these properties is independent of the number of modules if the number of modules is sufficiently large. This construction shows more of the computational advantages gained when investigating modular systems with symmetry between components. Other quotient automaton constructions shown in [6, 7, 27] which use permutation operators to define the state equivalence classes have a worst case size of  $\binom{(k+n-1)!}{(k-1)!n!}$  for testing state reachability properties and the size of this automaton grows as the number of modules grows. Note that with some mathematical manipulation it can be shown that

$$\begin{cases} \sum_{i=1}^n \binom{k}{i} & \text{if } n < k \\ 2^k - 1 & \text{if } n \geq k \end{cases} \leq \binom{(k+n-1)!}{(k-1)!n!}. \quad (6)$$

Therefore,  $|\tilde{X}_{\tilde{G}}| \leq \binom{(k+n-1)!}{(k-1)!n!}$ .

## 5 Verification for Local Specifications

Now topics related to the verification of similar module systems with respect to local specifications are discussed. First, the class of behavior language specifications used are defined.

**Definition 5** *A set of languages  $\{K_1, \dots, K_n\}$  is called a set of similar languages if for all  $i \in \{1, \dots, n\}$ ,  $K_i \subseteq \Sigma_i^*$  and  $\Psi_{1i}(K_1) = K_i$ .*

Suppose it needs to be checked if for all  $i$  that  $P_i(\mathcal{L}_m(G_1 \parallel \dots \parallel G_n)) = K_i$ . This property can be checked solely by verifying that  $\mathcal{L}_m(G_i) = K_i$  for similar module systems. The following corollary is a direct extension of Theorem 1.

**Corollary 1** *Given a local language specification  $K_i$  and similar module system  $\{G_1, \dots, G_n\}$ ,  $P_i(\mathcal{L}_m(G_1 \parallel \dots \parallel G_n)) = K_i$  if and only if  $\mathcal{L}_m(G_i) = K_i$ . Likewise,  $P_i(\mathcal{L}(G_1 \parallel \dots \parallel G_n)) = \overline{K_i}$  if and only if  $\mathcal{L}(G_i) = \overline{K_i}$ .*

Verifying  $\mathcal{L}_m(G_i) = K_i$  and  $\mathcal{L}(G_i) = \overline{K_i}$  are both known to be computationally simple if  $K_i$  is specified by a deterministic automaton and  $G_i$  is likewise deterministic. This greatly simplifies previously known verification methods based on more general modular systems because local behavior in a composed similar module system can be tested by investigating a single module.

## 6 Decomposition Properties

This section presents one of the main results of this paper: a polynomial time decomposition method for composed similar module systems that runs in polynomial time. Sufficient conditions are first shown such that if a language  $K$  is separable, then  $\overline{K}$  is also separable.

**Lemma 7** *Given a language  $K$  and a set of projections  $\{P_1, \dots, P_n\}$ , if  $K$  is separable with respect to  $\{P_1, \dots, P_n\}$  and the set of languages  $\{P_1^{-1}(P_1(K)), \dots, P_n^{-1}(P_n(K))\}$  are non-conflicting, then  $\overline{K}$  is separable with respect to  $\{P_1, \dots, P_n\}$ .*

**Proof:** Using Lemma 1, it is known that

$$\forall i \in \{1, \dots, n\}, P_i^{-1}(P_i(\overline{K})) = \overline{P_i^{-1}(P_i(K))}.$$

Consequently,

$$\cap_{i=1}^n \overline{P_i^{-1}(P_i(K))} = \cap_{i=1}^n P_i^{-1}(P_i(\overline{K})).$$

Due to the assumption that the languages  $\{P_1^{-1}(P_1(K)), \dots, P_n^{-1}(P_n(K))\}$  are non-conflicting,

$$\overline{\cap_{i=1}^n P_i^{-1}(P_i(K))} = \cap_{i=1}^n P_i^{-1}(P_i(\overline{K})).$$

Therefore, due to the separability of  $K$ ,

$$K = \cap_{i=1}^n P_i^{-1}(P_i(K))$$

and

$$\overline{K} = \cap_{i=1}^n P_i^{-1}(P_i(\overline{K})).$$

■

It is now demonstrated how symmetric and separable languages can be modeled using the similar module system framework.

**Theorem 6** *Suppose that a set of global events  $\Sigma_g$  is given with private events  $\{\Sigma_{p1}, \dots, \Sigma_{pn}\}$  that can be mapped to one another using  $\Psi_{ij}(\cdot)$  operations. If an automaton  $H = (X, x_0, \Sigma, \delta, X_m)$  is given that is trim and the language marked by the automaton  $\mathcal{L}_m(H)$  is symmetric with respect to  $\{\Psi_{11}, \dots, \Psi_{1n}\}$  and separable with respect to  $\{P_1, \dots, P_n\}$ , then there exists a similar module system  $\{H_1, \dots, H_n\}$  such that*

$$\mathcal{L}_m(H_1 \parallel \dots \parallel H_n) = \mathcal{L}_m(H)$$

and

$$\forall i \in \{1, \dots, n\} P_i(\mathcal{L}_m(H)) = \mathcal{L}_m(H_i).$$

**Proof:** Using standard methods, given  $\Sigma_i$  and  $H$ , an automaton  $H_i$  can be constructed such that  $\mathcal{L}_m(H_i) = P_i(\mathcal{L}_m(H))$  for all  $i$  such that  $i \in \{1, \dots, n\}$ . Because  $\mathcal{L}_m(H)$  is symmetric with respect to  $\{\Psi_{11}, \dots, \Psi_{1n}\}$ ,

$$\begin{aligned} \forall i, j \in \{1, \dots, n\} \Psi_{ij}(\mathcal{L}_m(H)) &= \mathcal{L}_m(H) \\ \Rightarrow \Psi_{ij}(P_i(\mathcal{L}_m(H))) &= P_j(\mathcal{L}_m(H)). \end{aligned}$$

$$\Rightarrow \Psi_{ij}(\mathcal{L}_m(H_i)) = \mathcal{L}_m(H_j).$$

Therefore, the set of automata  $\{H_1, \dots, H_n\}$  can be constructed such that they are copies of one another with respect to a renaming of their private events.

Because  $\mathcal{L}_m(H)$  is separable with respect to  $\{P_1, \dots, P_n\}$ ,

$$P_1^{-1}(P_1(\mathcal{L}_m(H))) \cap \dots \cap P_n^{-1}(P_n(\mathcal{L}_m(H))) = \mathcal{L}_m(H)$$

$$\Rightarrow P_1^{-1}(\mathcal{L}_m(H_1)) \cap \dots \cap P_n^{-1}(\mathcal{L}_m(H_n)) = \mathcal{L}_m(H)$$

$$\Rightarrow \mathcal{L}_m(H_1 \parallel \dots \parallel H_n) = \mathcal{L}_m(H) \text{ due to properties of the parallel composition operation.}$$

$$\Rightarrow \forall i \in \{1, \dots, n\} P_i(\mathcal{L}_m(H_1 \parallel \dots \parallel H_n)) = P_i(\mathcal{L}_m(H))$$

$$\Rightarrow \forall i \in \{1, \dots, n\} \mathcal{L}_m(H_i) = P_i(\mathcal{L}_m(H)).$$

This last step is by the result of Theorem 1 shown above. ■

In the proof of Theorem 6 any method can be used to construct the similar module system  $\{H_1, \dots, H_n\}$  from  $H$  such that  $\mathcal{L}_m(H_1 \parallel \dots \parallel H_n) = \mathcal{L}_m(H)$ . Note that there might not even be a unique set of automata  $\{H_1, \dots, H_n\}$  such that  $\mathcal{L}_m(H_1 \parallel \dots \parallel H_n) = \mathcal{L}_m(H)$ . However, the standard method of computing a set of automata  $\{H_1, \dots, H_n\}$  such that  $\mathcal{L}_m(H_1 \parallel \dots \parallel H_n) = \mathcal{L}_m(H)$  consists of converting all transitions labeled by events in  $\Sigma \setminus \Sigma_1$  in  $H$  to non-deterministic  $\epsilon$ -transitions. Then, this non-deterministic automaton is converted into a deterministic automaton  $H_1$ . Unfortunately, the determinization algorithm takes time and space exponential in the size of  $H$  in the worst case. The other automata  $\{H_2, \dots, H_n\}$  are copies of  $H_1$  with respect to  $\Psi_{1i}(\cdot)$  event translations.

There has been little discussion in the discrete-event systems literature on ways of more efficiently performing modular decompositions besides ad-hoc methods developed on a case-by-case basis. Developing formal methods for performing this operation is very important for many real-world problems where a large complicated system model would need to be converted into simpler, modular model blocks. This section presents a computationally efficient algorithm for performing this decomposition on similar module systems. As an example of a situation where such a decomposition algorithm would be useful, it can be at times difficult to model the behavior of a biological cell without observing its interactions with other biological cells. Notice that at the proper level of abstraction, a biological cellular network can be thought of as a similar module system. Therefore, if a DES model of a biological cellular network is developed from observations of the behaviors of the cellular network, then the decomposition algorithm would provide an efficient method to develop a model for a cell that comprises the cellular network.

Before the decomposition method is presented, an intuitive introduction to the algorithm's operation is now given. Let  $\{H_1, \dots, H_n\}$  be the desired modules that compose the given symmetric and separable  $H$ , i.e.,  $\mathcal{L}_m(H) = \mathcal{L}_m(H_1 \parallel \dots \parallel H_n)$ . Suppose distributed simulations of the composed behavior of  $\{H_1, \dots, H_n\}$  are run such that the only way for private events to occur would be in strings such as  $\sigma_{p1} \Psi_{12}(\sigma_{p1}) \dots \Psi_{1n}(\sigma_{pn}) = \sigma_{p1} \sigma_{p2} \dots \sigma_{pn}$  with no interleavings of other events. On the occurrence of these  $\sigma_{p1} \sigma_{p2} \dots \sigma_{pn}$  strings the local states  $\{H_1, \dots, H_n\}$  are forced to update with identical state transitions. Because  $\mathcal{L}_m(H) = \mathcal{L}_m(H_1 \parallel \dots \parallel H_n)$ , this lockstep simulation of private behaviors in the global system model can be used to calculate the local system behaviors. Given  $H$ , this automaton can be trimmed in a special manner by only allowing global events and strings of private events  $\sigma_{p1} \sigma_{p2} \dots \sigma_{pn}$  as outlined above to occur. The behavior of this specially trimmed automaton matches the behavior of the lockstep simulation of the  $\{H_1, \dots, H_n\}$  automata. Then, if the event  $\sigma_{p1}$  is substituted for every occurrence of a string  $\sigma_{p1} \sigma_{p2} \dots \sigma_{pn}$ , the behavior of the resulting automaton will match the behavior of the  $H_1$  module.

In the following algorithm, a slightly modified set notation for the state transition function is used, i.e., if  $\delta(x, s) = y$  for state transition function  $\delta(\cdot, \cdot)$ , states  $x, y$  and string  $s$ , then the

notation that  $(x, s, y) \in \delta$  is used. For simplicity it is assumed that the inputted automaton  $H$  is deterministic and trim.

**Algorithm 2** *Decomposition Construction Algorithm.*

*Input:*

$$H = (X^H, x_0^H, \Sigma, \delta^H, X_m^H)$$

$$\Sigma_1, \dots, \Sigma_n$$

(Note that  $\delta^H \subseteq (X^H \times \Sigma \times X^H)$ ).

*Output:*

$$\{H_1, \dots, H_n\} \text{ such that } \forall i \in \{1, \dots, n\},$$

$$H_i = (X^i, x_0^i, \Sigma_i, \delta_i, X_m^i).$$

(Note that  $\delta_i \subseteq (X^i \times \Sigma_i \times X^i)$ ).

*Assumptions:*

$H$  is trim.  
 $S$  is a stack with the normal push and pop operations.

*Initialize:*

$$X := \{x_0^H\};$$

$$x_0 := x_0^H;$$

$$\delta^1 := \emptyset;$$

$$S := [x_0];$$

*Repeat:*

$$\{$$

$$x_s = \text{pop}(S)$$

$$\text{Do the following for all } x_b \in X^H:$$

$$\{$$

$$\text{Do the following for all } \sigma_g \in \Sigma_g \text{ with } (x_s, \sigma_g, x_b) \in \delta^H:$$

$$\{$$

$$\delta^1 := \delta^1 \cup \{(x_s, \sigma_g, x_b)\};$$

$$\text{If } x_b \notin X$$

$$\text{Then}$$

$$\{$$

$$X := X \cup \{x_b\};$$

$$\text{push}(S, x_b);$$

$$\}$$

$$\}$$

$$\}$$

$$\text{Do the following for all } \sigma_{p1} \in \Sigma_{p1}, \dots, \sigma_{pn} \in \Sigma_{pn}$$

$$\text{such that } \sigma_{pi} = \Psi_{1i}(\sigma_{p1}), (x_s, \sigma_{p1}\sigma_{p2}\cdots\sigma_{pn}, x_b) \in \delta^H:$$

$$\{$$

$$\delta^1 := \delta^1 \cup \{(x_s, \sigma_{p1}, x_b)\};$$

$$\text{If } x_b \notin X$$

$$\text{Then}$$

$$\{$$

$$X := X \cup \{x_b\};$$

$$\}$$

$$\}$$

```

        push(S, x_b);
      }
    }
  }
}
Until S is empty;
X_m = X_m^H \cap X;
Construct {H_2, \dots, H_n} from H_1 using \Psi_{1i}(\cdot) operations;
Return {H_1, \dots, H_n}.

```

For Algorithm 2 it may be known that there exists a set of automaton  $\{H'_1, \dots, H'_n\}$  such that  $\mathcal{L}_m(H) = \mathcal{L}_m(H'_1 \parallel \dots \parallel H'_n)$ . However, the automata  $\{H'_1, \dots, H'_n\}$  might not be known explicitly, so Algorithm 2 gives a polynomial time method to compute a similar module system  $\{H_1, \dots, H_n\}$  such that  $\mathcal{L}_m(H) = \mathcal{L}_m(H_1 \parallel \dots \parallel H_n)$  and  $\forall i \in \{1, \dots, n\} P_i(\mathcal{L}_m(H)) = \mathcal{L}_m(H_i)$ . The correctness of the decomposition algorithm is demonstrated in the following theorem.

**Theorem 7** *Suppose there is a trim automaton  $H$  and there exists a set of similar module automata  $\{H'_1, \dots, H'_n\}$  such that  $\mathcal{L}_m(H) = \mathcal{L}_m(H'_1 \parallel \dots \parallel H'_n)$ . Then, if Algorithm 2 is run with  $H$  as input, a similar module system  $\{H_1, \dots, H_n\}$  is returned such that*

$$\mathcal{L}_m(H) = \mathcal{L}_m(H_1 \parallel \dots \parallel H_n)$$

and

$$\forall i \in \{1, \dots, n\} P_i(\mathcal{L}_m(H)) = \mathcal{L}_m(H_i).$$

**Proof:** From  $H$ , a specially trimmed version of  $H$  called  $H^{lock}$  can be constructed by trimming all private event transitions in  $H$  except those that correspond to occurrence of chains of private events

$$\sigma_{p1} \Psi_{12}(\sigma_{p1}) \cdots \Psi_{1n}(\sigma_{p1})$$

for  $\sigma_{p1} \in \Sigma_{p1}$  and no other events are allowed to be interleaved in these strings.

Therefore,

$$\mathcal{L}_m(H^{lock}) = \mathcal{L}_m(H) \cap (\{\sigma_{p1} \Psi_{12}(\sigma_{p1}) \cdots \Psi_{1n}(\sigma_{p1}) \mid \sigma_{p1} \in \Sigma_{p1}\} \cup \Sigma_g)^*.$$

Now consider the composition of the  $\{H'_1, \dots, H'_n\}$  automata where global events are allowed to occur freely and private events are restricted to occur in strings  $\sigma_{p1} \Psi_{12}(\sigma_{p1}) \cdots \Psi_{1n}(\sigma_{p1})$  for  $\sigma_{p1} \in \Sigma_{p1}$  with no other events interleaved as above. This automaton marks the same language as  $H^{lock}$  shown above because  $\mathcal{L}_m(H) = \mathcal{L}_m(H'_1 \parallel \dots \parallel H'_n)$ . Furthermore, due the construction of  $H^{lock}$ , it should be apparent that

$$P_i(\mathcal{L}_m(H^{lock})) = \mathcal{L}_m(H'_i)$$

because the marking behavior of  $H_i$  can be reclaimed from  $H^{lock}$  by replacing the  $\sigma_{p1} \Psi_{12}(\sigma_{p1}) \cdots \Psi_{1n}(\sigma_{p1})$  transitions with a single  $\sigma_{p1}$  transition. Therefore,

$$P_i[\mathcal{L}_m(H) \cap (\{\sigma_{p1} \Psi_{12}(\sigma_{p1}) \cdots \Psi_{1n}(\sigma_{p1}) \mid \sigma_{p1} \in \Sigma_{p1}\} \cup \Sigma_g)^*] = \mathcal{L}_m(H'_i).$$

Algorithm 2 constructs  $H_1$  by restricting the behavior of private events in  $H$  to strings such as  $\sigma_{p1}\Psi_{12}(\sigma_{p1})\cdots\Psi_{1n}(\sigma_{p1})$  and then converts these transition strings to  $\sigma_{p1}$  events. Therefore,

$$\mathcal{L}_m(H_1) = P_i[\mathcal{L}_m(H) \cap (\{\sigma_{p1}\Psi_{12}(\sigma_{p1})\cdots\Psi_{1n}(\sigma_{p1})\mid\sigma_{p1} \in \Sigma_{p1}\} \cup \Sigma_g)^*].$$

Consequently,  $\mathcal{L}_m(H'_1) = \mathcal{L}_m(H_1)$ . For all  $i \in \{1, \dots, n\}$ , the automata  $H'_i$  and  $H_i$  are copies of the  $H'_1$  and  $H_1$  automata with respect to a renaming of transition labels by the  $\Psi_{1i}(\cdot)$  functions. This implies that

$$\bigcap_{i \in \{1, \dots, n\}} \mathcal{L}_m(H'_i) = \bigcap_{i \in \{1, \dots, n\}} \mathcal{L}_m(H_i)$$

Therefore,

$$\mathcal{L}_m(H'_1 \parallel \cdots \parallel H'_n) = \mathcal{L}_m(H_1 \parallel \cdots \parallel H_n)$$

Hence,  $\mathcal{L}_m(H) = \mathcal{L}_m(H_1 \parallel \cdots \parallel H_n)$ .

By construction  $\{H_1, \dots, H_n\}$  is a similar module system, and by Theorem 1 it is known that  $P_i(\mathcal{L}_m(H_1 \parallel \cdots \parallel H_n)) = \mathcal{L}_m(H_i)$ . Therefore, by substitution,

$$\forall i \in \{1, \dots, n\} P_i(\mathcal{L}_m(H)) = \mathcal{L}_m(H_i).$$

■

Note that due to Theorem 7, because  $\mathcal{L}_m(H) = \mathcal{L}_m(H'_1 \parallel \cdots \parallel H'_n)$ ,  $\mathcal{L}_m(H) = \mathcal{L}_m(H_1 \parallel \cdots \parallel H_n)$  and Theorem 1, it is known that  $\mathcal{L}_m(H'_1) = \mathcal{L}_m(H_1)$ . Hence Algorithm 2 gives a method to compute automata  $\{H_1, \dots, H_n\}$  which mark the same languages as the similar module systems  $\{H'_1, \dots, H'_n\}$  which comprise the composed system  $H$  when  $\{H'_1, \dots, H'_n\}$  are known to exist. Corollary 2 also follows immediately from Theorems 7 and 6.

**Corollary 2** *Suppose there is a trim automaton  $H$  such that  $\mathcal{L}_m(H)$  is symmetric with respect to  $\{\Psi_{11}, \dots, \Psi_{1n}\}$  and separable with respect to the set of projections  $\{P_1, \dots, P_n\}$ . Then, if Algorithm 2 is run with  $H$  as input, a similar module system  $\{H_1, \dots, H_n\}$  is returned such that*

$$\mathcal{L}_m(H) = \mathcal{L}_m(H_1 \parallel \cdots \parallel H_n)$$

and

$$\forall i \in \{1, \dots, n\} P_i(\mathcal{L}_m(H)) = \mathcal{L}_m(H_i).$$

The main *Repeat – Until* loop in Algorithm 2 iterates  $|X|$  times, which is the size of the state space of  $H_1$ . The size of the state space of  $H_1$  is always at least as small as  $|X^H|$ , the size of the state space of  $H$ , and generally much more so. Inside the main iterative loop, there are two types of tests for transition existence, one for individual global events and one for chains of private events. The tests for transition existence are generally efficient and negligible depending on the encoding of  $H$ , especially if each state contains a list of its outgoing transitions. In this case, only the existence of  $|\Sigma_g|$  global event transitions and  $|\Sigma_{p1}|$  private event string transitions at each state need to be tested. If there is a  $\sigma_{p1} \in \Sigma_{p1}$  transition at the current state, it needs to be tested if there is a chain of transitions  $\sigma_{p1}\sigma_{p2}\dots\sigma_{pn}$  from the current state. Therefore, for every private event transition detected, at most  $n - 1$  other transitions need to be tested. Therefore, Algorithm 2 is in  $O(|X| * (|\Sigma_g| + n * |\Sigma_{p1}|))$ .

An example of a run of Algorithm 2 is now given using a trimmed version of the automaton  $G_1 \parallel G_2$  from Example 1.

**Example 8** Consider the automaton  $G$  that is the trimmed version of  $G_1 \parallel G_2$  from Example 1 with the states renamed for convenience. This automaton can be seen in Figure 3. Recall that  $\mathcal{L}_m(G) = ((\alpha_1\alpha_2 + \alpha_2\alpha_1)\gamma + (\beta_1\beta_2 + \beta_2\beta_1)\lambda)^*$ .

Algorithm 2 is now run with  $G$  as input for  $\Sigma_1 = \{\alpha_1, \beta_1, \gamma, \lambda\}$  and  $\Sigma_2 = \{\alpha_2, \beta_2, \gamma, \lambda\}$ .

To initialize:  $X := \{1\}$ ,  $x_0 := 1$ ,  $\delta^1 := \emptyset$ ,  $S := [1]$ .

The first state is popped off of  $S$ .  $x_s := 1$ .

There are no  $\Sigma_g$  transitions from state 1, but there are two private event chains,  $\alpha_1\alpha_2$  and  $\beta_1\beta_2$  starting from state 1 and respectively leading to states 4 and 7. Therefore, states 4, 7 are added to  $X$ ,  $(1, \alpha_1, 4)$  and  $(1, \beta_1, 7)$  are added to  $\delta^1$  and 4 and 7 are pushed onto  $S$ .

Now,  $X = \{1, 4, 7\}$   $\delta^1 = \{(1, \alpha_1, 4), (1, \beta_1, 7)\}$ ,  $S = [4, 7]$ .

Next, 4 is popped off  $S$  and  $x_s := 4$ . There is a  $\Sigma_g$  transition from 4  $(4, \gamma, 1)$ , but no private event chains. Therefore,  $(4, \gamma, 1)$  is added to  $\delta^1$  and no other changes are made.

Next, 7 is popped off  $S$  and  $x_s := 7$ . There is a  $\Sigma_g$  transition from 7  $(7, \lambda, 1)$ , but no private event chains. Therefore,  $(7, \lambda, 1)$  is added to  $\delta^1$  and no other changes are made. The stack  $S$  is empty, so the Repeat – Until loop has been completed.

The marked state list is now assigned  $X_m = \{1\}$  and this completes the construction of  $G_1$ .  $G_1$  is then copied as  $G_2$  by replacing  $\alpha_1$  with  $\alpha_2$  and  $\beta_1$  with  $\beta_2$ .

This results in the automata seen in Figure 8.

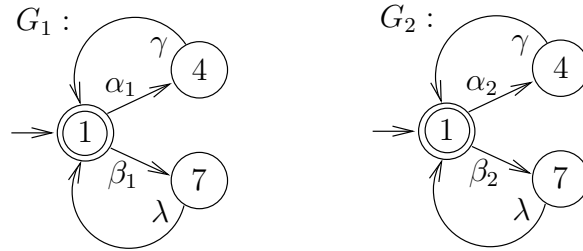


Figure 8: The automata  $G_1$  and  $G_2$  decomposed from  $G$  in Figure 3.

It is now shown that the  $\{H_1, \dots, H_n\}$  automata returned by Algorithm 2 are guaranteed to be trim.

**Theorem 8** Suppose a trim automaton  $H$  is given such that  $\mathcal{L}_m(H)$  is symmetric with respect to  $\{\Psi_{11}, \dots, \Psi_{1n}\}$  and separable with respect to the set of projections  $\{P_1, \dots, P_n\}$ . Let  $\{H_1, \dots, H_n\}$  be the automata constructed from  $H$  using Algorithm 2. Then, the set of automata  $\{H_1, \dots, H_n\}$  are all trim.

**Proof:** It suffices to show that  $H_1$  is trim.

It is known that all states  $X$  in  $H_1$  are reachable from the initial state, so to show that  $H_1$  is trim, it must be demonstrated that for every unmarked state in  $H_1$ , there must be a string to a marked state.

Suppose  $x \in X \setminus X_m$ . Because  $H$  is trim, there must be two strings  $s, t \in \Sigma$  such that  $s$  is in

$$(\{\sigma_{p1}\Psi_{12}(\sigma_{p1}) \cdots \Psi_{1n}(\sigma_{p1}) | \sigma_{p1} \in \Sigma_{p1}\} \cup \Sigma_g)^*$$

$\delta^H(x_0^H, s) = x$  and  $\delta^H(x, t) \in X_m^H$ . Define  $s_1, t_1$  such that  $s_1 = P_1(s)$  and  $t_1 = P_1(t)$ . Because of the construction in Algorithm 2, it must be true that  $\delta^1(x_0, s_1) = x$ . Furthermore, because  $\delta^H(x, t) \in X_m^H$  and  $t_1$  is the string of events that is relevant to  $H_1$  when  $t$  occurs, then it must be true that  $\delta^1(x, t_1) \in X_m$  because  $\mathcal{L}_m(H_1 \parallel \cdots \parallel H_n) = \mathcal{L}_m(H)$ . ■

## 7 Control Operations

Now that verification methods for similar module systems have been discussed, properties related to the control of these systems are explored. The control framework for these systems is first introduced. It is a version of the standard decentralized supervisory control framework specialized for similar module systems.

Given a supervisor  $S_1$  and a system  $G_1$ , the composed system of  $S_1$  supervising  $G_1$  is denoted as the supervised system  $S_1/G_1$ . Furthermore, because the parallel supervisors are assumed to be realized as finite-state automata,  $S_1/G_1$  is equivalent to  $S_1\|G_1$ . Supervisor  $S_1$  is said to be nonblocking for system  $G_1$  if  $S_1/G_1$  is nonblocking, i.e., if  $\overline{\mathcal{L}_m(S_1/G_1)} = \mathcal{L}(S_1/G_1)$ .

Controller  $S_i$  can observe a subset of system events  $\Sigma_{oi} \subseteq \Sigma_i$ . Furthermore, on the occurrence of observable events, Controller  $S_i$  may be given sufficient actuation to selectively disable the locally controllable events  $\Sigma_{ci} \subseteq \Sigma_i$ . Controllers should not be able to disable uncontrollable events and control actions should not update on the occurrence of unobservable events. Furthermore, let  $\Sigma_{uci} = \Sigma_i \setminus \Sigma_{ci}$ . Let  $P_{oi} : \Sigma^* \rightarrow \Sigma_{oi}^*$  be the natural projection that erases events in  $\Sigma \setminus \Sigma_{oi}$  and represents the projection operation for the local observations of controller  $S_i$ .

For the control systems model, it is assumed that the controller automata  $\{S_2, \dots, S_n\}$  are copies of the generic control module  $S_1$  with the local controller's events replaced according to the  $\Psi_{1i}(\cdot)$  mapping. The set of controllers  $\{S_1, \dots, S_n\}$  defined in this way is called a set of *Similar Controllers*.

**Definition 6** *A set of controllers  $\{S_1, \dots, S_n\}$  is called a set of Similar Controllers if all controllers are exact copies of one another with respect to  $\Psi_{ij}(\cdot)$  operations. In particular,  $\Sigma_{ci} = \Psi_{1i}(\Sigma_{c1})$ ,  $\Sigma_{oi} = \Psi_{1i}(\Sigma_{o1})$  and if controller  $S_1$  disables events  $\gamma_1 \subseteq \Sigma_{c1}$  after observing a string  $s_1$ , then controller  $S_i$  disables  $\Psi_{1i}(\gamma_1) \subseteq \Psi_{1i}(\Sigma_{c1})$  after observing string  $\Psi_{1i}(s_1)$ .*

Note that due to Definition 6, if a global event  $\sigma \in \Sigma_g$  is controllable (observable) by  $S_i$ , then it is controllable (observable) by all other controllers.

The controllers  $\{S_1, \dots, S_n\}$  are non-blocking for  $\{G_1, \dots, G_n\}$  if

$$\overline{\mathcal{L}_m((S_1/G_1)\|\dots\|(S_n/G_n))} = \mathcal{L}((S_1/G_1)\|\dots\|(S_n/G_n)).$$

## 8 Control for Local Specifications

There are potentially great reductions in computational effort for many similar module system control problems with local specifications. For a similar module system  $\{G_1, \dots, G_n\}$  and a set of similar language specifications  $\{K_1, \dots, K_n\}$ , suppose it is desirable to know if there exists a set of similar controllers  $\{S_1, \dots, S_n\}$  such that

$$\forall i \in \{1, \dots, n\} \left( \begin{array}{l} P_i [\mathcal{L}_m((S_1/G_1)\|\dots\|(S_n/G_n))] = K_i \\ \wedge P_i [\mathcal{L}((S_1/G_1)\|\dots\|(S_n/G_n))] = \overline{K_i}. \end{array} \right)$$

This problem can be solved by looking only at the local behavior of  $G_1$  and the locally observable and controllable event sets,  $\Sigma_{o1}$  and  $\Sigma_{c1}$ , respectively.

**Theorem 9** *For a similar module system  $\{G_1, \dots, G_n\}$  with local event sets  $\{\Sigma_1, \dots, \Sigma_n\}$ , observable event sets  $\{\Sigma_{o1}, \dots, \Sigma_{on}\}$ , controllable event sets  $\{\Sigma_{c1}, \dots, \Sigma_{cn}\}$  and local behavior specifications  $\{K_1, \dots, K_n\}$  such that for all  $i, j \in \{1, \dots, n\}$ ,  $K_i \neq \emptyset$  and  $K_i \subseteq \mathcal{L}_m(G_i)$ ,  $K_j = \Psi_{ij}(K_i)$ , there exists a set of similar controllers  $\{S_1, \dots, S_n\}$  such that for all  $i \in \{1, \dots, n\}$ ,*

$$P_i [\mathcal{L}_m((S_1/G_1)\|\dots\|(S_n/G_n))] = K_i$$

and

$$P_i [\mathcal{L}((S_1/G_1) \parallel \cdots \parallel (S_n/G_n))] = \overline{K_i}$$

if and only if

1.  $K_1$  is controllable with respect to  $\mathcal{L}(G_1)$  and  $\Sigma_{uc1}$ .
2.  $K_1$  is observable with respect to  $\mathcal{L}(G_1)$ ,  $P_{o1}$  and  $\Sigma_{c1}$ .
3.  $K_1$  is  $\mathcal{L}_m(G_1)$ -closed.

**Proof:** It was shown in Corollary 1 that

$$P_i [\mathcal{L}_m((S_1/G_1) \parallel \cdots \parallel (S_n/G_n))] = K_i$$

and

$$P_i [\mathcal{L}((S_1/G_1) \parallel \cdots \parallel (S_n/G_n))] = \overline{K_i}$$

if and only if  $\mathcal{L}_m(S_i/G_i) = K_i$  and  $\mathcal{L}(S_i/G_i) = \overline{K_i}$ . Due to the similar behavior of the systems, controllers and specifications,  $\mathcal{L}_m(S_i/G_i) = K_i$  and  $\mathcal{L}(S_i/G_i) = \overline{K_i}$  if and only if  $\mathcal{L}_m(S_1/G_1) = K_1$  and  $\mathcal{L}(S_1/G_1) = \overline{K_1}$ . The result then follows using the controllability and observability theorem of supervisory control theory [18].  $\blacksquare$

Theorem 9 shows that controller existence for local behavior specifications can be decided by testing controller existence locally and apart from the interaction of other modules. The controllability and observability theorem is known to be constructive, so therefore there are known methods for synthesizing the local controllers  $\{S_1, \dots, S_n\}$  such that local non-blocking specifications are satisfied when they interact and the conditions of Theorem 9 are satisfied. These results also generalize to cases where marking properties are not of a concern.

Despite these very positive results a caveat is in order with respect to the nature of language semantics. All local modules in a similar module system may be non-blocking and deadlock free, but blocking and deadlock may both still occur globally. Consider the following example.

**Example 9** Consider the automaton  $G_1 \parallel G_2$  introduced in Example 1. The composed automaton  $G_1 \parallel G_2$  is blocking and contains two deadlock states,  $(2, 3)$  and  $(3, 2)$ . These states are reached when an  $\alpha$  event is followed immediately by a  $\beta$  event or when a  $\beta$  event is followed immediately by an  $\alpha$  event. Neither  $G_1$  or  $G_2$  can observe the interleaving of  $\alpha$  and  $\beta$  events due to the restriction that the local systems cannot observe events that are private to other modules. However, when the language generated by the system  $G_1 \parallel G_2$  is projected down to either the  $\Sigma_1$  or  $\Sigma_2$  event sets, the behavior is equivalent to  $G_1$  or  $G_2$ , respectively, and these automata are individually non-blocking and deadlock-free.

Example 9 shows one of the major limitations of using local language specifications for similar module systems and motivates why attention should not be restricted solely to local behavior. Suppose a set of trim similar specification automata  $\{H_1, \dots, H_n\}$  is given such that  $\forall i \in \{1, \dots, n\} \mathcal{L}_m(H_i) = K_i$ . One way to ensure that the global composed system is non-blocking after the local controllers are designed would be to verify that  $\{K_1, \dots, K_n\}$  are non-conflicting by testing if  $H^\parallel$  is non-blocking. This can be verified fairly efficiently by testing if the  $\tilde{H}$  construction is non-blocking as seen in Theorem 5. For concurrent systems blocking and deadlock properties are inherently global in nature and cannot generally be investigated on a local level. Another approach to ensure that the global controlled system is non-blocking would be to use global control specifications instead of local specifications. This situation is explored in the following section.

## 9 Control for Global Specifications

Instead of considering a set of local similar specifications  $\{K_1, \dots, K_n\}$ , suppose a global language specification  $K$  is given and it needs to be decided if there exist non-blocking similar controllers  $\{S_1, \dots, S_n\}$  for a similar module system  $\{G_1, \dots, G_n\}$  such that  $\mathcal{L}_m((S_1/G_1) \parallel \dots \parallel (S_n/G_n)) = K$ . Due to the similarity of the controllers and system modules one would think that  $K$  would need to exhibit a degree of symmetry with respect to the occurrence of private events. This is exactly the case which is found in Theorem 10 below.

**Theorem 10** *For a similar module system  $\{G_1, \dots, G_n\}$  with respective local projection operations  $\{P_1, \dots, P_n\}$ , observation projections  $\{P_{o1}, \dots, P_{on}\}$ , controllable event sets  $\{\Sigma_{c1}, \dots, \Sigma_{cn}\}$  and global behavior specification  $K$  such that  $K \neq \emptyset$  and  $K \subseteq \mathcal{L}_m(G_1 \parallel \dots \parallel G_n)$ , a set of non-blocking similar controllers  $\{S_1, \dots, S_n\}$  exists such that  $\mathcal{L}_m((S_1/G_1) \parallel \dots \parallel (S_n/G_n)) = K$  if and only if*

1.  $P_1(K)$  is controllable with respect to  $\mathcal{L}(G_1)$  and  $\Sigma_{uc1}$ .
2.  $P_1(K)$  is observable with respect to  $\mathcal{L}(G_1)$ ,  $P_{o1}$  and  $\Sigma_{c1}$ .
3.  $P_1(K)$  is  $\mathcal{L}_m(G_1)$ -closed.
4.  $K$  is symmetric with respect to  $\{\Psi_{11}, \dots, \Psi_{1n}\}$ .
5.  $K$  is separable with respect to  $\{P_1, \dots, P_n\}$ .
6.  $\{P_1^{-1}(P_1(K)), \dots, P_n^{-1}(P_n(K))\}$  are non-conflicting.

**Proof:** This theorem is shown in two parts. Assume there exists a set of controllers  $\{S_1, \dots, S_n\}$  such that

$$K = \mathcal{L}_m((S_1/G_1) \parallel \dots \parallel (S_i/G_i) \parallel \dots \parallel (S_n/G_n)) \quad (7)$$

$$\bar{K} = \mathcal{L}((S_1/G_1) \parallel \dots \parallel (S_i/G_i) \parallel \dots \parallel (S_n/G_n)). \quad (8)$$

By the definition of  $\Psi_{1i}(\cdot)$ :

$$\begin{aligned} \Psi_{1i}(\mathcal{L}_m((S_1/G_1) \parallel \dots \parallel (S_i/G_i) \parallel \dots \parallel (S_n/G_n))) &= \\ \mathcal{L}_m((S_i/G_i) \parallel \dots \parallel (S_1/G_1) \parallel \dots \parallel (S_n/G_n)). \end{aligned}$$

The parallel composition operation is commutative, so:

$$\begin{aligned} \mathcal{L}_m((S_1/G_1) \parallel \dots \parallel (S_i/G_i) \parallel \dots \parallel (S_n/G_n)) &= \\ \mathcal{L}_m((S_i/G_i) \parallel \dots \parallel (S_1/G_1) \parallel \dots \parallel (S_n/G_n)). \end{aligned}$$

$\Rightarrow K = \Psi_{1i}(K)$ . This proves the fourth part of the implication.

By Equations 7 and 8 there exists a set of controllers  $\{S_1, \dots, S_n\}$  such that

$$\begin{aligned} K &= \mathcal{L}_m((S_1/G_1) \parallel \dots \parallel (S_i/G_i) \parallel \dots \parallel (S_n/G_n)) \\ \bar{K} &= \mathcal{L}((S_1/G_1) \parallel \dots \parallel (S_i/G_i) \parallel \dots \parallel (S_n/G_n)) \\ &\Rightarrow \end{aligned}$$

there exists a set of controllers  $\{S_1, \dots, S_n\}$  such that  $\forall i \in \{1, \dots, n\}$

$$\begin{aligned} P_i(K) &= P_i(\mathcal{L}_m((S_1/G_1) \parallel \dots \parallel (S_n/G_n))) \\ P_i(\bar{K}) &= P_i(\mathcal{L}((S_1/G_1) \parallel \dots \parallel (S_n/G_n))) \end{aligned}$$

$\Rightarrow$  using Corollary 1:

there exists a set of controllers  $\{S_1, \dots, S_n\}$  such that  $\forall i \in \{1, \dots, n\}$

$$\begin{aligned} P_i(K) &= \mathcal{L}_m(S_i/G_i) \\ P_i(\bar{K}) &= \mathcal{L}(S_i/G_i) \end{aligned}$$

$\Rightarrow$  from the controllability and observability theorem of supervisory control theory,

1.  $P_i(K)$  is controllable with respect to  $\mathcal{L}(G_i)$  and  $\Sigma_{uci}$ .
2.  $P_i(K)$  is observable with respect to  $\mathcal{L}(G_i)$ ,  $P_{oi}$  and  $\Sigma_{ci}$ .
3.  $P_i(K)$  is  $\mathcal{L}_m(G_i)$ -closed.

This proves the first three parts of the implication.

It is known that:

$$K = P_1^{-1}(\mathcal{L}_m(S_1/G_1)) \cap \dots \cap P_n^{-1}(\mathcal{L}_m(S_n/G_n))$$

$$\overline{K} = P_1^{-1}(\mathcal{L}(S_1/G_1)) \cap \dots \cap P_n^{-1}(\mathcal{L}(S_n/G_n))$$

$\Rightarrow$  using Corollary 1:

$$K = P_1^{-1}(P_1(K)) \cap \dots \cap P_n^{-1}(P_n(K)),$$

$$\overline{K} = P_1^{-1}(P_1(K)) \cap \dots \cap P_n^{-1}(P_n(K))$$

$\Rightarrow$  using Lemma 1:

$$K = \overline{P_1^{-1}(P_1(K))} \cap \dots \cap \overline{P_n^{-1}(P_n(K))},$$

$$\overline{K} = P_1^{-1}(P_1(K)) \cap \dots \cap P_n^{-1}(P_n(K))$$

This proves the fifth and sixth parts of the implication. This completes one direction of the proof.

It is now shown that the controllers exist if the six conditions hold.

Because of the first three conditions it is known that there exists a controller  $S_1$  such that  $P_1(K) = \mathcal{L}_m(S_1/G_1)$  and  $\overline{P_1(K)} = \mathcal{L}(S_1/G_1)$ .

Because of the separability condition,  $K = P_1^{-1}(P_1(K)) \cap \dots \cap P_n^{-1}(P_n(K))$ .

Because of the symmetry condition and Lemma 2 it is known that there exists controllers  $S_1, \dots, S_n$  such that  $\forall i \in \{1, \dots, n\} (P_i(K) = \mathcal{L}_m(S_i/G_i))$ .

Therefore by substitution,

$$K = P_1^{-1}(\mathcal{L}_m(S_1/G_1)) \cap \dots \cap P_n^{-1}(\mathcal{L}_m(S_n/G_n)).$$

This implies that

$$K = \mathcal{L}_m((S_1/G_1) \parallel \dots \parallel (S_n/G_n)).$$

Because of Condition 6 and Lemma 1:

$$P_1^{-1}(P_1(K)) \cap \dots \cap P_n^{-1}(P_n(K)) = P_1^{-1}(\overline{P_1(K)}) \cap \dots \cap P_n^{-1}(\overline{P_n(K)}).$$

$\Rightarrow$  Because of Conditions 5 and 6,  $\overline{K}$  is separable and:

$$\overline{K} = P_1^{-1}(\overline{P_1(K)}) \cap \dots \cap P_n^{-1}(\overline{P_n(K)}).$$

Because of  $\overline{P_1(K)} = \mathcal{L}(S_1/G_1)$ , the symmetry condition and Lemma 2 it is known that there exists controllers  $S_1, \dots, S_n$  such that  $\forall i \in \{1, \dots, n\}: (\overline{P_i(K)} = \mathcal{L}(S_i/G_i))$ .

Therefore, for the set of similar controllers  $\{S_1, \dots, S_n\}$ ,

$$\overline{K} = P_1^{-1}(\mathcal{L}(S_1/G_1)) \cap \dots \cap P_n^{-1}(\mathcal{L}(S_n/G_n))$$

$\Rightarrow$

$$\overline{K} = \mathcal{L}((S_1/G_1) \parallel \dots \parallel (S_n/G_n)).$$

Overall it has been shown that there exists a set of similar controllers  $\{S_1, \dots, S_n\}$  such that

$$K = \mathcal{L}_m((S_1/G_1) \parallel \dots \parallel (S_n/G_n))$$

$$\overline{K} = \mathcal{L}((S_1/G_1) \parallel \dots \parallel (S_n/G_n)). \quad \blacksquare$$

The six necessary and sufficient conditions for controller existence in Theorem 10 can be divided into two types. The first three conditions (local controllability, observability and  $\mathcal{L}_m$ -closure) are essentially existence conditions for local controllers to achieve local projections of global behavior. These conditions are inherent to many supervisory control problems and have been well known from the early papers in supervisory control theory such as [18, 24]. However, the combination of the last three conditions (symmetry, separability and non-conflictingness) is unique to this problem setting.

The separability condition ensures that when the specification  $K$  is decomposed into sets of desired local behaviors, if automata that mark the desired local behavior are composed, then the behavior of the composed automaton is equal to  $K$ . That is, the composition of the automata that mark the  $\{P_1(K), \dots, P_n(K)\}$  (i.e.,  $\{S_1/G_1, \dots, S_n/G_n\}$ ) is equal to  $K$ . The non-conflicting condition ensures that if the local automata  $(\{S_1/G_1, \dots, S_n/G_n\})$  are all non-blocking, then the composed automaton  $(S_1/G_1 \parallel \dots \parallel S_n/G_n)$  is nonblocking.

The symmetry condition ensures that if  $K$  is decomposed with respect to  $\{P_1, \dots, P_n\}$ , then the set of modules that mark  $\{P_1(K), \dots, P_n(K)\}$  is a similar module system. This condition in hindsight should be expected when one considers Theorem 6. If the behavior of all controllers operating on the modules is similar with respect to a renaming of local events, then the specification would necessarily be symmetric if it can be achieved.

Taken together, if Conditions 1 through 3 are satisfied, then Conditions 4 through 6 imply that the global specification  $K$  needs to be expressible as a set of similar non-conflicting local specifications  $\{P_1(K), \dots, P_n(K)\}$  if there exists a set of similar controllers  $\{S_1, \dots, S_n\}$  that can be coupled with the similar module system  $\{G_1, \dots, G_n\}$  to achieve the specification  $K$ .

Suppose a trim automaton  $H$  is given such that  $\mathcal{L}_m(H) = K$  where  $K$  satisfies the conditions of Theorem 10. Using the decomposition algorithm, Algorithm 2, a set of trim automata  $\{H_1, \dots, H_n\}$  can be constructed from  $H$  such that  $\forall i \in \{1, \dots, n\} \mathcal{L}_m(H_i) = K_i$  and  $P_1^{-1}(K_1) \cap \dots \cap P_n^{-1}(K_n) = K$  from Theorems 7 and 8. The local similar controllers  $\{S_1, \dots, S_n\}$  can then be synthesized using known centralized control methods.

Note that Conditions 1 through 6 together imply that the language  $K$  is controllable with respect to  $\mathcal{L}(G_1 \parallel \dots \parallel G_n)$ , co-observable with respect to  $\mathcal{L}(G_1 \parallel \dots \parallel G_n)$  and  $\mathcal{L}_m(G_1 \parallel \dots \parallel G_n)$ -closed, but the reverse implication does not hold because of the assumptions on the controllers being used. If the controllers were allowed to be asymmetric, a larger class of specifications could be achieved, but this would require an extra amount of coordination in the control synthesis. The considered framework is designed so that once one control module is designed, the implementation of more controllers is merely a matter of copying that first controller.

## 10 Discussion

A model for a similar module system was given in this paper that can be used to model a wide variety of real-world processes. In addition, a method has been shown that can be used to test global blocking and deadlock-freeness without enumerating all possible combinations of system states and it has been shown that verification of local behavior can be performed in an off-line manner without module interaction. Necessary and sufficient conditions for the existence of local controllers have also been introduced for the similar module system model.

A method has also been shown for decomposing an automaton representing global system behavior into automata representing the local, modular subsystems; this method runs in polynomial time with respect to the size of the local specification automaton. This result is important because the standard methods for performing decompositions currently known in the supervisory control literature take exponential time with respect to the size of the global model in the worst case.

It would be interesting to develop ad-hoc methods for controlling similar module systems when the necessary and sufficient conditions for controller existence do not hold. It might be desired to synthesize controllers that achieve behavior that is “maximal” in some sense when the behavioral specifications cannot be matched exactly. That is, to find a set of similar controllers

$\{S'_1, \dots, S'_n\}$  for a similar module system  $\{G_1, \dots, G_n\}$  such that for any other set of similar controllers  $\{S_1, \dots, S_n\}$ ,  $\mathcal{L}(S_1/G_1 \parallel \dots \parallel S_n/G_n) \subseteq \mathcal{L}(S'_1/G_1 \parallel \dots \parallel S'_n/G_n)$ .

The case where the control modules are similar has solely been investigated in this paper, but it might be the case in many real-world problems that the control systems may be asymmetric. For instance, one controller may be a “leader” that has more leeway in enforcing global control actions to avoid situations where blocking and deadlock may occur. It would also be interesting to investigate more general system models. For instance, it is possible that using a method besides parallel composition to model module interaction might lead to other results. Furthermore, it would be interesting to investigate more general system models where the modules might have some sort of similarity besides being isomorphic with respect to a renaming of private events.

## Acknowledgment

The authors would like to thank the associate editor and the reviewers for their useful comments.

## References

- [1] B. Alberts, D. Bray, J. Lewis, M. Raff, K. Roberts, and J.D. Watson. *Molecular Biology of the Cell*. Garland Publishing, Inc., New York, 1994.
- [2] C.G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, Boston, MA, 1999.
- [3] Y.L. Chen, S. Lafortune, and F. Lin. Design of nonblocking modular supervisors using event priority functions. *IEEE Trans. Auto. Contr.*, 45(3):432–452, March 2000.
- [4] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. The MIT Press, Cambridge, MA, 2002.
- [5] E.M. Clarke, S. Jha, R. Enders, and T. Filkhorn. Exploiting symmetry in temporal logic model checking. *Formal Methods in System Design*, 9(1/2):77–104, August 1996.
- [6] E.A. Emmerson and A.P. Sistla. Symmetry and model checking. *Formal Methods in System Design*, 9(1/2):105–131, August 1996.
- [7] J.M. Eyzell and J.E.R. Cury. Exploiting symmetry in the synthesis of supervisors for discrete event systems. *IEEE Trans. Auto. Contr.*, 46(9):1500–1505, September 2001.
- [8] P. Gohari and W.M. Wonham. On the complexity of supervisory control design in the RW framework. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 30(5):643–652, 2000.
- [9] D. Harel, O. Kupferman, and M.Y. Vardi. On the complexity of verifying concurrent transition systems. *Information and Computation*, 173:143–161, 2002.
- [10] C. Hoffman. *Graph Isomorphism and Permutation Groups, LNCS 132*. Springer-Verlag, 1982.
- [11] C.N. Ip and D.L. Dill. Better verification through symmetry. *Formal Methods in System Design*, 9(1/2):41–75, August 1996.

- [12] S. Jiang, V. Chandra, and R. Kumar. Decentralized control of discrete event systems with multiple local specializations. In *Proc. of 2001 American Control Conference*, pages 959–964, 2001.
- [13] S. Jiang and R. Kumar. Decentralized control of discrete event systems with specializations to local control and concurrent systems. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 30(5):653–660, 2000.
- [14] O. Kupferman, M. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, 2000.
- [15] R.J. Leduc, B. Brandin, M. Lawford, and W.M. Wonham. Hierarchical interface-based supervisory control: Serial case. In *Proc. 40th IEEE Conf. on Decision and Control*, pages 4116–4121, 2001.
- [16] R.J. Leduc, M. Lawford, and W.M. Wonham. Hierarchical interface-based supervisory control: AIP example. In *39th Allerton Conf. on Comm., Contr., and Comp.*, 2001.
- [17] F. Lin and W. M. Wonham. Decentralized supervisory control of discrete-event systems. *Information Sciences*, 44:199–224, 1988.
- [18] F. Lin and W. M. Wonham. On observability of discrete-event systems. *Information Sciences*, 44:173–198, 1988.
- [19] F. Lin and W. M. Wonham. Decentralized control and coordination of discrete-event systems with partial observation. *IEEE Trans. Auto. Contr.*, 35(12):199–224, December 1990.
- [20] F. Lin and W. M. Wonham. Verification of nonblocking in decentralized supervision. *Control Theory and Advanced Technology*, 7(1):223–232, March 1991.
- [21] P. Lincoln and A. Tiwari. Symbolic systems biology: Hybrid modeling and analysis of biological networks. In R. Alur and G. Pappas, editors, *Proc. of the 7th International Workshop, HSCC, number 2993 in LNCS*, pages 660–672. Springer-Verlag, 2004.
- [22] M. H. Queiroz and J. E. R. Cury. Modular control of composed systems. In *Proc. of 2000 American Control Conference*, 2000.
- [23] P.J. Ramadge. Some tractable supervisory control problems for discrete-event systems modeled by Büchi automata. *IEEE Trans. Auto. Contr.*, 34(1):10–19, 1989.
- [24] P.J. Ramadge and W.M. Wonham. Supervisory control of a class of discrete-event processes. *SIAM Journal of Control Optimization*, 25(1):206–230, 1987.
- [25] P.J. Ramadge and W.M. Wonham. The control of discrete-event systems. *Proc. IEEE*, 77(1):81–98, 1989.
- [26] K. Rohloff and S. Lafortune. On the computational complexity of the verification of modular discrete-event systems. In *Proc. 41st IEEE Conf. on Decision and Control*, Las Vegas, Nevada, December 2002.
- [27] K. Rohloff and S. Lafortune. The control and verification of similar agents operating in a broadcast network. In *Proc. 42nd IEEE Conf. on Decision and Control*, Maui, Hawaii, December 2003.

- [28] K. Rohloff and S. Lafortune. Supervisor existence for modular discrete-event systems. In *Proc. 2nd IFAC Conf. on Control Systems Design*, Bratislava, Slovakia, September 2003.
- [29] K. Rudie and W.M. Wonham. Think globally, act locally: Decentralized supervisory control. *IEEE Trans. Auto. Contr.*, 37(11):1692–1708, November 1992.
- [30] M. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115:1–37, 1994.
- [31] Y. Willner and M. Heyman. Supervisory control of concurrent discrete event systems. *International Journal of Control*, 54(5):1143–1169, 1991.
- [32] K.C. Wong and W.M. Wonham. Modular control and coordination of discrete-event systems. *Journal of Discrete Event Dynamical Systems: Theory and Applications*, 8:247–297, 1998.
- [33] W. M. Wonham and P. J. Ramadge. Modular supervisory control of discrete event systems. *Maths. of Control, Signals and Systems*, 1(1):13–30, 1988.