Discrete Event Dyn Syst
DOI 10.1007/s10626-009-0070-y

# Modular Supervisory Control with Equivalence-Based Abstraction and Covering-Based Conflict Resolution

**Richard C. Hill · Dawn M. Tilbury · Stéphane Lafortune**

Received: 3 January 2008 / Accepted: 13 May 2009

**Abstract** A modular approach to control is one way to reduce the complexity of supervisory controller design for discrete-event systems (DES). A problem, however, is that modular supervisors can conflict with one another. This paper proposes requirements on coordinating filters that will resolve this conflict. Abstractions are employed to reduce the complexity of the filter construction. Our specific approach is unique in that it employs a conflict-equivalent abstraction that offers the potential for greater reduction in model size than those abstractions employed in previous works on conflict resolution. The resulting control implemented by the modular supervisors in conjunction with coordinating filters meeting the proposed requirements is shown to be safe and nonblocking. Approaches for constructing these filters are discussed and a methodology that implements deterministic coordinating filter control laws by nondeterministic automata is presented. The covering-based filter law construction methodology presented here is further demonstrated to provide less restrictive control than existing results on state-feedback supervisory control.

**Keywords** Modular supervisory control · Conflict resolution · Abstraction · State feedback

This work was supported in part by NSF grants CMS-05-28287 and EECS-0624821. The majority of this work was performed while the first author was also with the University of Michigan.

R. C. Hill (✉)
University of Detroit Mercy, Detroit, MI 48221-3038, USA
e-mail: hillrc@udmercy.edu

D. M. Tilbury · S. Lafortune
University of Michigan, Ann Arbor, MI 48109-2125, USA

D. M. Tilbury
e-mail: tilbury@umich.edu

S. Lafortune
e-mail: stephane@umich.edu

Published online: 04 June 2009

## 1 Introduction

It is well-know that one of the primary obstacles facing the application of existing supervisory control theory is the complexity that arises when it is applied to large-scale discrete-event systems (DES). This complexity arises due to the explosion of the state space that occurs when considering systems with even a modest level of concurrency. To address the complexity problem, hierarchical and modular approaches have been considered in the literature. The overall goal of this paper is to generate a set of modular supervisors and conflict-resolving filters to control the behavior of a given plant to satisfy a set of specifications in a nonblocking manner. A further goal is to limit the computational complexity necessary for constructing the supervisors and filters.

Hierarchical approaches (Zhong and Wonham 1990; Wong and Wonham 1996; Hubbard and Caines 1998) hide aspects of a model to generate a "high-level" system. A controller is then constructed from the abstracted DES thereby reducing complexity and often times improving understandability. The control generated for the high-level system is then mapped back to the "low-level" for application to the actual system. A limitation of this approach is that generally the monolithic system is built first and then abstracted. In many cases, the state space of the monolithic system is too large to construct.

Modular control (Ramadge and Wonham 1988; Lin and Wonham 1988; de Queiroz and Cury 2000) leverages the fact that DES models and specifications are often generated in a component-wise manner. The traditional approach to supervisory control would compose these models and specifications to construct one large monolithic supervisor. In this process of composition, the state space may explode. The modular approach to supervisory control avoids this growth in complexity by building a series of modular supervisors to meet each of the component specifications individually, rather than a single monolithic supervisor to meet all specifications simultaneously. Even though each modular supervisor might satisfy its given specification and allow its associated subplant to reach a "completion" state when acting by itself, it is possible the individual modular supervisors can interfere with one another, preventing some of the subplants from reaching completion. Each of the component specifications could have conflicting goals. It is possible to check a priori that the modular supervisors will not conflict with one another, but this verification is often computationally expensive because it relies on a global analysis of the monolithic system.

Some noteworthy work allows it to be verified locally that modules are nonconflicting by adding interfaces between the system's components (Leduc et al. 2005a, b; Hill et al. 2008). These structural restrictions often result in suboptimal control, but can greatly reduce the complexity of verification. These works do not provide algorithms for the construction of the interfaces, but rather rely on the designer's understanding of the system. Other works exist to reduce the complexity associated with verifying that modules are nonconflicting (Pena et al. 2006; Flordal and Malik 2006) through the use of incremental analysis and abstraction. A question that is left, however, and that we will address in this paper, is what to do if conflict among the modules is detected.

Some research combines aspects of modular control with the abstraction of hierarchical approaches (Schmidt et al. 2005; Malik et al. 2007; Hill and Tilbury 2008). These works achieve a greater level of complexity reduction as compared to the monolithic approach without ever building the monolithic system and without having to verify that the modules are nonconflicting. These gains are achieved by putting further requirements on the structure of the modular supervisors. These restrictions can limit the overall reduction in complexity that can be achieved. Other works that combine aspects of modular and hierarchical control (Wong and Wonham 1998; Feng and Wonham 2006) employ abstraction and add coordinators on top of the modular supervisors to resolve conflict.

Most of the above works utilize a language projection with the observer property of Wong and Wonham (1996). The work of Wong and Wonham (1998) does not assume a language projection is used, but does require of its mapping the observer property. The observer property guarantees that the abstraction maintains a type of observation equivalence. In this paper we will employ a less restrictive abstraction that maintains only conflict properties. The notion of conflict equivalence was first introduced in Malik et al. (2006) and shown in Malik et al. (2007) to offer the potential for greater reduction in model size than can be achieved by observation-equivalent abstractions. Conflict-equivalent abstractions were employed by Flordal and Malik (2006) to reduce the complexity of verifying that automata are nonconflicting. They were also employed in Malik et al. (2007) for constructing nonconflicting modular supervisors. The work of Malik et al. (2007), however, does not specify how to construct the supervisors based on the abstracted models.

The contribution of this paper is to propose a unique methodology for constructing filters to resolve conflict among closed-loop modules in systems where it is detected. In contrast to existing work, our approach explicitly states how to construct nonconflicting control laws using conflict-equivalent abstractions. We build off of the approach of Flordal and Malik (2006) that proposes to incrementally abstract and compose modules to reduce the complexity associated with verifying that they are nonconflicting. Like Flordal and Malik (2006), we leverage the additional reduction in model size that a conflict-equivalent abstraction achieves. We specifically propose a set of novel requirements on the conflict-resolving filter laws, that if met, guarantee safe nonblocking control when acting in conjunction with traditionally built modular supervisors. We then specify one approach for constructing filters that meet these proposed requirements. The filter construction algorithm itself represents a significant contribution in that it generates a covering-based feedback law in the presence of nondeterminism that results in less restrictive control than can be achieved by existing state-feedback approaches (Li 1991; Takai and Kodama 1998). The ability to handle nondeterministic models is necessitated by the fact that the conflict-equivalent abstraction can introduce nondeterminism.

The outline of the rest of this paper is as follows. Section 2 introduces the necessary background and concepts. Section 3 provides a procedure for resolving conflict assuming that filters exist, while Section 4 proves that deterministic filters meeting the given language-based requirements provide safe nonblocking control when acting in conjunction with traditionally built modular supervisors. Section 5 then generates a set of analogous state-based requirements that allow the deterministic filter laws to be represented by nondeterministic automata. These state-based requirements are then shown to be satisfied by construction for the covering-based filter laws presented in Section 6. Section 7 applies these results to a manufacturing example and Section 8 summarizes the contributions of this paper. Preliminary and partial versions of our results were presented in papers (Hill et al. 2008a, b).

## 2 Notation and preliminaries

In this work we will consider DES modeled by possibly nondeterministic automata. Each automaton generates a language representing the set of possible behaviors of the DES and is represented by the five-tuple $G = (Q, \Sigma_\tau, \delta, q_0, Q_m)$, where $Q$ is the set of states, $\Sigma_\tau = \Sigma \cup \{\tau\}$ is the set of events including the silent event $\tau$, $\delta : Q \times \Sigma_\tau \to 2^Q$ is the state transition function, $q_0 \in Q$ is the initial state, and $Q_m \subseteq Q$ is the set of marked states representing successful termination of a process. There are algorithms in this paper that remove states from an automaton such that an empty state set may result. In this case we obtain what is commonly referred to in the literature as the "empty automaton." Let $\Sigma_\tau^*$ be the set of all finite strings of elements of $\Sigma_\tau$, including the empty string $\varepsilon$. The function $\delta$ is extended to $\delta : Q \times \Sigma_\tau^* \to 2^Q$ in the natural way. The notation $\delta(q, s)!$ for any $q \in Q$ and any $s \in \Sigma_\tau^*$ denotes that $\delta(q, s)$ is nonempty. The notation $\Sigma_G(q)$ will represent the set of *feasible events* of state $q$ in automaton $G$, that is, those events $\sigma \in \Sigma_\tau$ for which $\delta(q, \sigma)!$. A string $s \in \Sigma_\tau^*$ will be said to be *accepted* by an automaton $G$ if $\delta(q_0, s)!$.

Let $P_\tau : \Sigma_\tau^* \to \Sigma^*$ be the natural projection that erases the silent event $\tau$ from strings $s \in \Sigma_\tau^*$. The *generated* and *marked languages* of $G$, denoted by $\mathcal{L}(G)$ and $\mathcal{L}_m(G)$ respectively, are defined by $\mathcal{L}(G) = \{P_\tau(s) \in \Sigma^* \mid \delta(q_0, s)!\}$ and $\mathcal{L}_m(G) = \{P_\tau(s) \in \Sigma^* \mid \delta(q_0, s) \cap Q_m \neq \emptyset\}$. For the string $s = ru \in \Sigma_\tau^*$ formed from the catenation of the strings $r$ and $u$, $r$ is called a prefix of $s$ and is denoted $r \leq s$. The notation $\overline{K}$ represents the set of all prefixes of strings in the language $K$, and is referred to as the *prefix-closure* of $K$.

An automaton is said to be *nonblocking* when from all of its reachable states a marked state can be reached. From a language point of view, this is defined as $\overline{\mathcal{L}_m(G)} = \mathcal{L}(G)$. If an automaton enters a state from which it cannot reach a marked state, the automaton is said to have *blocked*.

### 2.1 Supervisory control

Supervisory control of DES requires that the event set $\Sigma_\tau$ be partitioned into controllable and uncontrollable events, $\Sigma_\tau = \Sigma_c \dot{\cup} \Sigma_u$, where controllable events can be disabled and uncontrollable events cannot. Traditionally, the theory of supervisory control (Ramadge and Wonham 1989) has been developed for application to deterministic automata models that do not include the silent event $\tau$ and are characterized by the fact that any string can take the automaton to only a single state. Nondeterministic automata can arise due to abstraction where events are hidden by replacing them by the silent event $\tau$. Note that $\tau \in \Sigma_u$. Let $\Sigma_h \subseteq \Sigma$ represent the set of events that are hidden for a given automaton. We will define a supervisor, denoted $\mathcal{S}$, to be a mapping that, upon observation of a string generated by a plant $G$, outputs a list of events to be enabled. Since uncontrollable events must always be enabled, the mapping $\mathcal{S} : \mathcal{L}(G) \to 2^{\Sigma_\tau}$ implicitly includes all uncontrollable events. It is the goal of supervisory control to restrict the behavior of the uncontrolled plant to meet some given specification.

Given a set of allowed behaviors $K \subseteq \mathcal{L}_m(G)$ and the set of uncontrollable events $\Sigma_u \subseteq \Sigma$, the existence of a supervisor that can successfully restrict the operation of the plant within the behavior allowed by the specification is guaranteed by satisfaction of the following *language controllability* condition (Cassandras and Lafortune 2007):

$$\overline{K}\Sigma_u \cap \mathcal{L}(G) \subseteq \overline{K} \tag{1}$$

The operation of two automata together is captured via the *synchronous composition* (parallel composition) operator, $\|$. By representing the supervisor mapping $\mathcal{S}$ as an automaton $S$, and the open-loop plant as a separate automaton $G$, the closed-loop or supervised behavior of the system can be modeled using the synchronous composition operator, $S\|G$. Throughout this paper we assume all automata have the same event set $\Sigma_\tau$. When two automata operate concurrently they will synchronize on all events except $\tau$, as specified by the following definition of synchronous composition.

**Definition 1** The *synchronous composition* of two automata $G_1$ and $G_2$, where $G_1 = (Q_1, \Sigma_\tau, \delta_1, q_{01}, Q_{m1})$ and $G_2 = (Q_2, \Sigma_\tau, \delta_2, q_{02}, Q_{m2})$ is the automaton

$$G_1\|G_2 = (Q_1 \times Q_2, \Sigma_\tau, \delta, (q_{01}, q_{02}), Q_{m1} \times Q_{m2})$$

where the transition function $\delta : (Q_1 \times Q_2) \times \Sigma_\tau \to 2^{(Q_1 \times Q_2)}$ is defined for $q_1 \in Q_1, q_2 \in Q_2$, and $\sigma \in \Sigma_\tau$ as:

$$\text{if } \sigma = \tau,\ \ \delta((q_1, q_2), \sigma) = \begin{cases} \delta_1(q_1, \tau) \times \{q_2\} & \text{if } \delta_1(q_1, \tau)! \text{ and } \neg\delta_2(q_2, \tau)! \\ \{q_1\} \times \delta_2(q_2, \tau) & \text{if } \neg\delta_1(q_1, \tau)! \text{ and } \delta_2(q_2, \tau)! \\ (\delta_1(q_1, \tau) \times \{q_2\}) \cup (\{q_1\} \\ \quad \times \delta_2(q_2, \tau)) & \text{if } \delta_1(q_1, \tau)! \text{ and } \delta_2(q_2, \tau)! \end{cases}$$

$$\text{if } \sigma \in \Sigma,\ \ \delta((q_1, q_2), \sigma) = \ \delta_1(q_1, \sigma) \times \delta_2(q_2, \sigma) \quad \text{if } \delta_1(q_1, \sigma)! \text{ and } \delta_2(q_2, \sigma)!$$

$$\text{else } \ \delta((q_1, q_2), \sigma) \ \text{ is empty.}$$

In terms of their generated languages, $\mathcal{L}(G_1)\|\mathcal{L}(G_2) = \mathcal{L}(G_1) \cap \mathcal{L}(G_2)$. Also note that $\|$ is a commutative and associative operation. The plant $G$ and specification $E$ are modeled by deterministic finite state automata given in the following component-wise manner:

$$G = G_1\| \cdots \|G_n \text{ and } E = E_1\| \cdots \|E_p$$

In this paper, the notation $\Sigma_{rel}(G)$ will represent the set of relevant events in $G$, a concept that is defined below:

**Definition 2** For an automaton $G = (Q, \Sigma_\tau, \delta, q_0, Q_m)$, the *relevant* event set $\Sigma_{rel}(G)$ is defined as:

$$\Sigma_{rel}(G) = \Sigma - \{\sigma \in \Sigma \mid \forall q \in Q,\ \delta(q, \sigma) = \{q\}\}$$

In words, the relevant event set of an automaton $G$ is made up of those events that are not $\tau$ and that are not only self-looped at every state. Events that are not relevant cannot exist as transitions between distinct states.

Modular supervisors will be built in the sense of de Queiroz and Cury (2000) as shown below in Definition 3. Let $H_i$ be an automaton realization of a closed-loop subsystem $\mathcal{S}_i/G_i'$ consisting of a plant $G_i'$ under the control of the supervisor $\mathcal{S}_i$. The notation $\sup\mathcal{C}(K, L)$ represents the supremal controllable sublanguage of $K$ with respect to the prefix-closed language $L$ and the given set of uncontrollable events. The supervisor synthesis of Definition 3 provides that the automaton $H_i$ represents both the closed-loop behavior of the $i^{\text{th}}$ module and its associated supervisor since $\mathcal{S}_i/G'_i = H_i \| G'_i = H_i$.

**Definition 3**

$$\begin{aligned} G'_i &= \ \|_{j\in J_i}\ G_j, \text{where:} \\ J_i &= \{j \in \{1,\ldots,n\} \mid \Sigma_{rel}(G_j) \cap \Sigma_{rel}(E_i) \neq \emptyset\} \\ \mathcal{L}_m(H_i) &= \sup\mathcal{C}(\mathcal{L}(E_i) \cap \mathcal{L}_m(G'_i), \mathcal{L}(G'_i)) \\ \mathcal{L}(H_i) &= \overline{\mathcal{L}_m(H_i)} \end{aligned}$$

In the above, the definition of the allowable language as $\mathcal{L}(E_i) \cap \mathcal{L}_m(G'_i)$ results in a *nonmarking* supervisor. That is, the supervisor does not affect the marking of the uncontrolled plant. Additionally, given that the uncontrolled plant is nonblocking, $\overline{\mathcal{L}_m(G'_i)} = \mathcal{L}(G'_i)$, this formulation will result in a nonblocking closed-loop subsystem. The results of this paper will be stated in terms of the closed-loop modules, $H_i$. Since the results do not depend on the supervisor employed, supervisors can be constructed in a manner different from Definition 3. If there are plant modules $G_j$ that do not share relevant events with any of the specifications $E_i$, they are treated as closed-loop modules $H_i$ on their own, without any additional supervision. Let $\{H_1, H_2, \ldots, H_q\}$ be the resulting set of automata representing the closed-loop modules.

The conjunction of modular supervisors succeeds in satisfying all of the component specifications, but does not guarantee nonblocking. In order to accomplish this goal, we need that the closed-loop subsystems $H_i$ be nonconflicting. A set of automata $H_1, H_2, \ldots, H_q$ is *nonconflicting* if the synchronous composition $H_1 \| H_2 \| \ldots \| H_q$ is nonblocking. If a set of automata is nonconflicting, its corresponding set of marked languages $K_1, K_2, \ldots, K_q$ is also nonconflicting. A set of languages is defined as nonconflicting if $\overline{K_1} \cap \overline{K_2} \cap \ldots \cap \overline{K_q} = \overline{K_1 \cap K_2 \cap \ldots \cap K_q}$.

Unfortunately, a priori verification of the nonconflicting condition is generally quite expensive computationally. With this in mind, we will incrementally apply abstraction to our modular supervisors as was done in Flordal and Malik (2006). In this paper we additionally propose a methodology for resolving the conflict in systems where it is detected.

## 2.2 Equivalence reductions

Many types of equivalence relations can be employed for reducing the complexity of a model. Specifically, equivalence relations can be used to merge equivalent states, thereby reducing the state size of the model. In this paper we are interested in generating reduced models that preserve conflict properties, a notion introduced in Malik et al. (2006).

**Definition 4** (Malik et al. 2006) Two automata $H_1$ and $H_2$ are said to be *conflict equivalent* if for any third automaton $T$, $H_1$ and $T$ are nonconflicting if and only if

$H_2$ and $T$ are nonconflicting. If the automata $H_1$ and $H_2$ are conflict equivalent we write, $H_1 \simeq_{\mathrm{conf}} H_2$.

Note that conflict equivalence respects the property of blocking. Also, two languages can be defined as conflict equivalent in a similar manner to Definition 4. Using the fact that two automata being nonconflicting implies that their marked languages are nonconflicting means that $H_1 \simeq_{\mathrm{conf}} H_2$ implies $\mathcal{L}_m(H_1) \simeq_{\mathrm{conf}} \mathcal{L}_m(H_2)$. The converse, however, does not hold since the automaton representation of a given language is not unique. Specifically, two automata can generate the same language and not be conflict equivalent. This is demonstrated by the example presented in Fig. 1. In the figure, automata $G_1$ and $G_2$ generate the same language, but $G_1$ conflicts with $G_3$ while $G_2$ does not.

More generally, when nondeterministic automata models are considered, language equivalence is insufficient for capturing certain system properties. A very strong equivalence relation on states of automata is *bisimulation equivalence* (Milner 1989).

**Definition 5** Let there be two (possibly nondeterministic) automata $G_1 = (Q_1, \Sigma_\tau, \delta_1, q_{01}, Q_{m1})$ and $G_2 = (Q_2, \Sigma_\tau, \delta_2, q_{02}, Q_{m2})$. A binary relation $\sim \subseteq Q_1 \times Q_2$ on the states of these automata is said to be a *bisimulation equivalence* if for any $q_1 \in Q_1$ and $q_2 \in Q_2$, $q_1 \sim q_2$ implies that for any $\sigma \in \Sigma_\tau$:

(i) if $q_1' \in \delta(q_1, \sigma)$ then $\exists q_2'$ such that $q_2' \in \delta(q_2, \sigma)$ and $q_1' \sim q_2'$;

(ii) if $q_2' \in \delta(q_2, \sigma)$ then $\exists q_1'$ such that $q_1' \in \delta(q_1, \sigma)$ and $q_1' \sim q_2'$;

(iii) (bisimulation with marking) $q_1 \in Q_{m1}$ if and only if $q_2 \in Q_{m2}$.

The existence of a bisimulation relation between two states can be thought of as the two states having the same future behavior (including the silent event $\tau$). Two automata are said to be bisimulation equivalent if there exists a bisimulation relation between the states of these automata such that their initial states are related by the relation, $q_{01} \sim q_{02}$. If two automata are bisimulation equivalent, most properties of interest are consistent between the two, including blocking.

Another equivalence relation called *weak bisimulation* or *observation equivalence* (Milner 1989) has also been defined on the states of automata. A weak bisimulation

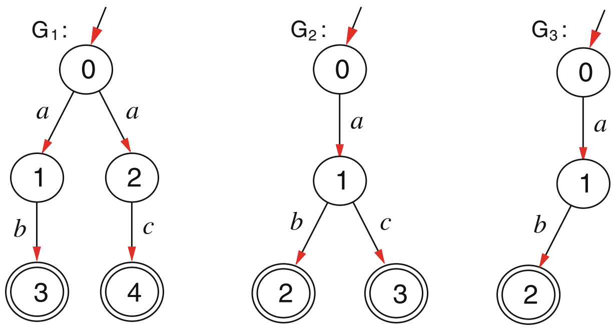

**Fig. 1** Illustrative example of nondeterminism (initial states are denoted by an *arrow* and marked states by *double circles*)

relation exists between two states if they both have the same "observed" futures. That is, all continuations from these states must be the same when the silent event $\tau$ is projected away. This concept of observation equivalence is similar to the notion of the *observer property* employed in Pena et al. (2006), Wong and Wonham (1998), Feng and Wonham (2006) and Hill and Tilbury (2008). Refer to Malik et al. (2007) for a more detailed examination of the relationship between conflict equivalence, observation equivalence, and projections with the observer property.

Conflict-equivalent abstraction in general provides a greater reduction in the state size of a model than either an observation-equivalent abstraction or a projection with the observer property (Malik et al. 2007). A drawback of a conflict-equivalent abstraction is that it is not as straightforward to implement; it is implemented via heuristics and a select set of rules (Flordal and Malik 2006; Flordal 2006). Also, a unique minimal reduction does not exist in general.

In this paper we will employ the notation $G_a$ to represent a conflict-equivalent abstraction of the automaton $G$. The abstracted automaton will specifically be generated in the following manner:

**Algorithm 1** *Conflict-Equivalent Abstraction*

Step 1: Given an automaton $G$, "hide" those events in the set $\Sigma_h$. One approach for constructing the set $\Sigma_h$ in the context of the approach of this paper is presented within Algorithm 2 of Section 3. These events are hidden by replacing their occurrences in the automaton $G$ by the silent event $\tau$ resulting in an intermediate automaton $G'$.

Step 2: Apply the conflict equivalence preserving rules that will be identified in Section 4.2 and are taken from Flordal and Malik (2006); Flordal (2006) to $G'$. The result of these rules is the reduced automaton $G_a$.

*Remark 1* While the intermediate automaton $G'$ and the abstraction $G_a$ are conflict equivalent per Definition 4 by construction, the original automaton $G$ and the reduced automaton $G_a$ are not. However, $G$ and $G_a$ do have equivalent conflict properties with respect to a third automaton $T$ if that automaton does not have any relevant events that were hidden in the process of generating $G'$. In other words, if $\Sigma_{rel}(T) \cap \Sigma_h = \emptyset$, then $G$ and $T$ are nonconflicting if and only if $G_a$ and $T$ are nonconflicting. In the remainder of this paper, we will only hide events in a manner consistent with this fact; that is, events are only hidden if they are not relevant to any remaining automata. In a slight abuse of notation, we will still write that $G \simeq_{\text{conf}} G_a$. Note also that $G$ and $G_a$ are consistent with respect to the property of blocking.

The software tool `Supremica` can be employed for generating conflict-equivalent abstractions (Supremica). Since each automaton in this paper has the same event set $\Sigma_\tau$, it is implied that any hidden events are self-looped at every state of the resulting automaton. However, we will not in general picture all of these self-looped transitions. Example 1 demonstrates a conflict-equivalent abstraction.

*Example 1* Consider automaton $G$ in Fig. 2 where event $f$ is not relevant to any other automata. Since $f$ is "local" to $G$, we can hide it by replacing all occurrences of $f$ by the silent event $\tau$ resulting in a new automaton $G'$. In $G'$, states 1 and 2 are not observation equivalent because state 1 has the observed continuation $bc$ while state

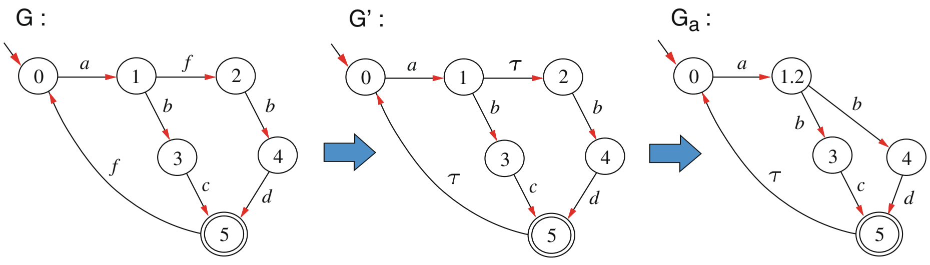

**Fig. 2** Illustrative example of a conflict-equivalent abstraction

2 does not. States 1 and 2, however, can be merged to achieve the conflict-equivalent automaton $G_a$. We will consider the abstraction $G_a$ to have the same alphabet as $G$, namely $\Sigma_\tau$, but will not picture an event (except $\tau$) if it is not relevant to the automaton. Therefore, one can imagine that $G_a$ has the event $f$ self-looped at every state. A consequence of this abstraction is that $G_a$ is nondeterministic.

In order to make the conflict-equivalent abstraction useful, we need to show that it is preserved under the synchronous composition operation $\|$. This result follows from Proposition 1 which is a reformulation of a result from Malik et al. (2006).

**Proposition 1** *Let $G$, $G_a$, and $H$ be automata. Also assume that any events hidden in the process of generating $G_a$ (Algorithm 1) are not relevant to $H$, that is, $\Sigma_{rel}(H) \cap \Sigma_h = \emptyset$. If $G \simeq_{conf} G_a$ then $G \| H \simeq_{conf} G_a \| H$. See Remark 1.*

The above proposition can be used to show that if no relevant events shared between $G_1$ and $G_2$ are hidden, then $G_1 \| G_2 \simeq_{\text{conf}} G_{1,a} \| G_{2,a}$. Analogous results can also be shown for conflict-equivalent languages.

## 3 Incremental conflict resolution using filters

The overall goal of this paper is to generate a set of modular supervisors and conflict-resolving filters that control the behavior of a given plant so that it satisfies a set of specifications in a nonblocking manner. A further goal is to limit the computational complexity of constructing the supervisors and filters. In this section we describe a procedure by which conflict is incrementally detected and resolved among closed-loop modules. Requirements on the conflict-resolving filters are presented in Section 4 and Section 5, while an algorithm for constructing the filters is presented in Section 6.

In the following procedure it will be assumed without loss of generality that the set of closed-loop modules $\{H_1, H_2, \ldots, H_q\}$ are addressed sequentially. More specifically, the procedure will begin with the automaton $H_1$ and after abstraction will be composed with an abstracted version of the automaton $H_2$. A filter automaton is added only if the resulting composition is blocking. The index $i$ increments on the number of automata that have been composed so far. Since a filter is constructed to resolve conflict in a given composition only when the composition is blocking, the filter index $j$ increments independently.

**Algorithm 2** *Conflict Resolution*

Step 1: Build modular supervisors according to Definition 3. Note that the supervisors may be constructed in other ways as long as the closed-loop automaton $H_i$ is employed in the subsequent steps. Any plant components that are not addressed by a specification are treated as additional closed-loop modules. Let $\{H_1, H_2, \ldots, H_q\}$ represent the resulting set of closed-loop modules. Section 4.4 identifies a special case where the full closed-loop automaton $H_i$ need not be employed.

Step 2: For each supervised subsystem $H_i$, generate a conflict-equivalent abstraction $H_{i,a}$ according to Algorithm 1. The set of hidden events in this step corresponds to those events relevant to only a single $H_i$, that is, $\Sigma_h = \Sigma - \bigcup_{i \neq j}(\Sigma_{rel}(H_i) \cap \Sigma_{rel}(H_j))$.

Step 3: Choose an abstracted subsystem $H_{1,a}$ with which to begin the procedure. Let $H'_{i,a} = H_{1,a}$ where $i = 1$ is the index for the individual closed-loop modules. Also initialize the filter index, $j = 1$.

Step 4: Choose one of the remaining abstracted subsystems, $H_{i+1,a}$, to compose with $H'_{i,a}$. This operation is performed via synchronous composition, $H'_{i,a} \| H_{i+1,a}$.

Step 5: If the composition $H'_{i,a} \| H_{i+1,a}$ is nonblocking, skip to Step 7, otherwise proceed to Step 6.

Step 6: At this point a coordinating filter law $\mathcal{H}_{filt,j} : \mathcal{L}(H'_{i,a} \| H_{i+1,a}) \longrightarrow 2^{\Sigma_\tau}$ must be generated to resolve the detected conflict in the preceding blocking composition. Otherwise stated, $\mathcal{H}_{filt,j}$ is built so that the controlled system $\mathcal{H}_{filt,j}/(H'_{i,a} \| H_{i+1,a})$ is nonblocking. Specific requirements for this filter will be presented in Section 4 and Section 5, and an approach for its construction will be proposed in Section 6.

Step 7: If all controlled subsystems have been addressed, then $i + 1 = q$ and the procedure is finished. Otherwise, more abstraction is performed according to Algorithm 1 and this overall procedure is repeated beginning at Step 4. The abstraction is performed in order to take advantage of the fact that some events are no longer relevant to any remaining abstracted subsystems and hence can now be hidden. More precisely, the set of hidden events becomes

$$\Sigma_h \leftarrow \Sigma_h \cup \left( \Sigma - \bigcup_{k > i+1} \Sigma_{rel}(H_{k,a}) \right) \tag{2}$$

and $H'_{i+1}$ is abstracted to generate $H'_{i+1,a}$ where $H'_{i+1} = \mathcal{H}_{filt,j}/(H'_{i,a} \| H_{i+1,a})$ if a filter had been constructed according to Step 6, otherwise, $H'_{i+1} = H'_{i,a} \| H_{i+1,a}$ if a filter had not been necessary. The index $i$ is then incremented before returning to Step 4. If a filter had been constructed in Step 6, then the filter index $j$ is also incremented at this time.

The process in Step 4 to Step 7 of abstracting and composing subsystems and adding filters as necessary to prevent blocking is repeated until there are no more subsystems remaining. The work of Flordal and Malik (2006) offers a sizable survey of heuristics for determining the ordering with which subsystems are addressed. The end result of this procedure is a set of filters that act in conjunction with the set of modular supervisors. If the controlled subsystems are nonconflicting on their own, no filters are needed. If a filter is generated that has an empty state set, it is possible that a nonempty filter can be found by abstracting away fewer details of the controlled subsystems, that is, by making $\Sigma_h$ smaller. A nonempty solution could also be found by addressing the modules in a different order. This approach to conflict resolution is the first to employ conflict-equivalent abstraction in the construction of coordinating filters for conflict resolution. The details of this coordinating level of control will be discussed in the following three sections. Section 4 provides a set of language-based requirements that are sufficient to guarantee safe nonblocking control when the filters are represented by deterministic automata. Section 5 then provides an analogous set of state-based requirements that allow the deterministic filter laws to be represented by possibly nondeterministic automata, while Section 6 introduces a methodology for constructing filters that satisfy these state-based requirements.

## 4 Language-based filter requirements

In the preceding section, Algorithm 2 was presented for incrementally resolving conflict among a set of supervised subsystems. In this section we will provide a set of language-based conditions on these deterministic filter laws and prove they are sufficient to provide safe nonblocking control when acting in conjunction with traditionally built modular supervisors. Within this process, we will examine the details of how a conflict-equivalent abstraction is generated. At the end of this section, we will also discuss how the closed-loop modules constructed as part of Algorithm 2 can be reduced to further improve the overall complexity of our approach.

In Algorithm 2, each filter law $\mathcal{H}_{filt,j}$ is built with respect to a blocking composition of abstracted automata that have preceded it. Here we will denote the associated blocking composition $B_{j,a}$. In the proofs that follow, we will assume that the control required by each filter law $\mathcal{H}_{filt,j}$ is realized by a <u>deterministic</u>, nonblocking automaton $H_{filt,j}$ and applied via synchronous composition, that is, $\mathcal{H}_{filt,j}/B_{j,a} = H_{filt,j} \| B_{j,a}$, therefore,

$$B_{j,a} = \left(H_{filt,j-1} \| \ldots \left(H_{filt,1} \| H_{1,a} \| H_{2,a}\right)_a \ldots\right)_a \| H_{i_j,a}$$

Furthermore, we will prove that deterministic filter automata meeting the following three requirements ($R1$–$R3$) will provide safe nonblocking control when acting in conjunction with the modular supervisors.

*Language-based filter requirements*

$R1$) $H_{filt,j} \| B_{j,a}$ is nonblocking
$R2$) $\mathcal{L}(H_{filt,j})$ is language controllable w.r.t $\mathcal{L}(B_{j,a})$
$R3$) $\Sigma_{rel}(H_{filt,j}) \cap \Sigma_h = \emptyset$

In the above, requirement $R3$ is meant to prevent a given filter law from trying to affect the occurrence of events that have been hidden. Since the set $\Sigma_h$ changes at each iteration, it is implicit in $R3$ that $\Sigma_h$ be the set taken at the time the filter $H_{filt,j}$ is constructed. We must now prove that these requirements are sufficient for guaranteeing safe nonblocking control can be realized.

In Section 5 we will present new state-based requirements that will allow our deterministic filter laws $\mathcal{H}_{filt,j}$ to be realized by possibly nondeterministic automata. These state-based requirements are also shown to be satisfied by filters constructed according to the algorithm of Section 6.

## 4.1 Nonblocking

Recall that the conjunction of modular supervisors satisfies the global specification $E$. Since the addition of filters only serves to further restrict the behavior of the system, the conjunction of filters and modular supervisors also provides safety. We will now demonstrate global nonblocking. In the following we will assume sequential ordering of the automata without loss of generality.

**Theorem 1** *Let $H_i$ be the automaton representing the behavior of the $i^{\text{th}}$ controlled subsystem where $i \in \{1, \ldots, q\}$. Also let there be filter automata $H_{filt,j}$, $j \in \{1, \ldots, k\}$, constructed according to Algorithm 2 and satisfying requirements $R1$ and $R3$. The conjunction of supervised subplants and filters $H_{filt,1} \| \ldots \| H_{filt,k} \| H_1 \| \ldots \| H_q$ is then nonblocking.*

*Proof*

- By the procedure of Section 3, automata are incrementally abstracted and composed. Assume the first two abstracted automata do not conflict. Therefore, $H_{1,a} \| H_{2,a}$ is nonblocking. Since $\Sigma_{rel}(H_1) \cap \Sigma_{rel}(H_2) \subseteq (\Sigma - \Sigma_h)$, $H_{1,a} \| H_{2,a} \simeq_{\text{conf}} H_1 \| H_2$ by Proposition 1. Therefore, $H_1 \| H_2$ is also nonblocking since conflict equivalence preserves blocking properties.
- Assume the addition of a third automaton also does not cause conflict, then $(H_{1,a} \| H_{2,a})_a \| H_{3,a}$ is nonblocking. Noting again that conflict equivalence holds across synchronous composition when shared relevant events are not abstracted away, $(H_{1,a} \| H_{2,a})_a \| H_{3,a}$ is conflict equivalent to $H_{1,a} \| H_{2,a} \| H_3$. Since those events made silent in the generation of $H_{1,a}$ and $H_{2,a}$ are not relevant to any of the remaining subsystems, Proposition 1 provides that $H_{1,a} \| H_{2,a} \| H_3 \simeq_{\text{conf}} H_1 \| H_2 \| H_3$. Furthermore, since equivalence relations are transitive, $(H_{1,a} \| H_{2,a})_a \| H_{3,a} \simeq_{\text{conf}} H_1 \| H_2 \| H_3$. Therefore, $H_1 \| H_2 \| H_3$ is also nonblocking.
- Assume for the first $i_1$ automata addressed, where $1 \leq i_1 \leq q$, no conflict is detected. Therefore, the resulting nested composition given below is nonblocking.

$$H'_{i_1} = \Big(\Big(\ldots\big(\big(H_{1,a} \| H_{2,a}\big)_a \| H_{3,a}\big)_a \| \ldots\Big)_a \| H_{i_1-1,a}\Big)_a \| H_{i_1,a} \tag{3}$$

  Following the logic above, the expression in Eq. 3 is conflict equivalent to $H_1 \| H_2 \| \ldots \| H_{i_1}$. Therefore, $H_1 \| H_2 \| \ldots \| H_{i_1}$ is nonblocking since the expression in Eq. 3 is.
- If $i_1 = q$, then there are no filters and we are done. Otherwise, the filter $H_{filt,1}$ is needed to resolve the conflict in $H'_{i_1,a} \| H_{i_1+1,a}$, where $H'_{i_1,a}$ is the further abstraction of the expression in Eq. 3. By $R1$, $H_{filt,1} \| H'_{i_1,a} \| H_{i_1+1,a}$ is nonblocking. By Proposition 1 and the above, $H'_{i_1,a} \| H_{i_1+1,a}$ is conflict equivalent to $H_1 \| \ldots \| H_{i_1+1}$. Therefore, $H_{filt,1} \| H'_{i_1,a} \| H_{i_1+1,a}$ is conflict equivalent to $H_{filt,1} \| H_1 \| \ldots \| H_{i_1+1}$ by

Proposition 1 since no events in $\Sigma_h$ at this point are relevant to $H_{filt,1}$ by $R3$. Therefore, $H_{filt,1} \| H_1 \| \ldots \| H_{i_1+1}$ is nonblocking since $H_{filt,1} \| H'_{i_1,a} \| H_{i_1+1,a}$ is.

- Let automata $H_{i_1+2,a}$ through $H_{i_2,a}$ be added such that the following expression is nonblocking, where $i_1 + 2 \leq i_2 \leq q$.

$$H'_{i_2} = \Big(\ldots \big(\big(H_{filt,1} \| H'_{i_1,a} \| H_{i_1+1,a}\big)_a \| H_{i_1+2,a}\big)_a \| \ldots\Big)_a \| H_{i_2,a} \tag{4}$$

  Following the logic employed above, it can then be shown that the expression in Eq. 4 is conflict equivalent to $H_{filt,1} \| H_1 \| \ldots \| H_{i_1+1} \| H_{i_1+2} \| \ldots \| H_{i_2}$, which is in turn nonblocking also.
- If $i_2 = q$, then there are no more filters and we are done. Otherwise, the filter $H_{filt,2}$ is needed to resolve the conflict in the composition $H'_{i_2,a} \| H_{i_2+1,a}$, where $H'_{i_2,a}$ is the further abstraction of the expression in Eq. 4. By $R1$, $H_{filt,2} \| H'_{i_2,a} \| H_{i_2+1,a}$ is nonblocking. By Proposition 1 and the above, $H'_{i_2,a} \| H_{i_2+1,a}$ is conflict equivalent to $H_{filt,1} \| H_1 \| \ldots \| H_{i_2+1}$. Therefore, $H_{filt,2} \| H'_{i_2,a} \| H_{i_2+1,a}$ is conflict equivalent to $H_{filt,2} \| H_{filt,1} \| H_1 \| \ldots \| H_{i_2+1}$ by Proposition 1 since no events in $\Sigma_h$ at this point are relevant to $H_{filt,2}$ by $R3$. Therefore, $H_{filt,2} \| H_{filt,1} \| H_1 \| \ldots \| H_{i_2+1}$ is nonblocking since $H_{filt,2} \| H'_{i_2,a} \| H_{i_2+1,a}$ is.
- Repeating this process, supervised subsystems and filters are added to the composition until they have all been addressed. The resulting composition $H_{filt,1} \| \ldots \| H_{filt,k} \| H_1 \| \ldots \| H_q$ is, therefore, shown to be nonblocking. □

## 4.2 Conflict-equivalence preserving rules

We now need to demonstrate that the control required of these filters is realizable. For deterministic automata, this corresponds to demonstrating language controllability. In order to demonstrate this, we will require that our conflict-equivalent abstraction satisfies the following property, where $P_h : \Sigma^* \rightarrow (\Sigma - \Sigma_h)^*$ is the natural projection that erases those events that have been hidden.

$$P_h(\mathcal{L}(H)) = P_h(\mathcal{L}(H_a)) \tag{5}$$

In words, we need that the original and reduced automata generate the same projected languages. We will specifically demonstrate which rules of Flordal and Malik (2006) and Flordal (2006) applied in Step 2 of Algorithm 1 achieve the property required by Eq. 5. We must first, however, introduce the following equivalence relation from Flordal and Malik (2006). This relation is employed in some of the reduction rules to follow. Here the notation $q \overset{\sigma}{\Rightarrow} q'$ will be employed to denote that there exists a string $s \in \Sigma_\tau^*$ such that $q' \in \delta(q, s)$ and $P_\tau(s) = \sigma$.

**Definition 6** Let $G = (Q, \Sigma_\tau, \delta, q_0, Q_m)$ be an automaton. The binary relation $\sim_{\text{inc}} \subseteq Q \times Q$ is defined such that $q \sim_{\text{inc}} q'$ if:

$$q_0 \overset{\varepsilon}{\Rightarrow} q \iff q_0 \overset{\varepsilon}{\Rightarrow} q';$$
$$\forall p \in Q \text{ and } \forall \sigma \in \Sigma \;:\; p \overset{\sigma}{\Rightarrow} q \iff p \overset{\sigma}{\Rightarrow} q'.$$

The relation $\sim_{\text{inc}}$ defines two states as being equivalent if they are reached in the same observed manner. In a sense, this relation is dual to observation equivalence where states with the same observed future are equated. The following two rules from Flordal and Malik (2006) employ the relation $\sim_{\text{inc}}$ to identify *conflict-equivalent states*. Two states are defined to be conflict equivalent if they have future behaviors that cannot be distinguished by conflict equivalence. The reduction of the automaton model is then achieved by merging conflict-equivalent states.

1) *Active Events Rule:* Two states that are equivalent with respect to $\sim_{\text{inc}}$ and have the same set of *active events* are conflict equivalent. The active event set of a state $q$ is defined here to be those events $\sigma \in \Sigma$ for which there exists a string $t \in \Sigma_\tau^*$ such that $\delta(q, t)!$ and $P_\tau(t) = \sigma$.
2) *Silent Continuation Rule:* Two states that are equivalent with respect to $\sim_{\text{inc}}$ and from which states without outgoing $\tau$ transitions can be reached via a *nonempty* sequence of $\tau$ transitions, are conflict equivalent.

Observation-equivalence provides another rule for identifying conflict-equivalent states since observation equivalence implies conflict equivalence (Flordal and Malik 2006).

3) *Observation Equivalence Rule:* Two states that are observation equivalent are also conflict equivalent.

The requirement presented in Eq. 5 can now be demonstrated for automata abstracted by applying the *Active Events Rule* and the *Silent Continuation Rule* based on their reliance on the binary relation $\sim_{\text{inc}}$.

**Proposition 2** *Let there be two (possibly nondeterministic) automata $H$ and $H_a$, where $H = (Q, \Sigma_\tau, \delta_h, q_0, Q_m)$ and $H_a = (Q_a, \Sigma_\tau, \delta_a, q_{0,a}, Q_{m,a})$ is an abstraction generated by Algorithm* 1. *If only the* Active Events Rule *and the* Silent Continuation Rule *are applied in the process of abstraction, then $P_h(\mathcal{L}(H_a)) = P_h(\mathcal{L}(H))$, where $P_h : \Sigma^* \to (\Sigma - \Sigma_h)^*$.*

*Proof*

- Following the first step of Algorithm 1, let $H' = (Q, \Sigma_\tau, \delta'_h, q_0, Q_m)$ be the automaton generated by replacing those transitions of $H$ that are in $\Sigma_h$ by the silent event $\tau$. It is then apparent that:

$$P_h(\mathcal{L}(H')) = P_h(\mathcal{L}(H)) \tag{6}$$

- Next, assume that $H_a$ is generated from $H'$ by first merging the single pair of distinct states $q, q' \in Q$. See Fig. 3. Since it is assumed that equivalent states are identified by only the *Active Events Rule* and the *Silent Continuation Rule*, we then have that $q \sim_{\text{inc}} q'$.
- Let $l \in \Sigma_\tau^*$ be a string accepted by $H'$ or $H_a$. We must show that $P_h(P_\tau(l)) \in P_h(\mathcal{L}(H')) \Leftrightarrow P_h(P_\tau(l)) \in P_h(\mathcal{L}(H_a))$. If this property holds for any $l$, then it holds for all $l$, thereby leading to the desired property that $P_h(\mathcal{L}(H_a)) = P_h(\mathcal{L}(H))$.

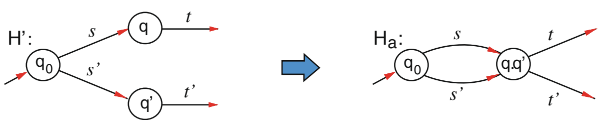

**Fig. 3** Example of an abstraction using an equivalence relation

**Case 1**

- Let $l$ be a string that does not pass through either of the states to be merged (in the case of $H'$) or through the merged state (in the case of $H_a$). It logically follows that $P_\tau(l) \in \mathcal{L}(H') \Leftrightarrow P_\tau(l) \in \mathcal{L}(H_a)$ and further that $P_h(P_\tau(l)) \in P_h(\mathcal{L}(H')) \Leftrightarrow P_h(P_\tau(l)) \in P_h(\mathcal{L}(H_a))$.

**Case 2**

- We will now examine those strings that do pass through one of the states to be merged (in the case of $H'$) or through the merged state (in the case of $H_a$).
- ($\subseteq$) Let $l$ be a string accepted by $H'$ that passes through at least one of the states to be merged, $q$ or $q'$, and may pass through them multiple times. Without loss of generality, let $q$ be the state of the pair to be merged that is last visited by the string $l$ and let $s \leq l$ be the prefix of $l$ that last reaches $q$. Therefore, $l = st$ where $q \in \delta'_h(q_0, s)$ and $\nexists v \in \Sigma_\tau^* - \{\varepsilon\}$ such that $v \leq t$ and $q \in \delta'_h(q, v)$.
  Merging $q$ and $q'$ means that if the string $l = st$ is accepted by $H'$, then $l = st$ is also accepted by $H_a$ (see Fig. 3). Therefore, $P_\tau(l) \in \mathcal{L}(H') \Rightarrow P_\tau(l) \in \mathcal{L}(H_a)$. Furthermore, $P_h(P_\tau(l)) \in P_h(\mathcal{L}(H')) \Rightarrow P_h(P_\tau(l)) \in P_h(\mathcal{L}(H_a))$. Note, the acceptance of other instances of the string $l$ by $H'$ will not affect this result.
- ($\supseteq$) Let $l$ be a string accepted by $H_a$ that passes through the merged state $q.q'$, possibly multiple times. Without loss of generality, let $l_1 \leq l$ be the prefix of $l$ that last reaches the merged state. Therefore, $l = l_1 l_2$ where $q.q' \in \delta_a(q_{0,a}, l_1)$ and $\nexists v \in \Sigma_\tau^* - \{\varepsilon\}$ such that $v \leq l_2$ and $q.q' \in \delta_a(q.q', v)$.
  Since $l = l_1 l_2$ is accepted by $H_a$, $l_1$ is a prefix of some string accepted by $H'$ that passes through $q$ or $q'$ and $l_2$ is some continuation that starts at $q$ or $q'$ and does not return to $q$ or $q'$. Let the strings $s, s', s'', \ldots$ be all of the traces that satisfy the assumptions on $l_1$ and let the continuations $t, t', t'', \ldots$ be all of the traces that satisfy the assumptions on $l_2$. Figure 3 shows one such example.
  Therefore, we must show that for all combinations of strings and continuations $\{st, s't, s''t, \ldots, st', s't', s''t', \ldots\}$ that are accepted by $H_a$, their projections are in the language generated by $H'$. Recall that $q \sim_{\text{inc}} q'$. Referring to Definition 6, this means that either $P_\tau(s) = P_\tau(s') = P_\tau(s'') = \ldots = \varepsilon$, or that $s = ru$, $s' = ru'$, $s'' = ru''$, $\ldots$ where $p = \delta'_h(q_0, r)$ and $P_\tau(u) = P_\tau(u') = P_\tau(u'') = \ldots = \sigma$ for some $\sigma \in \Sigma$. In either case, $P_\tau(s) = P_\tau(s') = P_\tau(s'') = \ldots$. The following equation, therefore, holds for any continuation $t$:

$$P_\tau(st) = P_\tau(s't) = P_\tau(s''t) = \ldots \tag{7}$$

  Since for any continuation $t$, one element of the set $\{st, s't, s''t, \ldots\}$ must be accepted by $H'$, Eq. 7 can be employed to show that $P_\tau(st)$, $P_\tau(s't)$, $P_\tau(s''t), \ldots \in \mathcal{L}(H_a) \Rightarrow P_\tau(st) = P_\tau(s't) = P_\tau(s''t) = \ldots \in \mathcal{L}(H')$. This logic holds for any continuation $t$ that satisfies the assumptions on $l_2$. Therefore, we have the desired result that $P_h(P_\tau(l)) \in P_h(\mathcal{L}(H_a)) \Rightarrow P_h(P_\tau(l)) \in P_h(\mathcal{L}(H))$.

- Taking Case 1 and Case 2 together, since it is true $\forall l$ accepted by $H'$ or $H_a$ that $P_h(P_\tau(l)) \in P_h(\mathcal{L}(H')) \Leftrightarrow P_h(P_\tau(l)) \in P_h(\mathcal{L}(H_a))$, we have that $P_h(\mathcal{L}(H')) = P(\mathcal{L}(H_a))$ if a single pair of states have been merged.
- If we further abstract $H_a$ by merging another pair of states that satisfy the binary relationship $\sim_{\mathrm{inc}}$, we can repeat the logic above. Therefore, we can show that $P_h(\mathcal{L}(H')) = P_h(\mathcal{L}(H_a))$ in general. This in conjunction with Eq. 6, therefore, proves our ultimate desired result. □

Similar logic to the above proposition can be employed to show that for two observation equivalent automata $H$ and $H_a$, Eq. 5 also holds. This fact is noted by Su and Thistle (2006). Other rules found in Flordal (2006) can be derived based on *Rules 1–3* mentioned above, therefore, they will also satisfy Eq. 5. In this paper, we will only apply rules that derive from the three rules mentioned above. If other rules that meet Eq. 5 can be identified, then they can be employed in our application as well.

Now recall that a reduced automaton $H_a$ has replaced all hidden events with the $\tau$ event then added self-loops at every state for each hidden event, therefore, none of the events that have been made silent are relevant to $H_a$. This in turn means that $P_h^{-1}(P_h(\mathcal{L}(H_a))) = \mathcal{L}(H_a)$. Here $P_h^{-1}$ is an inverse projection that expands the alphabet from $(\Sigma - \Sigma_h)$ to $\Sigma$. In terms of automata, $P_h^{-1}$ adds self loops at every state for all events in $\Sigma_h$. This logic along with Eq. 5 then provides that:

$$\mathcal{L}(H_a) = P_h^{-1}(P_h(\mathcal{L}(H_a))) = P_h^{-1}(P_h(\mathcal{L}(H))) \supseteq \mathcal{L}(H)$$

Defining the languages marked by these automata as $K = \mathcal{L}_m(H)$ and $K_a = \mathcal{L}_m(H_a)$ and assuming the automata are nonblocking, we then have that:

$$\overline{K_a} \supseteq \overline{K} \tag{8}$$

A final reduction rule of Flordal and Malik (2006), the *certain conflicts rule*, is not guaranteed to satisfy the containment of Eq. 8. However, this rule is only relevant to blocking automata and hence is not employed in our approach. The certain conflicts rule could be modified to add self-loops for events that have not been hidden in order to provide the containment of Eq. 8 without affecting conflict equivalence.

Repeated application of Eq. 8 can be used to generate the expression in Eq. 9 where each $K_i$ is either the marked language of a closed-loop module or a coordinating filter. Below we will show a few steps of the logic that leads us to Eq. 9.

Beginning with the expression $\overline{K_1} \cap \overline{K_2} \cap \ldots \cap \overline{K_{k-1}} \cap \overline{K_k}$, Eq. 8 can be used to show that $\overline{K_{1,a}} \supseteq \overline{K_1}$ and that $\overline{K_{2,a}} \supseteq \overline{K_2}$. Therefore,

$$\overline{K_{1,a}} \cap \overline{K_{2,a}} \cap \overline{K_3} \cap \ldots \cap \overline{K_{k-1}} \cap \overline{K_k} \supseteq \overline{K_1} \cap \overline{K_2} \cap \overline{K_3} \cap \ldots \cap \overline{K_{k-1}} \cap \overline{K_k}$$

Applying Eq. 8 again, we have that $(\overline{K_{1,a}} \cap \overline{K_{2,a}})_a \supseteq \overline{K_{1,a}} \cap \overline{K_{2,a}}$ and that $\overline{K_{3,a}} \supseteq \overline{K_3}$. Combining these results with the above expression, we then have that:

$$(\overline{K_{1,a}} \cap \overline{K_{2,a}})_a \cap \overline{K_{3,a}} \cap \ldots \cap \overline{K_{k-1}} \cap \overline{K_k} \supseteq \overline{K_1} \cap \overline{K_2} \cap \overline{K_3} \cap \ldots \cap \overline{K_{k-1}} \cap \overline{K_k}$$

Repeating the above logic then leads us to Eq. 9 that will be employed in the proofs of the next section.

$$\left(\left(\left(\ldots\left(\overline{K_{1,a}}\cap\overline{K_{2,a}}\right)_a\cap\overline{K_{3,a}}\right)_a\cap\ldots\right)_a\cap\overline{K_{k-1,a}}\right)_a\cap\overline{K_{k,a}}\supseteq\overline{K_1}\cap\overline{K_2}\cap\overline{K_3}\cap\ldots\cap\overline{K_{k-1}}\cap\overline{K_k} \tag{9}$$

## 4.3 Controllability

Equation 9 and the following well-known propositions will help to demonstrate that our filters acting in conjunction with the modular supervisors will be language controllable with respect to the global uncontrolled plant language $L$.

The first proposition is a result from Brandin et al. (2004) that is useful in our incremental approach since it demonstrates that if an allowable language is controllable with respect to a subset of plant subsystems, it will be controllable with respect to the global plant.

**Proposition 3** (Brandin et al. 2004) *Let $K$, $L \subseteq L' \subseteq \Sigma^*$ be languages. If $K$ is language controllable with respect to $L'$, then $K$ is language controllable with respect to $L$.*

The next proposition shows that the intersection of two nonconflicting controllable languages is itself controllable. This well-known result can be found in Cassandras and Lafortune (2007).

**Proposition 4** (Cassandras and Lafortune 2007) *Let $K_1$, $K_2$, and $L \subseteq \Sigma^*$ be languages and let $K = K_1 \cap K_2$. If $K_1$ and $K_2$ are nonconflicting and $K_1$ and $K_2$ are language controllable with respect to $L$, then $K$ is language controllable with respect to $L$.*

The following lemma demonstrates language controllability for the conjunction of a single filter and its associated blocking composition. This result will then be applied repeatedly to show language controllability of the conjunction of all modular supervisors and filters. We will denote the languages marked and generated by the filters $H_{filt,j}$ as $K_{filt,j} = \mathcal{L}_m(H_{filt,j})$ and $\overline{K_{filt,j}} = \mathcal{L}(H_{filt,j})$.

**Lemma 1** *Let $K_{filt}$, $K_1$, $K_2$, ..., $K_k$, and $L \subseteq \Sigma^*$ be languages and $L$ be prefix-closed. Let the subscript* a *represent an abstraction satisfying $\overline{K_a} \supseteq \overline{K}$. Also let $K_{filt}$, $K_1$, $K_2$, ..., $K_k$ be a nonconflicting set. Let $\Sigma_u \subseteq \Sigma$ be the set of uncontrollable events. If $K_{filt}$ is language controllable with respect to $L'_a = (\ldots(\overline{K_{1,a}} \cap \overline{K_{2,a}})_a \cap \ldots)_a \cap \overline{K_{k,a}}$ and $K_1$, $K_2$, ..., $K_k$ are each language controllable with respect to $L$, then $K_{filt} \cap K_1 \cap \ldots \cap K_k$ is language controllable with respect to $L$.*

*Proof*

- It is given that $K_{filt}$ is language controllable w.r.t. $L'_a$:

$$\overline{K_{filt}}\Sigma_u \cap L'_a \subseteq \overline{K_{filt}}$$

- Noting Eq. 9, intersection of both sides of the above with $L' = \overline{K_1} \cap \overline{K_2} \cap \ldots \cap \overline{K_k}$ gives us that:

$$\overline{K_{filt}}\Sigma_u \cap L' \subseteq \overline{K_{filt}} \cap L' \tag{10}$$

- It is further given that $K_1$, $K_2$, ..., $K_k$ are each language controllable w.r.t. $L$. Hence, $L'\Sigma_u \cap L \subseteq L'$. This fact combined with Eq. 10 gives us that:

$$\overline{K_{filt}}\Sigma_u \cap (L'\Sigma_u \cap L) \subseteq \overline{K_{filt}}\Sigma_u \cap L' \subseteq \overline{K_{filt}} \cap L'$$

  and substituting the expression for $L'$ we get

$$(\overline{K_{filt}} \cap \overline{K_1} \cap \ldots \cap \overline{K_k})\Sigma_u \cap L \subseteq \overline{K_{filt}} \cap \overline{K_1} \cap \ldots \cap \overline{K_k}$$

- Also recalling that it is given that the set $K_{filt}$, $K_1$, $K_2$, ..., $K_k$ is nonconflicting, we have our desired result:

$$(\overline{K_{filt} \cap K_1 \cap \ldots \cap K_k})\Sigma_u \cap L \subseteq \overline{K_{filt} \cap K_1 \cap \ldots \cap K_k} \qquad \square$$

The following theorem provides the language controllability result for the global system that we require.

**Theorem 2** *Let $K_i = \mathcal{L}_m(H_i)$ be the language representing the behavior of the $i^{\text{th}}$ subplant $L'_i = \mathcal{L}(G'_i)$ under the supervision of the $i^{\text{th}}$ modular supervisor where $i \in \{1, \ldots, q\}$. Furthermore, let there be filters $K_{filt,j}$, $j \in \{1, \ldots, k\}$, constructed as part of Algorithm 2 and satisfying requirements $R1$ and $R2$. The conjunction of supervised languages and filters $K_{filt,1} \cap \ldots \cap K_{filt,k} \cap K_1 \cap \ldots \cap K_q$ is then language controllable with respect to the global uncontrolled plant $L = \mathcal{L}(G) = L'_1 \cap \ldots \cap L'_q$.*

*Proof*

- Each supervised language $K_i$ is language controllable with respect to its associated subplant $L'_i$ by construction. Since $L \subseteq L'_i$ for each local subplant, each closed-loop language is also language controllable with respect to the global plant $L$ by Proposition 3.
- Let the sets $K_1, \ldots, K_{i_1}$ be nonconflicting, where $1 \leq i_1 \leq q$. Since each $K_i$ is language controllable with respect to $L$, $K_1 \cap \ldots \cap K_{i_1}$ is also language controllable with respect to $L$ by Proposition 4.
- If $i_1 = q$, then there are no filters and we are done. Otherwise, the filter $K_{filt,1}$ is needed to resolve the conflict in the composition $K'_{i_1,a} \cap K_{i_1+1,a}$, where $K'_{i_1,a} = ((\ldots(K_{1,a} \cap K_{2,a})_a \cap \ldots)_a \cap K_{i_1,a})_a$. By $R1$ and Theorem 1, the set $K_{filt,1}$, $K_1, \ldots, K_{i_1+1}$ is nonconflicting. Also by $R2$, $K_{filt,1}$ is language controllable with respect to $\overline{K'_{i_1,a}} \cap \overline{K_{i_1+1,a}}$, where $\overline{K'_{i_1,a}} = ((\ldots(\overline{K_{1,a}} \cap \overline{K_{2,a}})_a \cap \ldots)_a \cap \overline{K_{i_1,a}})_a$. Therefore, $K_{filt,1} \cap K_1 \cap \ldots \cap K_{i_1+1}$ is language controllable with respect to $L$ by Lemma 1.
- Let $K_{i_1+2}, \ldots, K_{i_2}$ be chosen such that the set $K_{filt,1}, K_1, \ldots, K_{i_1+1}, K_{i_1+2}, \ldots, K_{i_2}$ is nonconflicting, where $i_1 + 2 \leq i_2 \leq q$. Also, since $K_{filt,1} \cap K_1 \cap \ldots \cap K_{i_1+1}$ and each $K_i$ is language controllable with respect to $L$, Proposition 4 provides that $K_{filt,1} \cap K_1 \cap \ldots \cap K_{i_2}$ is language controllable with respect to $L$.

- If $i_2 = q$, then there are no more filters and we are done. Otherwise, the filter $K_{filt,2}$ is needed to resolve the conflict in the composition $K'_{i_2,a} \cap K_{i_2+1,a}$, where $K'_{i_2,a} = ((\ldots(K_{filt,1,a} \cap K'_{i_1,a})_a \cap K_{i_1+2,a} \ldots)_a \cap K_{i_2,a})_a$. By $R1$ and Theorem 1, the set $K_{filt,2}, K_{filt,1}, K_1, \ldots, K_{i_2+1}$ is nonconflicting. Also by $R2$, $K_{filt,2}$ is language controllable with respect to $\overline{K'_{i_2,a}} \cap \overline{K_{i_2+1,a}}$, where $\overline{K'_{i_2,a}} = ((\ldots(\overline{K_{filt,1,a}} \cap \overline{K'_{i_1,a}})_a \cap \overline{K_{i_1+2,a}} \ldots)_a \cap \overline{K_{i_2,a}})_a$. Hence, $K_{filt,1} \cap K_{filt,2} \cap K_1 \cap \ldots \cap K_{i_2+1}$ is language controllable with respect to $L$ by Lemma 1.
- Repeating this logic, supervised modules and filters are added to the composition until they have all been addressed. The resulting composition $K_{filt,1} \cap \ldots \cap K_{filt,k} \cap K_1 \cap \ldots \cap K_q$ is, therefore, shown to be language controllable with respect to $L$. □

Theorem 1 and Theorem 2, therefore, provide the desired result that filter laws represented by deterministic automata and built to satisfy requirements $R1$, $R2$, and $R3$ provide safe nonblocking control when acting in conjunction with the modular supervisors.

## 4.4 Supervisor reduction

A further improvement over Algorithm 2 presented in Section 3 is that in some instances a closed-loop module $H_i$ can be replaced by a reduction in the sense of Su and Wonham (2004), prior to the module being abstracted using a conflict-equivalent abstraction. For a plant $G'_i$ and supervisor automaton $S_i$, Su and Wonham (2004) shows how to generate a reduced supervisor $C_i$ that provides the same behavior as $S_i$ when acting on the given plant, that is, $C_i \| G'_i = S_i \| G'_i$.

In our procedure, we note that it may be possible to replace a closed-loop module $H_i = S_i \| G'_i$ by the reduced supervisor $C_i$ if the components making up the associated plant $G'_i$ are included in other plant modules $\{G'_j | j < i\}$ that have already been addressed; here it is assumed the modules are addressed in numerical order. Otherwise stated, $J_i \subseteq \cup_{j<i} J_j$, where $J_k$ is the set of indices of subplants in the composition that produces $G'_k$. The following proposition formalizes this idea for a situation involving three closed-loop modules and a single filter. Figure 4 can be referenced to help visualize the result. In the following, each of the closed-loop modules $H_i = S_i \| G'_i$ are supervised such that they satisfy the corresponding specification $E_i$. The corresponding subplants are defined as $G'_1 = G_1 \| G_2$, $G'_2 = G_3 \| G_4$, and $G'_3 = G_2 \| G_3$.

**Proposition 5** *Let $H_1$, $H_2$, and $H_3$ be closed-loop modules. Let $C_3$ be the reduction constructed in the manner of Su and Wonham (2004) corresponding to the module*

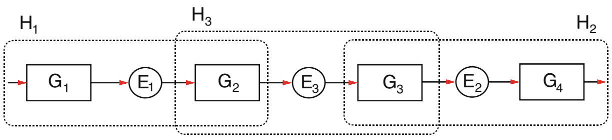

**Fig. 4** Three specification example for demonstrating supervisor reduction

*$H_3 = S_3 \| G'_3$ where $S_3$ and $G'_3$ are respectively the supervisor and plant. Also, let $H_{filt,1}$ be a filter automaton constructed to satisfy requirements $R1$ and $R3$ with respect to the blocking composition $(H_{1,a} \| H_{2,a})_a \| C_{3,a}$. If $H_1 \| H_2 = H_1 \| H_2 \| G'_3$, then $H_{filt_1} \| (H_{1,a} \| H_{2,a})_a \| C_{3,a} \simeq_{conf} H_{filt_1} \| H_1 \| H_2 \| H_3$.*

*Proof*

- Since requirements $R1$ and $R3$ are given, the same logic employed in the proof of Theorem 1 can be applied here to show that

$$H_{filt_1} \| (H_{1,a} \| H_{2,a})_a \| C_{3,a} \simeq_{\text{conf}} H_{filt,1} \| H_1 \| H_2 \| C_3 \tag{11}$$

- Since it is given that $H_1 \| H_2 = H_1 \| H_2 \| G'_3$,

$$H_{filt,1} \| H_1 \| H_2 \| C_3 = H_{filt,1} \| H_1 \| H_2 \| G'_3 \| C_3 \tag{12}$$

- Since $C_3$ is a reduced supervisor of $S_3$, we further have that $H_3 = S_3 \| G'_3 = C_3 \| G'_3$. Therefore,

$$H_{filt,1} \| H_1 \| H_2 \| G'_3 \| C_3 = H_{filt,1} \| H_1 \| H_2 \| H_3 \tag{13}$$

- Since equality implies conflict equivalence, by Eqs. 11, 12, and 13 we have the desired result that

$$H_{filt_1} \| (H_{1,a} \| H_{2,a})_a \| C_{3,a} \simeq_{\text{conf}} H_{filt_1} \| H_1 \| H_2 \| H_3 \qquad \square$$

The idea of employing supervisor reduction in conflict resolution was first employed by Feng (2007) in constructing a different sort of conflict-resolving coordinator. The real advantage of this result is that since the relevant event set of $C_i$ is smaller than the relevant event set of $H_i$, more reduction can take place in generating the conflict-equivalent abstractions of the preceding $H_j$.

## 5 State-based filter requirements

In this section we will propose a new set of state-based requirements that are analogous to the language-based requirements of Section 4 ($R1$–$R3$). These new requirements are necessary because of the nondeterminism introduced into our models by the process of abstraction. The language-based requirements are not sufficient to guarantee safe nonblocking control when applied to nondeterministic filter automata.

### 5.1 Supervisory control in the presence of nondeterminism

A difficulty that arises in considering how to construct filters is that each blocking composition $B_{j,a}$ is possibly nondeterministic because of the abstraction employed.

Determinization is not appropriate in this instance because it can change the blocking properties of the automaton model. In addition, it can result in a new model with a state space that is exponentially larger than the original nondeterministic model. To avoid the determinization process, we need to specify a set of requirements and a filter construction algorithm that addresses nondeterminism.

An alternate way to think about our problem is that each blocking composition $B_{j,a}$ is like our uncontrolled plant and we are trying to build a supervisor (the filter $H_{filt,j}$) to achieve a specification in a nonblocking manner. If we consider our specification to be the language $\Sigma^*$, then we have a situation where the "plant" is nondeterministic and the "specification" is deterministic. Of the existing research on supervisory control in the presence of nondeterminism, some address the situation where either only the plant is nondeterministic (Kumar and Shayman 1996; Park and Lim 2000) or only the specification is nondeterministic (Fabian and Lennartson 1996). Still other research allows the supervisors to be nondeterministic but only in application to partially observed deterministic plants and deterministic specifications (Inan 1993; Kumar et al. 2005). The works applicable to our situation (Kumar and Shayman 1996; Park and Lim 2000) demonstrate conditions for supervisor existence, but do not provide a supervisor construction algorithm.

Another, perhaps more intuitive, way to think about our situation is to consider our specification to be the trim of $B_{j,a}$. Therefore, we have a situation where our "plant" and "specification" are both nondeterministic. Research that addresses this situation is presented in Overkamp (1997), Zhou et al. (2006) and Heymann and Lin (1996). The work of Overkamp (1997) only addresses deadlock avoidance and its construction algorithm for building supervisors has exponential complexity. In the work of Zhou et al. (2006), conditions are presented under which a supervisor exists that can achieve behavior that is bisimilar to the given specification. A limitation of Zhou et al. (2006) is that supervisor synthesis is not addressed other than to mention that a search can be performed over the cartesian product of the plant and specification state spaces. The work of Heymann and Lin (1996) handles the situation of a nondeterministic plant and specification by converting the models to partially observed deterministic ones. At this point, traditional techniques for control under partial observation can be applied. This approach could be applied to our situation, but we hope to avoid the conversion process and the exponential complexity of the traditional techniques.

As existing works do not provide a methodology for constructing the filters required by Algorithm 2 with less than exponential complexity, we will propose our own approach for constructing deterministic filter laws that meet the requirements $R1$, $R2$, and $R3$. We will represent these deterministic control laws by possibly nondeterministic automata in order to keep the representation compact and in order to avoid determinizing the model. One problem that arises is that language controllability is insufficient to assess the realizability of a control law in regards to nondeterministic automata, as demonstrated by the following example.

*Example 2* Consider the automata in Fig. 5 where $G$ is the plant and $H$ is the specification and event $b$ is uncontrollable. Since the string $ab$ is in $\mathcal{L}(G)$ as well as in $\mathcal{L}(H)$, $\mathcal{L}(H)$ is language controllable with respect to $\mathcal{L}(G)$. However, the automaton $H$ still requires that the uncontrollable event $b$ be disabled at state 2.

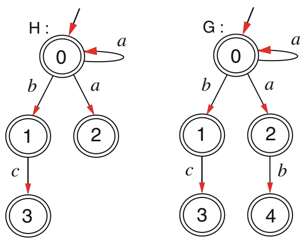

**Fig. 5** State controllability example

## 5.2 State controllability and observability

One solution to address the limitation of language controllability with respect to nondeterministic automata is to require a *state controllability* property similar to what was done in Fabian and Lennartson (1996) and Zhou et al. (2006). Language controllability requires that following an observed string $s$, if there is an uncontrollable continuation $\sigma$ allowed in the plant automaton, then at least one instance of $\sigma$ must be allowed following a string with the same observation $s$. With the state controllability property of Fabian and Lennartson (1996) and Zhou et al. (2006), it is rather required that the continuation $\sigma$ be allowed following every string with the observation $s$. In the case of subautomata as defined below, we can apply a slightly weaker notion of state controllability.

**Definition 7** $H = (Q_h, \Sigma_\tau, \delta_h, q_{0h}, Q_{mh})$ is a *subautomaton* of $G = (Q_g, \Sigma_\tau, \delta_g, q_{0g}, Q_{mg})$ denoted $H \sqsubseteq G$ if and only if

$$Q_h \subseteq Q_g, q_{0h} = q_{0g}, Q_{mh} = Q_{mg} \cap Q_h \text{ and}$$
$$p \in \delta_h(q, \sigma) \Rightarrow p \in \delta_g(q, \sigma).$$

The idea of this weaker notion is that following a string with an observation $s$, we will require that an instance of an uncontrollable event $\sigma$ must be allowed only if the event $\sigma$ is possible in that particular state of the plant automaton. That is, if there is a string with an observation $s$ that leads to a state in the plant automaton where $\sigma$ is not possible, then $\sigma$ does not have to be enabled at that state. Both state controllability properties imply language controllability. We will now formally define our state controllability property for a subautomaton. Recall that the silent event $\tau$ is uncontrollable.

**Definition 8** Let $\Sigma_u \subseteq \Sigma_\tau$. Subautomaton $H$ of $G$ is *state controllable in G* if

$$\text{for all } s \text{ and } q \in \delta_h(q_0, s), \text{ and all } \sigma \in \Sigma_u,\ \ \delta_h(q, \sigma) = \delta_g(q, \sigma)$$

State controllability as a property, however, is not sufficient to provide that the subautomaton $H$ represents a deterministic control law with respect to $G$. If the same observed string leads to two different states, those two states could require conflicting control actions. As such, we need a new observability-type requirement.

**Definition 9** Let $\Sigma_c \subseteq \Sigma$. Subautomaton $H$ of $G$ is *state observable in* $G$ with respect to the event set $\Sigma_c$ if

$$\text{for all } s \text{ and } q \in \delta_h(q_0, s), \text{ and all } \sigma \in \Sigma_c,\ \ P_\tau(s)\sigma \in \mathcal{L}(H) \Rightarrow \delta_h(q,\sigma) = \delta_g(q,\sigma)$$

Taken together, state controllability and state observability provide that $H$ represents a deterministic control law with respect to $G$. This is demonstrated formally by the following theorem that shows that a deterministic automaton $H_{obs}$ that generates and marks the same languages as $H$ will produce a result that is bisimulation equivalent to $H$ when it is composed with $G$. In essence, $H_{obs}$ can be considered a deterministic supervisor that achieves the specification represented by the nondeterministic automaton $H$ for the nondeterministic plant model $G$. Similar results for generating control for bisimulation equivalence can be found in Qin and Lewis (1991), Madhusudan and Thiagarajan (2002) and Tabuada (2004), but none demonstrate the following specific result. Here we implicitly assume the states of the automata are reachable.

**Theorem 3** *Let* $H = (Q_h, \Sigma_\tau, \delta_h, q_0, Q_{mh})$ *and* $G = (Q_g, \Sigma_\tau, \delta_g, q_0, Q_{mg})$ *be (possibly nondeterministic) automata such that* $H \sqsubseteq G$ *and* $H$ *is state controllable and state observable with respect to* $\Sigma_c$ *in* $G$*. Also let* $\Sigma_\tau = \Sigma_c \dot{\cup} \Sigma_u$ *where* $\tau \in \Sigma_u$*. If* $H_{obs} = (Q_{ho}, \Sigma_\tau, \delta_{ho}, p_0, Q_{mho})$ *is a deterministic automaton for which* $\mathcal{L}(H_{obs}) = \mathcal{L}(H)$ *and* $\mathcal{L}_m(H_{obs}) = \mathcal{L}_m(H)$*, then the synchronous composition* $H_{obs} \| G = (Q_\|, \Sigma_\tau, \delta_\|, (p_0, q_0), Q_{m\|})$ *is bisimulation equivalent to* $H$*.*

*Proof*

- Since $H \sqsubseteq G$, Definition 7 implies that $\mathcal{L}(H) \subseteq \mathcal{L}(G)$. Also, since $\mathcal{L}(H) = \mathcal{L}(H_{obs})$, $\mathcal{L}(H_{obs} \| G) = \mathcal{L}(H_{obs}) \cap \mathcal{L}(G) = \mathcal{L}(H)$. Therefore, $H_{obs} \| G$ and $H$ are equivalent in the sense that they generate the same language. Moreover, we will show in the following that they are also bisimulation equivalent.
- The automata $H_{obs} \| G$ and $H$ can be shown to be bisimulation equivalent by demonstrating that there exists a bisimulation relation between their initial states. Consider the following candidate relation $\sim$ over $(Q_{ho} \times Q_g) \times Q_h$:

$$(p, q_g) \sim q_h \Leftrightarrow q_g = q_h \text{ and there exists } t \in \Sigma_\tau^* \text{ such that}$$
$$\delta_{ho}(p_0, P_\tau(t)) = \{p\} \text{ and } q_h \in \delta_h(q_0, t).$$

- Therefore, we need to demonstrate that the relation $\sim$ exists for the initial states, $(p_0, q_0) \sim q_0$, and satisfies Points $(i)-(iii)$ of Definition 5. In words, Point $(i)$ means that if a string takes the state $(p_0, q_0)$ to a state $(p, q) \in Q_\|$, then the same string must take $q_0$ to the state $q \in Q_h$, where the relation $\sim$ on $(p, q)$ and $q$ exists and is a bisimulation. Point $(ii)$ likewise means that if a string takes the state $q_0$ to a state $q \in Q_h$, then the same string must take $(p_0, q_0)$ to the state $(p, q) \in Q_\|$, where the relation $\sim$ on $(p, q)$ and $q$ exists and is again a bisimulation. Point $(iii)$ further requires that all pairs $(p, q)$ and $q$ have the same marking.
- Since the existence of a bisimulation relation $\sim$ on the states $(p_0, q_0)$ and $q_0$ depends on the relation being a bisimulation for states subsequently reached by the same strings, we will perform this proof by induction on the length of an arbitrary string $t$.

*Base Step*: String $t$ has length 0, that is, $t = \varepsilon$.
(Point *i* and Point *ii*) Based on the definitions of the automata $Q_{\|}$ and $Q_h$, it is trivially satisfied that $(p_0, q_0) \in \delta_{\|}((p_0, q_0), \varepsilon)$ and $q_0 \in \delta_h(q_0, \varepsilon)$. Definition 1 also implies that $\delta_{ho}(p_0, \varepsilon) = \{p_0\}$ and $q_0 \in \delta_g(q_0, \varepsilon)$. Hence, $(p_0, q_0) \sim q_0$ also.
(Point *iii*) We now need to show that $(p_0, q_0) \in Q_{m\|}$ if and only if $q_0 \in Q_{mh}$.

> ($\Rightarrow$) Let $(p_0, q_0) \in Q_{m\|}$. This implies that $q_0 \in Q_{mg}$ by Definition 1 which in turn implies that $q_0 \in Q_{mh}$ by Definition 7 since we already have that $q_0 \in Q_h$.
> ($\Leftarrow$) Let $q_0 \in Q_{mh}$. Definition 1 then implies that $q_0 \in Q_{mg}$. Additionally, since $\mathcal{L}_m(H) = \mathcal{L}_m(H_{obs})$, $\varepsilon \in \mathcal{L}_m(H)$ implies that $\varepsilon \in \mathcal{L}_m(H_{obs})$ and thus $p_0 \in Q_{mho}$. Therefore, by Definition 1 $(p_0, q_0) \in Q_{m\|}$.

*Inductive Step*: String $t = s\sigma$ has length $n + 1$ and is the catenation of the string $s \in \Sigma_\tau^*$ and the event $\sigma \in \Sigma_\tau$.
(Point *i*) Let $\sigma \in \Sigma_\tau$. We need to show that $(p', q') \in \delta_{\|}((p, q), \sigma)$ implies that $q' \in \delta_h(q, \sigma)$ where $(p, q) \in \delta_{\|}((p_0, q_0), s)$ and $q \in \delta_h(q_0, s)$. Additionally, we need to show the relation $\sim$ exists on the pair $(p', q')$ and $q'$. Assuming $(p', q') \in \delta_{\|}((p, q), \sigma)$, $\delta_{ho}(p, P_\tau(\sigma)) = \{p'\}$ and $q' \in \delta_g(q, \sigma)$ by Definition 1. The following then shows that $q' \in \delta_h(q, \sigma)$:

> —If $\sigma \in \Sigma_u$, then $q' \in \delta_h(q, \sigma)$ since $q' \in \delta_g(q, \sigma)$ and $H$ is state controllable in $G$.
> —If $\sigma \in \Sigma_c$, then $P_\tau(s)\sigma = P_\tau(s\sigma) \in \mathcal{L}(H)$ since $P_\tau(s)\sigma = P_\tau(s\sigma) \in \mathcal{L}(H_{obs}\|G) = \mathcal{L}(H_{obs})$. Since it is also known that $q' \in \delta_g(q, \sigma)$ and $H$ is state observable in $G$, we then have that $q' \in \delta_h(q, \sigma)$.

This also provides that $(p', q') \sim q'$.
(Point *ii*) Now we need to show that $q' \in \delta_h(q, \sigma)$ implies $(p', q') \in \delta_{\|}((p, q), \sigma)$ and that the relation $\sim$ exists on the pair $(p', q')$ and $q'$. Since $H \sqsubseteq G$, $q' \in \delta_h(q, \sigma)$ implies $q' \in \delta_g(q', \sigma)$ by Definition 7.

> —If $\sigma = \tau$, then Definition 1 provides that $(p, q') \in \delta_{\|}((p, q), \sigma)$ and we can let $(p', q') = (p, q')$.
> —If $\sigma \neq \tau$, then $P_\tau(s)\sigma = P_\tau(s\sigma) \in \mathcal{L}(H) = \mathcal{L}(H_{obs})$. Since $H_{obs}$ is deterministic, we then have that $\delta_{ho}(p, \sigma)$ has a single element that we will call $p'$. Therefore, $(p', q') \in \delta_{\|}((p, q), \sigma)$ again by Definition 1.

In either case we have that $\delta_{ho}(p_0, P_\tau(s\sigma)) = \{p'\}$ and $(p', q') \sim q'$.
(Point *iii*) We now need to show that $(p, q) \in Q_{m\|}$ if and only if $q \in Q_{mh}$.

> ($\Rightarrow$) Let $(p, q) \in Q_{m\|}$. This implies that $q \in Q_{mg}$ by Definition 1 which in turn implies that $q \in Q_{mh}$ by Definition 7 since we already have that $q \in Q_h$.
> ($\Leftarrow$) Let $q \in Q_{mh}$. Definition 7 then implies that $q \in Q_{mg}$. Additionally, since $\mathcal{L}_m(H) = \mathcal{L}_m(H_{obs})$, $P_\tau(s) \in \mathcal{L}_m(H)$ implies that $P_\tau(s) \in \mathcal{L}_m(H_{obs})$ and thus $p \in Q_{mho}$. Therefore, by Definition 1 $(p, q) \in Q_{m\|}$.

This completes the induction proof.

- Therefore, the relation $\sim$ on the initial states, $(p_0, q_0) \sim q_0$, exists and is a bisimulation and $H_{obs} \| G$ is bisimulation equivalent to $H$. □

## 5.3 State-based requirements

The above theorem allows our filters to now be represented by possibly nondeterministic automata models, $H_{filt,j}$. More specifically, it allows us to replace the language-based requirements of Section 4 with the following set of state-based requirements:

*State-based filter requirements*

$R1'$) $H_{filt,j}$ is a nonblocking subautomaton of $B_{j,a}$

$R2'$) $H_{filt,j}$ is state controllable and state observable in $B_{j,a}$

$R3'$) $\Sigma_{rel}(H_{filt,j}) \cap \Sigma_h = \emptyset$

These results imply that a determinized version of the automaton $H_{filt,j}$, that we will denote $H_{filt,j,obs}$, will meet the previously established requirements $R1$, $R2$, and $R3$. Specifically, conditions $R1'$ and $R2'$ together with Theorem 3 imply that $H_{filt,j,obs} \| B_{j,a}$ is nonblocking since it is bisimulation equivalent to $H_{filt,j}$. Therefore, requirement $R1$ is satisfied. Furthermore, since $H_{filt,j}$ is state controllable in $B_{j,a}$, the generated language $\mathcal{L}(H_{filt,j})$ is language controllable with respect to $\mathcal{L}(B_{j,a})$. This then implies that $\mathcal{L}(H_{filt,j,obs})$ is also language controllable with respect to $\mathcal{L}(B_{j,a})$, thereby satisfying requirement $R2$. Also, $R3'$ are $R3$ are equivalent.

Thus far we have demonstrated that determinized versions of the filter automata $H_{filt,j}$ meeting requirements $R1'$, $R2'$, and $R3'$ will provide safe nonblocking control when acting in conjunction with traditionally built modular supervisors. However, we would like to avoid the determinization process. Since a nondeterministic filter automaton $H_{filt,j}$ possesses all the information that $H_{filt,j,obs}$ does, it turns out that $H_{filt,j,obs}$ never actually has to be constructed. However, the control required by the automaton $H_{filt,j}$ cannot be implemented via the synchronous composition operation. Rather, following the observation of a string $s \in \Sigma^*$, all continuations feasible at all states reached by strings with the same observation must be allowed. In essence, we are using $H_{filt,j}$ and its transition function $\delta_{filt}$ to generate an online implementation of $H_{filt,j,obs}$. Consider the following mathematical definition of the filter law $\mathcal{H}_{filt,j} : \mathcal{L}(B_{j,a}) \rightarrow 2^{\Sigma_\tau}$ that determines which events are to be enabled.

$$\mathcal{H}_{filt,j}(s) := \bigcup_{q \in T(s)} \Sigma_{H_{filt}}(q), \text{ where } T(s) = \{q \mid q \in \delta_{filt}(q_0, t) \text{ and } t \in P_\tau^{-1}(s)\} \quad (14)$$

In order to make this more clear, consider the automaton $G_a$ in Fig. 2. If we consider $G_a$ to be a nondeterministic representation of a deterministic control law, then following an observation of the string $ab$ we do not know if we are in state 3 or state 4, therefore, we have to allow both event $c$ and event $d$ to occur. The result of Theorem 3 also allows the composition $H_{filt,j,obs} \| B_{j,a}$ to be replaced by the subautomaton $H_{filt,j}$ in the course of Algorithm 2.

## 6 Filter law construction

Having established that we can employ filter laws represented by nondeterministic automata, the final question that remains is how to construct $H_{filt,j}$ so that requirements $R1'$, $R2'$, and $R3'$ are satisfied. Since we are ultimately trying to find a subautomaton, we are in essence trying to solve a state avoidance problem. This type of problem can be solved by a state-feedback approach to control. In other words, the control applied depends only on the state the system is in, not on the path taken to get there. It is well-established that a static control law of this type is potentially more restrictive than a dynamic control law in the case of partial observation (Kumar et al. 1993). However, we are willing to make this sacrifice in order to avoid exponential complexity. In this section we introduce some existing results on state-based approaches to control that can be employed to generate the conflict-resolving filters required by the approach of this paper. Additionally, we develop an improved covering-based approach to control that is less restrictive than existing state-feedback approaches.

### 6.1 State-based supervisory control

A *state-feedback supervisory controller* is a function $f : Q_g \to 2^{\Sigma_\tau}$ that determines the set of events to be enabled based on the current state of the system under control $G = (Q_g, \Sigma_\tau, \delta_g, q_0, Q_{mg})$. In the context of our larger approach to conflict resolution, the "plant" $G$ represents a given blocking composition $B_{j,a}$. The closed-loop system $f/G$ then represents the allowable set of states, that is, the subautomaton representing the coordinating filter $H_{filt,j}$. Assuming $q_0 \in G$, $f/G$ is defined iteratively as that portion of $G$ that is reachable via transitions that are allowed by $f$:

**Definition 10**

$$f/G = (Q_f, \Sigma_\tau, \delta_f, q_0, Q_{mf}) \tag{15}$$

*Iterative Definition of $f/G$:*

Step 1: $q_0 \in Q_f$.
Step 2: If $q \in Q_f$ and $\delta_g(q, \sigma)!$ for some $\sigma \in f(q)$, then $q' \in Q_f$, $\forall q' \in \delta_g(q, \sigma)$. Also, $\delta_f(q, \sigma) = \delta_g(q, \sigma)$. Otherwise, $\delta_f(q, \sigma)$ is empty.
Step 3: Every state in $Q_f$ and every transition for which $\delta_f$ is nonempty is obtained as in Step 1 and Step 2. Also, $Q_{mf} = Q_f \cap Q_{mg}$.

Note from the above definition that all of the states of $f/G$ are reachable and inherit their marking from $G$. The existence of a state-feedback controller that can keep the behavior of $G$ within a set of "good" states represented by the subset $Q_h \subseteq Q_g$ requires a property called $\Sigma_u$-invariance (Ramadge and Wonham 1987). In terms of nondeterministic automata and the notation of this paper, $\Sigma_u$-invariance of a state set $Q_h \subseteq Q_g$ is equivalent to the state controllability of a subautomaton of $G$, $H = (Q_h, \Sigma_\tau, \delta_h, q_0, Q_{mh})$. If the state space of $G$ is not fully observable, then additional considerations must be addressed.

In existing work on state-feedback control under partial observation, the concept of a "mask" is employed. A mask $M$ is defined as a function $M : Q_g \to Y$ that maps elements from the state space $Q_g$ to the observation space $Y$. Under partial observation two states $q$ and $q'$ might not be distinguishable, that is, they could have the same observation $M(q) = M(q') = y$. It is then necessary that the state-feedback control $f(q)$ be determined based on $M(q)$. Specifically, it is required that:

$$\text{For any } q, q' \in Q_g, M(q) = M(q') \Rightarrow f(q) = f(q') \tag{16}$$

In existing state-feedback work (Li 1991; Takai and Kodama 1998) it is assumed the mask $M$ is given. In this paper we will assume the mask $M$ is constructed to satisfy the following constraint.

$$\begin{gathered}\text{The mask } M : Q_g \to Y \text{ is defined such that if } \exists s, s' \in \Sigma_\tau^* \text{ with}\\ q \in \delta_g(q_0, s), q' \in \delta_g(q_0, s') \text{ and } P_\tau(s) = P_\tau(s'), \text{then } M(q) = M(q').\end{gathered} \tag{17}$$

In the above, states $q \in \delta_g(q_0, s)$ and $q' \in \delta_g(q_0, s')$ for which $P_\tau(s) = P_\tau(s')$ are defined to be *indistinguishable*. In other words, two states that are reached by strings that have the same projection are indistinguishable and the mask $M$ is constructed such that all indistinguishable states map to the same observation under $M$.

Of the existing work for generating state-feedback control under partial observation, the least restrictive control strategy is proposed in Takai and Kodama (1998) and builds off the prior results of Takai et al. (1995) and Takai and Kodama (1997). We will now outline their strategy using notation consistent with this paper and extensions we have added to handle nondeterminism. Specifically, Takai and Kodama (1998) presents an algorithm for constructing a state-feedback controller that satisfies Eq. 16. This algorithm is based on the following sets $A_H(q) \subseteq \Sigma_c$ that define which events must be disabled at state $q$ for a set of allowable states represented by the subautomaton $H \sqsubseteq G$. In essence, the sets $A_H(q)$ capture which events at a given state will cause a violation of the observability-type property captured by Eq. 16.

$$A_H(q) = \{\sigma \in \Sigma_c \mid (\exists q' \in Q_h) : [M(q) = M(q')] \wedge [\exists p \in \delta_g(q', \sigma) \text{ such that } p \notin Q_h]\} \tag{18}$$

In the above equation, $A_H(q)$ is also defined for $q' = q$ since a state is always considered indistinguishable from itself, that is, $M(q) = M(q)$. The resulting state-feedback control law is thus defined:

$$f(q) = \Sigma_\tau - A_H(q) \tag{19}$$

In order to guarantee that the control law defined by Eq. 19 is able to achieve the specification required by the subautomaton $H$, Takai and Kodama (1998) requires the following property in addition to $\Sigma_u$-invariance:

$$Q_h \subseteq R(Q_h) \tag{20}$$

In the above, $R$ is a transformation that represents which states of $H$ are reachable by permissible transitions, that is, those transitions that are not prohibited by the sets $A_H(q)$. Recall, the sets $A_H(q)$ enforce the observability-type requirement prescribed in Eq. 16. If $q_0 \notin Q_h$, then $R(Q_h) = \emptyset$. Otherwise, the set of states represented by $R(Q_h)$ can be constructed iteratively in a similar manner to Takai and Kodama (1998):

**Algorithm 3** *$R(Q_h)$ Construction*

```
Step 1:  $q_0 \in R(Q_h)$.
Step 2:  If $q \in R(Q_h)$ and $\delta_g(q,\sigma) \subseteq Q_h$ for some $\sigma \in \Sigma_\tau - A_H(q)$, then $q' \in$
         $R(Q_h)\ \forall q' \in \delta_g(q,\sigma)$.
Step 3:  Every state satisfying $R(Q_h)$ is obtained as in Step 1 and Step 2.
```

A specification represented by the set of states $Q_h$ that is $\Sigma_u$-invariant and satisfies Eq. 20 is defined to be *M-controllable* in Takai et al. (1995). The work of Takai et al. (1995) further demonstrates that a state-feedback controller that can achieve the behavior prescribed by the state set $Q_h$ exists if and only if the state set is $M$-controllable. In particular, the control law given by Eq. 19 will achieve the $M$-controllable state set $Q_h$ and is further the supremal state-feedback control law that satisfies Eq. 16.

If a given state set is not $M$-controllable, then Takai and Kodama (1998) prescribes an approach for finding an $M$-controllable subset, $R(Q_h^{\uparrow})$. Here the $\uparrow$ operation generates the supremal $\Sigma_u$-invariant subset of states constructed according to Ramadge and Wonham (1987). While the resulting $R(Q_h^{\uparrow})$ is not necessarily maximal or supremal, it does represent a larger state set than can be achieved by other existing state-feedback approaches Li (1991); Takai and Kodama (1997).

The subautomaton that results from the state-feedback controller of Takai and Kodama (1998) can be shown to be state controllable and state observable in $G$. Therefore, the results given above could be directly applied to the construction of filters required by our approach to conflict resolution. In the next section, however, we will propose an improved covering-based approach that generates a more permissive control law than Takai and Kodama (1998).

### 6.2 Covering-based supervisory control

Our improvement over Takai and Kodama (1998) derives from the fact that the requirement of Eq. 16 is stronger than necessary for the achievement of state observability. Therefore, we can apply a new covering-based approach that will result in a less restrictive control law. In this section we will additionally address blocking.

The observability-type requirement Eq. 16 is too strong based on the character of the mask $M$. The fact that $M$ is a function implies that when the state space is observed through this mask it is effectively partitioned into disjoint sets of states that have the same observation. For example, if $q$ and $q'$ have the same observation $M(q) = M(q')$, and $q'$ and $q''$ have the same observation $M(q') = M(q'')$, it then follows that $q$ and $q''$ must have the same observation $M(q) = M(q'')$. Therefore, all three states $q$, $q'$, and $q''$ must be in the same partition of the state space. For achievement of state observability, however, it may not be necessary that the same control action be applied at $q$ and $q''$ if they are not indistinguishable. In other words, if the observed string that reaches $q$ and $q'$ is different than the observed string that reaches $q'$ and $q''$, then the control applied at $q$ and $q''$ may be allowed to be different. Figure 6 illustrates this situation where $\sigma$ is disabled at $q''$, but need not be disabled at $q$ since these states are not both reached by the same observed string. In essence, we would like to base our control on a covering of the state space rather than a partition like that imposed by the mask $M$. If the event $\sigma$ was possible at the state $q'$, then $\sigma$ would need to be disabled at all three states for our covering-based approach too.

In order to present our covering-based approach, we will employ the mapping $I_H$ defined as follows:

**Definition 11** Let $I_H : Q_h \to 2^{Q_h}$ be a mapping defined $\forall q, q' \in Q_h$ as follows: $q' \in I_H(q)$ if $q$ and $q'$ are indistinguishable, that is, if $\exists s, s' \in \Sigma_\tau^*$ such that $q \in \delta_h(q_0, s)$, $q' \in \delta_h(q_0, s')$, and $P_\tau(s) = P_\tau(s')$.

Whereas the mask $M$ represents an equivalence relation on the states of an automaton that is by definition reflexive, symmetric, and transitive, the mapping $I_H$ represents a relation that is reflexive and symmetric, but not necessarily transitive. Reflexivity is apparent based on the fact that a state $q$ is always considered indistinguishable from itself, that is, $q \in I_H(q)$. Symmetry can be seen from the fact that $q \in I_H(q')$ implies $q' \in I_H(q)$. Transitivity, however, is not guaranteed. For example, $q \in I_H(q')$ and $q' \in I_H(q'')$ does not necessarily imply that $q \in I_H(q'')$. The notion of transitivity is discussed further below.

With $I_H$, we can then define new sets of prohibited events, $A'_H(q) \subseteq \Sigma_c$. At a state $q$ in the uncontrolled plant $G$, a controllable transition $\sigma$ that is defined in the uncontrolled plant $G$ is prohibited if it leads to a state outside of $Q_h$ or if it is prohibited at a state $q'$ that is indistinguishable from $q$ in $H$.

Since the definition of prohibited events at a state $q$, $A'_H(q)$, depends on the prohibited events of other states, each set $A'_H(q)$ is constructed iteratively. In words, if there is a string of indistinguishable states defined:

$$q \in I_H(q'),\ q' \in I_H(q''), \ldots,\ q^{(m-1)} \in I_H(q^{(m)})$$

each with $\sigma$ possible in $G$ and such that $\sigma$ is not possible in $H$ at $q^{(m)}$, then $\sigma$ is again added to $A'_H(q)$. This construction indicates a transitivity similar to that imposed

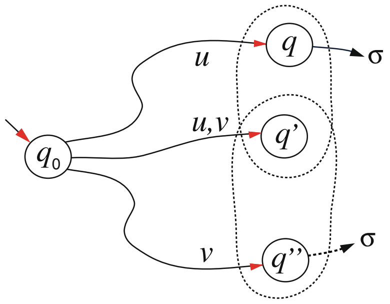

**Fig. 6** Example of a covering for indistinguishable states

by $M$, *except* that here the transitivity is limited to indistinguishable states where $\sigma$ is possible in $G$. Assuming the mapping $I_H$ is given, $A'_H(q)$ can be constructed according to Algorithm 4 given below. The mapping $I_H$ can be constructed with polynomial complexity using results from Wang et al. (2007).

**Algorithm 4** Prohibited Events Determination

```
Input: automaton $G$, subautomaton $H \sqsubseteq G$ and mapping $I_H : Q_h \to 2^{Q_h}$
For each $q \in Q_h$
    For each event $\sigma \in \Sigma_G(q) \cap \Sigma_c$
        If $\exists p \in \delta_g(q, \sigma)$ such that $p \notin \delta_h(q, \sigma)$ then
          add $\sigma$ to the set of prohibited events at $q$, $A'_H(q) \leftarrow \{\sigma\} \cup A'_H(q)$.
        End if
        If $\sigma \in A'_H(q)$ then
          define a set of states $T$ that is initialized with the state $q$, $T \leftarrow \{q\}$. Also
          let $\mathcal{M} : Q_h \to \{0, 1\}$ be a partial function marking whether or not states in
          the set $T$ have been addressed yet. Set $\mathcal{M}(q) = 0$.
          For each $q' \in T$ with $\mathcal{M}(q') = 0$
              For each $q'' \in I_H(q')$ that is not in $T$
                  If $\delta_g(q'', \sigma)!$ then
                    add state $q''$ to the set $T$, $T \leftarrow \{q''\} \cup T$, and add event $\sigma$ to
                    the set of prohibited events at $q''$, $A'_H(q'') \leftarrow \{\sigma\} \cup A'_H(q'')$. Set
                    $\mathcal{M}(q'') = 0$.
                  End if
              End for
              Set $\mathcal{M}(q') = 1$.
          End for
          Clear $T$ and $\mathcal{M}$.
        End if
    End for
End for
Output: the sets $A'_H(q)$
```

Algorithm 4 has complexity $\mathcal{O}(mn^2)$ where $m$ is the cardinality of the event set and $n$ is the cardinality of the state space. Assuming the subautomaton $H$ has a finite number of transitions, this algorithm will terminate in finite time. The sets $A'_H(q)$ defined by Algorithm 4 satisfy the following equation by construction:

$$A'_H(q) = \{\sigma \in \Sigma_c \mid (\exists q' \in Q_h) : [\delta_g(q, \sigma)!] \wedge [q' \in I_H(q)] \wedge [(\exists p \in \delta_g(q', \sigma) \text{ such that } p \notin Q_h) \vee (\sigma \in A'_H(q'))]\} \tag{21}$$

In order to explicitly compare the sets $A_H(q)$ and $A'_H(q)$, we now define the following mapping $J_H$ that reflects the partition implicitly imposed by a given mask $M$ on the state set $Q_h$:

**Definition 12** Let $J_H : Q_h \to 2^{Q_h}$ be a mapping defined $\forall q, q' \in Q_h$ as follows: $q' \in J_H(q)$ if $M(q) = M(q')$.

The definition of $A_H(q)$ given in Eq. 18 can then be rewritten in terms of the mapping $J_H$ as follows:

$$A_H(q) = \{\sigma \in \Sigma_c \mid (\exists q' \in Q_h) : [q' \in J_H(q)] \wedge [\exists p \in \delta_g(q', \sigma) \text{ such that } p \notin Q_h]\} \tag{22}$$

It can then be shown that $A'_H(q) \subseteq A_H(q)$. Let $\sigma$ be an arbitrary event in $A'_H(q)$. By Eq. 21, there exists a state $q' \in Q_h$ that is indistinguishable from $q$ ($q' \in I_H(q)$) such that either $\sigma$ leads from $q'$ to a state outside of $Q_h$ or $\sigma$ is prohibited at $q'$ ($\sigma \in A'_H(q')$). If it is the former, then $\sigma \in A_H(q)$ based on Eq. 22 and the fact that $q' \in J_H(q)$ since $I_H(q) \subseteq J_H(q)$. If it is the latter, then by recursive application of Eq. 21 there exists a chain of indistinguishable states leading to a state $q''$ for which $\sigma$ does lead to a state outside of $Q_h$. The construction of the sets $A'_H(q)$ by Algorithm 4 guarantees that for this latter case such a state $q''$ exists in the chain. Furthermore, as only automata with finite state spaces are addressed by the work of this paper, this chain will terminate in a finite number of steps. Since the mask $M$ imparts a partition on the state space, $M(q) = M(q'')$ and $q'' \in J_H(q)$ thereby implying $\sigma \in A_H(q)$ based on Eq. 22. These facts together provide the desired result that $A'_H(q) \subseteq A_H(q)$.

This new $A'_H(q)$ can then be employed to generate a new transformation $R'$. $R'$ is defined in the same manner as Algorithm 3 that constructs $R$ except for the different definition of prohibited events that is employed. $R'$ is again a transformation that retains those states of $H$ that are reachable by permissible transitions. As we construct the state set $R'(Q_h)$ below, we will additionally construct an associated subautomaton of $H$, $R'(H)$, with a state set $R'(Q_h)$ and a transition function $\delta_{R'}$. Based on our definition of a subautomaton given in Definition 7, the marking of $R'(H)$ will be consistent with the marking of $H$. The use of $A'_H(q)$ in the construction process will result in the transformed subautomaton $R'(H)$ being state observable in $G$.

If $q_0 \notin Q_h$, then $R'(Q_h) = \emptyset$. Otherwise, $R'(Q_h)$ and $R'(H)$ are constructed as follows:

**Algorithm 5** *Construction of $R'(Q_h)$ and $R'(H)$*

Step 1: $q_0 \in R'(Q_h)$.
Step 2: If $q \in R'(Q_h)$ and $\delta_g(q, \sigma) \subseteq Q_h$ for some $\sigma \in \Sigma_\tau - A'_H(q)$, then $q' \in R'(Q_h)$ $\forall q' \in \delta_g(q, \sigma)$ and $\delta_{R'}(q, \sigma) = \delta_g(q, \sigma)$. Otherwise, $\delta_{R'}(q, \sigma)$ is empty.
Step 3: Every state satisfying $R'(Q_h)$ and every transition for which $\delta_{R'}$ is nonempty is obtained as in Step 1 and Step 2.

From the above algorithm and employing logic from Takai et al. (1995), for any $q \in R'(Q_h) - \{q_0\}$, there exist $q_1, q_2, \ldots, q_m \in Q_g$ and $\sigma_0, \sigma_1, \ldots, \sigma_{m-1} \in \Sigma_\tau$ satisfying the following conditions:

$C1.1)$ $\delta_g(q_i, \sigma_i) = q_{i+1}$ for $i = 0, 1, \ldots, m-1$
$C1.2)$ $q_i \in Q_h$ for $i = 0, 1, \ldots, m$
$C1.3)$ $\sigma_i \in \Sigma_\tau - A'_H(q_i)$ for $i = 0, 1, \ldots, m-1$
$C1.4)$ $q_m = q$

The following result that employs the logic of Lemma 1 in Takai and Kodama (1997) can now be presented.

**Proposition 6** *For any subautomaton $H \sqsubseteq G$*

$$R(Q_h) \subseteq R'(Q_h) \tag{23}$$

*Proof* If $q_0 \notin Q_h$, then $R(Q_h) = R'(Q_h) = \emptyset$. Consider the case that $q_0 \in Q_h$. Since $A'_H(q) \subseteq A_H(q)$ for any $q \in Q_h$, we have $R(Q_h) \subseteq R'(Q_h)$. □

Although the result of $R(Q_h^{\uparrow})$ represents a larger state set than can be achieved by any prior state-feedback work, Proposition 6 demonstrates that we can generate a potentially larger state set $R'(Q_h^{\uparrow}) \supseteq R(Q_h^{\uparrow})$. This together with the example of Section 6.4 shows that our covering-based approach is less restrictive than existing state-feedback approaches.

## 6.3 Covering-based filter construction

In this section we will specify how the subautomaton $R'(H^{\uparrow})$ is constructed. Here $R'(H^{\uparrow})$ is defined according to Algorithm 5, where $H^{\uparrow} \sqsubseteq H$ is the subautomaton that possesses the state set $Q_h^{\uparrow}$. Specifically, the transition structure of $H^{\uparrow}$ is defined such that it includes all transitions of $H$ for which the source state $q$ and destination state $q' \in \delta_h(q, \sigma)$ are in $Q_h^{\uparrow}$.

Since a subautomaton $R'(H^{\uparrow})$ will be employed to represent each coordinating filter, we must first demonstrate that $R'(H^{\uparrow})$ is state observable and state controllable in its associated $G$. We will specifically show that state observability holds directly as a result of the $R'$ transformation. We will then demonstrate that the $R'$ operation did not destroy the state controllability achieved by the $\uparrow$ operation. In order to accomplish this goal, we first propose the following hypothetical state-feedback control law $f'$ that achieves the specification $R'(Q_h^{\uparrow})$; though, our covering-based law will be ultimately be implemented according to Eq. 14.

$$f'(q) = \Sigma_\tau - A'_{H^{\uparrow}}(q) \tag{24}$$

The subautomaton of $G$, $f'/G = (Q_{f'}, \Sigma_\tau, \delta_{f'}, q_0, Q_{mf'})$, is defined in the same manner as Definition 10. Also, for any state $q \in Q_g - \{q_0\}$ that is also in $Q_{f'}$, there exist $q_1, q_2, \ldots, q_m \in Q_g$ and $\sigma_0, \sigma_1, \ldots, \sigma_{m-1} \in \Sigma_\tau$ satisfying the following conditions (Li and Wonham 1993):

$C2.1)$ $q_{i+1} \in \delta_g(q_i, \sigma_i)$ for $i = 0, 1, \ldots, m-1$
$C2.2)$ $\sigma_i \in f'(q_i)$ for $i = 0, 1, \ldots, m-1$
$C2.3)$ $q_m = q$

Based on the manner in which the sets $A'_{H^{\uparrow}}(q)$ are constructed, it can also be seen that the following relation is implied where $I_{f'}$ is defined for the automaton $f'/G$.

$$\text{For any } q,\ q' \in Q_{f'} \text{ with } q' \in I_{f'}(q),\ \sigma \in f'(q) \cap \Sigma_G(q) \Rightarrow \sigma \in f'(q') \tag{25}$$

The above then leads to a result that is similar to Eq. 16:

$$\text{For any } q, q' \in Q_{f'}, q' \in I_{f'}(q) \Rightarrow f'(q) \cap \Sigma_G(q) \cap \Sigma_G(q') = f'(q') \cap \Sigma_G(q) \cap \Sigma_G(q') \tag{26}$$

The above expression captures that the control applied by $f'$ is consistent between states that are indistinguishable for those feasible events that are shared between the states. The limitation of consistency to those feasible events shared between states demonstrates the limited transitivity that provides the improvement of our covering-based approach. With the above property, we can now demonstrate that state observability is achieved.

**Proposition 7** *The state-feedback law $f'$ given by Eq. 24 generates a subautomaton $f'/G$ that is state observable in $G$.*

*Proof*

- Let $q \in \delta_{f'}(q_0, s)$, $\sigma \in \Sigma_c$, $P_\tau(s)\sigma \in \mathcal{L}(f'/G)$ and $p \in \delta_g(q, \sigma)$.
- $P_\tau(s)\sigma \in \mathcal{L}(f'/G)$ implies there exists an $s' \in \Sigma_\tau^*$ and $q' \in \delta_{f'}(q_0, s')$ for which $P_\tau(s') = P_\tau(s)$ and $\sigma \in f'(q')$.
- Since $P_\tau(s') = P_\tau(s)$, $q' \in I_{f'}(q)$. Therefore, we have that $f'(q) \cap \Sigma_G(q) \cap \Sigma_G(q') = f'(q') \cap \Sigma_G(q) \cap \Sigma_G(q')$ by Eq. 26.
- Since $\sigma \in f'(q') \cap \Sigma_G(q) \cap \Sigma_G(q')$, $\sigma \in f'(q) \cap \Sigma_G(q) \cap \Sigma_G(q')$ also.
- Since $\sigma \in f'(q)$ and $p \in \delta_g(q, \sigma)$, $p \in \delta_{f'}(q, \sigma)$ by definition of $\delta_{f'}$. Thus we have shown that $f'/G$ is state observable in $G$. □

This control law $f'$ achieves the set of states $R'(Q_h^\uparrow)$. This fact is mathematically represented $Q_{f'} = R'(Q_h^\uparrow)$ and is proven in the following proposition employing logic from Theorem 1 of Takai et al. (1995):

**Proposition 8** *If for the subautomaton $H \sqsubseteq G$, $q_0 \in H^\uparrow$ and $f'$ is given by Eq. 24, then $Q_{f'} = R'(Q_h^\uparrow)$.*

*Proof* By assumption, $q_0 \in H^\uparrow$. Step 1 of Algorithm 5 then provides that $q_0 \in R'(Q_h^\uparrow)$. Additionally, since $f'$ only disables transitions, $q_0$ will be reachable under control. That is, $q_0 \in Q_{f'}$.

($\subseteq$) We will next show that $Q_{f'} \subseteq R'(Q_h^\uparrow)$. Let $q \in Q_{f'}$. For any $q \in Q_{f'} - \{q_0\}$, there exist $q_1, q_2, \ldots, q_m \in Q_g$ and $\sigma_0, \sigma_1, \ldots, \sigma_{m-1} \in \Sigma_\tau$ satisfying conditions ($C$2.1-$C$2.3). By induction on the index of the states in this chain, we can then show that $q \in R'(Q_h^\uparrow)$. For the basis step, we already have $q_0 \in R'(Q_h^\uparrow)$. For the induction step, suppose that $q_k \in R'(Q_h^\uparrow) \subseteq Q_h^\uparrow$. We now want to show that $q_{k+1} \in \delta_g(q_k, \sigma_k) \subseteq R'(Q_h^\uparrow)$. Consider two cases:

1) If $\sigma_k \in \Sigma_u$, we then have that $q_{k+1} \in \delta_g(q_k, \sigma_k) \subseteq Q_h^\uparrow$ since $q_k \in Q_h^\uparrow$ and $Q_h^\uparrow$ is $\Sigma_u$-invariant. Furthermore, since $\sigma_k \in \Sigma_u$ we have that $\sigma_k \notin A'_{H^\uparrow}(q_k)$. Therefore, $q_{k+1} \in \delta_g(q_k, \sigma_k) \subseteq R'(Q_h^\uparrow)$ by Step 2 of Algorithm 5.

2) If $\sigma_k \in \Sigma_c$, then by condition $C2.2$ and Eq. 24, we have $\sigma_k \in f'(q_k) = \Sigma_\tau - A'_{H^\uparrow}(q_k)$. Therefore, by Step 2 of Algorithm 5 we again have that $q_{k+1} \in \delta_g(q_k, \sigma_k) \subseteq R'(Q_h^\uparrow)$.

This completes the induction.

($\supseteq$) We will now show that $Q_{f'} \supseteq R'(Q_h^\uparrow)$. Let $q \in R'(Q_h^\uparrow)$. For any $q \in R'(Q_h^\uparrow) - \{q_0\}$ there exist $q_1, q_2, \ldots, q_m \in Q_g$ and $\sigma_0, \sigma_1, \ldots, \sigma_{m-1} \in \Sigma_\tau$ satisfying conditions analogous to ($C1.1$-$C1.4$), but for $Q_h^\uparrow$ instead of $Q_h$. Since $q \in R'(Q_h^\uparrow)$ implies $q \in Q_g$, to show that $q \in Q_{f'}$, it is sufficient to prove that $\sigma_i \in f'(q_i) (i = 0, 1, \ldots, m-1)$. By condition $C1.3$ and Eq. 24, we have $\sigma_i \in \Sigma_\tau - A'_{H^\uparrow}(q_i) = f'(q_i)$. □

Based on the above proposition and its proof, we have that $q_0 \in R'(Q_h^\uparrow)$ and that $Q_{f'} = R'(Q_h^\uparrow)$, therefore, $R'(Q_h^\uparrow)$ is $\Sigma_u$-invariant by Theorem 6 of Li and Wonham (1993). This in turn implies that the associated subautomaton $f'/G = R'(H^\uparrow)$ is state controllable in $G$. Therefore, we have demonstrated that $R'(H^\uparrow)$ is state controllable and state observable in $G$. The only property that has not been addressed yet is blocking. By construction, the marking of $R'(H^\uparrow)$ is consistent with the marking of $G$. Taking the trim of $R'(H^\uparrow)$ makes the subautomaton nonblocking. In this instance, the trim operation will simply remove those states of $R'(H^\uparrow)$ that are blocking. This, however, can destroy state controllability and/or state observability. Therefore, following the trim operation, it may be necessary to repeat the $\uparrow$ and $R'$ operations again. A summary of this algorithm is given below.

We can then employ the resulting automaton as our filter $H_{filt,j}$ and can implement our filter law $\mathcal{H}_{filt,j}$ according to Eq. 14. Note, the hypothetical state-feedback law $f'$ working under full observation achieves the same behavior as the covering-based law of Eq. 14 under partial observation.

**Algorithm 6** *Filter Law Construction*

Step 1: Given a blocking automaton $G = B_{j,a}$, let the subautomaton $H = \text{trim}(G)$ be our "specification."

Step 2: Find $Q_h^\uparrow$, the supremal $\Sigma_u$-invariant subset of $Q_h$. The algorithm of Ramadge and Wonham (1987) can be employed. Let $H^\uparrow$ be the subautomaton with the state set corresponding to $Q_h^\uparrow$.

Step 3: Construct the mapping $I_{H^\uparrow}$ of indistinguishable states of the subautomaton $H^\uparrow$. The algorithm of Wang et al. (2007) can be used for this purpose.

Step 4: Construct the sets of prohibited transitions $A'_{H^\uparrow}(q)$ to satisfy Eq. 21. Algorithm 4 can be employed.

Step 5: Follow Algorithm 5 to construct the state set $R'(Q_h^\uparrow)$ and subautomaton $R'(H^\uparrow)$.

Step 6: If the subautomaton $R'(H^\uparrow)$ is nonblocking, then this represents our filter automaton $H_{filt,j}$ and we are done. Otherwise, redefine $H = \text{trim}(R'(H^\uparrow))$ and return to Step 2.

In the above, Step 3 and Step 4 could be addressed simultaneously by a single algorithm. The end result of this procedure is a (possibly nondeterministic) subautomaton $H_{filt,j}$ that satisfies requirements $R1'$, $R2'$, and $R3'$ with respect to the blocking automaton $G = B_{j,a}$.

Each step in the above procedure has polynomial complexity in the number of states and transitions of the initial automaton, therefore, each iteration of the algorithm will also have polynomial complexity. In addition, each pass through the algorithm either removes a state from the subautomaton or reaches a fixpoint. Assuming the initial automaton has a finite number of states, at most $n$ iterations must be performed where $n$ is the number of states in the initial automaton. Therefore, the overall complexity of the algorithm is polynomial. The resulting coordinating filter law produces more permissive control than existing state-feedback approaches, but it is not necessarily maximal. This fact is demonstrated by Remark 4 following the example of the next section.

## 6.4 Filter construction example

The following example helps to illustrate our filter construction procedure introduced in Algorithm 6 of the previous section.

*Example 3* Consider the blocking automaton $G$ pictured on the left of Fig. 7 with event set partitioned into controllable and uncontrollable events as follows, $\Sigma_c = \{a, b, c, d, f\}$ and $\Sigma_u = \{e, \tau\}$. By Step 1 of Algorithm 6, our specification $H_0 = \mathrm{trim}(G)$ is a subautomaton of $G$ where the blocking state 9 has been removed. Since state 8 of $H_0$ then requires that the uncontrollable event $e$ be disabled, the $\uparrow$ operation of Step 2 will remove state 8 resulting in the subautomaton $H_0^{\uparrow}$.

Following Step 3 of Algorithm 6, the mapping $I_{H_0^{\uparrow}}$ representing which states are indistinguishable is then constructed. We will represent $I_{H_0^{\uparrow}}$ as Table 1 that was constructed using the algorithm from Wang et al. (2007). Specifically, the left-hand column enumerates each state $q$ in the state space of $H_0^{\uparrow}$ and the center column lists the corresponding set of indistinguishable states $I_{H_0^{\uparrow}}(q)$.

Step 4 of Algorithm 6 then constructs the sets $A'_{H_0^{\uparrow}}(q)$. Examining state 3, $\delta_g(3, b) = 8 \in Q_g$, but $8 \notin Q^{\uparrow}_{H_0}$, therefore, $b \in A'_{H_0^{\uparrow}}(3)$. It then follows that $b$ is also

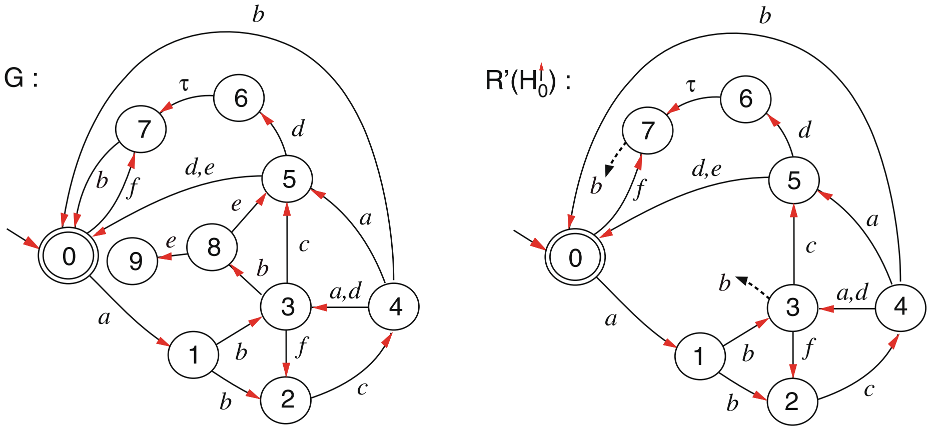

**Fig. 7** Filter construction example

**Table 1** Table representing the maps $I_{H_0^\uparrow}$ and $A'_{H_0^\uparrow}$

| $q$ | $I_{H_0^\uparrow}(q)$ | $A'_{H_0^\uparrow}(q)$ |
|---|---|---|
| 0 | 0, 3, 6, 7 | |
| 1 | 1 | |
| 2 | 2, 3, 7 | |
| 3 | 3, 2, 5, 6, 7, 0 | b |
| 4 | 4, 5 | |
| 5 | 5, 4, 3 | |
| 6 | 6, 7, 3, 0 | |
| 7 | 7, 6, 3, 0, 2 | b |

in the set $A'_{H_0^\uparrow}(7)$ since $b$ is defined at state 7 and $7 \in I_{H_0^\uparrow}(3)$. Since there are no other states in $I_{H_0^\uparrow}(3)$ or $I_{H_0^\uparrow}(7)$ for which a $b$ event is defined, and since there are no other events actively disabled by $H_0^\uparrow$, all the sets $A'_{H_0^\uparrow}(q)$ are now completely defined. Step 5 of the algorithm then applies the transformation $R'$. Since $\delta_g(7, b) = 0 \in Q_{H_0}^\uparrow$ and $b \in A'_{H_0^\uparrow}(7)$, event $b$ must be disabled at state 7. The resulting subautomaton $R'(H_0^\uparrow)$ is displayed on the right-hand side of Fig. 7.

According to Step 6, since $R'(H_0^\uparrow)$ is blocking, we must then take the trim and start over at Step 2 of the algorithm. Let $H_1 = \text{trim}(R'(H_0^\uparrow))$ and refer to the left-hand side of Fig. 8 for an illustration.

Since the only actively disabled events $b$, $d$, and $f$ are controllable, the $\uparrow$ operation does not remove any states, $H_1^\uparrow = H_1$. We now construct the map $I_{H_1^\uparrow}$; the result is shown below in Table 2.

Next we build the sets $A'_{H_1^\uparrow}(q)$. Noting which transitions of $G$ are not included in $H_1^\uparrow$ allows us to determine that $f \in A'_{H_1^\uparrow}(0)$, $b \in A'_{H_1^\uparrow}(3)$, and $d \in A'_{H_1^\uparrow}(5)$. Examining the table describing the mapping $I_{H_1^\uparrow}$, we then also have that $f \in A'_{H_1^\uparrow}(3)$ since $3 \in I_{H_1^\uparrow}(0)$ and $\delta_g(3, f)!$. Likewise, $d \in A'_{H_1^\uparrow}(4)$ since $4 \in I_{H_1^\uparrow}(5)$ and $\delta_g(4, d)!$.

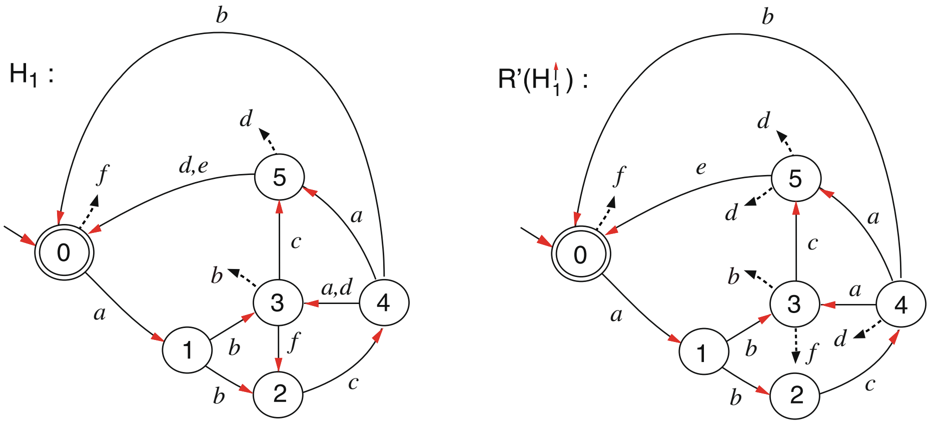

**Fig. 8** Filter construction example

**Table 2** Table representing the maps $I_{H_1^{\uparrow}}$ and $A'_{H_1^{\uparrow}}$

| $q$ | $I_{H_1^{\uparrow}}(q)$ | $A'_{H_1^{\uparrow}}(q)$ |
|---|---|---|
| 0 | 0, 3 | f |
| 1 | 1 | |
| 2 | 2, 3 | |
| 3 | 3, 2, 5, 0 | b, f |
| 4 | 4, 5 | d |
| 5 | 5, 4, 3 | d |

Now according to Step 5 we construct $R'(H_1^{\uparrow})$. First, note that one instance of a $d$ event is disabled at state 5 according to $H_1^{\uparrow}$, therefore, the remaining $d$ transition at state 5 must also be disabled. Since $\delta_g(3, f) = 2 \in Q_{H_1}^{\uparrow}$ and $f \in A'_{H_1^{\uparrow}}(3)$, the $f$ event at state 3 must also be disabled. Likewise, the $d$ event at state 4 must be disabled. The resulting $R'(H_1^{\uparrow})$ is shown on the right-hand side of Fig. 8. Since this subautomaton is nonblocking, we are done. Therefore, our deterministic filter law $\mathcal{H}_{filt}$ is represented by the nondeterministic automaton $H_{filt} = R'(H_1^{\uparrow})$ that satisfies requirements $R1'$, $R2'$, and $R3'$.

In view of the above example, we make the following observations regarding the new results presented in this section.

*Remark 2* In traditional state-feedback control employing a mask $M$, states 3 and 4 would be in the same partition since state 3 is indistinguishable from state 5 and state 5 is indistinguishable from state 4. Therefore, traditional approaches would have disabled the $b$ event at state 4. This example along with Eq. 23 demonstrates the advantage of our covering-based approach over the state-feedback control approaches of Li (1991) and Takai and Kodama (1998).

*Remark 3* Our approach, however, still produces a static control law with respect to the those events feasible at a given state. For example, if for some reason the $b$ event at state 3 needed to be disabled following the string $ab$, but not following the string $abca$, our covering-based control law would not be able to make that distinction. This more restrictive control law is again chosen to avoid the exponential complexity that would come with implementing an event-feedback law.

*Remark 4* If we had recalculated the mapping $I$ after the event $d$ at state 5 was disabled, then states 0 and 3 would no longer be indistinguishable. This new $I$ mapping would then not have required that the $f$ event at state 3 be disabled. If we allowed the $I$ mapping to change within the calculation of the transformation $R'$, then the resulting subautomaton would be dependent on the order in which the states were addressed. This dependence is an issue common also to event-feedback approaches to control under partial observation.

*Remark 5* We have shown that our covering-based approach is an improvement over the state-feedback approach proposed by Takai and Kodama (1998). The approach of Takai and Kodama (1998) in turn has been shown to provide more permissive control than the construction of the supremal controllable and normal subset of states presented in Li (1991). Namely, if for all $q$ in the allowed state set the set $M^{-1}(M(q))$ is a subset of the allowed state set, where

$$M^{-1}(M(q)) = \{q' \mid M(q) = M(q')\},$$

then the state set is normal. The example of this subsection, therefore, shows how the approach of Li (1991) is more restrictive than our approach. Since in our example states 0 and 6 are reached by the same string *abcd*, they both have the same observation under $M$, but state 0 is in the state set $R'(Q^{\uparrow}_{H_1})$ while state 6 is not. Therefore, the state set $R'(Q^{\uparrow}_{H_1})$ violates normality. Construction of the supremal normal and controllable subset of states would then require removal of state 0 leading to an empty state set.

## 7 Application to a flexible manufacturing system (FMS)

In this section we will demonstrate the approach of this paper for generating nonblocking modular supervisory control through a Flexible Manufacturing System (FMS) example.

*Example 4* The FMS we will specifically employ is a reduced version of the system given in de Queiroz et al. (2005) and is shown in Fig. 9. The basic idea is that parts enter from the left via the conveyor *Con2*. From *Con2* the parts pass through buffer *B2* to a handling robot. This robot then passes parts, through buffer *B4*, to a lathe that can generate two different types of parts. After the lathe has finished an operation and returned a part to the robot, again through buffer *B4*, the robot then passes the part to either buffer *B6* or buffer *B7* depending on the part type. If passed to *B7*, the part is then sent to a painting machine *PM* via conveyor *Con3* and buffer *B8*. Once the painting operation is finished, the part is passed back through the same sequence by which it arrived. From buffers *B6* and *B7*, the two different parts are passed to the machine *AM* for finishing.

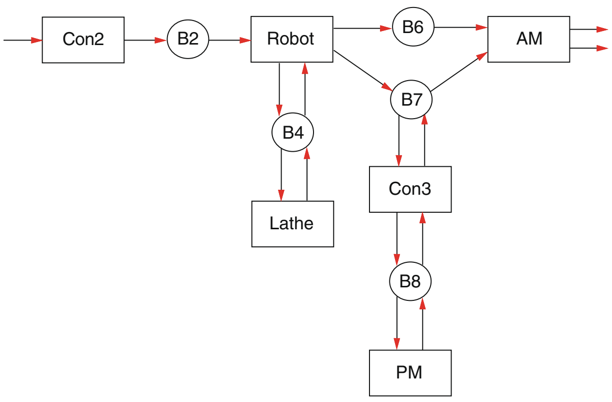

**Fig. 9** Flexible manufacturing system (FMS)

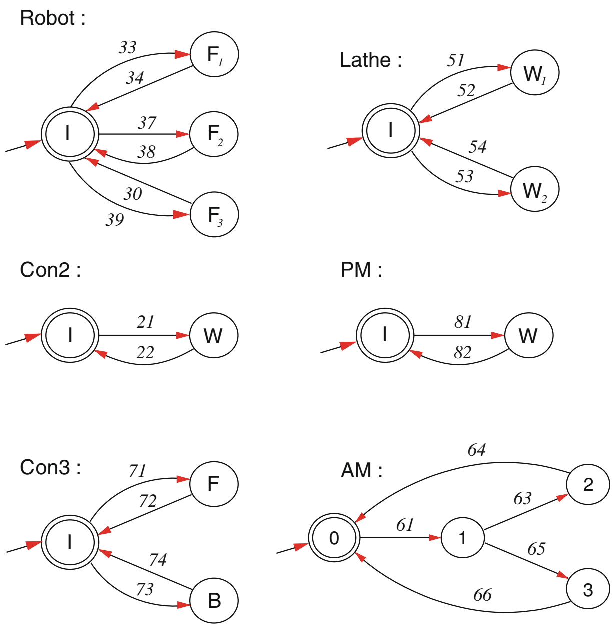

**Fig. 10** Automata modeling the components of the open-loop plant

The machines *Con2*, *Robot*, *Lathe*, *Con3*, *PM*, and *AM* can be thought of as components of the open-loop plant. The automata models for these machines are given Fig. 10.

The buffers *B2*, *B4*, *B6*, *B7*, and *B8* can be thought of as the component specifications for the system where it is desired that the buffers not underflow or overflow. The automata models for these machines are given Fig. 11. In these automata odd

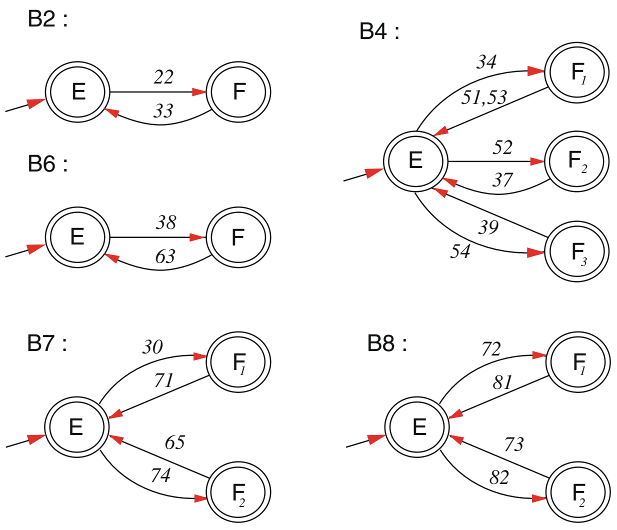

**Fig. 11** Automata modeling the component buffer specifications

labels represent controllable events and even labels represent uncontrollable events. Furthermore, all automata have the same event set $\Sigma_\tau$, though only relevant events are pictured.

Following Algorithm 2 of Section 3, the first step is to generate a set of local modular supervisory controllers. We will ultimately choose to address the buffer specifications in the order *B7* → *B6* → *B4* → *B8* → *B2*. Specifically, $H_1$ is the automaton representation of the supervised module corresponding to specification *B7* and subplant $G'_1 = \mathit{Robot}\|\mathit{AM}\|\mathit{Con3}$. Likewise, $H_2$ to corresponds to the specification *B6* and subplant $G'_2 = \mathit{Robot}\|\mathit{AM}$, $H_3$ to specification *B4* and subplant $G'_3 = \mathit{Robot}\|\mathit{Lathe}$, $H_4$ to specification *B8* and subplant $G'_4 = \mathit{Con3}\|\mathit{PM}$, and $H_5$ to specification *B2* and subplant $G'_5 = \mathit{Con2}\|\mathit{Robot}$.

Recognizing that the plant components making up $G'_2$ are a subset of the plant components making up $G'_1$, we have in this instance that $H_1\|G'_2 = H_1$ and can employ a reduction for $H_2$ based on the logic of Proposition 5. We will denote the resulting reduction $C_2$. In order to make the connection to Proposition 5 more clear, the $H_1$ of this example corresponds to the $H_1\|H_2$ of Proposition 5, while the $H_2$ of this example corresponds to the $H_3$ of Proposition 5. While the original closed-loop module $H_2$ of this example has 28 states and 71 transitions, its reduction $C_2$ can be represented by an automaton with 2 states and 3 transitions. Furthermore, the events 61, 64, 65, and 66 are not relevant to $C_2$, though they were to $H_2$. This result demonstrates how the use of supervisor reduction can result in fewer shared relevant events, thereby allowing additional abstraction in the preceding modules.

Step 2 then instructs us to generate conflict-equivalent abstractions for each supervised subsystem employing Algorithm 1. Initially, module $H_1$ is represented by an automaton with 80 states and 259 transitions that we will write 80(259). If a transition is self-looped at every state, then we will not count it in the total number of transitions. Since events 61, 64, 65, and 66 are relevant to only the current subsystem $H_1$, we will "hide" them, that is, we will replace their occurrence in $H_1$ by the silent event $\tau$. As such, we can apply the rules from Section 4.2 to generate the abstraction $H_{1,a}$ that has size 31(115). The relevant event set of $C_2 : 2(3)$ is $\{37, 38, 63\}$ and is contained in the relevant event set of $H_{1,a}$ and hence can be reduced no further. The reduced size of the relevant event set of $C_2$ as compared to $H_2$, however, did enable the hiding of the events 61, 64, 65, and 66. Events 51, 52, 53, and 54 are relevant to only subsystem $H_3 : 9(10)$, leading to the abstraction $H_{3,a} : 6(7)$. Similarly, $H_4 : 6(6)$ has events 81 and 82 that can be hidden, leading to $H_{4,a} : 4(4)$. Also, events 21 and 22 are relevant to only $H_5 : 12(24)$ allowing the abstraction $H_{5,a} : 4(6)$.

The next step is to pick an initial subsystem. Some considerations for how to pick a "good" ordering of subsystems will be discussed at the end of this section, but for now we will choose $H_{1,a}$ as our starting point. Following Step 4 of the procedure, we will then choose the next subsystem to be $C_2$ and will generate the composition $H_{1,a}\|C_2 : 53(174)$. The next step is to check for blocking. Since it turns out the $H_{1,a}\|C_2$ is nonblocking, we skip to Step 7. At this point, event 63 is not relevant to any of the remaining subsystems and hence can now be hidden. This leads to the abstraction $(H_{1,a}\|C_2)_a : 27(79)$.

Since other subsystems still have not yet been addressed, we return to Step 4 and add $H_{3,a}$ to the composition, $(H_{1,a}\|C_2)_a\|H_{3,a} : 45(120)$. Note, the process of abstraction has led $H_{3,a}$ and the resulting composition to be nondeterministic. At this point we again check for blocking. Since the composition is nonblocking, we skip to Step 7. All the relevant events of the composition $(H_{1,a}\|C_2)_a\|H_{3,a}$ are still relevant to the remaining subsystems, therefore, no more events can be hidden at this point. However, the composition can still be reduced further by applying the conflict equivalence preserving rules introduced earlier, $((H_{1,a}\|C_2)_a\|H_{3,a})_a : 42(115)$.

Returning to Step 4 again, $H_{4,a}$ is added to the composition. The result $((H_{1,a}\|C_2)_a\|H_{3,a})_a\|H_{4,a} : 61(52)$ turns out to be blocking. Therefore, according to Step 6 of the procedure, a filter must be built to resolve the conflict. The blocking composition $B_{1,a} = ((H_{1,a}\|C_2)_a\|H_{3,a})_a\|H_{4,a}$ is in essence the uncontrolled "plant" and we can apply Algorithm 6 to construct the subautomaton of $B_{1,a}$ that will serve as the filter that supervises the system and prevents the blocking. Taking the trim of $B_{1,a}$ removes two blocking states, but leaves the resulting automaton not state controllable with respect to $B_{1,a}$. Applying Steps 2 through 5 of Algorithm 6 leaves a state controllable and state observable automaton, but it is again blocking. Performing another iteration of the algorithm leaves us a nonblocking subautomaton that is state controllable and state observable with respect to $B_{1,a}$. This subautomaton has 41 states and 120 transitions and serves as our coordinating filter law $H_{filt,1}$.

Since a determinized version of $H_{filt,1}$ composed with $(H_{1,a}\|C_2)_a\|H_{3,a}\|H_{4,a}$ is bisimulation equivalent to $H_{filt,1}$, we will replace the composition $(H_{1,a}\|C_2)_a\|H_{3,a}\|H_{4,a}$ by $H_{filt,1}$ as we proceed to Step 7. At this point, the events 71, 72, 73, and 74 have become local and can thus be hidden leading to the abstraction $(H_{filt,1})_a : 9(17)$.

Returning to Step 4 once more, the last subsystem $H_{5,a}$ is added to the composition, $(H_{filt,1})_a\|H_{5,a} : 9(17)$. Since the composition is nonblocking and no further subsystems remain, we are done. The resulting modular control achieved by the five original modular supervisors along with the conflict-resolving law $\mathcal{H}_{filt,1}$ satisfies the given specifications in a nonblocking manner and is nonempty. Table 3 summarizes the procedure applied in this example.

It turns out that the resulting modular solution is more restrictive than the monolithic solution in that it allows only five pieces to be operated on by the FMS at a given time, while the monolithic solution allows six pieces to be active at once. The loss of optimality of our approach arises in two ways due to the hiding of events. Namely, hiding an event means that we lose the ability to disable it. Also, the hiding of events causes us to lose information about what state the underlying plant is in, and as such forces us to employ a more conservative control law. This loss of optimality, however, is often worth the reduction in complexity the modular approach provides. Specifically, a measure of the complexity of the modular solution in the above example is that the largest automaton that had to be built had 128 states and 420 transitions. This automaton was constructed in the process of generating the modular supervisor $H_1$. In the monolithic approach, the composition of all the machines and buffers leads to an automaton with 13,248 states and 46,424 transitions. While the size of the resulting automata does not account for the complexity of the algorithms involved in generating the control laws and the abstractions, it does give some indication of the benefits of this approach.

Noting that the largest automaton constructed in the modular approach was the result of building a single modular supervisor, the overall complexity could possibly be reduced further by employing abstraction in the construction of the modular supervisors, in addition to using abstraction in the construction of the conflict-resolving filters. Specifically, results for the construction of individual supervisors from Wong and Wonham (1996), Feng and Wonham (2006) and Hill and Tilbury (2008) could be investigated.

**Table 3** Application of Algorithm 2 to FMS example

| Step | Automaton Built | States(Transitions) | Notes |
|---|---|---|---|
| 1 | $H_1$ | 80(259) | Note $G'_1 \| B_1$ : 128(420) |
| | $H_2$ | 28(74) | |
| | $H_3$ | 9(10) | |
| | $H_4$ | 6(6) | |
| | $H_5$ | 12(24) | |
| | $H_2 \rightarrow C_2$ | 2(3) | Supervisor reduction |
| 2 | $H_1 \rightarrow H_{1,a}$ | 31(115) | {61,64,65,66} hidden |
| | $H_3 \rightarrow H_{3,a}$ | 6(7) | {51,52,53,54} hidden $H_{3,a}$ is nondeterministic |
| | $H_4 \rightarrow H_{4,a}$ | 4(4) | {81,82} hidden |
| | $H_5 \rightarrow H_{5,a}$ | 4(6) | {21,22} hidden |
| 3 | | | $H_{1,a}$ chosen as the initial subsystem |
| 4 | $H_{1,a} \| C_2$ | 53(174) | $C_2$ chosen from neighboring subsystems |
| 5 | | | $H_{1,a} \| C_2$ is determined to be nonblocking |
| 6 | | | this step is skipped |
| 7 | $H_{1,a} \| C_2 \rightarrow (H_{1,a} \| C_2)_a$ | 27(79) | {63} hidden |
| 4 | $(H_{1,a} \| C_2)_a \| H_{3,a}$ | 45(120) | $H_{3,a}$ chosen from neighboring subsystems |
| 5 | | | $(H_{1,a} \| C_2)_a \| H_{3,a}$ is determined to be nonblocking |
| 6 | | | This step is skipped |
| 7 | $(H_{1,a} \| C_2)_a \| H_{3,a} \rightarrow ((H_{1,a} \| C_2)_a \| H_{3,a})_a$ | 42(115) | No further events hidden at this point |
| 4 | $((H_{1,a} \| C_2)_a \| H_{3,a})_a \| H_{4,a}$ | 61(52) | $H_{4,a}$ chosen from neighboring subsystems |
| 5 | | | $((H_{1,a} \| C_2)_a \| H_{3,a})_a \| H_{4,a}$ is determined to be blocking |
| 6 | $H_{filt,1}$ | 41(120) | Employ Algorithm 6 $H_{filt,1}$ replaces $((H_{1,a} \| C_2)_a \| H_{3,a})_a \| H_{4,a}$ |
| 7 | $H_{filt,1} \rightarrow (H_{filt,1})_a$ | 9(17) | {71,72,73,74} hidden |
| 4 | $(H_{filt,1})_a \| H_{5,a}$ | 9(17) | $H_{5,a}$ chosen from neighboring subsystems |
| 5 | | | $(H_{filt,1})_a \| H_{5,a}$ is determined to be nonblocking |
| 6 | | | This step is skipped |
| 7 | | | No further subsystems left, done |

Also, an improved modular solution can often be arrived at by changing the order in which subsystems are addressed or by changing the set of events that are considered silent along the way. For instance, if in the above example we had chosen not to hide the events 61, 63, and 65, the resulting modular solution would have allowed six pieces to be operated on by the FMS at a given time, just like the monolithic solution. This solution would have resulted in slightly larger automata, with the largest automaton constructed having 150 states and 352 transitions.

One drawback of this approach is that there is not a single approach to ordering subsystems that will result in the "best" overall solution. Some ordering heuristics that can help keep the overall complexity of the procedure down include first choosing subsystems that are either small or that offer the possibility of larger reduction. The work of Flordal and Malik (2006) offers a sizable survey of ordering heuristics applied to a variety of examples. Furthermore, implementation of a conflict-equivalent abstraction relies on an incomplete set of heuristic rules that do not in general provide a unique result.

Some heuristics for improving the optimality of this approach include "hiding" fewer events. In this way, reduction is traded for optimality. Furthermore, it is possible to change the outcome by not building a filter immediately following the detection of blocking. The idea here is that sometimes conflict is resolved by composition with other subsystems and ultimately a filter is not needed. The advantage of waiting here is that a filter cannot disable uncontrollable events and hence sometimes must remove states from an automaton, while interaction with other subsystems can prevent an uncontrollable event from happening in the first place so that it does not need to be actively disabled. The drawback of waiting to build the filter is that often the process of abstraction hides transitions that could be used to prevent blocking or violations of controllability. In general, ordering heuristics remain an open area for investigation.

This approach in general is well-suited to systems that are loosely coupled, as are other modular approaches to control. If a component specification shares relevant events with all plant components, then the achievable reduction will likely be modest, though in most cases it will still result in smaller automata being built than with the monolithic solution.

## 8 Conclusions

This paper has proposed a new approach for resolving conflict among supervised subsystems. Requirements are presented for conflict-resolving filter laws that guarantee safe nonblocking control when acting in conjunction with traditionally built modular supervisors. A methodology for building filter laws that avoids exponential complexity is also proposed. The coordinating filters are constructed based on conflict-equivalent abstractions, that offer the potential for a greater reduction in state-size than existing work on conflict resolution. Additionally, the covering-based feedback approach employed in building the filters generates a less restrictive control law than is achieved by existing state-feedback methodologies. A manufacturing example is also presented showing the overall potential of this approach.

A direction for future work is to investigate different ordering heuristics. Another important direction for this work is to develop a better understanding of how conflict-equivalent abstractions can be generated. Specifically, it would be useful to investigate the complexity associated with generating conflict-equivalent abstractions based on the heuristic rules of Flordal and Malik (2006) and Flordal (2006). It could also be interesting to explore the possibility that other rules could be developed for generating the abstractions. Another direction for this work is to find new ways to construct the conflict-resolving filters. Finally, the results of this work could be combined with other modular and hierarchical approaches to supervisory control to achieve even greater reduction in complexity.

**Acknowledgements** The authors thank the anonymous reviewers for their thorough comments that have helped to improve the clarity and accuracy of this paper. The kind assistance of the Associate Editor is also acknowledged.

## References

Brandin BA, Malik R, Malik P (2004) Incremental verification and synthesis of discrete-event systems guided by counter examples. IEEE Trans Control Syst Technol 12(3):387–401

Cassandras CG, Lafortune S (2007) Introduction to discrete event systems, 2nd edn. Springer, New York

de Queiroz MH, Cury JER (2000) Modular supervisory control of composed systems. In: Proceedings American control conf., Chicago, pp 4051–4055

de Queiroz MH, Cury JER, Wonham WM (2005) Multitasking supervisory control of discrete-event systems. Discret Event Dyn Syst Theory Appl 15:375–395

Fabian M, Lennartson B (1996) On non-deterministic supervisory control. In: Proceedings 35th IEEE Conf. decision & control, pp 2213–2218

Feng L (2007) Computationally efficient supervisor design in discrete-event systems. Ph.D. dissertation, University of Toronto, Toronto

Feng L, Wonham WM (2006) Computationally efficient supervisor design: abstraction and modularity. In: Proceedings int. workshop on discrete event systems (WODES), Ann Arbor, pp 3–8

Flordal H (2006) Compositional approaches in supervisory control. Ph.D. dissertation, Chalmers University of Technology, Gothenburg

Flordal H, Malik R (2006) Modular nonblocking verification using conflict equivalence. In: Proceedings int. workshop on discrete event systems (WODES), Ann Arbor, pp 100–106

Heymann M, Lin F (1996) Nonblocking supervisory control of nondeterministic systems. Technion, Israel Institute of Technology, Haifa, Tech. Rep. CIS-9620

Hill RC, Cury JER, de Queiroz MH, Tilbury DM (2008) Modular requirements for hierarchical interface-based supervisory control with multiple levels. In: Proceedings American control conf., Seattle, pp 483–490

Hill RC, Tilbury DM (2008) Incremental hierarchical construction of modular supervisors for discrete-event systems. Int J Control 81(9):1364–1381

Hill RC, Tilbury DM, Lafortune S (2008a) Covering-based supervisory control of partially observed discrete-event systems for state avoidance. In: Proceedings int. workshop on discrete event systems (WODES), Gothenburg, pp 2–8

Hill RC, Tilbury DM, Lafortune S (2008b) Modular supervisory control with equivalence-based conflict resolution. In: Proceedings American control conf., Seattle, pp 491–498

Hubbard P, Caines PE (1998) A state aggregation approach to hierarchical supervisory control with applications to a transfer line example. In: Proceedings int. workshop on discrete event systems (WODES), Cagliari

Inan K (1993) Supervisory control: theory and application to the gateway synthesis problem. In: Belgian-French-Netherlands summer school on discrete event systems, Spa

Kumar R, Shayman MA (1996) Non-blocking supervisory control of nondeterministic discrete-event systems via prioritized synchronoziation. IEEE Trans Automat Contr 41(8):1160–1175

Kumar R, Jiang S, Zhou C, Qiu W (2005) Polynomial synthesis of supervisor for partially observed discrete-event systems by allowing nondeterminism in control. IEEE Trans Automat Contr 50(4):463–475

Kumar R, Garg VK, Marcus SI (1993) Predicates and predicate transformers for supervisory control of discrete event dynamical systems. IEEE Trans Automat Contr 38:232–247

Leduc RJ, Brandin BA, Lawford M, Wonham WM (2005a) Hierarchical interface-based supervisory control—part I: serial case. IEEE Trans Automat Contr 50(9):1322–1335

Leduc RJ, Lawford M, Wonham WM (2005b) Hierarchical interface-based supervisory control—part II: parallel case. IEEE Trans Automat Contr 50(9):1336–1348

Li Y (1991) Control of vector discrete-event systems. Ph.D. dissertation, University of Toronto, Toronto

Li Y, Wonham WM (1993) Control of vector discrete event systems—part I: the base model. IEEE Trans Automat Contr 38(8):1215–1227

Lin F, Wonham WM (1988) Decentralized supervisory control of discrete-event systems. Inf Sci 44:199–224

Madhusudan P, Thiagarajan PS (2002) Branching time controlers for discrete event systems. Theor Comp Sci 274:117–149

Malik R, Streader D, Reeves S (2006) Conflicts and fair testing. Int J Found Comput Sci 17(4):797–813

Malik R, Flordal H, Pena P (2007) Conflicts and projections. In: Proceedings 1st IFAC workshop on dependable control of discrete systems (DCDS'07), pp 63–68

Malik P, Malik R, Streader D, Reeves S (2007) Modular synthesis of discrete controllers. In: Proceedings 12th IEEE international conference on engineering complex computer systems (ICECCS'07), pp 21–30

Milner R (1989) Communication and concurrency. Prentice-Hall, London

Overkamp A (1997) Supervisory control using failure semantics and partial specification. IEEE Trans Automat Contr 42:498–510

Park SJ, Lim JT (2000) Nonblocking supervisory control of nondeterministic systems based on multiple deterministic model approach. IEICE Trans Inf Syst E83-D(5):1177–1180

Pena P, Cury JER, Lafortune S (2006) Testing modularity of local supervisors: an approach based on abstractions. In: Proceedings int. workshop on discrete event systems (WODES), Ann Arbor, pp 107–112

Qin H, Lewis P (1991) Factorization of finite state machines under strong and observational equivalences. Form Asp Comput 3:284–307

Ramadge PJG, Wonham WM (1987) Modular feedback logic for discrete event systems. SIAM J Control Optim 25(5):1202–1218

Ramadge PJ, Wonham WM (1988) Modular supervisory control of discrete event systems. Math Control Signals Syst 1:13–30

Ramadge PJ, Wonham WM (1989) The control of discrete event systems. In: Proceedings of IEEE, vol 77, no. 1, pp 81–98

Schmidt K, Moor T, Perk S (2005) A hierarchical architecture for nonblocking control of discrete event systems. In: Mediterranean conf. control and automation, Limassol, pp 902–907

Su R, Thistle J (2006) A distributed supervisor synthesis approach based on weak bisimulation. In: Proceedings int. workshop on discrete Event systems (WODES), Ann Arbor, pp 64–69

Su R, Wonham WM (2004) Supervisor reduction for discrete-event systems. Discret Event Dyn Syst Theory Appl 14:31–53

Supremica (2009) Supremica homepage. http://www.supremica.org

Tabuada P (2004) Open maps, alternating simulations and control synthesis. In: International conference on concurrency theory, pp 466–480

Takai S, Kodama S (1997) M-controllable subpredicates arising in state feedback control of discrete event systems. Int J Control 67(4):553–566

Takai S, Kodama S (1998) Characterization of all M-controllable subpredicates of a given predicate. Int J Control 70(4):541–549

Takai S, Ushio T, Kodama S (1995) Static-state feedback control of discrete-event systems under partial observation. IEEE Trans Automat Contr 40(11):1950–1954

Wang W, Lafortune S, Lin F (2007) An algorithm for calculating indistinguishible states and clusters in finite state automata with partially observable transitions. Syst Control Lett 56:656–661

Wong KC, Wonham WM (1996) Hierarchical control of discrete-event systems. Discret Event Dyn Syst Theory Appl 6:241–273

Wong KC, Wonham WM (1998) Modular control and coordination of discrete-event systems. Discret Event Dyn Syst Theory Appl 8:247–297

Zhong H, Wonham WM (1990)On the consistency of hierarchical supervision in discrete-event systems. IEEE Trans Automat Contr 35(10):1125–1134

Zhou C, Kumar R, Jiang S (2006) Control of nondeterministic discrete-event systems for bisimulation equivalence. IEEE Trans Automat Contr 51(5):754–765

**Richard C. Hill** received the B.S. degree in Mechanical Engineering, *summa cum laude*, from the University of Southern California in 1998, and the M.S. degree in Mechanical Engineering from the University of California, Berkeley in 2000. From 2000 to 2002, he worked at Lockheed Martin Corporation on satellite attitude determination and control. He then spent two years as a high school math and science teacher. In 2008 he received the Ph.D. degree in Mechanical Engineering and the M.S. degree in Applied Mathematics from the University of Michigan, Ann Arbor. In 2008 Dr. Hill joined the faculty of the Mechanical Engineering Department at the University of Detroit Mercy. His research interests lie in the control and diagnosis of discrete-event systems, modular and hierarchical control, nonlinear control, and engineering education.

**Dawn M. Tilbury** received the B.S. degree in Electrical Engineering, *summa cum laude*, from the University of Minnesota in 1989, and the M.S. and Ph.D. degrees in Electrical Engineering and Computer Sciences from the University of California, Berkeley, in 1992 and 1994, respectively. In 1995, she joined the Mechanical Engineering Department at the University of Michigan, Ann Arbor, where she is currently Professor and Associate Chair, with a joint appointment as Professor of EECS. She won the EDUCOM Medal (jointly with Professor William Messner of Carnegie Mellon University) in 1997 for her work on the web-based Control Tutorials for Matlab. An expanded version, *Control Tutorials for Matlab and Simulink*, was published by Addison-Wesley in 1999. She is co-author (with Joseph Hellerstein, Yixin Diao, and Sujay Parekh) of the textbook *Feedback Control of Computing Systems*. She received an NSF CAREER award in 1999, and is the 2001 recipient of the Donald P. Eckman Award of the American Automatic Control Council. She was a member of the 2004–2005 class of the Defense Science Study Group (DSSG) and is a current member of DARPA's Information Science and Technology Study Group (ISAT). Her research interests include distributed control of mechanical systems with network communication, logic control of manufacturing systems,

cooperative control under uncertainty, and dynamic systems modeling of physiological systems. She belongs to ASME, IEEE, and SWE. She is an appointed member of the IEEE Control Systems Society Board of Governors for 2008, and began a five-year term on the ASME Dynamic Systems and Control Division Executive Committee in July 2008.

**Stéphane Lafortune** received the B. Eng degree from Ecole Polytechnique de Montréal in 1980, the M. Eng. degree from McGill University in 1982, and the Ph.D. degree from the University of California at Berkeley in 1986, all in electrical engineering. Since September 1986, he has been with the University of Michigan, Ann Arbor, where he is a Professor of Electrical Engineering and Computer Science. Dr. Lafortune is a Fellow of the IEEE (1999). He received the Presidential Young Investigator Award from the National Science Foundation in 1990 and the George S. Axelby Outstanding Paper Award from the Control Systems Society of the IEEE in 1994 (for a paper co-authored with S. L. Chung and F. Lin) and in 2001 (for a paper co-authored with G. Barrett). At the University of Michigan, he received the EECS Department Research Excellence Award in 1994–95, the EECS Department Teaching Excellence Award in 1997–98, and the EECS Outstanding Achievement Award in 2003–04. Dr. Lafortune is a member of the editorial boards of the Journal of Discrete Event Dynamic Systems: Theory and Applications and of the International Journal of Control. His research interest are in discrete event systems modeling, diagnosis, control, and optimization. He is co-developer of the software packages DESUMA and UMDES. He co-authored, with C. Cassandras, the textbook *Introduction to Discrete Event Systems - Second Edition* (Springer, 2007). Recent publications and software tools are available at the Web site www.eecs.umich.edu/umdes.