Supervision Equivalence

Hugo Flordal
Department of Signals and Systems
Chalmers University of Technology, Gothenburg, Sweden
flordal@chalmers.se

Robi Malik
Department of Computer Science
University of Waikato, Hamilton, New Zealand
robi@cs.waikato.ac.nz

***Abstract*— This paper presents a general framework for modular synthesis of supervisors for discrete event systems. The approach is based on compositional minimisation, using concepts of process equivalence. Its result is a compact representation of a least restrictive supervisor that ensures controllability and nonblocking. The method is demonstrated to reduce the number of states to be constructed for a simple manufacturing example, and the framework is proven to be sound.**

I. Introduction

The standard (monolithic) way to synthesise a controllable and nonblocking supervisor for a discrete event system is to build the synchronous composition of all components and search the state space. This method is known to suffer from the state-space explosion problem and therefore is only feasible for small systems. To handle larger systems, modular approaches to supervisor synthesis are of great interest and have long been studied in supervisory control theory [1]–[3].

Several approaches to modular synthesis of discrete event systems have been suggested in the literature [4], [5], but so far most of them rely on structure to be provided by users and hence are hard to automate. Those that can be automated [6]–[9] either do not consider nonblocking, or are guaranteed to produce a least restrictive supervisor only under certain constraints.

Using ideas of process equivalence [10], this paper proposes a general framework for modular synthesis of least restrictive controllable and nonblocking supervisors, which can be fully automated. The method efficiently handles supervisory control problems given as a large set of small finite-state automata.

Section II demonstrates the proposed method using a simple example. Then, Section III provides the formal notation needed for automata and supervisory control, and Section IV explains the synthesis framework in detail, with a formal proof of its soundness. Finally, Section V contains some concluding remarks.

II. Motivating Example

This section demonstrates the ideas of the modular synthesis procedure using a simple manufacturing example, the transfer line originally given in [1]. It consists of two machines M_1 and M_2 and a test unit T, linked by two buffers B_1 and B_2. A finite-state automata model of this system is shown in Fig. 1. Automata M_1, M_2, and T constitute the plant model, while B_1 and B_2 are specifications. Uncontrollable events are prefixed by an exclamation mark (!).

The modular synthesis procedure presupposes the system model to be given as a set of *plant* models only. Therefore, the buffer *specifications* B_1 and B_2 are first transformed into plants B_1' and B_2'. This is a straightforward operation: wherever an uncontrollable event is disabled, a transition on that event to a dump state $\perp$ is added. The result is shown in Fig. 2. This transformation produces an equivalent supervisory control problem using only plants if both controllability and nonblocking are considered.

Modular synthesis is performed as a series of small steps that in the end result in a simplified representation of the least restrictive supervisor. The intermediate steps strive to avoid state explosion by simplifying different parts of the system to something that preserves all information necessary for the synthesis. That is, to something *supervision equivalent*.

For example, when composing the modules M_2 and B_2', the uncontrollable event f_2 becomes local to the subsystem $M_2 \,\|\, B_2'$, i.e., transitions associated with f_2 can never be disabled in future compositions with other modules. Therefore, the identity of event f_2 is no longer important, so all its occurrences are replaced by τ_u, an identity-less uncontrollable event. The resultant automaton, $H_1' = (M_2 \parallel B_2') \setminus_! \{f_2\}$, where $\setminus_!$ denotes this special kind of hiding, is shown in Fig. 3. In a similar fashion, all local controllable events are replaced by the identity-less controllable event τ_c.

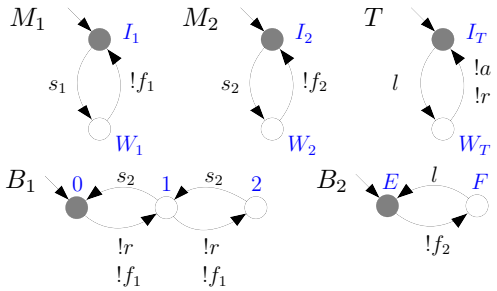

Fig. 1. The transfer line example.

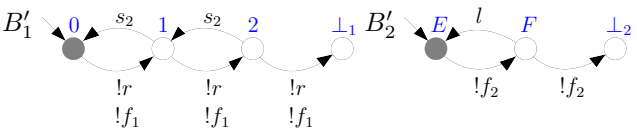

Fig. 2. The buffer specifi cations as plant models.

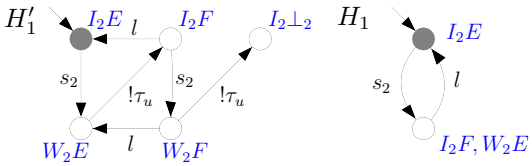

Fig. 3. The simplification of $H_1' = (M_2 \parallel B_2') \setminus_! \{f_2\}$ (to the left) results in the automaton H_1 (to the right).

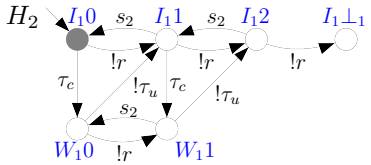

Fig. 4. The result of the simplification of $(M_1 \parallel B_1') \setminus_! \{f_1, s_1\}$.

The intermediate automata obtained by such manipulations can obviously be nondeterministic. However, this nondeterminism in interpreted in a special way by the modular synthesis procedure—it is assumed that a potential supervisor can individually enable or disable each controllable *transition* of a nondeterministic automaton. This is an appropriate assumption, because the nondeterministic automata have been obtained by abstraction from deterministic automata in the original model, so the supervisor actually can distinguish between the different transitions. The information needed for this distinction is carried along using *state labels* that relate the states of the intermediate automata to the states in the original model. For example, the state label I_2E in Fig. 3 represents a state of the original system, where automaton M_2 is in state I_2 and B_2 is in state E.

After hiding, the five-state automaton H_1' can be replaced by the supervision equivalent two-state automaton H_1, also shown in Fig. 3. To see that these automata are equivalent, consider a supervisor that is to enforce controllability and nonblocking. Clearly, state $I_2\perp_2$ must be avoided since it is blocking. Moreover, state W_2F must be avoided, too, since there is an outgoing uncontrollable transition leading to a blocking state, which no other process can disable. So, both $I_2\perp_2$ and W_2F must be avoided whatever the other modules in the system look like. Thus, already at this point it is clear that the controllable transition from I_2F to W_2F must be and can be prevented by a supervisor—states $I_2\perp_2$ and W_2F can be removed. Furthermore, states I_2F and W_2E can be merged into a single state since a supervisor that allows the plant to reach W_2E cannot do anything but accept that the plant may uncontrollably transit to I_2F.

Figures 4 and 5 show further simplification results, H_2

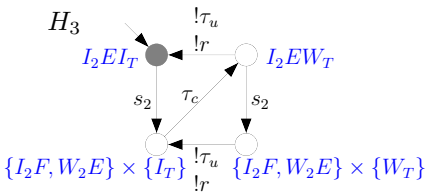

Fig. 5. The result of the simplification of $(H_1 \parallel T) \setminus_! \{a, l\}$.

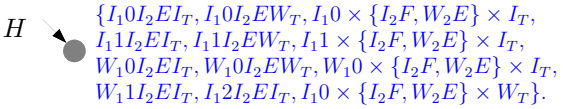

Fig. 6. The result of the simplification of $(H_2 \parallel H_3) \setminus_! \{s_2, r\}$.

derives from $(M_1 \parallel B_1') \setminus_! \{f_1, s_1\}$ and H_3 from $(H_1 \parallel T) \setminus_! \{a, l\}$. Fig. 6 shows the end result H, a simplified version of $(H_2 \parallel H_3) \setminus_! \{s_2, r\}$. This final result does not share any events with other components so all events can be hidden. After simplification this always results in an automaton with one or, if controllable and nonblocking supervision is not possible, zero states.

In the process of producing the final result, the largest intermediate automaton, $H_2 \parallel H_3$, has 21 states and 45 transitions. These figures should be compared to the corresponding values for the monolithic approach, which calculates the supervisor directly from the composed system $G = M_1 \parallel B_1 \parallel M_2 \parallel B_2 \parallel T$, an automaton with 48 states and 120 transitions.

Now, how is the result supposed to be used? Given the final result H and the original automata M_1, B_1, M_2, B_2, and T, it is possible to construct a supervisor that yields the least restrictive behaviour that is controllable and nonblocking. The original automata are used to observe the system and determine the current global state. The objective of supervision is to avoid "bad" states, i.e., states that are blocking or uncontrollable or that may lead to such states. The final result H can be used to determine whether a state is "bad", and thereby to decide which events can be enabled. This is possible because, at each stage in the construction of H, there is a clear correspondence between the states of the intermediate and the original automata. This correspondence is propagated and stored using *labels* in such a way that the single state of H has labels representing all states that are reachable under a least restrictive supervisor.

For instance, assume that the transfer line system is in the global state $I_12I_2EI_T$. An inspection of the modular model shows two possible transitions, associated with controllable events s_1 and s_2. Event s_1 would lead the system to state $W_12I_2EI_T$, but since its label *cannot* be found in H (actually no label starting with "W_12" can be found there) the supervisor *disables* s_1. Event s_2, on the other hand, leads to state $I_11W_2EI_T$ whose label *can* be found in H (on the second line in Fig. 6 if the last expression is unfolded). Therefore, the supervisor *enables* s_2.

III. Notation and Preliminaries

A. Events and Strings

Event sequences and languages are a simple means to describe discrete system behaviours. Their basic building blocks are *events*, which are taken from a finite *alphabet* Σ. For the purpose of supervisory control, the event alphabet Σ is partitioned into the set Σ_c of *controllable* events and the set Σ_u of *uncontrollable* events. There are two special events, the *silent* controllable event τ_c and the *silent* uncontrollable event τ_u. These do not belong to Σ, Σ_c, Σ_u. If they are to be

included, the alphabets $\Sigma_\tau = \Sigma \cup \{\tau_c, \tau_u\}$, $\Sigma_{c,\tau} = \Sigma_c \cup \{\tau_c\}$, and $\Sigma_{u,\tau} = \Sigma_u \cup \{\tau_u\}$ are used instead.

Σ^* denotes the set of all finite *strings* of the form $\sigma_1\sigma_2\cdots\sigma_k$ of events from Σ, including the *empty string* ε. The *catenation* of two strings $s, t \in \Sigma^*$ is written as st.

B. Nondeterministic Automata

System behaviours are represented using finite-state automata. Nondeterminism is used to support hiding, which is essential for modular synthesis in this approach.

Definition 1: A (nondeterministic) *finite-state automaton* is a 5-tuple $G = \langle Q, \Sigma, \rightarrow, Q^i, Q^m\rangle$, where Σ is a finite alphabet of events, Q is a finite set of *states*, $\rightarrow \subseteq Q \times \Sigma_\tau \times Q$ is the *state transition relation*, $Q^i \subseteq Q$ is the set of *initial states*, and $Q^m \subseteq Q$ is the set of *marked states*. ■

Note that the silent events are allowed in $\rightarrow$ even though they are never included in the alphabet of an automaton.

The transition relation is written in infix notation $p \xrightarrow{\sigma} q$, and is extended to strings in Σ_τ^* by letting

$$p \xrightarrow{\varepsilon} p \qquad \text{for all } p \in Q\ ; \tag{1}$$

$$p \xrightarrow{s\sigma} q \qquad \text{if } p \xrightarrow{s} r \text{ and } r \xrightarrow{\sigma} q \text{ for some } r \in Q\ . \tag{2}$$

For state sets $Q_1, Q_2 \subseteq Q$, $Q_1 \xrightarrow{s} Q_2$ denotes the existence of $q_1 \in Q_1$ and $q_2 \in Q_2$ such that $q_1 \xrightarrow{s} q_2$. Similarly, $p \rightarrow q$ means that there exists a string $s \in \Sigma_\tau^*$ such that $p \xrightarrow{s} q$. Finally, $p \xrightarrow{s}$ denotes that there exists a state q such that $p \xrightarrow{s} q$, and for an automaton G, $G \xrightarrow{s} q$ means $Q^i \xrightarrow{s} q$.

A state q is called *reachable* in an automaton G if $G \rightarrow q$; if this holds for all $q \in Q$, then G is called *accessible*. If G is not accessible, it can easily be made so by removing all states that are not reachable. Therefore, in the following all automata are assumed to be accessible.

Definition 2: An automaton $G = \langle Q, \Sigma, \rightarrow, Q^i, Q^m\rangle$ is *deterministic* if Q^i is a singleton, $p \xrightarrow{\sigma} q_1$ and $p \xrightarrow{\sigma} q_2$ always implies $q_1 = q_2$, and $\rightarrow$ contains no transitions labelled τ_c or τ_u. ■

Various operations can be used to modify or combine automata. For modular synthesis, synchronous composition [11] and hiding are the most important.

Definition 3: Let $G_1 = \langle Q_1, \Sigma, \rightarrow_1, Q_1^i, Q_1^m\rangle$ and $G_2 = \langle Q_2, \Sigma, \rightarrow_2, Q_2^i, Q_2^m\rangle$ be two automata using the same alphabet. The *synchronous product* of G_1 and G_2 is

$$G_1 \parallel G_2 \ = \ \langle Q_1 \times Q_2, \Sigma, \rightarrow, Q_1^i \times Q_2^i, Q_1^m \times Q_2^m\rangle \tag{3}$$

where

$$\begin{aligned}
&(p,q) \xrightarrow{\sigma} (p',q') \text{ if } \sigma \in \Sigma,\ p \xrightarrow{\sigma}_1 p' \text{ and } q \xrightarrow{\sigma}_2 q'\ ;\\
&(p,q) \xrightarrow{\sigma} (p',q) \text{ if } \sigma \in \{\tau_c, \tau_u\} \text{ and } p \xrightarrow{\sigma}_1 p'\ ;\\
&(p,q) \xrightarrow{\sigma} (p,q') \text{ if } \sigma \in \{\tau_c, \tau_u\} \text{ and } q \xrightarrow{\sigma}_2 q'\ .
\end{aligned}$$ ■

If the two automata to be combined do not use the same alphabet, they first have to be *extended* to their united alphabet. An automaton using alphabet Σ is extended to $\Sigma' \supseteq \Sigma$ by adding *selfloops* $q \xrightarrow{\sigma} q$ for all states $q \in Q$ and all events $\sigma \in \Sigma' \setminus \Sigma$.

Definition 4: Let $G = \langle Q, \Sigma, \rightarrow, Q^i, Q^m\rangle$ be an automaton, and let $\Upsilon \subseteq \Sigma$. The result of *controllability preserving hiding*, hiding henceforth, of Υ from G is

$$G \setminus_! \Upsilon \ = \ \langle Q, \Sigma \setminus \Upsilon, \rightarrow_!, Q^i, Q^m\rangle \tag{4}$$

where $\rightarrow_!$ is obtained from $\rightarrow$ by replacing each transition $p \xrightarrow{\sigma} q$ such that $\sigma \in \Upsilon$ by $p \xrightarrow{\tau_c}_! q$ if $\sigma \in \Sigma_c$ or by $p \xrightarrow{\tau_u}_! q$ if $\sigma \in \Sigma_u$. ■

Hiding removes the identity of the events in Υ and in general produces a nondeterministic automaton.

By introducing concepts of *subautomata* and *union* of automata, the set of automata can be considered as a lattice.

Definition 5: Let $G_1 = \langle Q_1, \Sigma, \rightarrow_1, Q^i, Q_1^m\rangle$ and $G_2 = \langle Q_2, \Sigma, \rightarrow_2, Q^i, Q_2^m\rangle$ be two automata with the same alphabet and initial states. G_1 is a *subautomaton* of G_2, $G_1 \subseteq G_2$, if $Q_1 \subseteq Q_2$, $\rightarrow_1 \subseteq \rightarrow_2$, and $Q_1^m \subseteq Q_2^m$. ■

Definition 6: Let $G_j = \langle Q_j, \Sigma, \rightarrow_j, Q^i, Q_j^m\rangle$, $j \in J$ be a family of automata all having the same alphabet and set of initial states. Define

$$\bigcup_{j\in J} G_j \ = \ \langle \bigcup_{j\in J} Q_j, \Sigma, \bigcup_{j\in J} \rightarrow_j, Q^i, \bigcup_{j\in J} Q_j^m\rangle\ .$$ ■

C. Supervision and Synthesis

Supervisors are used to restrict the behaviour of systems represented by automata. A supervisor observes the sequence of events occurring in the system and then enables or disables certain controllable events, but it cannot disable any uncontrollable events. Formally, this can be considered as a map S, where $S(s)$ represents the set of events enabled by the supervisor after observing the system execute the string s.

Definition 7: Let $G = \langle Q, \Sigma, \rightarrow, Q^i, Q^m\rangle$ be an automaton. A *supervisor* for G is a map $S\colon \Sigma^* \rightarrow 2^\Sigma$, such that $S(s) \supseteq \Sigma_u$ for all $s \in \Sigma^*$. ■

Given a plant behaviour G, and a desired behaviour K, it is of interest to construct a supervisor for G that yields exactly the behaviour K. Supervisory control theory shows that the behaviour K has to be *controllable* for a supervisor to exist [2], [3]. Below are two definitions of controllability used for the nondeterministic setting of this paper.

Definition 8: Let G and K be two automata using the same alphabet Σ. K is *controllable* with respect to G if, for every string $s \in \Sigma^*$, every state q of K, and every uncontrollable event $\upsilon \in \Sigma_u$ such that $K \xrightarrow{s} q$ and $G \xrightarrow{s\upsilon}$, it holds that $q \xrightarrow{\upsilon}$. ■

Definition 9: Let G be an automaton, and let $K \subseteq G$ be a subautomaton of G. K is *controllable* in G if, for every string $s \in \Sigma_\tau^*$, every uncontrollable event $\upsilon \in \Sigma_{u,\tau}$, and all states $p, q \in Q$ such that $K \xrightarrow{s} p$ and $G \xrightarrow{s} p \xrightarrow{\upsilon} q$, it holds that $K \xrightarrow{s} p \xrightarrow{\upsilon} q$. ■

Def. 8 corresponds to the original controllability definition from [2]. The two definitions coincide when K is a subautomaton of a deterministic automaton G. In the nondeterministic case, Def. 9 assumes that a supervisor can disable each *transition* individually unlike in traditional supervisory control. Here, this definition makes sense because nondeterministic automata always derive from deterministic automata, and the supervisor is assumed to distinguish

different transitions using its knowledge about the global state. In the following, be aware of the difference between controllable *with respect to* and controllable *in*.

In addition to controllability, the behaviour of a supervised system is typically also required to be nonblocking [2], [12].

Definition 10: An automaton $G = \langle Q, \Sigma, \rightarrow, Q^i, Q^m \rangle$ is called *nonblocking* if, for every state q such that $Q^i \rightarrow q$ it holds that $q \rightarrow Q^m$. ■

Similar to traditional supervisory control theory [2], it can be shown that the union of controllable and nonblocking subautomata of a given automaton is again controllable and nonblocking. This justifies the following definition.

Definition 11: Let G be an automaton. The supremal controllable and nonblocking subautomaton of G is

$$\sup\mathcal{CN}(G) \;=\; \bigcup \{\, G' \subseteq G \mid G' \text{ is controllable in } G \text{ and nonblocking} \,\} \;. \qquad \blacksquare$$

Unlike traditional synthesis [2], $\sup\mathcal{CN}$ merely describes a controllable and nonblocking sub-behaviour of the given *plant* G. *Specifications* are not considered here since they can easily be translated into plants.

D. Translation of Specifications into Plants

A specification automaton can be transformed into a plant by adding, for every uncontrollable event that is not enabled in a state, a transition to a new blocking state $\perp$. The result of synthesis remains the same after this transformation.

Definition 12: Let $K = \langle Q, \Sigma, \rightarrow, Q^i, Q^m \rangle$ be a specification. The *complete plant automaton* $K_\perp$ for K is

$$K_\perp \;=\; \langle Q \cup \{\perp\}, \Sigma, \rightarrow_\perp, Q^i, Q^m \rangle \tag{5}$$

where $\perp \notin Q$ is a new state and

$$\rightarrow_\perp \;=\; \rightarrow \cup \{\, (q, \upsilon, \perp) \mid q \in Q, \upsilon \in \Sigma_u, q \not\xrightarrow{\upsilon} \,\} \;. \qquad \blacksquare$$

Whenever the specification disallows an uncontrollable event enabled by the plant, after the transformation, synchronous composition results in an uncontrollable transition to a blocking state.

In this way, controllability problems are translated into blocking problems. The synthesis procedure treats all possible causes of nonblocking alike. Thus, it resolves controllability problems in the same way as it resolves blocking—a computed nonblocking supervisor will in addition yield a controllable solution to the original problem.

Proposition 1: Let G, K, and K' be deterministic automata over the same alphabet Σ. Then the following two statements are equivalent.

(i) $K' \subseteq G \parallel K$ is nonblocking and controllable with respect to G.

(ii) $K' \subseteq G \| K_\perp$ is nonblocking and controllable in $G \| K_\perp$. □

Proof: First, assume that (i) holds. Clearly, since $K \subseteq K_\perp$, it follows that $K' \subseteq G \parallel K \subseteq G \parallel K_\perp$. Also, K' is nonblocking by assumption.

It remains to show that K' is controllable in $G \parallel K_\perp$. Let $s \in \Sigma^*$ and $\upsilon \in \Sigma_u$ be such that $K' \xrightarrow{s} p$ and $G \parallel K_\perp \xrightarrow{s} p \xrightarrow{\upsilon} q$. By the definition of $\parallel$, it is clear that $G \xrightarrow{s\upsilon}$. Thus, since K' is controllable with respect to G, it follows that $K' \xrightarrow{s\upsilon}$. Since K' is a deterministic automaton, this implies $K' \xrightarrow{s} p \xrightarrow{\upsilon} q$.

Second, assume that (ii) holds. Since, by the assumption, K' is nonblocking, it holds that $K' \not\rightarrow (q, \perp)$ for every state q in G. Thus, since $K_\perp$ is the complete plant automaton for K, it holds that $K' \subseteq G \parallel K_\perp$ implies $K' \subseteq G \parallel K$.

It remains to show that K' is controllable with respect to G. Let $s \in \Sigma^*$ and $\upsilon \in \Sigma_u$ such that $K' \xrightarrow{s} (p_G, p_K)$ and $G \xrightarrow{s} p_G \xrightarrow{\upsilon} q_G$. Since $K' \subseteq G \| K_\perp$ it holds that $K_\perp \xrightarrow{s} p_K$. Since $\upsilon \in \Sigma_u$ and since $K_\perp$ is a complete plant automaton for K, there exists a state $q_\perp$ such that $K_\perp \xrightarrow{s} p_K \xrightarrow{\upsilon} q_\perp$. This implies $G \parallel K_\perp \xrightarrow{s} (p_G, p_K) \xrightarrow{\upsilon} (q_G, q_\perp)$. Since K' is controllable in $G \| K_\perp$, it holds that $(p_G, p_K) \xrightarrow{\upsilon}$ in K'. ■

An immediate consequence of this result is that synthesis of the least restrictive nonblocking and controllable behaviour allowed by a specification K with respect to a plant G can be achieved by computing

$$\sup\mathcal{CN}(G \parallel K_\perp) \;. \tag{6}$$

This automaton can be used to implement a supervisor, enabling precisely the events enabled in $\sup\mathcal{CN}(G \parallel K_\perp)$. However, if hiding and/or abstractions have been made the (potentially nondeterministic) automaton no longer explicitly shows which events can be enabled in a certain state. To overcome this problem, state labels are introduced to convey the necessary information.

E. Kripke-Structures

The intermediate results in the construction of the supervisor carry state labels to establish a correspondence to the global system states. To this end, labelled automata or *Kripke-structures* are used.

Definition 13: An (extended) *Kripke-structure* is a 7-tuple $G = \langle Q, \Sigma, \rightarrow, Q^i, Q^m, \Omega, B \rangle$ where $\langle Q, \Sigma, \rightarrow, Q^i, Q^m \rangle$ is an automaton, Ω is a set of *state labels*, and $B : Q \rightarrow 2^\Omega$ is a map that associates each state with a set of labels. ■

A Kripke-structure can be considered as an automaton, simply by ignoring its labels. Conversely, an unlabelled automaton $G = \langle Q, \Sigma, \rightarrow, Q^i, Q^m \rangle$ can be extended to a Kripke-structure G' by labelling each state with its name,

$$G' \;=\; \langle Q, \Sigma, \rightarrow, Q^i, Q^m, \Omega, B \rangle \tag{7}$$

where $\Omega = Q$ and $B(q) = \{q\}$. Simplification may result in states with more than one label associated to them. In the following, the letter G is used to represent both an automaton and its Kripke-structure. All concepts and notations that can be applied to automata, such as transitions and controllability, are extended to Kripke-structures in the straightforward way.

Synchronous composition produces state tuples as labels and is extended to Kripke-structures using

$$\Omega_{G_1 \| G_2} \;=\; \Omega_{G_1} \times \Omega_{G_2} \;; \tag{8}$$

$$B_{G_1 \| G_2}(p, q) \;=\; B_{G_1}(p) \times B_{G_2}(q) \;. \tag{9}$$

Here, the resulting labels should not depend on the order in which automata are composed. Therefore pairs (p, q)

and (q, p), e.g., are considered as equivalent when occurring as labels. To this effect, it is also assumed that all composed automata use different state names.

Sometimes it is of interest to know the set of all reachable labels in a Kripke-structure G, which is defined as

$$B(G) \;=\; \{\, \omega \in \Omega \;\mid \exists q \in Q\colon G \to q,\; \omega \in B(q) \,\}\;. \quad (10)$$

IV. Modular Synthesis

A modular supervisory control problem consists of a plant $G = G_1 \parallel \cdots \parallel G_n$ and a specification $K = K_1 \parallel \cdots \parallel K_m$, each composed of *deterministic* automata. The task is to find the supremal controllable and nonblocking sub-behaviour of

$$G \parallel K \;=\; G_1 \parallel \cdots \parallel G_n \parallel K_1 \parallel \cdots \parallel K_m\;, \quad (11)$$

or, equivalently, the largest subautomaton of $G \parallel K$ that is nonblocking and controllable (in $G \parallel K$).

Proposition 1 shows that this can be represented equivalently by a set of plant automata. Therefore, in the following it is assumed without loss of generality that the synthesis problem consists of finding a nonblocking and controllable supervisor for a modular deterministic plant

$$G \;=\; G_1 \parallel \cdots \parallel G_n\;, \quad (12)$$

represented as Kripke-structures where each state is labelled by its name. This is the starting point for modular synthesis.

A. Compositional Minimisation

The supervisor should result in the least restrictive nonblocking sub-behaviour of the system G in (12). Such a supervisor can be described by

$$S^{\circ}_G(s) \;=\; \{\, \sigma \in \Sigma \mid \sup\mathcal{CN}(G) \xrightarrow{s\sigma} \,\}\;. \quad (13)$$

To avoid monolithic synthesis and compute this supervisor in a modular way, the system G of plant automata is transformed into a simpler system

$$H \;=\; H_1 \parallel \cdots \parallel H_m \quad (14)$$

that should be related to the original system G in an appropriate way. The simplified system H is assumed to use the same state labels as the original system G, i.e., it is labelled by the names of the states in G. Using these labels, the simplified system can also be used to define a supervisor as follows.

$$S_H(s) \;=\; \{\, \sigma \in \Sigma \mid G \xrightarrow{s\sigma} q \text{ and } q \in B(\sup\mathcal{CN}(H)) \,\}\;. \quad (15)$$

To determine whether an event σ should be enabled after executing a string s, this supervisor first determines whether the original system G can execute string $s\sigma$. If this is the case, it checks whether the state reached by G is a reachable label of $\sup\mathcal{CN}(H)$. If this is the case, the event σ is enabled, otherwise it is disabled.

To be useful, the supervisor constructed in this way should produce exactly the same behaviour as a monolithic supervisor computed for the original system. Clearly, this can only be guaranteed if the simplified system H stands in a certain relationship to the original system G.

Three operations are proposed that can be applied to a system of plant automata in such a way that the resultant supervisor yields the same behaviour.

Synchronous Composition. Any two plant automata can be replaced by their synchronous product.

Hiding. If an event σ is used by only one automaton G_i, then G_i can be replaced by $G_i \setminus\!! \{\sigma\}$.

Simplification. A plant automaton can be replaced by a simplified automaton, provided that that simplified automaton is *supervision equivalent* to the original.

The simplification step clearly relies on an appropriate notion of equivalence to guarantee that the resultant supervisor remains unchanged. The following definition introduces a general equivalence that relates two automata with equivalent synthesis results in combination with any other system. Synthesis results for Kripke-structures can be considered as equivalent if they result in the same sets of state labels.

Definition 14: Two Kripke-structures G and H are said to be *supervision equivalent*, denoted $G \simeq_{\sup} H$, if, for any automaton T,

$$B(\sup\mathcal{CN}(G \parallel T)) \;=\; B(\sup\mathcal{CN}(H \parallel T))\;. \qquad \blacksquare$$

Definition 15: The following rules can be used to rewrite sets of Kripke-structures $G_1, \ldots, G_n$.

$$\frac{\{G_1, \ldots, G_n\}}{\{G_1 \parallel G_2, G_3, \ldots, G_n\}}$$

$$\frac{\{G_1, \ldots, G_n\}}{\{G_1 \setminus\!! \Upsilon, G_2, \ldots, G_n\}} \quad \text{if events } \Upsilon \subseteq \Sigma \text{ are unused in } G_2, \ldots, G_n;$$

$$\frac{\{G_1, \ldots, G_n\}}{\{H_1, G_2, \ldots, G_n\}} \quad \text{if } G_1 \simeq_{\sup} H_1.$$

If $\{G_1, \ldots, G_n\}$ can be transformed into $\{H_1, \ldots, H_m\}$ using one of the above rules, then this is denoted by $\{G_1, \ldots, G_n\} \succ \{H_1, \ldots, H_m\}$. The reflexive and transitive closure of the rewrite relation $\succ$ is denoted by $\succ^*$. $\blacksquare$

B. Main Result

In order to construct the supervisor efficiently, the system of plants (12) is repeatedly rewritten and simplified using the rewrite rules given in Def. 15, until all automata have been composed and reduced to a one-state automaton.

The following result shows that this method is sound. The sequence of rewrite steps always leads to a supervisor that is equivalent to the monolithic supervisor for the original system, and therefore yields the least restrictive behaviour that can be achieved by control.

Proposition 2: Let $G = G_1 \parallel \cdots \parallel G_n$ be a system of deterministic plant automata, and let $H = H_1 \parallel \cdots \parallel H_m$ be a system of Kripke-structures such that

$$\{G_1, \ldots, G_n\} \succ^* \{H_1, \ldots, H_m\}\;. \quad (16)$$

Then their synthesised supervisors yield the same behaviour, i.e., $S^{\circ}_G = S_H$. $\square$

Proof: The claim is proved by induction on the number of rewrite steps used to transform G into H.

Base case. First assume that $G = H$, i.e., no rewrite steps have been used. It needs to be shown that the monolithic

and modular supervisors obtained from G are equal, i.e., that $S^\circ_G = S_G$. This is the case because, for arbitrary $s \in \Sigma^*$,

$$\begin{aligned} S^\circ_G(s) &= \{\, \sigma \in \Sigma \mid \mathrm{sup}\mathcal{CN}(G) \xrightarrow{s\sigma} \,\} \\ &= \{\, \sigma \in \Sigma \mid G \xrightarrow{s\sigma} q, q \text{ reachable in } \mathrm{sup}\mathcal{CN}(G) \,\} \\ &= \{\, \sigma \in \Sigma \mid G \xrightarrow{s\sigma} q, q \in B(\mathrm{sup}\mathcal{CN}(G)) \,\} \\ &= S_G(s) \ . \end{aligned}$$

Inductive Step. Assume that G is rewritten into H in $k+1$ rewrite steps as follows

$$G = G^0 \succ \cdots \succ G^k \succ G^{k+1} = H \ . \tag{17}$$

By inductive assumption, the modular supervisor for G^k behaves like the monolithic supervisor for G, i.e., $S^\circ_G = S_{G^k}$. It remains to be shown that

$$S_{G^k} \ = \ S_H \ , \tag{18}$$

where H is obtained from G^k using one of the rewrite rules from Def. 15.

Case 1. H is obtained via synchronous composition from G^k. Let $G^k = \{G_1, G_2, \ldots, G_n\}$, and let $H = \{G_1 \parallel G_2, G_3, \ldots, G_n\}$. By definition of synchronous composition and synthesis, and using the assumption that state labels are constructed in such a way that the order of composition does not matter, it follows that

$$\begin{aligned} B(\mathrm{sup}\mathcal{CN}(G^k)) &= B(\mathrm{sup}\mathcal{CN}(G_1 \parallel G_2 \parallel \cdots \parallel G_n)) \\ &= B(\mathrm{sup}\mathcal{CN}((G_1 \parallel G_2) \parallel \cdots \parallel G_n)) \\ &= B(\mathrm{sup}\mathcal{CN}(H)) \ . \end{aligned}$$

Case 2. H is obtained via hiding from G^k. Let $G^k = \{G_1, G_2, \ldots, G_n\}$ and $H = \{G_1 \setminus_! \Upsilon, G_2, \ldots, G_n\}$ where all events in Υ are unused in $G_2 \parallel \cdots \parallel G_n$. Since synthesis treats the silent events τ_c and τ_u in the same way as ordinary controllable and uncontrollable events, it follows that

$$\begin{aligned} B(\mathrm{sup}\mathcal{CN}(G^k)) &= B(\mathrm{sup}\mathcal{CN}(G_1 \parallel \cdots \parallel G_n)) \\ &= B(\mathrm{sup}\mathcal{CN}((G_1 \parallel \cdots \parallel G_n) \setminus_! \Upsilon)) \\ &= B(\mathrm{sup}\mathcal{CN}((G_1 \setminus_! \Upsilon) \parallel G_2 \parallel \cdots \parallel G_n)) \\ &= B(\mathrm{sup}\mathcal{CN}(H)) \ . \end{aligned}$$

Case 3. H is obtained via simplification from G^k. Let $G^k = \{G_1, G_2, \ldots, G_n\}$ and $H = \{H_1, G_2, \ldots, G_n\}$ where $G_1 \simeq_{\mathrm{sup}} H_1$. By letting $T = G_2 \parallel \cdots \parallel G_n$ in Def. 14,

$$\begin{aligned} B(\mathrm{sup}\mathcal{CN}(G^k)) &= B(\mathrm{sup}\mathcal{CN}(G_1 \parallel G_2 \parallel \cdots \parallel G_n)) \\ &= B(\mathrm{sup}\mathcal{CN}(H_1 \parallel G_2 \parallel \cdots \parallel G_n)) \\ &= B(\mathrm{sup}\mathcal{CN}(H)) \ . \end{aligned}$$

Thus, the equation $B(\mathrm{sup}\mathcal{CN}(G^k)) = B(\mathrm{sup}\mathcal{CN}(H))$ holds in all three cases. This implies, for arbitrary $s \in \Sigma^*$,

$$\begin{aligned} S^\circ_G(s) &= S_{G^k}(s) \\ &= \{\, \sigma \in \Sigma \mid G \xrightarrow{s\sigma} q, q \in B(\mathrm{sup}\mathcal{CN}(G^k)) \,\} \\ &= \{\, \sigma \in \Sigma \mid G \xrightarrow{s\sigma} q, q \in B(\mathrm{sup}\mathcal{CN}(H)) \,\} \\ &= S_H(s) \ . \end{aligned}$$

■

C. Supervision Equivalence Preserving Operations

Simplification is the only step that reduces the size of intermediate automata, and is therefore crucial for the performance of the method. Since the state space tends to grow exponentially with the number of components, even a small reduction, particularly at an early stage, can greatly reduce effort later in the process.

There are various ways how an automaton can be rewritten to a simpler supervision equivalent version. A simple but powerful method is called *"half-way" synthesis*. The idea is to perform synthesis on a subsystem but, to guarantee that the end result is maximally permissive, the synthesis must take into consideration that all uncontrollable events except τ_u may actually become disabled by other subsystems. Thus, transitions associated with such events are not sure to cause uncontrollability and must be retained to guarantee maximal permissiveness.

Definition 16: Let $G = \langle Q, \Sigma, \rightarrow, Q^i, Q^m \rangle$. The set of "bad" states for "half-way" synthesis is the smallest state set $Q^x \subseteq Q$ that satisfies the following two conditions.

- If for some state $p \in Q$, no path to a marked state

$$p = q_0 \rightarrow q_1 \rightarrow \cdots \rightarrow q_n \in Q^m \tag{19}$$

  exists that does not pass through a "bad" state $q_i \in Q^x$, then $p \in Q^x$.
- If for some state $p \in Q$ it holds that $p \xrightarrow{\tau_u} Q^x$ then $p \in Q^x$.

If $Q^x \cap Q^i = \emptyset$, the "half-way" synthesis result for G is

$$\mathrm{synth}_h(G) = \langle (Q \setminus Q^x) \cup \{\bot\}, \Sigma, \rightarrow_h, Q^i, Q^m \setminus Q^x \rangle \tag{20}$$

where $\bot \notin Q$, and $p \xrightarrow{\sigma}_h q$ if $p \xrightarrow{\sigma} q$ and $p, q \notin Q^x$, and $p \xrightarrow{\upsilon}_h \bot$ if $p \xrightarrow{\upsilon} Q^x$ for some $p \notin Q^x$ and $\upsilon \in \Sigma_u$. ■

The set Q^x can be shown to be well-defined. If there is a "bad" initial state, it is already known that there does not exist any controllable and nonblocking supervisor.

"Half-way" synthesis can be implemented using a fixed point algorithm similar to standard synthesis [2]. Until a fixed point is reached, all blocking states, and all states from which blocking states can be reached via τ_u are replaced by the new non-marked state $\bot$. Finally, all incoming controllable transitions and all outgoing transitions are removed from $\bot$.

Proposition 3: Let G be a Kripke-structure. Then

$$G \ \simeq_{\mathrm{sup}} \ \mathrm{synth}_h(G) \ . \qquad \square$$

Proof: (Sketch) It suffices to show for arbitrary T,

$$\mathrm{sup}\mathcal{CN}(\mathrm{synth}_h(G) \parallel T) \ = \ \mathrm{sup}\mathcal{CN}(G \parallel T). \tag{21}$$

This is true because synth_h only removes controllable transitions that $\mathrm{sup}\mathcal{CN}$ would also remove, and all blocking situations remain. ■

This shows that "half-way" synthesis is a sound way of simplification for the modular synthesis procedure. Methods from process-algebraic testing theory [10] can be used to derive several other simplification procedures, but they have to be omitted here for lack of space.

V. Conclusions

A general method for modular synthesis of controllable and nonblocking supervisors for discrete event systems has been proposed. The monolithic representation of the state space is avoided by the use of simplified automata at the intermediate stages of the algorithm. The supervisor is produced in an efficient representation using a symbolic mapping of state labels and therefore remains modular.

The proposed framework can be extended and enhanced in different ways. In the future, the authors would like to study and evaluate additional algorithms for the minimisation of automata in a way that preserves supervision equivalence. Furthermore, it is interesting to consider coarser equivalences than supervision equivalence that take some aspects of the rest of the system considered into account.

References

[1] W. M. Wonham, "Notes on control of discrete event systems," Department of Electrical and Computer Engineering, University of Toronto, Tech. Rep., 1999.

[2] P. J. G. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proc. IEEE*, vol. 77, no. 1, pp. 81–98, Jan. 1989.

[3] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Kluwer, Sept. 1999.

[4] R. J. Leduc, B. A. Brandin, and W. M. Wonham, "Hierarchical interface-based non-blocking verification," in *Proc. Canadian Conf. Electrical and Computer Engineering*, May 2000, pp. 1–6.

[5] K. C. Wong and W. M. Wonham, "Modular control and coordination of discrete-event systems," *Discrete Event Dynamic Systems: Theory and Applications*, vol. 8, no. 3, pp. 247–297, Oct. 1998.

[6] B. A. Brandin, R. Malik, and P. Malik, "Incremental verification and synthesis of discrete-event systems guided by counter-examples," *IEEE Trans. Contr. Syst. Technol.*, vol. 12, no. 3, pp. 387–401, May 2004.

[7] K. Åkesson, H. Flordal, and M. Fabian, "Exploiting modularity for synthesis and verification of supervisors," in *Proc. of the 15th Triennial World Congress of the International Federation of Automatic Control*, Barcelona, Spain, July 2002.

[8] F. Lin and W. M. Wonham, "Decentralized control and coordination of discrete-event systems with partial observation," *IEEE Trans. Automat. Contr.*, vol. 35, no. 12, pp. 1330–1337, Dec. 1990.

[9] J. E. R. Cury, *Modular Supervisory Control of Large Scale Discrete Event Systems*. Springer, 2000, pp. 103–110.

[10] R. D. Nicola and M. C. B. Hennessy, "Testing equivalences for processes," *Theoretical Comput. Sci.*, vol. 34, no. 1–2, pp. 83–133, 1984.

[11] C. A. R. Hoare, *Communicating Sequential Processes*. Prentice-Hall, 1985.

[12] R. Malik, D. Streader, and S. Reeves, "Fair testing revisited: A process-algebraic characterisation of conflicts," in *Proc. 2nd Int. Symp. Automated Technology for Verification and Analysis, ATVA 2004*, ser. LNCS, F. Wang, Ed., vol. 3299. Taipei, Taiwan: Springer, 2004, pp. 120–134.