The Control and Verification of Similar Agents Operating in a Broadcast Network Environment [1]

Kurt Rohloff and Stéphane Lafortune
Department of Electrical Engineering and Computer Science
The University of Michigan
1301 Beal Ave., Ann Arbor, MI 48109-2122, USA
{krohloff,stephane}@eecs.umich.edu; www.eecs.umich.edu/umdes

Abstract

We explore issues related to the control and verification of similar agents that interact through events broadcast over a network. The similar agents are modeled as discrete-event systems that have identical structure. System events are partitioned into global and private events that respectively affect all agents or exactly one agent. We show how the state explosion problem inherent to many concurrent systems is not as problematic in this setting. We give a procedure to test if these systems are globally deadlock-free or nonblocking. We explore control and verification problems related to both local and global specifications on these systems. For each module there is exactly one controller and all controllers enforce the same control policy. Necessary and sufficient conditions for achieving local and global specifications in this setting are identified.

1 Introduction

Many real-world systems are comprised of interacting agents that can be modeled in a modular manner. Often many of these agents exhibit a degree of similarity such that they can be modeled as interacting modular subsystems that are exact copies of one another. It is possible in principle to convert the set of modular subsystems to a monolithic model to simplify the control and verification of the behavior of these interacting agents. However, this conversion from modular to monolithic systems is generally not computationally feasible as this operation suffers from the well-known state explosion problem. Regardless of the difficulties inherent to the conversion of modular systems to monolithic systems, it is generally more intuitive to manipulate concurrent processes as modular systems. It has been found that in general the verification and control of modular systems is complicated [3, 4, 6, 16, 17, 18, 20].

We explore a model for similar interacting agents that hopefully leads to more computationally tractable methods for solving verification and control problems. The modules are exact copies of one another except for a renaming of private events. Many relevant and important real-world processes are included in this class of systems such as platoons of similar unmanned aerial vehicles, sensor networks operating in an unknown environment and communication systems all running the same protocol at all nodes of a network. We attempt to make our model as general as possible so we do not specify the exact broadcast medium the similar agents will use for interaction. However, it could be conceived that the agents are communicating over a common data link such as an Ethernet LAN or through the use of radio transmissions that can be received by all modules.

The system events are partitioned into global and local behavior events. The behavior of the modules is coordinated only through the occurrence of global events, and the occurrence of a local event affects only the module where it occurs. We assume that there is exactly one controller per module that makes local observations and enforces local control actions, but local controllers can disable global events. As with the system modules, the controllers are similar in that all controllers enforce the same control policy at each of their respective subsystems except for a renaming of the respective local events. There is no explicit communication between controllers.

We generally follow the framework of "supervisory control theory"(SCT) introduced in the seminal work of [15]. The interested reader may consult the text [1] for a general introduction to discrete-event systems and SCT. The finite automaton modeling formalism which we use here is generally considered to be the simplest one for discrete-event systems that is expressive enough to approximate the behavior of real-world systems in a reasonable manner. We assume each local controller is a parallel control system modeled as a finite automaton. Namely, the controller is realized as an automaton that is coupled to the uncontrolled system through parallel composition. The controllers make observations of global and local behavior and enact control actions at the local site as introduced in [9]. We also use the parallel composition operation to model interaction between our modules, which is currently the standard method to model the interaction between automata in discrete-event system theory. We

[1] This research was supported in part by NSF grant CCR-0082784.

call our model for interacting similar module systems "Isomorphic Module Systems" or IMS's because the modules are isomorphic to one another.

Specifications for modular systems can be made at both the local and global levels. We assume that our specifications are given as languages and we wish our systems to exactly match the behavior of these specifications. We give a method for testing if a system is globally deadlock-free or nonblocking that avoids enumerating over all system states in an IMS. We show that for local specifications, control and verification problems are computationally very simple. We find that global language specifications are more problematic for our models, but we give necessary and sufficient conditions for the existence of controllers to achieve a global specification.

The supervision of modular systems is currently receiving much attention from the control research community; see, e.g., [2, 5, 7, 8, 9, 11, 12, 13, 14, 21, 22, 23]. Some of the earlier results relating to modular supervision are shown in [9, 11, 14, 21, 23]. Properties of modular discrete-event systems when the modules have disjoint alphabets are investigated in [13, 14]. Various local specification and concurrent supervision problems, respectively, are investigated in [5]. The supervision of modular systems using specific architectures is discussed in [7, 8]. A form of modular control where each controller has a different objective is discussed in [2]. Situations when local nonblocking behavior implies global nonblocking behavior are discussed in [12]. To the best of our knowledge however, problems of interacting similar systems have received very little attention in the literature.

In the next section of this paper we formally define the models used in this paper. In Section 3 we discuss classes of equivalent states in our models and how we can decrease the number of states that need to be checked in the worst-case in order to verify global properties. In Section 4 we discuss local specification problems and in Section 5 we discuss problems related to global specifications. We close this paper with a discussion of our results and possible areas for further research in Section 6.

2 Model Definition

We now introduce the models used in the rest of the paper. Automaton $G_1 = (X, x_0, \Sigma_1, \delta_1, X_m)$ models the behavior of a generic module in our system. The overall system is composed of a set of system modules $\{G_1, \ldots, G_n\}$ isomorphic to G_1. The module event set Σ_1 is partitioned into the distinct subsets Σ_g and Σ_{p1}, the global event set and the private event set for module 1, respectively. Let $\Sigma_{p2}, \ldots, \Sigma_{pn}$ be additional event sets that represent copies of the private event sets for $G_2, \ldots, G_n$, respectively, such that for all $i, j, i \neq j$, $\Sigma_g \cap \Sigma_{pi} = \emptyset$ and $\Sigma_{pi} \cap \Sigma_{pj} = \emptyset$. Define $\Sigma_i = \Sigma_g \cup \Sigma_{pi}$ and let Σ denote $\Sigma_1 \cup \cdots \cup \Sigma_n$. The events in Σ_i are the events whose occurrence affect behavior in G_i.

We use the function $\Psi_i : \Sigma_1 \to \Sigma_i$ that maps the private event set of the first module to the ith private event set and maps global events to themselves. This function is extended in the usual manner to map strings from Σ_1^* to strings from Σ_i^*. Let Ψ_i^{-1} denote the corresponding inverse mapping from Σ_i^* to Σ_1^*. We also use the function $\Psi_{1i} : \Sigma^* \to \Sigma^*$ to denote the operation where, for $K \subseteq \Sigma^*$, $\Psi_{1i}(K)$ is the set of strings $t \in K$ except all events $\sigma_1 \in \Sigma_{p1}$ in t are replaced with the corresponding event from $\sigma_i \in \Sigma_{pi}$ and all events σ_i in t are replaced with the corresponding event from σ_1 according to $\Psi_i(\cdot)$. Let $P_i : \Sigma^* \to \Sigma_i^*$ be the natural projection that erases events in $\Sigma \setminus \Sigma_i$ and let the inverse projection $P_i^{-1} : \Sigma_i^* \to 2^{\Sigma^*}$ be defined in the usual manner [1].

When constructing $G_i \in \{G_1, \ldots, G_n\}$ from G_1, we replace all events in Σ_{p1} with the respective events from Σ_{pi} according to the $\Psi_i(\cdot)$ mapping. To formalize, let $G_i = (X, x_0, \Sigma_i, \delta_i, X_m)$. For $x \in X, \gamma \in \Sigma_i$, we define $\delta_i(x, \gamma) = \delta_1(x, \Psi_i^{-1}(\gamma))$. We do not index the state sets X and X_m because it does not matter if the state labels are replicated. We take advantage of this replication of state labels in the next section.

As was mentioned in the introduction, we use the parallel composition, denoted by $\|$, to model the interaction between modules. Therefore, given a set of modules $\{G_1, \ldots, G_n\}$, the composed monolithic equivalent of these interacting modules can be expressed as $G_1 \| \cdots \| G_n$. We call the system just described with $\{G_1, \ldots, G_n\}$ as defined an IMS. Let $\vec{X}$ be the state space of $G_1 \| \cdots \| G_n$. We call the states of the individual modules (i.e., $x \in X$) *module states* and the states of the composed system $G_1 \| \cdots \| G_n$ (i.e., $\vec{x} \in \vec{X}$) *composed states*. Let $\vec{x}^i$ be the ith module state in $\vec{x}$. Observe that $\mathcal{L}_m(G_1 \| \cdots \| G_n) = P_1^{-1}(\mathcal{L}_m(G_1)) \cap \cdots \cap P_n^{-1}(\mathcal{L}_m(G_n))$ and $\mathcal{L}(G_1 \| \cdots \| G_n) = P_1^{-1}(\mathcal{L}(G_1)) \cap \cdots \cap P_n^{-1}(\mathcal{L}(G_n))$. We introduce the following definitions which will be used below.

Definition 1 *A language $K \subseteq \Sigma^*$ is said to be* symmetric *with respect to the private event sets $\Sigma_{p1}, \ldots, \Sigma_{pn}$ and the mappings $\Psi_{11}, \ldots, \Psi_{1n}$ if $\forall i \in \{1, \ldots, n\} \Psi_{1i}(K) = K$.*

This symmetry definition is used to convey the intuition that K is identical with respect to a relabeling of private events for the IMS $\{G_1, .., G_n\}$.

Definition 2 *[19] A language $K \subseteq \Sigma^*$ is* decomposable *with respect to the projections $\{P_1, \ldots, P_n\}$ if $K = P_1^{-1}(P_1(K)) \cap \cdots \cap P_n^{-1}(P_n(K))$.*

A language is decomposable if given the local knowledge of K at all sites, i.e., $P_1(K), \ldots, P_n(K)$, we can recover K.

Note that this definition is slightly altered from the definition given in [19] in that we disregard system behavior.

Definition 3 *[23] A set of languages* $\{L_1, \ldots, L_n\}$ *is said to be* nonconflicting *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 nonblocking automata need not be nonblocking unless the respective languages marked by the automata are nonconflicting.

Given a supervisor S_1 and a system G_1, we denote the composed system of S_1 supervising G_1 as the supervised system S_1/G_1. Furthermore, because we assume we are using parallel supervisors 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 $\overline{\mathcal{L}_m(S_1 \| G_1)} = \mathcal{L}(S_1 \| G_1)$. For our control systems model, we assume that the controller automata $\{S_2, \ldots, S_n\}$ are copies of the generic control module S_1 with the private events replaced according to the $\Psi_i(\cdot)$ mapping. We call $\{S_1, \ldots, S_n\}$ a set of *Isomorphic Module Controllers*.

We assume that a local controller S_i operating on module G_i observes and controls only locally relevant events ($\Sigma_{oi} \subseteq \Sigma_i$ and $\Sigma_{ci} \subseteq \Sigma_i$) respectively. Furthermore, let $\Sigma_{uci} = \Sigma_i \setminus \Sigma_{ci}$. Let $P_{oi} : \Sigma^* \to \Sigma_{oi}^*$ be the natural projection that erases events in $\Sigma \setminus \Sigma_{oi}$. P_{oi} represents the projection operation for the local observations of controller S_i. Let the inverse projection $P_{oi}^{-1} : \Sigma_i^* \to 2^{\Sigma^*}$ be defined in the usual manner [1].

An automaton G is blocking (resp. deadlocking) by definition if there is a reachable blocking (resp. deadlocking) state x in G. A state x is blocked if there is no path to reach a marked state from x according to the transition rules of G. A state x is deadlocked if there are no transitions from x according to the transition rules of G. Note that testing if a system is deadlock-free or nonblocking can be reduced to problems of testing reachability. As a reminder to the reader, a system that is nonblocking is not deadlock-free if a marked state deadlocks.

3 Verification

Given a set of automata $\{G_1, \ldots, G_n\}$ such that the size of their respective state spaces is bounded by k, the composed automaton $G_1 \| \cdots \| G_n$ has k^n reachable states in the worst case. As n grows, this state space can become unbearably large and we would understandably be unable to perform any procedures that require us to enumerate over all of the reachable states of $G_1 \| \cdots \| G_n$ in a time-efficient manner. Therefore we would be unable to efficiently perform many control or verification procedures on the composed system $G_1 \| \cdots \| G_n$ with respect to a global specification K using current known methods.

However, with the added structure for isomorphic module systems we introduced above, there are several shortcuts to decrease our computation time when we wish to perform control and verification procedures. We show how the symmetries in the state space of the composed systems decrease the number of states we need to check to verify the behavior of the composed model. We now show an example to demonstrate how the states of an IMS can be partitioned into sets of "equivalent" states.

Example 1 *Consider the 2-module system composed of the similar modules* G_1 *and* G_2 *seen in Figure 1. Let* $\Sigma_g = \{\gamma, \lambda\}$, $\Sigma_{p1} = \{\alpha_1, \beta_1\}$ *and* $\Sigma_{p2} = \{\alpha_2, \beta_2\}$. G_1 *and* G_2 *are both locally deadlock-free and nonblocking.*

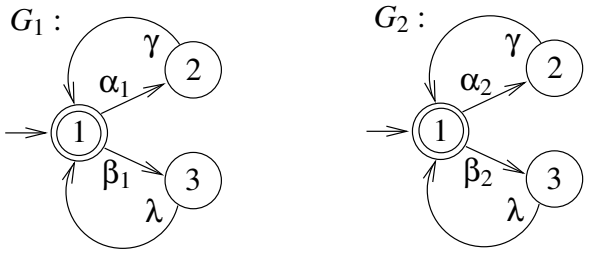

Figure 1: The automata G_1 and G_2.

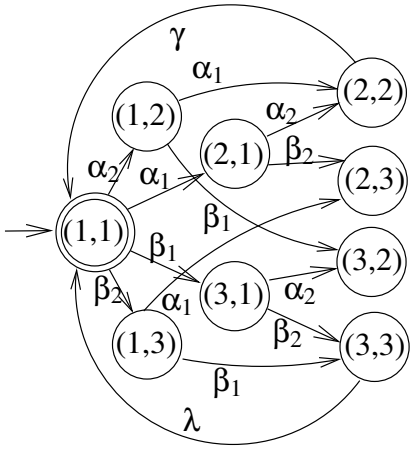

Figure 2: The automaton $G_1 \| G_2$.

Now consider the composed system $G_1 \| G_2$ *as seen in Figure 2. The automaton* $G_1 \| G_2$ *has a large degree of symmetry due to the similarity of the component automata* G_1 *and* G_2. *Consider the states* $(1,2)$ *and* $(2,1)$. *These states are reached by the occurrence of* α_2 *and* α_1 *events respectively. Consider also the* $(2,3)$ *and* $(3,2)$ *states in* $G_1 \| G_2$. *If we swap the subscript labels of the events in the strings leading to state* $(2,3)$ *the resulting string will lead instead to state* $(3,2)$. *The parallel composition operation is commutative and so, the order of parallel composition is arbitrary. Due to the isomorphic structure of the automata it should not matter if we alter the order of the component au-*

tomata when verifying state properties in the composed system. This is because the modules are identical with respect to a renaming of private events. Therefore, any property of state $(2,3)$ is also held by state $(3,2)$ because by swapping state locations we are in effect swapping the order of parallel composition of the component automata.

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

Now consider a third component G_3 similar to G_1 and G_2 as seen in Figure 1. A state (x_a,x_b,x_c) in $G_1 \| G_2 \| G_3$ is equivalent to (x_a,x_c,x_b), (x_b,x_a,x_c), (x_b,x_c,x_a), etc... with respect to a shuffling of the components G_1, G_2 and G_3.

The intuition we are trying to develop with Example 1 is that state orderings do not matter when testing global properties. Given a state space $X = \{x_1,\ldots,x_k\}$ for an IMS $G_1,\ldots,G_n$, we can place an ordering on the states in X such that $x_1 < x_2 < \cdots < x_k$. Therefore, for any state $\vec{x}$ in the composed machine $G_1 \| \cdots \| G_n$, we can reorder the component states in $\vec{x}$ such that the component states have the correct ordering with respect to the list $x_1 < x_2 < \cdots < x_k$. We call this rearranged state with the correct component state ordering the *standard representation* of the original state. The notion of a state's standard representations establishes several classes of equivalent states such that two states are in an equivalence class if they have the same standard representation. We define a function $SR : \vec{X} \to \vec{X}$ that maps a composed state to its standard representation and let $SR^{-1}(\cdot)$ be the inverse function that returns a set of states that have the same standard representation as its input. The initial state $\vec{x_0}$ is its own standard representation because $\vec{x_0} = (x_0,\ldots,x_0)$.

We now define an automaton $\vec{G} = \left(\vec{X}, \vec{x_0}, \Sigma, \vec{\delta}, \vec{X_m}\right)$ constructed from $\{G_1,\ldots,G_n\}$. $\vec{G}$ uses the set of standard representations of the reachable states in $G_1 \| \ldots \| G_n$ as its state space and $\vec{X_m}$ is the set of all marked states in $\vec{X}$. $\vec{x_o}$ is the initial state of $G_1 \| \cdots \| G_n$. The state transition function $\vec{\delta} : \vec{X} \to \vec{X}$ is defined as follows:

$$\vec{\delta}(\vec{x},\sigma) = \left\{ \begin{array}{l} SR\left(\left(\delta_1(\vec{x}^1,\sigma),\ldots,\delta_n(\vec{x}^n,\sigma)\right)\right) \text{ if } \sigma \in \Sigma_g \\ SR\left(\left(\vec{x}^1,\ldots,\delta_i(\vec{x}^i,\sigma),\ldots,\vec{x}^n\right)\right) \text{ if } \sigma \in \Sigma_{pi} \\ \text{undefined otherwise.} \end{array} \right\}$$

We can extend the definition of $\vec{\delta}(\cdot,\cdot)$ to allows strings of arbitrary length using the usual methods. We call $\vec{G}$ the reduced state space composed automaton. $\vec{G}$ can be constructed efficiently using standard breadth-first methods. We now use the IMS introduced in Example 1 to demonstrate the construction of a reduced state space composed automaton $\vec{G}$.

Example 2 *Consider the IMS G_1, G_2 introduced in Example 1 above. The set of standard representations in $G_1 \| G_2$ is $\{(1,1),(1,2),(1,3),(2,2),(2,3),(3,3)\}$. The reduced state space composed automaton $\vec{G}$ can be seen in Figure 3. Note that $\vec{G}$ has a smaller state space than $G_1 \| G_2$.*

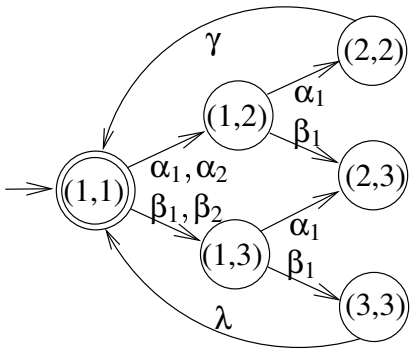

Figure 3: The automaton $\vec{G}$ constructed from G_1, G_2.

We can use $\vec{G}$ to perform verification on the composed behavior of $G_1 \| \cdots \| G_n$. First, we show reachability in $\vec{G}$ implies a form of reachability in $G_1 \| \cdots \| G_n$ and vice-versa.

Theorem 1 *Given two states $\vec{x},\vec{y}$ in a reduced state space composed automaton $\vec{G}$ constructed from an IMS composed of $G_1,\ldots,G_n$, $\vec{x}$ is reachable from $\vec{y}$ in exactly p steps according to the transition rules of $\vec{G}$ if and only if all states in $SR^{-1}(\vec{x})$ are reachable from a state in $SR^{-1}(\vec{y})$ in exactly p steps according to the transition rules of $G_1 \| \cdots \| G_n$.*

Due to space considerations we do not show the proof of this theorem, but it is fundamentally a proof by induction. A similar proof can also be used to show that given two states $\vec{x},\vec{y}$ in $\vec{G}$, $\vec{x}$ is reachable from $\vec{y}$ in exactly p steps if and only if from all states in $SR^{-1}(\vec{y})$ there is a set of transitions that leads to a state in $SR^{-1}(\vec{x})$ in exactly p steps according to the transition rules of $G_1 \| \cdots \| G_n$. We can extend Theorem 1 to the following useful corollary using a proof by contradiction.

Corollary 1 *Given two states $\vec{x},\vec{y}$ in a reduced state space composed automaton $\vec{G}$ constructed from an IMS composed of $G_1,\ldots,G_n$, $\vec{x}$ is reachable from $\vec{y}$ according to the transition rules of $\vec{G}$ if and only if all states in $SR^{-1}(\vec{x})$ are reachable from a state in $SR^{-1}(\vec{y})$ according to the transition rules of $G_1 \| \cdots \| G_n$.*

Based on the definitions of deadlock-free and nonblocking systems, Corollary 1 implies the following corollary.

Corollary 2 *Given an IMS composed of $G_1,\ldots,G_n$, the composed system $G_1 \| \cdots \| G_n$ is nonblocking (resp.*

deadlock-free) if and only if $\vec{G}$ is nonblocking (resp. deadlock-free).

Corollary 2 shows two of the main advantages of using the automaton $\vec{G}$: the construction allows us to test if an IMS is deadlock-free or nonblocking in less time using a construction with a smaller space than would be needed if we were to construct the full system $G_1 \| \cdots \| G_n$ and enumerate over all states.

From here a fruitful question to ask would be given an IMS $G_1, \ldots, G_n$ based on the model we introduced in Section 2 where $k = |X|$, how many sets of equivalent states are there? This would put an upper limit on the size of the reachable state space of $\vec{G}$. This question is equivalent to the question from bag theory where given a set of k elements and a bag that can contain n objects, how many ways are there to fill the bag? This problem is equivalent to a bin and ball problem from combinatorics where we are asked to find how many ways there are to fill $(n+1)$ bins using $(k-1)$ balls. For this problem, there are

$$\left(\begin{array}{c} k+n-1 \\ n \end{array} \right) = \left(\frac{(k+n-1)!}{(k-1)!n!} \right) \quad (1)$$

ways to fill the bins. Therefore, there are the same number of classes of states that need to be verified to check a global property.

Although this is still a large number of classes of states that need to be verified, it is rather smaller than k^n. We demonstrate the reduction in computation using the IMS introduced in Example 1.

Example 3 *Suppose instead of a 2-module IMS $\{G_1, G_2\}$ in Example 1 we have an n-module system, $\{G_1, \ldots, G_n\}$ with the same isomorphic structure as $\{G_1, G_2\}$. For this example $k = 3$, so $k^n = 3^n$ and*

$$\left(\frac{(2+n)!}{2!n!} \right) = \left(\frac{(n+1)(n+2)}{2} \right) = \left(\frac{n^2+3n+2}{2} \right) \quad (2)$$

Although not all problems have as dramatic a reduction in the number of states that need to be enumerated over in order to perform verification, we have found that for many problem instances the number of equivalent state classes is on the order of $n^k/k!$. The time required to construct $\vec{G}$ and test for deadlock-freeness or nonblocking is linear in the size of the state-space of $\vec{G}$. If a module of an IMS has a relatively small state space and there are a fairly large number of modules, the reductions in computation time and space can be significant when performing verification using the $\vec{G}$ construction described here.

When verifying similar module systems for local specifications, there are other approaches besides the brute-force method of enumerating over all possible states. For instance, we can take advantage of the symmetric system structure and use a divide-and-conquer approach. We explore these possibilities in the next section.

4 Local Specifications

We now show that to verify local module behavior in an IMS we only need to investigate the local module without the interaction of other modules. Suppose we are given a language specification $K_1 \subseteq \Sigma_1^*$ for behavior relevant to module G_1. We assume that K_1 is also replicated as $\{K_2, \ldots, K_n\}$ according to the mapping $\Psi_i(\cdot)$. We may for example wish to check if for all i, $P_i(\mathcal{L}_m(G_1 \| \cdots \| G_n)) = K_i$. We can verify this property by solely looking at the behavior $\mathcal{L}_m(G_i)$. This follows from the following theorem whose proof we do not show due to space considerations.

Theorem 2 *For an IMS $\{G_1, \ldots, G_n\}$ as introduced above with respective local projection operations $\{P_1, \ldots, P_n\}$ and for $i \in \{1, \ldots, n\}$, $P_i(\mathcal{L}_m(G_1 \| \cdots \| G_n)) = \mathcal{L}_m(G_i)$.*

We can now present the following corollary which is a direct extension of Theorem 2.

Corollary 3 *Given a local language specification K_i and IMS $\{G_1, \ldots, G_n\}$, $P_i(\mathcal{L}_m(G_1 \| \cdots \| G_n)) = K_i$ if and only if $\mathcal{L}_m(G_i) = K_i$.*

Likewise, we can extend the results of Theorem 2 and Corollary 3 to the case of generated languages rather than marked languages. 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 an automaton. This means we can test local behavior in a composed IMS by looking at a single module. This greatly simplifies previously known verification methods based on more general modular systems.

Similar reductions in computational effort also hold for many control problems related to IMS's with local specifications. For an IMS $\{G_1, \ldots, G_n\}$ and similar language specifications $\{K_1, \ldots, K_n\}$, suppose we would like to know if there exists a nonblocking set of control automata $\{S_1, \ldots, S_n\}$ as discussed above such that $\forall i \in \{1, \ldots, n\}$ $P_i[\mathcal{L}_m((S_1/G_1) \| \cdots \| (S_n/G_n))] = K_i$ and $P_i[\mathcal{L}((S_1/G_1) \| \cdots \| (S_n/G_n))] = \overline{K_i}$. This problem can be solved by looking only at the local behavior of G_1 and the locally observable and controllable event sets, Σ_{o1} and Σ_{c1}, respectively, as seen in the following corollary whose proof is based on the controllability and observability theorem as seen in [10].

Corollary 4 *For an IMS $\{G_1, \ldots, G_n\}$ as introduced above with respective local projection operations $\{P_1, \ldots, P_n\}$,*

observation projections $\{P_{o1},\ldots,P_{on}\}$*, controllable event sets* $\{\Sigma_{c1},\ldots,\Sigma_{cn}\}$ *and local behavior specifications* $\{K_1,\ldots,K_n\}$ *such that* $K_1 \neq \emptyset$*,* $K_1 \subseteq \mathcal{L}_m(G_i)$ *and* $\forall i \in \{1,\ldots,n\}\ K_i = \Psi_i(K_1)$*, there exists a set of isomorphic module controllers* $\{S_1,\ldots,S_n\}$ *such that* $P_i[\mathcal{L}_m((S_1/G_1)\|\cdots\|(S_n/G_n))] = K_i$ *and* $P_i[\mathcal{L}((S_1/G_1)\|\cdots\|(S_n/G_n))] = \overline{K_i}$ *if and only if*

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

The controllability and observability theorem is known to be constructive so we have a method for synthesizing the local controllers $\{S_1,\ldots,S_n\}$ such that local nonblocking specifications are satisfied at all nodes when the modules interact. Our results also generalize to the cases where we may not be concerned with marking properties.

5 Global Specifications

Suppose we are given a global specification K and we are asked to decide if there exist nonblocking isomorphic module controllers $\{S_1,\ldots,S_n\}$ for an IMS $\{G_1,\ldots,G_n\}$ such that $K = \mathcal{L}_m((S_1/G_1)\|\cdots\|(S_n/G_n))$ and $\overline{K} = \mathcal{L}((S_1/G_1)\|\cdots\|(S_n/G_n))$. Due to the similarity of the controllers and system modules one would think that K would have to exhibit a degree of symmetry with respect to the occurrence of private events. This is exactly the case which we find in Theorem 3 below. We do not show the proof due to space considerations, but it is a constructive proof and it gives a method for synthesizing the local controllers to achieve the specification.

Theorem 3 *For an IMS* $\{G_1,\ldots,G_n\}$ *as introduced above with respective local projection operations* $\{P_1,\ldots,P_n\}$*, observation projections* $\{P_{o1},\ldots,P_{on}\}$*, controllable event sets* $\{\Sigma_{c1},\ldots,\Sigma_{cn}\}$ *and global behavior specification* K *such that* $K \neq \emptyset$ *and* $K \subseteq \mathcal{L}_m(G_1\|\cdots\|G_n)$*, there exists a set of isomorphic module controllers* $\{S_1,\ldots,S_n\}$ *such that* $\mathcal{L}_m((S_1/G_1)\|\cdots\|(S_n/G_n)) = K$ *and* $\mathcal{L}((S_1/G_1)\|\cdots\|(S_n/G_n)) = \overline{K}$ *if and only if*

1. K *is symmetric w.r.t.* $\{\Sigma_1,\ldots,\Sigma_n\}$.
2. K *is decomposable w.r.t.* $\{P_1,\ldots,P_n\}$.
3. $\{P_1^{-1}(P_1(K)),\ldots,P_n^{-1}(P_n(K))\}$ *are nonconflicting.*
4. $P_1(K)$ *is controllable w.r.t.* $\mathcal{L}(G_1)$ *and* Σ_{uc1}.
5. $P_1(K)$ *is observable w.r.t.* $\mathcal{L}(G_1)$*,* P_{o1} *and* Σ_{c1}.
6. $P_1(K)$ *is* $\mathcal{L}_m(G_1)$*-closed.*

The six necessary and sufficient conditions for controller existence in Theorem 3 can be divided into two types. The first three conditions (symmetry, decomposability and nonconflicting inverse projections) show that the global specifications can be decomposed into symmetric local specifications such that the global behavior can be regained by recomposing the local behavior. The last 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 controller problems and have been well known since the early papers in supervisory control theory [10, 15]. Given a set of local specifications $\{K_1,\ldots,K_n\}$ obtained from the local projections of the global specification K, the controllers $\{S_1,\ldots,S_n\}$ can be synthesized very easily using known methods.

The first condition demonstrates that any specification K must contain a symmetry with respect to the local behavior required at the local sites in $\{G_1,\ldots,G_n\}$. The behavior of all controllers operating on the modules should be similar with respect to a renaming of local events, so the specification is necessarily symmetric if it can be achieved.

The decomposability condition ensures that we can reconstruct the specification K from its local behavior projections $P_1,\ldots,P_n$. Therefore, if we know the local specifications are achieved by the local behavior, i.e., $P_i(K) = \mathcal{L}_m(S_i/G_i)$, the decomposability condition forces the global controlled behavior to be equivalent to the global behavior specification.

As is known from the standard literature in SCT, local nonblocking behavior does not imply global nonblocking behavior. The nonconflicting condition ensures that local nonblocking behavior implies global nonblocking behavior.

Taken together, the first three conditions imply that the global specification can be expressed as isomorphic nonconflicting local specifications if there exists a set of isomorphic controllers that can be coupled with the IMS to achieve the specification.

Note that all six conditions imply that the system $G_1\|\cdots\|G_n$ is globally controllable, coobservable and $\mathcal{L}_m$-closed, but the reverse implication does not hold because of the assumptions on the controllers we are using. If we were to allow the controllers to be asymmetric we would be able to achieve a larger class of specifications, but this would require an extra amount of coordination in the control synthesis. Our model is set up so that once one control module is designed, the implementation of more controllers is merely a matter of copying that first controller.

6 Discussion

We have introduced a model for a isomorphic module system that can be used to model a wide variety of real-world concurrent processes. A method that can be used to test global deadlock-freeness or nonblocking without enumerating all possible combinations of system states is shown. We demonstrated that we can perform verification of local behavior in an off-line manner without module interaction. We have also shown necessary and sufficient conditions for the existence of local controllers for the similar module model we introduced.

We have solely investigated the case where the control modules are similar, 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 deadlock or blocking 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.

References

[1] C.G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, Boston, MA, 1999.

[2] 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.

[3] 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.

[4] D. Harel, O. Kupferman, and M.Y. Vardi. On the complexity of verifying concurrent transition systems. *Information and Computation*, 173:143–161, 2002.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] F. Lin and W. M. Wonham. Decentralized supervisory control of discrete-event systems. *Information Sciences*, 44:199–224, 1988.

[10] F. Lin and W. M. Wonham. On observability of discrete-event systems. *Information Sciences*, 44:173–198, 1988.

[11] 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.

[12] F. Lin and W. M. Wonham. Verification of nonblocking in decentralized supervision. *Control-Theory and Advanced Technology*, 7(1):223–232, March 1991.

[13] M. H. Queiroz and J. E. R. Cury. Modular control of composed systems. In *Proc. of 2000 American Control Conference*, 2000.

[14] 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.

[15] P.J. Ramadge and W.M. Wonham. The control of discrete-event systems. *Proc. IEEE*, 77(1):81–98, 1989.

[16] 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.

[17] K. Rohloff and S. Lafortune. Recent results on computational issues in supervisory control. In *Proc. of the ATPN-Workshop on Discrete Event Systems Control*, Eindhoven, The Netherlands, June 2003.

[18] 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.

[19] K. Rudie and W.M. Wonham. Think globally, act locally: Decentralized supervisory control. *IEEE Trans. Auto. Contr.*, 37(11):1692–1708, November 1992.

[20] M. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115:1–37, 1994.

[21] Y. Willner and M. Heyman. Supervisory control of concurrent discrete event systems. *International Journal of Control*, 54(5):1143–1169, 1991.

[22] 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.

[23] 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.