Model checking in isomorphic module systems

Wang, Weilin; Su, Rong; Lin, Liyong; Gong, Chaohui

2018

Wang, W., Su, R., Lin, L., & Gong, C. (2018). Model checking in isomorphic module systems. IEEE Transactions on Automatic Control, 64(2), 728-735. doi:10.1109/TAC.2018.2844683

<https://hdl.handle.net/10356/137873>

<https://doi.org/10.1109/TAC.2018.2844683>

© 2018 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works. The published version is available at:
<https://doi.org/10.1109/TAC.2018.2844683>

Downloaded on 26 Apr 2023 17:54:04 SGT

Model Checking in Isomorphic Module Systems

Weilin Wang, *Senior Member, IEEE*, Rong Su, *Senior Member, IEEE*, Liyong Lin, and Chaohui Gong

Abstract—With a growing number of interacting modules in today’s complex systems, finding an effective tool for detecting design errors has become increasingly challenging. We present procedures for analyzing blocking or deadlock in systems consisting of isomorphic modules instantiated from a template. These procedures identify all possible blocking or deadlock sources and directly compute the maximum number of isomorphic modules under which the system is guaranteed to be nonblocking or deadlock-free.

Index Terms—Model Checking, Formal Methods, Discrete Event Systems, Parameterized Systems, Automata

I. INTRODUCTION

MANY modern systems, including computer systems with concurrent processors [1], multi-agent systems [2], [3], biological systems [4], and communication systems [5], consist of a varied number of functionally similar modules. Such variations have caused great challenges for model checking. Since the proper and safe operation of these systems require certain system properties to be satisfied, the challenges that model-checking for these systems faces attracts considerable attention amongst researchers [6], [7], [8], [9], [10], [11]. For a specific type of systems that consists of an arbitrary number of functionally-isomorphic modules, we develop a novel approach for analyzing blocking or deadlock without explicit module synchronization.

When the number of modules is given, the research on analyzing blocking or deadlock focuses on methods for efficiently composing modules. Among such techniques are the conflict equivalence techniques of [12] and the nonconflicting testing techniques of [13]. Researchers have explored group-theoretic methods and system symmetry [1], [6], [7]. Researchers also restrict the module structure by investigating port automata [14] or asynchronous automata networks [15]. Most work on this topic using discrete event formalisms concern supervisory control for mitigating blocking or deadlock [16], [17]. In [16], supervisory control for blocking prevention is improved using a cognitive heuristic to search through available alternatives. In [17], a procedure of checking nonconflict is developed based on automaton abstractions that allows the check to be performed on relatively small automata. However, as far as we know, no method for identifying blocking / deadlock in modular systems could avoid explicitly synchronizing modules.

If the number of replicated module copies is arbitrary, a verification problem with a sufficiently rich modeling frame-

work can be undecidable [18], [9], [19], [20]. Apart from decidability, some researchers study these systems by identifying useful decidable modeling-structures. In [18], the authors show the usefulness of rings for structuring concurrent processes, and they show how to reduce the checking of correctness properties for rings of all sizes to the checking of these properties for rings of small sizes. By imposing stipulations on the organization of the system processes, [9] shows that the undecidable problem of testing a temporal property for an instance of any size is equivalent to model checking related systems whose number of components is up to a certain cutoff size. Based on the automatic calculation of candidate inductive assertions, [10] presents techniques to verify some properties of a parameterized system. For properties such as mutual exclusion, the verification can be abstracted to verifying them for a reduced system with limited number of components [11]. These works also include an extension of the Murphi verifier in [21] and the quotient model in [22].

We consider systems consisting of identical copies of the same component, and develop techniques to identify all sources where blocking or deadlock situations appear. From our results, the maximum number of module copies, under which the system is nonblocking or deadlock-free could be obtained directly. In order to model the interaction among module copies, the event set of the module template is partitioned into two subsets: global events and private events. A global event is shared by all module copies, while an instantiation of each private event can only belong to one single copy.

The module template in this paper is constructed to be a state-partition automaton with respect to the natural projection defined by global events. In a state-partition automaton, sets of states accept trajectories with the same projected image partition the state space. State-partition automaton has been used extensively in supervisory control [23], [24]. We develop in this paper techniques to identify blocking or deadlock situations by examining each subgraph of the template that corresponds to an element in such a partition.

To compute the maximum number of module copies with which the system stays nonblocking or deadlock-free, [22] proposes to add modules until these properties are violated. To verify systems consisting of an indefinite number of isomorphic module copies, the approaches in [21], [22] utilize extensive combinations of states before compressing them into abstract states. In contrast, our methods directly compute the maximum number of module copies with which the non-blocking or deadlock-free property holds. Since all possible blocking or deadlock sources can be identified within the state-partition automaton, this paper provides a tool for correcting design errors in these systems.

The rest of the paper is organized as follows. Concepts of system instantiation are introduced in Section II, and

Weilin Wang is with University of Shanghai for Science and Technology, China. He is also with the Institute of Transport Studies, Monash University, Australia. Rong Su and Liyong Lin are with the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore. Chaohui Gong is with University of Shanghai for Science and Technology, China. Emails: wllwang@usst.edu.cn (Weilin Wang), rsu@ntu.edu.sg (Rong Su), llin5@e.ntu.edu.sg (Liyong Lin), and chgong@usst.edu.cn (Chaohui Gong).

techniques for template refinement and precedence graph construction are provided in Section III. After presenting techniques to check nonblockingness and deadlock freeness in Section IV, conclusions are drawn in Section V. A preliminary version of this paper appeared in [25] without proofs.

II. MODULAR SYSTEM INSTANTIATED FROM A TEMPLATE

In this section, we explain how an isomorphic modular system is constructed using a template.

A deterministic finite automaton is represented by a 5-tuple $G = (X, E, f, x_0, X_m)$, where X is a finite set of states, E is a finite set of events, f is the transition function, x_0 is the initial state, and X_m is a set of marked states.

Definition 1. A template is an accessible deterministic automaton G over an event set E that is partitioned into a set of global events E^g and a set of private events E^p . $\square$

Example 1. Fig. 1 shows template $G = (X, E, f, x_0, X_m)$ for the running example. $x_0 = 0$ is the initial state. $E = \{a, b, c, e, f, o, p, q\}$ is partitioned into $E^g = \{e, f, o, p, q\}$ and $E^p = \{a, b, c\}$. A marked state in $X_m = \{4, 7, 8, 9, 11\}$ is denoted graphically by a double circle. $\blacksquare$

We extend f from E to E^* such that $f(x, se) := f(f(x, s), e)$ for all $se \in E^*$, where E^* is the set of all finite sequences of events in E . Given G , let $\mathcal{L}(G) = \{s \in E^* | f(x_0, s) \text{ is defined}\}$ be its closed behavior, and $\mathcal{L}_m(G) = \{s \in \mathcal{L}(G) | f(x_0, s) \in X_m\}$ be its marked behavior.

Given some $n \in \mathbb{N}$, for $i = 1, \dots, n$, define a parametrized isomorphism κ_i from $(\cup_i E_i)^*$ to $(E^g \cup \cup_i (E^p \times i))^*$ so that $\kappa_i(e) = e$ if $e \in E^g$, and $\kappa_i(e) = (e, i)$ if $e \in E^p$. κ^{-1} denotes the inverse of κ . We generalize κ to automaton G such that $\kappa_i(G) = (X, E_i, f_i, x_0, X_m)$ is obtained by replacing $(x, e, y) \in f$ by $(x, \kappa_i(e), y) \in f_i$.

The model of an isomorphic module system is a modular system consisting of modules $\kappa_i(G)$, $i = 1, \dots, n$ for some $n \in \mathbb{N}$. The dynamic of a modular system is determined by the parallel composition of all modules. See [26] for the definitions of modular systems and parallel composition. $G^n = \parallel_{i=1}^n \kappa_i(G)$ is the result of the parallel composition of $\kappa_i(G)$, $i = 1, \dots, n$, with respect to the set of global events E^g . With a slight abuse of notation, let $G^n = (X^n, \cup_{i=1}^n E_i, F^n, \vec{x}_0, X_m^n)$ in which both X^n and X_m^n in G^n only include reachable states. Given an $s \in \mathcal{L}(G^n)$, $F_i^n(\vec{x}_0, s)$ denotes the i^{th} element in $F^n(\vec{x}_0, s)$. A state $\vec{x} \in X^n$ is marked if $\vec{x} \in X_m^n$.

Example 2. This is a small motivation example. Consider the management system of a battery-powered autonomous vehicle served in a resort whose model $G = (X, E, f, x_0, X_m)$ is shown in Fig. 2. The resort has a set of four attractions $A = \{1, 2, 3, 4\}$. E is partitioned into $E^g = \{e, d_1, d_2, d_3, d_4\}$

and $E^p = \{c_1, c_2, c_3, c_4\}$. In E , e : the tourists enter the vehicle and, then, the doors of the vehicle are closed behind; for $i = 1, \dots, 4$, c_i : a tourist scan his/her ticket and select attraction i to visit; for $i = 1, \dots, 4$, d_i : after the vehicle visits three attractions in $A \setminus \{i\}$, it returns to the management center to change the battery. Suppose one ticket can only be used to select one attraction and the vehicle can only change its battery at the management center. Also suppose the vehicle can only visit up to three attractions before return to the management center to change its battery. State 0 is the only marked state that corresponds to vehicle being at the management center.

Intuitively, if four tourists or more are in the vehicle who select all four attractions, the vehicle does not have enough battery power to return to the management center, i.e., marked state 0 becomes unreachable. To guarantee the reachability of state 0, it is required that all states in $\{1, 2, 3, 4\}$ cannot be occupied simultaneously, i.e., the maximum number of attracts that can be selected in one round of trips is three. However, even if four or more tourists are in the vehicle, they can select no more than three attractions. Next, we develop formal procedures to analyze systems like this one. $\blacksquare$

Given sets of events E_1 and E_2 with $E_1 \supseteq E_2$, define natural projection $P : E_1^* \rightarrow E_2^*$ such that, for any $s \in E_1^*$, $P(s)$ is the sequence obtained from erasing occurrences of all events in $E_1 \setminus E_2$ in s . For example, if $E_1 = \{a, b, c\}$ and $E_2 = \{b, c\}$ and $s = aabcb$, then $P(s) = bcb$. The natural projections used in this paper include $P : (\cup_{i=1}^n E_i)^* \rightarrow (E^g)^*$, $P^g : E^* \rightarrow (E^g)^*$, $P_i : (\cup_{j=1}^n E_j)^* \rightarrow (E_i)^*$, and $P_{\downarrow} : (\cup_{i=1}^{n+1} E_i)^* \rightarrow (\cup_{i=1}^n E_i)^*$.

Given an arbitrary $s \in \mathcal{L}(G^n)$, let $s_i = \kappa^{-1}(P_i(s))$ for $i = 1, \dots, n$. Since, for all $x, x' \in X$ and $e \in E$, $f(x, e) = x'$ iff $f_i(x, \kappa_i(e)) = x'$, applying parallel composition, we can write $F^n(\vec{x}_0, s) = [f(x_0, s_1), \dots, f(x_0, s_n)]$. The sequences of local modules in G^n can run concurrently iff their projected images of global events are identical.

A directed graph $W = (V, A)$ is an ordered pair comprised of a set of vertices V and a set of directed edges $A \subseteq V \times V$. Given $s \in E^*$, $\bar{s} = \{t \in E^* | (\exists t' \in E^*) s = tt'\}$ is the set of all prefixes of s , and $|s|$ is the length (the number of event occurrences) of s . Given set A , $|A|$ is the cardinality of A . Given vector $\vec{x}$, x_i denotes the i^{th} entry of $\vec{x}$.

III. PRECEDENCE GRAPH

In this section, we prepare techniques for identifying blocking and deadlock in isomorphic module systems.

Without loss of generality, we assume G is a state-partition automaton with respect to projection P^g (see e.g., [27] for the formal definition). In a state-partition automaton with respect to a given projection, sets of states accepting trajectories with the same projected image form a partition of the state space.

Fig. 1. Template G for Example 1.

Fig. 2. G for Example 2.

If G is not a state-partition automaton, the algorithm for powerset construction in [28] can compute a deterministic automaton G^D with $P^g(\mathcal{L}(G)) = \mathcal{L}(G^D)$. Then, $G||G^D$ is a state-partition automaton with $\mathcal{L}(G||G^D) = \mathcal{L}(G)$ and $\mathcal{L}_m(G||G^D) = \mathcal{L}_m(G)$. The complexity of this construction is $\mathcal{O}(|E||X|2^{|X|})$. We use $G||G^D$ instead of G in all analyses.

We label states in a state-partition automaton G using nonnegative integers in $U = \{0, \dots, K\}$ for some K such that states have the same label iff they belong to the same element in the partition. With a slight abuse of notion, we write $(x, i) \in X$ where i is the label for x and use $(x_0, 0) \in X$ to denote the initial state.

Given $u \in U$, $\pi(u) = \{(x, u), e, (y, u)) \in f : e \in E^p\}$ is called the u -component of G , which includes all states labeled with u and transitions of private events among these states. Correspondingly, $X|_{\pi(u)} \subseteq X$ and $f|_{\pi(u)}$ denote the set of states of $\pi(u)$ and the set of transitions of $\pi(u)$, respectively.

Fig. 3. G^D for G in Fig. 1.

Example 3. Continue with Example 1, we construct state-partition automaton $G' = (X', E, f'(x_0, 0), X'_m) := G||G^D$ that corresponds to G in Fig. 1. Fig. 3 depicts G^D . $\pi(0)$ is shown in Fig. 4. Other components $\pi(u)$ do not contain transitions. The sets of states that correspond to $\pi(u)$, $u \in U \setminus \{0\}$ are as follows: $X'|_{\pi(1)} = \{(9, 1), (10, 1), (11, 1)\}$, $X'|_{\pi(2)} = \{(8, 2), (10, 2), (11, 2)\}$, $X'|_{\pi(3)} = \{(8, 3), (9, 3), (10, 3)\}$, $X'|_{\pi(4)} = \{(8, 4), (9, 4), (10, 4), (11, 4)\}$, and $X'|_{\pi(5)} = \{(8, 5), (9, 5), (11, 5)\}$. From now on, in all examples, state partition automaton $G = (X, E, f, (x_0, 0), X_m)$ refers to G' constructed in this example. ■

Lemma 1 states that, no matter which sequence a module takes to reach an $(x, u) \in X|_{\pi(u)}$, $u \in U$, it is possible for other modules to simultaneously reach all other states in $X|_{\pi(u)}$ when this module reaches (x, u) .

Lemma 1. Given an arbitrary $s \in \mathcal{L}(G)$, if $f((x_0, 0), s) \in X|_{\pi(u)}$ for some $u \in U$, then, $(\forall (y, u) \in X|_{\pi(u)})(\exists t \in \mathcal{L}(G))P^g(t) = P^g(s) \wedge f((x_0, 0), t) = (y, u)$. □

Lemma 2 states that all modules are always in the same $\pi(u)$ for some $u \in U$.

Lemma 2. Given G , for arbitrary $s, t \in \mathcal{L}(G)$, $P^g(s) = P^g(t)$ implies the existence of a unique $u \in U$ such that both $f((x_0, 0), s)$ and $f((x_0, 0), t)$ are in $X|_{\pi(u)}$. □

Using $\pi(u)$, $u \in U$, as an input, algorithms in [29] com-

pute maximal strongly-connected components and precedence relations among these components.

Given $\pi(u)$, let $\Omega(u)$ be the set of all maximal strongly-connected components in $\pi(u)$. $d_u(j)$, $j = 0, \dots, |\Omega(u)| - 1$, denotes an element in $\Omega(u)$.

For each $d_u(j) \in \Omega(u)$, $X|_{d_u(j)}$ is the set of all states in $d_u(j)$ and $f|_{d_u(j)}$ is the set of all transitions in $d_u(j)$. For any $d_u(k), d_u(l) \in \Omega(u)$, $d_u(k)$ is *precedent* to $d_u(l)$, denoted by $d_u(k) \prec d_u(l)$, if there exists $(x, u) \in X|_{d_u(k)}$, $(y, u) \in X|_{d_u(l)}$, and $e \in E^p$ such that $f((x, u), e) = (y, u)$. $D(u) = (\Omega(u), A(u))$, $u \in U$, denotes the precedence graph such that $(\forall d_u(k), d_u(l) \in \Omega(u))(d_u(k), d_u(l)) \in A(u) \Leftrightarrow d_u(k) \prec d_u(l)$. We distinguish $d_u(j)$ from other components of G by calling them nodes.

The following lemma is about properties of $D(u)$, $u \in U$, whose justifications can be found in [29].

Lemma 3. Given $\pi(u)$, $D(u)$ has the following properties.

- 1) Nodes are disjoint, i.e., $d_u(k) \neq d_u(l) \Rightarrow [X|_{d_u(k)} \cap X|_{d_u(l)} = \emptyset \wedge f|_{d_u(k)} \cap f|_{d_u(l)} = \emptyset]$.
- 2) Suppose $(x, u) \in X|_{d_u(k_i)}$ and $(y, u) \in X|_{d_u(k_j)}$ for some $d_u(k_i), d_u(k_j) \in \Omega(u)$ with $d_u(k_i) \neq d_u(k_j)$. Then, there exists an $s \in (E^p)^*$ with $f((x, u), s) = (y, u)$ iff there exists a path from $d_u(k_i)$ to $d_u(k_j)$ in $D(u)$, i.e., $(\exists d_u(k_1), d_u(k_2), \dots, d_u(k_j)) \in \Omega(u) d_u(k_i) \prec d_u(k_1) \prec \dots \prec d_u(k_j)$.
- 3) $D(u)$ is acyclic. □

Example 4. Continue with Example 3. In $\Omega(0)$, $d_0(0)$ is shown in Fig. 5. Nodes $d_0(1)$, $d_0(2)$, $d_0(3)$, $d_0(4)$, and $d_0(5)$ only contain respective state $(3, 0)$, $(4, 0)$, $(5, 0)$, $(6, 0)$, and $(7, 0)$ without transition. The precedence graph $D(0)$ includes $d_0(0) \prec d_0(i)$ for each $i = 1, \dots, 5$. All other nodes are listed as follows: $d_1(0) = \{(9, 1), \emptyset\}$, $d_1(1) = \{(10, 1), \emptyset\}$, $d_1(2) = \{(11, 1), \emptyset\}$, $d_2(0) = \{(8, 2), \emptyset\}$, $d_2(1) = \{(10, 2), \emptyset\}$, $d_2(2) = \{(11, 2), \emptyset\}$, $d_3(0) = \{(8, 3), \emptyset\}$, $d_3(1) = \{(9, 3), \emptyset\}$, $d_3(2) = \{(10, 3), \emptyset\}$, $d_4(0) = \{(8, 4), \emptyset\}$, $d_4(1) = \{(9, 4), \emptyset\}$, $d_4(2) = \{(10, 4), \emptyset\}$, $d_4(3) = \{(11, 4), \emptyset\}$, $d_5(0) = \{(8, 5), \emptyset\}$, $d_5(1) = \{(9, 5), \emptyset\}$, $d_5(2) = \{(11, 5), \emptyset\}$. ■

Fig. 4. $\pi(0)$ for G in Fig. 1.

IV. NONBLOCKINGNESS & DEADLOCK FREENESS

In this section, we present our main results for identifying blocking and deadlock situations.

The following notations will be used in this section. If $d_u(k)$ is not precedent to any other component in $\Omega(u)$, $d_u(k)$ is called a leaf-node. $\Phi(u)$ is the set of all leaf-nodes in $\Omega(u)$. Given $D(u)$, $d_u(k) \in \Omega(u)$ is a *singleton leaf-node* if $d_u(k)$ is a leaf-node and $f|_{d_u(k)} = \emptyset$. $\hat{\Phi}(u)$ is the set of all singleton leaf-nodes in $D(u)$. An event $e \in E^g$ is defined at $d_u(k) \in \Omega(u)$ if it is defined at some states in $X|_{d_u(k)}$. Let $\lambda(d_u(k)) = \{e \in E^g \mid (\exists v \in U)(\exists (r, v) \in X|_{\pi(v)})(\exists (w, u) \in X|_{d_u(k)})(\{(w, u), e, (r, v)\} \in f)\}$ be the set of global events defined at $d_u(k)$. Let $\Lambda(u) = \bigcap_{d_u(k) \in \hat{\Phi}(u)} \lambda(d_u(k))$ be the set of global events defined at every leaf node in $\Omega(u)$.

Definition 2. Given template G and integer $n \geq 1$, G^n is nonblocking for $s \in \mathcal{L}(G^n)$ if there exists a sequence t such that $st \in \mathcal{L}_m(G^n)$. G^n is nonblocking if, for any $s \in \mathcal{L}(G^n)$, there exists a sequence t such that $st \in \mathcal{L}_m(G^n)$. Moreover, template G is nonblocking if G^n is nonblocking for all $n \in \mathbb{N}$.

The system G^n is in a blocked state when it is not possible for all modules to reach marked states together. Note that, by [22], template G is nonblocking iff $G^{|X|}$ is nonblocking.

Lemma 4. If G^n is nonblocking, then, for all $k \leq n$, G^k is nonblocking. $\square$

Lemma 4 is true because all paths blocking G^k appear in G^{k+1} when $\kappa_{k+1}(G)$ mimics one of the modules of G^k .

Algorithm 1 checks the reachability of marked states by labelling nodes in precedence graphs $D(u)$, $u \in U$. In the algorithm, $\rho(d_u(i))$ denotes the set of labels that $d_u(i) \in \Omega(u)$ has. Given label k , $\mu(u, k) = \{d_u(i) \in \Omega(u) \mid k \in \rho(d_u(i))\}$ denotes the set of nodes in $\Omega(u)$ that have label k . The algorithm ensures that the window of opportunity for all modules to simultaneously reach some marked states is open iff nodes with modules in them have a common label.

In Algorithm 1, the foreach-loop on line 3 determines if a node contains a marked state or a module in this node is able to reach another node in the same macro state that has marked states. $1 \in \rho(d_u(i))$ means that a module in $d_u(i)$ is able to reach a marked state without leaving $\pi(u)$. The subroutine $\text{REACH}(D(u), \beta)$ computes the set of nodes in $\Omega(u)$ that can reach a node in β via a path in $D(u)$. The while-loop on line 9 computes maximal sets of nodes such that when all modules are exclusively in these nodes, they are able to simultaneously reach another set of nodes that all contain marked states. $I(v, k)$ on line 11 includes all pairs comprised of q -component, for some $q \in U$, and a global event e such that modules in $\pi(q)$ enter $\pi(v)$ if e occurs. β computed on lines 14 and 15 is a maximal set of nodes in $\pi(q)$ such that modules in $\pi(q)$ can reach those nodes labeled by k in $\pi(v)$ via respective sequences in which e is the only global event occurrence. If, in this case, all modules are able to simultaneously reach marked states when they are exclusively distributed in nodes labeled by k in $\pi(v)$, then all modules are able to simultaneously reach marked states when all of them are distributed in nodes in β .

Note that, if modules exclusively distributed in every node in a set of nodes are able to simultaneously reach some marked states, then modules exclusively distributed in nodes in a subset of that set are also able to simultaneously reach some marked states. Correspondingly, the if-condition on line 16 tests if there already exists a set of nodes labeled by the same label in $\pi(q)$ that is a superset or equal set of β . The algorithm only records β if no such set exists (line 18). Moreover, in $\pi(q)$, if there exists a set of nodes such that all nodes in that set are labeled by the same label k that is a subset of β , then, by implementing the foreach-loop on line 20, we remove k in the label set of each node in that set of nodes.

Example 5. Continue with Example 4, we demonstrate Algorithm 1. By line 2, $A = U = \{0, 1, 2, 3, 4, 5\}$. Correspondingly, for each $u \in U$, $\pi(u)$ has some marked states. Since $A \neq \emptyset$, the calculation enters the foreach-

Algorithm 1: TRACK-BLOCKING

Input: $G = (X, E, f, (x_0, 0), X_m)$ and $D(u)$ for all $u \in U$;
Output: Label sets $\rho(d_q(i))$ for each $d_q(i) \in \Omega(q)$ and each $q \in U$;

```

1   $\forall u \in U, l(u) \leftarrow 0$  and,  $\forall d_u(j) \in \Omega(u), \rho(d_u(j)) \leftarrow \emptyset$ ;
2   $A \leftarrow \{u \in U \mid X|_{\pi(u)} \cap X_m \neq \emptyset\}$ , and  $B \leftarrow \emptyset$ ;
3  foreach  $u \in A$  do
4       $l(u) \leftarrow l(u) + 1$ ;
5       $\beta \leftarrow \{d_u(i) \in \Omega(u) \mid X|_{d_u(i)} \cap X_m \neq \emptyset\}$ ;
6       $\beta \leftarrow \text{REACH}(D(u), \beta)$ ;
7       $\forall d_u(i) \in \beta, \rho(d_u(i)) \leftarrow \{l(u)\}$ ;
8       $B \leftarrow B \cup \{(u, l(u))\}$ ;
9  while  $B \neq \emptyset$  do
10     Pick up a  $(v, k) \in B, B \leftarrow B \setminus \{(v, k)\}$ ;
11      $I(v, k) \leftarrow \{(q, e) \in U \times E^g \mid ((\exists d_v(j) \in \Omega(v)) k \in \rho(d_v(j))) \exists ((x, q), e, (y, v)) \in f \text{ for some } (x, q) \in X|_{\pi(q)} \text{ and } (y, v) \in X|_{d_v(j)}\}$ ;
12     while  $I(v, k) \neq \emptyset$  do
13         Pick up a  $(q, e) \in I(v, k), \beta \leftarrow \emptyset$  and
14          $I(v, k) \leftarrow I(v, k) \setminus \{(q, e)\}$ ;
15          $\beta \leftarrow \{d_q(j) \in \Omega(q) \mid ((\exists (x, q) \in X|_{d_q(j)}) [((x, q), e, (z, v)) \in f \text{ for some } d_v(i) \in \Omega(v) \text{ s.t. } [(z, v) \in X|_{d_v(i)} \wedge k \in \rho(d_v(i))])])]\}$ ;
16          $\beta \leftarrow \text{REACH}(D(q), \beta)$ ;
17         if  $(\nexists m \leq l(q)) \beta \subseteq \mu(q, m)$  then
18              $l(q) \leftarrow l(q) + 1$ ;
19              $\forall d_q(j) \in \beta, \rho(d_q(j)) \leftarrow \rho(d_q(j)) \cup \{l(q)\}$ ;
20              $B \leftarrow B \cup \{(q, l(q))\}$ ;
21             foreach  $\mu(q, m) \subset \beta$  do
22                  $\forall d_q(i) \in \mu(q, m),$ 
23                  $\rho(d_q(i)) \leftarrow \rho(d_q(i)) \setminus \{m\}$ ;
24                  $B \leftarrow B \setminus \{(q, m)\}$ ;

```

Procedure $\text{REACH}(D(u), \beta)$

```

1  Set every element in  $\beta$  unflagged;
2  while there exists an unflagged element  $d_u(i) \in \beta$  do
3      foreach  $d_u(j) \in \Omega(u) \setminus \beta$  do
4          if  $d_u(j) \prec d_u(i)$  then
5               $\beta \leftarrow \beta \cup \{d_u(j)\}$  with  $d_u(j)$  unflagged;
6          Flag  $d_u(i) \in \beta$ ;
7  return  $\beta$ ;

```

loop on line 3. By line 4, $l(0) = 1$. By lines 5, 6, and 7, $\rho(d_0(0)) = \rho(d_0(2)) = \rho(d_0(5)) = \{1\}$ and $\rho(d_0(1)) = \rho(d_0(3)) = \rho(d_0(4)) = \emptyset$. Correspondingly, modules in states in $d_0(0)$ or $d_0(2)$ or $d_0(5)$ can reach marked states without leaving $\pi(0)$ but modules in states in $d_0(1)$ or $d_0(3)$ or $d_0(4)$ cannot reach marked states without leaving $\pi(0)$. By line 8, $A = \{1, 2, 3, 4, 5\}$ and $B = \{(0, 1)\}$. $(0, 1) \in B$ means that some nodes of $\pi(0)$ labeled by 1 need to be processed by the while-loop on line 9. After iterating the for-loop on line 3, $\rho(d_1(0)) = \rho(d_1(2)) = \{1\}$, $\rho(d_1(1)) = \emptyset$, $\rho(d_2(0)) = \rho(d_2(2)) = \{1\}$, $\rho(d_2(1)) = \emptyset$,

$\rho(d_3(0)) = \rho(d_3(1)) = \{1\}$, $\rho(d_3(2)) = \emptyset$, $\rho(d_4(2)) = \emptyset$, $\rho(d_4(0)) = \rho(d_4(1)) = \rho(d_4(3)) = \{1\}$, and $\rho(d_5(0)) = \rho(d_5(1)) = \rho(d_5(2)) = \{1\}$. Correspondingly, $B = \{(0,1), (1,1), (2,1), (3,1), (4,1), (5,1)\}$. Then, the calculation enters the while-loop on line 9. Suppose $(5,1) \in B$ is picked when line 10 is executed, Reset $B = \{(0,1), (1,1), (2,1), (3,1), (4,1)\}$. By line 11, $I(5,1) = \{(1,e), (2,f), (3,p), (5,o)\}$. Suppose $(1,e) \in I(5,1)$ is picked when line 13 is executed. Note that $(1,e) \in I(5,1)$ means that modules in some states in $X|_{\pi(1)}$ may reach some states in nodes labeled by 1 in $\pi(5)$ after $e \in E^s$ occurs. Set $I(5,1) = \{(2,f), (3,p), (5,o)\}$. By line 14, $\beta = \{d_1(0), d_1(1), d_1(2)\}$. The if-condition on line 16 is true. By lines 17 and 18, $\rho(d_1(0)) = \rho(d_1(2)) = \{1,2\}$, and $\rho(d_1(1)) = \{2\}$. Correspondingly, when all modules are exclusively in states in $X|_{d_1(0)} \cup X|_{d_1(1)} \cup X|_{d_1(2)}$, the door for them to reach marked states together is open. This fact is recorded by assigning new label 2 to each of $d_1(0)$, $d_1(1)$, and $d_1(2)$. By line 19, set $B = \{(0,1), (1,1), (1,2), (2,1), (3,1), (4,1)\}$. After iterating the foreach-loop on line 20, $\rho(d_1(0)) = \rho(d_1(1)) = \rho(d_1(2)) = \{2\}$ and $B = \{(0,1), (1,2), (2,1), (3,1), (4,1)\}$. After iterating the while-loop on line 12, $\rho(d_2(0)) = \rho(d_2(1)) = \rho(d_2(2)) = \{2\}$ and $\rho(d_3(0)) = \rho(d_3(1)) = \rho(d_3(2)) = \{2\}$. Correspondingly $B = \{(0,1), (1,2), (2,2), (3,2), (4,1)\}$. Iterate the while-loop on line 9 for $(1,2), (2,2), (3,2) \in B$ in this order. The first picks on line 13 for iterations regarding $(1,2), (2,2), (3,2) \in B$ are $(0,e), (0,f), (0,q)$, respectively. After these iterations, $\rho(d_0(0)) = \{2,3,4\}$, $\rho(d_0(1)) = \{3,4\}$, $\rho(d_0(2)) = \{2,4\}$, $\rho(d_0(3)) = \{2,3,4\}$, $\rho(d_0(4)) = \{2,3,4\}$, and $\rho(d_0(5)) = \{2,3\}$. Correspondingly, $B = \{(0,2), (0,3), (0,4), (4,1)\}$. Iterate the while-loop on line 9 for $(0,2), (0,3), (0,4) \in B$ in this order; no updates occur. After that, $B = \{(4,1)\}$. Iterate line 9 for $(4,1) \in B$, by line 11, $I(4,1) = \{(4,e), (4,f), (4,o), (4,p), (0,o), (0,p)\}$. Iterating the while-loop on line 12 for $(4,e), (4,f), (4,o), (4,p) \in I(4,1)$ in this order gives $\rho(d_4(0)) = \{1,3,4\}$, $\rho(d_4(1)) = \{1,2,4\}$, $\rho(d_4(2)) = \{2,3,4\}$, $\rho(d_4(3)) = \{1,2,3\}$. Correspondingly, $B = \{(4,2), (4,3), (4,4)\}$. Then, iterating the while-loop on line 12 for $(0,o), (0,p) \in I(4,1)$ in this order gives $\rho(d_0(0)) = \{2,3,4,5\}$, $\rho(d_0(1)) = \{3,4,5\}$, $\rho(d_0(2)) = \{2,4,5\}$, $\rho(d_0(3)) = \{2,3,4\}$, $\rho(d_0(4)) = \{2,3,4\}$, and $\rho(d_0(5)) = \{2,3,5\}$. Correspondingly, $B = \{(4,2), (4,3), (4,4), (0,5)\}$. Further iterations do not update results. ■

Lemma 5. Algorithm 1 terminates within finite iterations. □

Given $A \subseteq \Omega(u)$ for some $u \in U$, by Lemma 1, there exists an $s \in \mathcal{L}_m(G^n)$, for some $n \geq |A|$, such that s leads to the situation in which at least one module is in each node in A and all modules are in an element in A . Then, given a set of modules exclusively distributed in nodes in $A \subseteq \Omega(u)$ and every node in A has a module in it, the window for these modules to reach some marked states together is open iff all nodes in A have a common label, i.e., $\bigcap_{d_u(i) \in A} \rho(d_u(i)) \neq \emptyset$.

For all $u \in U$, define $\Upsilon(u) = \{A \subseteq \Omega(u) \mid A \neq \emptyset \wedge \bigcap_{d_u(i) \in A} \rho(d_u(i)) = \emptyset\}$ to be the set of subsets of $\Omega(u)$ such

that nodes in each of these subsets do not have a common label. Theorem 1 below identifies all situations in which the system is blocking and computes the minimum number of module copies with which the system is blocking.

Theorem 1. Given an $s \in \mathcal{L}(G^n)$ for some $n \in \mathbb{N}$, we write $f((x_0, 0), \kappa^{-1}(P_1(s))) = (x, u)$ for some $(x, u) \in X$. Then, G^n is blocking for s iff $\Upsilon(u) \neq \emptyset$ and after s at least one module is in each node in an $A \in \Upsilon(u)$, i.e., $[(\exists A \in \Upsilon(u))(\forall d_u(i) \in A)(\exists k \in \{1, \dots, n\})f((x_0, 0), \kappa^{-1}(P_k(s))) \in X|_{d_u(i)}]$. Moreover, G^n is nonblocking iff either, there exists $u \in U$ such that $\Upsilon(u) \neq \emptyset$, $n < n_0$ for

$$n_0 = \min\{|A| \mid (\exists u \in U) A \subseteq \Phi(u) \wedge A \in \Upsilon(u)\} \quad (1)$$

or, for all $u \in U$, $\Upsilon(u) = \emptyset$. □

Note that, (1) is undefined if, for all $u \in U$, $\Upsilon(u) = \emptyset$, i.e.,

$$(\forall u \in U) [\bigcap_{d_u(i) \in \Phi(u)} \rho(d_u(i)) \neq \emptyset] \quad (2)$$

By definition, template G is nonblocking iff (2) is true.

Example 6. Continue with Example 5. In $\pi(0)$, leaf nodes of states $(3,0)$, $(4,0)$, $(5,0)$, and $(7,0)$ do not have a common label. In $\pi(0)$, leaf nodes of states $(3,0)$, $(4,0)$, $(6,0)$, and $(7,0)$ do not have a common label. In $\pi(4)$, leaf nodes of states $(8,4)$, $(9,4)$, $(10,4)$, and $(11,4)$ do not have a common label. For $u = 1, 2, 3, 5$, leaf nodes in $\Phi(u)$ have a common label. By Theorem 1, n_0 , with which the system G^{n_0} is blocking, is 4. By definition, template G is blocking. ■

Theorem 2. Given G is not a state-partition automaton, the complexity for computing the minimum $n_0 \in \mathbb{N}$ with G^{n_0} being blocking or proving its nonexistence is $\mathcal{O}(|E|16^{|X|})$. □

Next, we discuss deadlock freeness.

Definition 3. Given a template G and an integer $n \geq 1$, G^n is deadlock free for $s \in \mathcal{L}(G^n)$ if there exists at least one event in $\bigcup_{i=1}^n E_i$ defined at $F^n(\vec{x}_0, s)$. G^n is deadlock free if for any $s \in \mathcal{L}(G^n)$, there exists at least one event in $\bigcup_{i=1}^n E_i$ defined at $F^n(\vec{x}_0, s)$. Moreover, template G is deadlock free if G^n is deadlock free for all $n \in \mathbb{N}$.

The system G^n is deadlocked when it is in a state at which no transition of any local module is active. Note that, by [22], template G is deadlock free iff $G^{|X|}$ is deadlock free.

Lemma 6. If G^n is deadlock free, then, for all $k \leq n$, G^k is deadlock free. □

Lemma 6 is true because all paths causing G^k to be deadlocked appear in G^{k+1} when $\kappa_{k+1}(G)$ only mimics one of the modules of G^k .

Given $A \subseteq \Omega(u)$ for some $u \in U$, by Lemma 1, there exists an $s \in \mathcal{L}(G^n)$ for some $n \geq |A|$, leading to the situation that at least one module is in each node in A and all modules are in an element in A . When a node in A is not a singleton leaf node, a private event could occur next and the system is not deadlocked after s . When all nodes in A are singleton leaf nodes, the system is not deadlocked after s iff a global event is defined at every node in A , i.e., $\bigcap_{d_u(i) \in A} \lambda(d_u(i)) \neq \emptyset$.

Define $\Psi(u) := \{A \subseteq \dot{\Phi}(u) \mid A \neq \emptyset \wedge \bigcap_{d_u(i) \in A} \lambda(d_u(i)) = \emptyset\}$. Theorem 3 below identifies all situations that the system is deadlocked and computes the minimum number of module copies with which the system is deadlocked.

Theorem 3. *Given an $s \in \mathcal{L}(G^n)$ for some $n \in \mathbb{N}$, we write $f((x_0, 0), \kappa^{-1}(P_1(s))) = (x, u)$. G^n is deadlocked for s iff after s at least one module is in each element in an $A \in \Psi(u)$, i.e.,*

$$(\exists A \in \Psi(u))(\forall d_u(i) \in A)(\exists k \in \{1, \dots, n\}) \\ f((x_0, 0), \kappa^{-1}(P_k(s))) \in X|_{d_u(i)}$$

Moreover, G^n is deadlock free iff either, there exists $u \in U$ such that $\Psi(u) \neq \emptyset$, $n < n_0$ for

$$n_0 = \min\{|A| \mid (\exists u \in U) A \in \Psi(u)\} \quad (3)$$

or, for all $u \in U$, $\Psi(u) = \emptyset$. $\square$

Note that (3) is undefined if, for all $v \in U$, $\Psi(v) = \emptyset$, i.e.,

$$(\forall u \in U)(\dot{\Phi}(u) \neq \emptyset \implies \bigcap_{d_u(k) \in \dot{\Phi}(u)} \lambda(d_u(k)) \neq \emptyset) \quad (4)$$

In this case, regardless of the number of modules, there is always an event that could occur next. By definition, template G is deadlock free iff (4) is true.

Example 7. *Continue with Example 4. As shown in Fig. 3, $U = \{0, \dots, 5\}$. For $u = 1, \dots, 5$, at least one global event is commonly defined at each node in $\dot{\Phi}(u)$ and, correspondingly, $\Psi(u) = \emptyset$. For $u = 0$, we have $\dot{\Phi}(0) = \{d_0(1), d_0(2), d_0(3), d_0(4), d_0(5)\}$ and $\Psi(0) = \{\dot{\Phi}(0)\}$. Thus, by (3) of Theorem 3, $n_0 = |\dot{\Phi}(0)| = 5$. By definition, template G is not deadlock free. ■*

Theorem 4. *Given G is not a state-partition automaton, the complexity for computing the minimum $n_0 \in \mathbb{N}$ with G^{n_0} having deadlock or proving its nonexistence is $\mathcal{O}(|E||X|2^{|X|})$. $\square$*

Example 8. *Continue with Example 2. It can be verified that G in Fig. 2 is a state-partition automaton. By Theorem 1, the minimum number of tourists that could cause the system to be blocked is 4. By Theorem 3, the minimum number of tourists that could cause the system to be deadlocked is also 4. ■*

Note that $|A|$ for A in (1) and $|A|$ for A in (3) are both upper bounded by $|X|$. So, monolithic systems G^n , $n = 1, \dots, |X|$, composed by parallel composition can be used to compute both n_0 in (1) and n_0 in (3). However, given G , the complexity for calculating $G^{|X|}$ is $|X|^{|X|}$, which is much higher than the exponential complexity of the approaches in this paper. Moreover, the results in this paper can be used to identify blocking or deadlock situations for arbitrary number of modules and eliminate these situations using supervisory control [30], while the monolithic model is restricted to specific number of modules.

V. CONCLUSION

We have proposed a novel approach for analyzing blocking or deadlock for systems consisting of an arbitrary number of isomorphic module copies without using explicit module composition. This paper has presented both procedures and theoretical proofs showing that all sources where blocking or

deadlock originates can be identified. These results are used to compute the maximum number of module copies with which the system is nonblocking or deadlock-free, test if the system is nonblocking or deadlock-free for an arbitrary number of module copies, and correct the relevant design errors.

ACKNOWLEDGMENT

This work is supported in part by National Natural Science Foundation of China Grant CNSF-61374058, Australian Research Council Grant DP-130100156, Shanghai Key Lab of Modern Optical System, The Program for Professor of Special Appointment (Eastern Scholar) at Shanghai Institute of Higher Learning, Singapore Ministry of Education Tier 1 Academic Research Grant RG84/13 (2013-T1-002-177), and National Research Foundation of Singapore Delta-NTU Corporate Lab Program (DELTA-NTU CORP LAB-SMA).

APPENDIX

Lemma 7. *Given synchronized system G^n , for all $s \in (\bigcup_{i=1}^n E_i)^*$, $s \in \mathcal{L}(G^n)$ iff, for all $i = 1, \dots, n$, there exists $s_i \in \mathcal{L}(G)$ such that $s_i = \kappa^{-1}(P_i(s))$ and $P^g(s_i) = P(s)$. $\square$*

Proof. The proof is from the definition of parallel composition and the definition of G^n . $\square$

Lemma 8 says that every sequence in $\mathcal{L}(G^n)$ can be extended to a sequence in $\mathcal{L}(G^{n+1})$ and the dynamic of newly added module in G^{n+1} copies the dynamic of a module in G^n .

Lemma 8. *Given template G and arbitrary $n \in \mathbb{N}$, for all $s \in \mathcal{L}(G^n)$, there always exists an $s' \in \mathcal{L}(G^{n+1})$ such that, in s' , the dynamics of newly added module $\kappa_{n+1}(G)$ mimics one of modules in G^n , i.e., $(\forall s \in \mathcal{L}(G^n))(\exists s' \in \mathcal{L}(G^{n+1}))[P_\downarrow(s') = s \wedge [(\forall i \in \{1, \dots, n\})P_i(s) = P_i(s')] \wedge [(\exists j \in \{1, \dots, n\})P_{n+1}(s') = \kappa_{n+1}(\kappa^{-1}(P_j(s)))]$. $\square$*

Proof. The proof is by directly applying Lemma 7. $\square$

Proofs of Lemmas 1 and 2. The proofs of these two lemmas can be found in [23], [24]. $\square$

Proofs of Lemmas 4 and 6. Suppose G^k is blocking (deadlocked). By Lemma 8, in G^{k+1} the dynamics of $\kappa_{k+1}(G)$ is always able to mimic that of $\kappa_k(G)$. This way, all the paths blocking (deadlock) G^k appear in G^{k+1} . Thus, adding new modules cannot eliminate any of blocking (deadlock) situations in G^k . $\square$

Proof of Lemma 5. The number of iterations of the foreach-loop on line 3 and the while-loop on line 12 are upper-bounded by $|A| \leq |U|$ and $|I(v, k)| \leq |E^g||U|$, respectively. After $\text{REACH}(D(u), \beta)$ is called on lines 6 and 15, the number of iterations of that subroutine is upper-bounded by $|\Omega(q)|^2$ for some $q \in U$. Restricted by the if-condition on line 16, the number of iterations of the while-loop on line 9 is upper-bounded by the number of all possible (v, k) that could be included in B , which cannot be larger than $|U|2^{\max\{|\Omega(u)| \mid u \in U\}}$. Thus, all loops in Algorithm 1 terminate finitely. $\square$

Proof of Theorem 1. We prove the first part as follows.

Since $s \in \mathcal{L}(G^n)$, by Lemma 7, for all $i = 1, \dots, n$, there exists $s_i \in \mathcal{L}(G)$ such that $s_i = \kappa^{-1}(P_i(s))$ and $P^g(s_i) = P(s)$. We write $f((x_0, 0), s_i) \in X|_{d_u(i)}$ for some $d_u(i) \in \Omega(u)$. Note

that, since s_i and s_j , $i \neq j$ could end in the same node, $d_u(i)$ and $d_u(j)$ can be the same node. Let $\Theta = \{d_u(1), \dots, d_u(n)\}$, where $d_u(i)$, $i = 1, \dots, n$, is the node that has state $f((x_0, 0), s_i)$.

To prove the “if” part, suppose $\mathcal{L}(G^n)$ is nonblocking for s . Then, there exists $st \in \mathcal{L}(G^n)$ such that $F^n((x_0, 0), st) \in X_m^n$, where $(x_0, 0)$ is the vector of n initial states $(x_0, 0)$. Since, for all $i = 1, \dots, n$, $s_i = \kappa^{-1}(P_i(s))$, by Lemma 7, there exists $t_i \in E^*$ such that $s_i t_i \in \mathcal{L}(G)$ with $P(st) = P^g(s_i t_i)$ and $\kappa^{-1}(P_i(st)) = s_i t_i$. We write $f((x_0, 0), s_i t_i) \in X|_{\pi(v)}$ for some $v \in U$. By parallel composition, $f((x_0, 0), s_i t_i) = F_i^n((x_0, 0), st)$. Since $F^n((x_0, 0), st) \in X_m^n$, for all $i = 1, \dots, n$, $f((x_0, 0), s_i t_i) \in X_m$. Moreover, for all $i = 1, \dots, n$, since $P(s) = P^g(s_i)$ and $P(st) = P^g(s_i t_i)$, $P(t) = P^g(t_i)$. For each $i = 1, \dots, n$, we write $s_i t_i = s_i t_i^0 e_i^1 t_i^1 e_i^2 \dots e_i^w t_i^w$ for some $t_i^z \in (E^P)^*$ and $e_i^z \in E^g$, $z = 1, \dots, w$. We write $s_i t_i^z = s_i t_i^0 e_i^1 t_i^1 e_i^2 \dots e_i^z t_i^z$, $z = 1, \dots, w$. We also write $f((x_0, 0), s_i t_i^z) \in X|_{d_v(i)}$, $i = 1, \dots, n$. Since, for all $i = 1, \dots, n$, $f((x_0, 0), s_i t_i) \in X_m$ and $s_i t_i = s_i t_i^w$, by lines 5, 6, and 7, nodes $d_v(i)$, $i = 1, \dots, n$, have at least one label in common. By lines 5, 6, and 7 and Lemma 3, this statement is also true for $f((x_0, 0), s_i t_i^{w-1} e_i^w)$, $i = 1, \dots, n$. By line 8, $(v, 1) \in B$. After the foreach-loop on line 3 is executed, the calculation enters the while-loop on line 9. By Lemma 3, after lines 14, 15, 17, and 18 are executed, this statement is true for $f((x_0, 0), s_i t_i^{w-1})$ and $f((x_0, 0), s_i t_i^{w-2} e_i^{w-1})$, $i = 1, \dots, n$. For $i = 1, \dots, n$ all together, repeat these arguments for $f((x_0, 0), s_i t_i^{w-2}), f((x_0, 0), s_i t_i^{w-3} e_i^{w-2}), \dots, f((x_0, 0), s_i)$ in this order. At the end, we find a label $l \in \bigcap_{d(i) \in \Theta} \rho(d(i), u)$. Thus, there does not exist $A \in Y(u)$ such that $A \subseteq \Theta$.

Note that, even if the if-condition on line 16 is false or some (q, m) is deleted from B by executing the foreach-loop on line 20, this statement remains true. Also note that, by line 19, the conditions of both while-loops are satisfied in the above reasoning and the algorithm does not terminate before these repeats are completed.

To prove the “only if”, suppose there does not exist $A \in Y(u)$ such that $A \subseteq \Theta$, which gives $\bigcap_{d(i) \in \Theta} \rho(d(i), u) \neq \emptyset$. Then, quoting Algorithm 1 in the same way as proving the “if” part. We have there exists $v \in U$, for all $i = 1, \dots, n$, there exists $t_i \in E^*$ such that $P^g(t_i) = P^g(t_1)$ and $f(f((x_0, 0), s_i), t_i) \in X_m \cap X|_{\pi(v)}$. Thus, for all $i = 1, \dots, n$, there exists $s_i t_i \in \mathcal{L}(G)$ such that $P^g(s_i t_i) = P^g(s_1 t_1)$ and $f((x_0, 0), s_i t_i) \in X_m \cap X|_{\pi(v)}$. Construct a sequence $t \in (\bigcup_{i=1}^n E_i)^*$ such that for all $i = 1, \dots, n$, $P^g(s_i t_i) = P(st)$ and $s_i t_i = \kappa^{-1}(P_i(st))$. By Lemma 7, $st \in \mathcal{L}(G^n)$. Since, for all $i = 1, \dots, n$, $s_i t_i \in \mathcal{L}_m(G)$, $F^n((x_0, 0), st) \in X_m^n$.

Next, we prove the second part.

We first prove the case that there exists $u \in U$ with $Y(u) \neq \emptyset$.

Suppose $\pi(v)$, $v \in U$, is the component and $A' \subseteq \Phi(v)$ is the set of leaf nodes by which n_0 in (1) is computed. We write $A' = \{d_v(1), \dots, d_v(n_0)\}$. Then, by Lemma 1, for all $i = 1, \dots, n_0$, there exists $s_i \in \mathcal{L}(G)$ such that $f((x_0, 0), s_i) \in X|_{d_v(i)}$ and $P^g(s_1) = \dots = P^g(s_{n_0})$. We construct a sequence $s \in (\bigcup_{i=1}^{|A'|} E_i)^*$ from s_i , $i = 1, \dots, n_0$, such that, for all $i = 1, \dots, n_0$, $P^g(s_i) = P(s)$ and $s_i = \kappa^{-1}(P_i(s))$. By Lemma 7, $s \in \mathcal{L}(G^{n_0})$. Moreover, by parallel composition, for all $i = 1, \dots, n_0$, $F_i^{n_0}((x_0, 0), s) \in X|_{d_v(i)}$. Then, since $A' \in Y(v)$, by

the first part, $\mathcal{L}(G^{n_0})$ is blocking for s . By Lemma 4, for all $n \geq n_0$, G^n is blocking.

Pick an $s \in \mathcal{L}(G^{\hat{n}})$ with $\hat{n} = n_0 - 1$ and extend s to reach a marked state in $X_m^{\hat{n}}$ as follows. By Lemma 7, for all $i = 1, \dots, \hat{n}$, there exists $s_i \in \mathcal{L}(G)$ such that $P^g(s_i) = P(s)$ and $s_i = \kappa^{-1}(P_i(s))$. We write $f((x_0, 0), s_1) \in X|_{\pi(u)}$ for some $u \in U$. By Lemma 2, for all $i = 1, \dots, \hat{n}$, $f((x_0, 0), s_i) \in X|_{\pi(u)}$. By parallel composition, for all $i = 1, \dots, \hat{n}$, $f((x_0, 0), s_i) = F_i^{\hat{n}}((x_0, 0), s)$. By Lemma 3, for all $i = 1, \dots, \hat{n}$, there exists $w_i \in (E^P)^*$ such that $f(f((x_0, 0), s_i), w_i) \in X|_{d_u(i)}$ for some $d_u(i) \in \Phi(u)$. Then, for all $i = 1, \dots, \hat{n}$, $s_i w_i \in \mathcal{L}(G)$ and $P^g(s_i w_i) = P^g(s_1 w_1)$. Construct $sw \in (\bigcup_{i=1}^{\hat{n}} E_i)^*$ from $s_i w_i$, $i = 1, \dots, \hat{n}$, such that, for all $i = 1, \dots, \hat{n}$, $P^g(s_i w_i) = P(swz)$ and $s_i w_i = \kappa^{-1}(P_i(sw))$. By Lemma 7, $sw \in \mathcal{L}(G^{\hat{n}})$. Then, since $\hat{n} = n_0 - 1$, by (1) and the first part, $\mathcal{L}(G^{\hat{n}})$ is nonblocking for $sw \in \mathcal{L}(G^{\hat{n}})$, which implies $\mathcal{L}(G^{\hat{n}})$ is nonblocking for $s \in \mathcal{L}(G^{\hat{n}})$. Since $s \in \mathcal{L}(G^{\hat{n}})$ is arbitrary, $\mathcal{L}(G^{\hat{n}})$ is nonblocking. Therefore, by Lemma 4, $\mathcal{L}(G^n)$ is nonblocking for all $n < n_0$.

The proof for the case that, for all $u \in U$, $Y(u) = \emptyset$ is by showing that, for an arbitrarily given n and an arbitrary $s \in \mathcal{L}(G^n)$, we are always able to extend s to reach a marked state in X_m^n , which is similar to the previous paragraph. $\square$

Proof of Theorem 2. The complexity of Algorithm 1 is determined by the product of the following three numbers of iterations: the number of iterations of the while-loop on line 9, the number of iterations of the while-loop on line 12, and the maximum number of iterations for implementing subroutine REACH() on line 15, the if-condition on line 16, and the foreach-loop on line 20. At most, the loop on line 9 has $|U|2^{\max\{|\Omega(u)| | u \in U\}}$ iterations, the loop on line 12 has $|U||E|$ iterations, the subroutine REACH() on line 15 has $(\max\{|\Omega(u)| | u \in U\})^2$ iterations, implementing the if-condition on line 16 needs $2^{\max\{|\Omega(u)| | u \in U\}}$ iterations, and the foreach-loop on line 20 has $2^{\max\{|\Omega(u)| | u \in U\}}$ iterations. Thus, the complexity of Algorithm 1 is $\mathcal{O}(|U|^2 |E| 4^{\max\{|\Omega(u)| | u \in U\}})$. Moreover, given outputs of Algorithm 1, the complexity of computing n_0 in (1) or proving its nonexistence is $\mathcal{O}(|U| 2^{\max\{|\Omega(u)| | u \in U\}})$. Note that $U \leq 2^{|X|}$ and $(\forall u \in U) |\Omega(u)| \leq |X|$. Therefore, the overall complexity of computing n_0 using Theorem 1 or proving its nonexistence is the maximum of the complexity of computing $G|G^D$, the complexity of Algorithm 1, the complexity of computing $D(u)$ for all $u \in U$, and the complexity of implementing (1), which is $\mathcal{O}(|E| 16^{|X|})$. $\square$

Proof of Theorem 3. The Proof of the first part is directly by Definition 3, parallel composition, and Lemma 7.

The Proof of the second part is as follows.

We first prove the case that there exists $u \in U$ such that $Y(u) \neq \emptyset$. For n_0 computed by (3), we show that $\mathcal{L}(G^{n_0})$ is not deadlock free as follows. Let $A \subseteq \Phi(u)$ for some $u \in U$ be the set of singleton leaf nodes by which n_0 is computed. We write $A = \{d_u(1), \dots, d_u(n_0)\}$ and $d_u(i)$ associates with respective state (y_i, u) . Since G is reachable, there exists $s_1 \in \mathcal{L}(G)$ with $f((x_0, 0), s_1) = (y_1, u)$. Then, by Lemma 1, for all $j = 2, \dots, k$, there exists $s_j \in \mathcal{L}(G)$ with $f((x_0, 0), s_j) = (y_j, u)$ and $P^g(s_j) = P^g(s_1)$. By Lemma 7, there exists an $s \in \mathcal{L}(G^{n_0})$ such that, for all $i = 1, \dots, k$,

we have $P(s) = P^g(s_i)$ and $s_i = \kappa^{-1}(P_i(s))$. Repeatedly apply parallel composition along s , we have $F_i^{n_0}(\overrightarrow{(x_0, 0)}, s) = (y_i, u)$ for all $i = 1, \dots, n_0$. Since $\bigcap_{d(i) \in A} \lambda(d(i)) = \emptyset$, by parallel composition, no global event is defined after s . Moreover, since $A \subseteq \dot{\Phi}(u)$, for every $i = 1, \dots, n_0$, no private event in E_i^p is defined after s . Consequently, by Definition 3, $\mathcal{L}(G^{n_0})$ is not deadlock free. By Lemma 6, for all $n \geq n_0$, $\mathcal{L}(G^n)$ is not deadlock free.

Given an arbitrary $s \in G^{\hat{n}}$ for some $\hat{n} \in \mathbb{N}$, by Lemma 7, for every $i = 1, \dots, \hat{n}$, there exists $s_i \in \mathcal{L}(G)$ such that $P^g(s_i) = P(s)$ and $s_i = \kappa^{-1}(P_i(s))$. Since $P^g(s_i) = P(s)$ for all $i = 1, \dots, \hat{n}$, we have $P^g(s_i) = P^g(s_l)$ for all $i, l = 1, \dots, \hat{n}$. Then, by Lemma 2, there exists a $u \in U$ such that, for all $i = 1, \dots, \hat{n}$, $f(\overrightarrow{(x_0, 0)}, s_i) \in X|_{\pi(u)}$. Then, repeatedly apply parallel composition along sequence s , $F_i^{\hat{n}}(\overrightarrow{(x_0, 0)}, s) = f(\overrightarrow{(x_0, 0)}, s_i)$. Suppose $\hat{n} = n_0 - 1$. We consider $f(x_0, s_i) \in X|_{d_u(i)}$ for some $d_u(i) \in \Phi(u)$ for every $i = 1, \dots, \hat{n}$. Let $A = \{d_u(1), \dots, d_u(\hat{n})\}$. Since $\hat{n} < n_0$, by (3), $A \subseteq \dot{\Phi}(u)$ such that $\bigcap_{d_u(i) \in A} \lambda(d_u(i)) \neq \emptyset$ or $A \not\subseteq \dot{\Phi}(u)$. In the former case, a global event can occur after s . In the later case, some module has local event defined after s . By parallel composition, in both cases $\mathcal{L}(G^{\hat{n}})$ is not deadlocked after s . Since s is arbitrary, $\mathcal{L}(G^{\hat{n}})$ is deadlock free. Since $\hat{n} = n_0 - 1$, by Lemma 6, for all $n < n_0$, $\mathcal{L}(G^n)$ is deadlock free.

The proof for the case that, for all $u \in U$, $\Psi(u) = \emptyset$ is by showing that the system is not deadlocked for an arbitrarily given n and an arbitrary $s \in \mathcal{L}(G^n)$, which is similar to the previous paragraph. $\square$

Proof of Theorem 4. Given $\pi(u)$, $u \in U$, since the cardinality of the set of singleton leaf nodes is less than $|\Omega(u)|$, computing A with the smallest cardinality and $\bigcap_{d(i) \in A} \lambda(d(i)) = \emptyset$ can be executed by examining all subsets of $\dot{\Phi}(u)$ for all global events, whose cardinality is upper bounded by $|E|2^{\max\{|\Omega(u)| | u \in U\}}$. Note that $|U| \leq 2^{|X|}$ and $\max\{|\Omega(u)| | u \in U\} \leq |X|$. Given $D(u)$, $u \in U$, the complexity of implementing (3) is upper bounded by the product of above term and $|U|$, which gives $\mathcal{O}(|E|4^{|X|})$. Since the complexity of computing precedence relations for all $u \in U$ has lower order and the complexity of computing $G||G^D$ has the same order, the complexity of computing n_0 using Theorem 3 is $\mathcal{O}(|E|4^{|X|})$. $\square$

REFERENCES

- [1] E. Emmerson and A. Sistla, "Symmetry and model checking," *Formal Methods in System Design*, vol. 9, no. 1/2, pp. 105–131, Aug. 1996.
- [2] P. Kouvaros and A. Lomuscio, "Parameterised verification for multi-agent systems," *Artificial Intelligence*, vol. 234, pp. 152–189, 2016.
- [3] T. T. Johnson and S. Mitra, "Invariant synthesis for verification of parameterized cyber-physical systems with applications to aerospace systems," in *AIAA Infotech@ Aerospace (I@ A) Conference*, 2013, p. 4811.
- [4] L. Brim, M. Češka, and D. Šafránek, "Model checking of biological systems," in *Formal Methods for Dynamical Systems*. Springer, 2013, pp. 63–112.
- [5] G. D. Plotkin, N. Bjørner, N. P. Lopes, A. Rybalchenko, and G. Varghese, "Scaling network verification using symmetry and surgery," in *ACM SIGPLAN Notices*, vol. 51, no. 1. ACM, 2016, pp. 69–83.
- [6] E. Clarke, S. Jha, R. Enders, and T. Filkhorn, "Exploiting symmetry in temporal logic model checking," *Formal Methods in System Design*, vol. 9, no. 1/2, pp. 77–104, Aug. 1996.
- [7] E. A. Emerson, R. J. Treffer, and T. Wahl, "Reducing model checking of the few to the one," in *International Conference on Formal Engineering Methods*. Springer, 2006, pp. 94–113.
- [8] S. Jiang and R. Kumar, "Decentralized control of discrete event systems with specializations to local and concurrent systems," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 30, no. 5, pp. 653–660, Oct. 2000.
- [9] E. A. Emerson and V. Kahlon, "Reducing model checking of the many to the few," in *International conference on automated deduction*. Springer, 2000, pp. 236–254.
- [10] T. Arons, A. Pnueli, S. Ruah, Y. Xu, and L. Zuck, "Parameterized verification with automatically computed inductive assertions," in *Computer Aided Verification*. Springer, 2001, pp. 221–234.
- [11] A. Pnueli, J. Xu, and L. Zuck, "Liveness with $(0, 1, \infty)$ -counter abstraction," in *Computer Aided Verification*. Springer, 2002, pp. 107–122.
- [12] H. Flordal and R. Malik, "Modular nonblocking verification using conflict equivalence," in *Proc. of International Workshop on Discrete Event Systems 2006*, Ann Arbor, USA, 2006, pp. 100–106.
- [13] P. Pena, A. G. da Cunha, J. Cury, and S. Lafortune, "New results on the nonconflict test of modular supervisors," in *Proc. of International Workshop on Discrete Event Systems 2008*, Göteborg, Sweden, 2008, pp. 468–473.
- [14] E. W. Stark, "On the relations computable by a class of concurrent automata," in *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Programming Languages*. ACM, 1990, pp. 329–340.
- [15] P. Dömösi and C. Nehaniv, *Algebraic Theory of Automata Networks: A Introduction*, ser. Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, 2004.
- [16] E. Chen and S. Lafortune, "Dealing with blocking in supervisory control of discrete event systems," *IEEE Trans. Automatic Control*, vol. 36, no. 6, pp. 724–735, Jun. 1991.
- [17] R. Su, J. H. van Schuppen, J. E. Rooda, and A. T. Hofkamp, "Nonconflict check by using sequential automaton abstractions based on weak observation equivalence," *Automatica*, vol. 46, no. 6, pp. 968–978, Jun. 2010.
- [18] E. A. Emerson and K. S. Namjoshi, "Reasoning about rings," in *Proceedings of the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. ACM, 1995, pp. 85–94.
- [19] S. German and A. P. Sistla, "Reasoning about systems with many processes," *Journal of the ACM*, vol. 39, pp. 675–735, 1992.
- [20] S. Nazari and J. G. Thistle, "Blocking in fully connected networks of arbitrary size," *IEEE Trans. Automatic Control*, vol. 57, no. 5, pp. 1233–1242, 2012.
- [21] C. N. Ip and D. L. Dill, "Verifying systems with replicated components in $\text{Mur}\phi$," in *International Conference on Computer Aided Verification*. Springer, 1996, pp. 147–158.
- [22] K. Rohloff and S. Lafortune, "The verification and control of interacting similar discrete-event systems," *SIAM Journal on Control and Optimization*, vol. 45, no. 2, pp. 634–667, Jun. 2006.
- [23] M. P. Schützenberger, "On the definition of a family of automata," *Information and control*, vol. 4, no. 2-3, pp. 245–270, 1961.
- [24] H. Cho and S. I. Marcus, "On supremal languages of classes of sublanguages that arise in supervisor synthesis problems with partial observation," *Mathematics of Control, Signals, and Systems (MCSS)*, vol. 2, no. 1, pp. 47–69, 1989.
- [25] W. Wang, R. Su, and L. Lin, "On analysis of deadlock and blocking freeness in isomorphic module systems," in *Proc. of 2013 American Control Conference*, 2013.
- [26] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems – Second Edition*. Springer, 2007.
- [27] G. Jirásková and T. Masopust, "On properties and state complexity of deterministic state-partition automata," in *IFIP International Conference on Theoretical Computer Science*. Springer, sep 2012, pp. 164–178.
- [28] K. Schneider, *Verification of reactive systems: formal methods and algorithms*. Springer Science & Business Media, 2013.
- [29] E. W. Dijkstra, *A discipline of programming*. Prentice-hall Englewood Cliffs, 1976, vol. 1.
- [30] L. Lin, A. Stefanescu, W. Wang, R. Su, and W. M. Wonham, "Symbolic reachability analysis and maximally permissive entrance control for globally synchronized templates," *Automatica*, vol. 87, pp. 290–300, 2018.