Yet Another Approach to Compositional Synthesis of Discrete Event Systems

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

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

***Abstract*— A two-pass algorithm for compositional synthesis of modular supervisors for large-scale systems of composed finite-state automata is proposed. The first pass provides an efficient method to determine whether a supervisory control problem has a solution, without explicitly constructing the synchronous composition of all components. If a solution exists, the second pass yields an *over-approximation* of the least restrictive solution which, if nonblocking, is a modular representation of the least restrictive supervisor. Using a new type of equivalence of nondeterministic processes, called *synthesis equivalence*, a wide range of abstractions can be employed to mitigate state-space explosion throughout the algorithm.**

I. Introduction

Modular approaches to supervisor synthesis are of great interest in *supervisory control theory* [1], [2], firstly in order to find more comprehensible supervisor representations, and secondly to overcome the problem of *state-space explosion* for systems with a large number of components.

Most approaches studied so far rely on structure to be provided by users [3], [4] and hence are hard to automate. Those that can be automated do not consider both nonblocking and least restrictiveness [5]–[9]. *Supervisor reduction* [10] has been used successfully to reduce the size of synthesised supervisors, but it relies on a monolithic supervisor to be constructed first, and thus remains limited by its size.

A different approach is proposed in [11], where *language projection* is used to simplify finite-state machines during synthesis and to construct modular supervisors. To ensure that nonblocking and maximal permissiveness are preserved, the *observer property* and *output-control consistency* are imposed on the projection.

In [12], the authors present another framework for compositional synthesis, using abstractions based on a process equivalence called *supervision equivalence*. Using nondeterministic automata, the method supports a wide range of simplifications and can hide both controllable and uncontrollable events, while still ensuring a least restrictive result. Yet, there is room for improvement. Due to its reliance on *state labels*, supervision equivalence is not preserved under bisimulation [13], which suggests that this is not the best possible equivalence for reasoning about synthesis. Furthermore, the procedure described in [12] produces an efficient representation of a *monolithic* supervisor, making further analysis of the supervisor troublesome.

This paper introduces another equivalence relation on automata, called *synthesis equivalence*, that does not suffer from these drawbacks. Synthesis equivalence is coarser than both bisimulation equivalence and supervision equivalence, and the compositional synthesis procedure proposed in this paper produces a *modular* supervisor.

Section II introduces notation from supervisory control theory and defines the synthesis procedure for nondeterministic automata used. Then, Section III defines synthesis equivalence and presents the main results that lead to the compositional synthesis procedure. Afterwards, Section IV demonstrates the procedure by applying it to a medium-scale example, and Section V finishes with some concluding remarks. A more detailed version of this paper including all the proofs can be found in [14].

II. Preliminaries

A. Events and Languages

Event sequences and languages are a simple means to describe discrete system behaviours. Their basic building blocks are *events*, taken from a finite *alphabet* Σ. For the purpose of supervisory control, the 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, or Σ_u. If they are to be included, the alphabets $\Sigma_\tau = \Sigma \cup \{\tau_c, \tau_u\}$, $\Sigma_{\tau,c} = \Sigma_c \cup \{\tau_c\}$, and $\Sigma_{\tau,u} = \Sigma_u \cup \{\tau_u\}$ are used instead [12].

Σ^* denotes the set of all finite *strings* of the form $\sigma_1\sigma_2\ldots\sigma_k$ of events from Σ, including the *empty string* ε. The *concatenation* of two strings $s, t \in \Sigma^*$ is written as st. A subset $\mathcal{L} \subseteq \Sigma^*$ is called a *language*.

B. Nondeterministic Automata

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

Definition 1: A (nondeterministic) *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 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 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 $x \xrightarrow{\sigma} y$, and extended to strings in Σ_τ^* in the standard way.

For state sets $X, Y \subseteq Q$, $X \xrightarrow{s} Y$ denotes the existence of $x \in X$ and $y \in Y$ such that $x \xrightarrow{s} y$. Similarly, $x \to y$ means that there exists a string $s \in \Sigma_\tau^*$ such that $x \xrightarrow{s} y$, and $x \xrightarrow{s}$ means that there exists a state $y \in Q$ such that $x \xrightarrow{s} y$. For an automaton G, $G \xrightarrow{s} x$ means $Q^i \xrightarrow{s} x$. Given this notation, the *marked language* of an automaton is

$$\mathcal{M}(G) \;=\; \{\, s \in \Sigma^* \mid G \xrightarrow{s} Q^m \,\} \;. \tag{1}$$

Definition 2: An automaton G is *deterministic* if Q^i is a singleton, $x \xrightarrow{\sigma} y_1$ and $x \xrightarrow{\sigma} y_2$ always implies $y_1 = y_2$, and $\to$ contains no transitions labelled τ_c or τ_u.

Various operations are used to modify or combine automata. For compositional synthesis, synchronous composition [2], [15] and hiding are the most important.

Definition 3: Let $G_1 = \langle Q_1, \Sigma_1, \to_1, Q_1^i, Q_1^m\rangle$ and $G_2 = \langle Q_2, \Sigma_2, \to_2, Q_2^i, Q_2^m\rangle$ be two automata. The *synchronous product* of G_1 and G_2 is

$$G_1 \| G_2 \;=\; \langle Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \to, Q_1^i \times Q_2^i, Q_1^m \times Q_2^m\rangle \tag{2}$$

where

$$\begin{aligned}
(x,y) &\xrightarrow{\sigma} (x',y') \text{ if } \sigma \in \Sigma_1 \cap \Sigma_2,\; x \xrightarrow{\sigma}_1 x',\; \text{and}\; y \xrightarrow{\sigma}_2 y' ;\\
(x,y) &\xrightarrow{\sigma} (x',y) \text{ if } \sigma \in (\Sigma_1 \setminus \Sigma_2) \cup \{\tau_c, \tau_u\} \text{ and } x \xrightarrow{\sigma}_1 x' ;\\
(x,y) &\xrightarrow{\sigma} (x,y') \text{ if } \sigma \in (\Sigma_2 \setminus \Sigma_1) \cup \{\tau_c, \tau_u\} \text{ and } y \xrightarrow{\sigma}_2 y' .
\end{aligned}$$

Definition 4: Let $G = \langle Q, \Sigma, \to, Q^i, Q^m\rangle$ be an automaton, and let $\Upsilon \subseteq \Sigma$. The result of *controllability preserving hiding* [12], *hiding* henceforth, of Υ from G is

$$G \setminus_! \Upsilon \;=\; \langle Q, \Sigma \setminus \Upsilon, \to_!, Q^i, Q^m\rangle \tag{3}$$

where $\to_!$ is obtained from $\to$ 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 $\sigma \in \Sigma_u$.

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

Definition 5: Let $G_1 = \langle Q_1, \Sigma, \to_1, Q_1^i, Q_1^m\rangle$ and $G_2 = \langle Q_2, \Sigma, \to_2, Q_2^i, Q_2^m\rangle$ be two automata with the same alphabet. G_1 is a *subautomaton* of G_2, written $G_1 \subseteq G_2$, if $Q_1 \subseteq Q_2$, $\to_1 \subseteq \to_2$, $Q_1^i \subseteq Q_2^i$, and $Q_1^m \subseteq Q_2^m$.

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

$$\bigcup_{j\in J} G_j \;=\; \langle \bigcup_{j\in J} Q_j, \Sigma, \bigcup_{j\in J} \to_j, \bigcup_{j\in J} Q_j^i, \bigcup_{j\in J} Q_j^m\rangle \;. \tag{4}$$

C. Synthesis

In this paper, synthesis is applied to a single nondeterministic automaton, considered as a *plant*. Section II-D below shows how traditional control problems involving *specifications* [1] can be treated in this formalism. In a "plant-only" control problem, the objective is to find a subautomaton of a given plant automaton G that is both controllable and nonblocking according to the following definitions.

Definition 7: Let $G = \langle Q_G, \Sigma, \to_G, Q_G^i, Q_G^m\rangle$ and $K = \langle Q_K, \Sigma, \to_K, Q_K^i, Q_K^m\rangle$ be automata such that $K \subseteq G$. K is *controllable* in G if, for all states $x \in Q_K$ and $y \in Q_G$ and for every uncontrollable event $\upsilon \in \Sigma_{\tau,u}$ such that $x \xrightarrow{\upsilon}_G y$, it also holds that $x \xrightarrow{\upsilon}_K y$.

Definition 8: Let $G = \langle Q, \Sigma, \to, Q^i, Q^m\rangle$. A state $x \in Q$ is called *reachable* in G if $G \to x$, and *coreachable* in G if $x \to Q^m$. The automaton G is called reachable or coreachable if every state in G has this property. G is called *nonblocking* if every reachable state is coreachable.

Such definitions also appear in [12] and extend the standard definitions [1] to the nondeterministic case considered here. The synthesis computation is done by iteratively calculating state sets $X \subseteq Q$ and *restricting* the automaton to these states.

Definition 9: Let $G = \langle Q, \Sigma, \to, Q^i, Q^m\rangle$. The *restriction* of G to $X \subseteq Q$ is $G_{|X} = \langle X, \Sigma, \to_{|X}, Q^i \cap X, Q^m \cap X\rangle$ where $\to_{|X} = \{\, (x,\sigma,y) \mid x, y \in X \,\}$.

Definition 10: Let $G = \langle Q, \Sigma, \to, Q^i, Q^m\rangle$. The *synthesis step operator* $\Theta_G \colon 2^Q \to 2^Q$ is defined by

$$\begin{aligned}
\Theta_G(X) \;=\; \{\, x \in X \mid \text{For all } u \in \Sigma_{\tau,u}^* \text{ and all } y \in Q \\
\text{such that } x \xrightarrow{u} y \text{ it holds that } y \to_{|X} Q^m \,\} \;.
\end{aligned}$$

$\Theta_G(X)$ contains all states $x \in X$ such that all states reachable from x by uncontrollable transitions are coreachable within X. This operator captures both controllability and nonblocking, and allows for a more succinct description of the synthesis procedure than previously in [12].

The synthesis step operator is monotonic and has a greatest fixpoint, which turns out to be the least restrictive controllable and nonblocking subautomaton of a given automaton G. It follows that the greatest fixpoint of the synthesis step operator exists and characterises an optimal synthesis result.

Theorem 1: Let $G = \langle Q, \Sigma, \to, Q^i, Q^m\rangle$. The synthesis step operator Θ_G has a greatest fixpoint $\hat{X}_G \subseteq Q$, such that $G_{|\hat{X}_G}$ is the greatest subautomaton of G that is both controllable in G and coreachable. If the state set Q is finite, the sequence $X^0 = Q$, $X^{i+1} = \Theta_G(X^i)$ reaches this fixpoint in a finite number of steps, i.e., $\hat{X}_G = X^n$ for some $n \in \mathbb{N}$.

Proof: See [14]. ∎

Accordingly, the *synthesis result* for an automaton G,

$$\sup\mathcal{CN}(G) \;=\; G_{|\hat{X}_G} \;, \tag{5}$$

is obtained by restricting G to the fixpoint $\hat{X}_G$ (unreachable states can be removed). If $\hat{X}_G$ contains no initial states, there is no feasible solution to the synthesis problem, otherwise $\sup\mathcal{CN}(G)$ is the least restrictive solution. Supervisory control theory focuses on the language of this solution,

$$\mathcal{M}^\uparrow(G) \;=\; \mathcal{M}(\sup\mathcal{CN}(G)) \;. \tag{6}$$

In slight abuse of notation, the above $\mathcal{M}^\uparrow(G)$ denotes both the language accepted by the least restrictive synthesis result as well as its minimal deterministic recogniser.

If G is deterministic, then $\sup\mathcal{CN}(G)$ is also deterministic and can be used to implement a *supervisor* that achieves the behaviour $\mathcal{M}^\uparrow(G)$. In this paper, any nondeterministic automaton is an *abstraction* of an originally deterministic model built using transformations ensuring that a meaningful supervisor can also be constructed.

D. Translation of Specifications into Plants

A traditional supervisory control problem [1] consists of a *plant* G and a *specification* K, given as deterministic automata. In this context, the following controllability requirement is used instead of Def. 7.

Definition 11: 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 x of K, and every uncontrollable event $\upsilon \in \Sigma_u$ such that $K \xrightarrow{s} x$ and $G \xrightarrow{s\upsilon}$, it holds that $x \xrightarrow{\upsilon}$ in K.

Using the nonblocking condition, such control problems can be represented *equivalently* only using plants. A specification automaton is 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 following construction from [12] essentially transforms all potential controllability problems into potential blocking problems, eliminating the need for explicitly checking controllability.

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

$$K^\perp \;=\; \langle Q \cup \{\perp\}, \Sigma, \rightarrow^\perp, Q^i, Q^m \rangle \tag{7}$$

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

$$\rightarrow^\perp \;=\; \rightarrow \cup \left\{ (x, \upsilon, \perp) \mid x \in Q, \upsilon \in \Sigma_u, x \overset{\upsilon}{\not\rightarrow} \right\} . \tag{8}$$

Proposition 1: Let G, K, and K' be deterministic automata over the same alphabet Σ, and let K' be reachable. Then $K' \subseteq G \parallel K^\perp$ is nonblocking and controllable in $G \parallel K^\perp$ if and only if $K' \subseteq G \parallel K$ is nonblocking and controllable with respect to G.

Proof: See [12] or [14]. ■

According to this result, synthesis of the least restrictive nonblocking and controllable behaviour allowed by a specification K with respect to a plant G—both deterministic—can be achieved by computing $\sup\mathcal{CN}(G \parallel K^\perp)$.

III. Compositional Synthesis

This section outlines the proposed compositional synthesis procedure and presents the underlying theoretical results. As discussed in Section II-D, the synthesis problem can be reduced to the task of finding the supremal nonblocking and controllable supervisor for a deterministic plant

$$G \;=\; G_1 \parallel \cdots \parallel G_n \; . \tag{9}$$

The synthesis calculation presented here is a two-pass procedure. The first pass is a compositional minimisation where the automata in (9) are simplified and composed step-by-step; all intermediate results are stored. The result of this pass is an automaton representing a highly abstract description of the monolithic behaviour of the supervised system. In the second pass, this abstract behaviour, in the form of a marked language, is passed backwards, and used to find a supervisor component to control the part of the behaviour that was abstracted at each step of the first pass.

In the *first pass*, the modular plant (9) is simplified step-by-step using a similar strategy as proposed in [12], [13], [16]. At each step, a subsystem of (9) is chosen and modified in one of the following three ways.

1) A component G_i can be *simplified* and replaced by an equivalent component G_i', provided that the new component is *synthesis equivalent* to the original component G_i according to the definition given below.
2) A component can be modified by *hiding local events*. If $\Upsilon_i \subseteq \Sigma$ is a set of events that appear only in G_i, then G_i can be replaced by $G_i \setminus\!! \Upsilon_i$.
3) Two or more components can be *composed* and replaced by their synchronous product.

Simplification and hiding are typically performed together, since it usually is the removal of local events that makes more simplification possible. Composition typically is only used as a last resort, when no hiding and simplification is possible. For simplification to work correctly, it must be guaranteed that synthesis results are not changed despite the simplification. The condition imposed for this purpose is *synthesis equivalence*.

Definition 13: Two automata G_1 and G_2 are *synthesis equivalent*, denoted $G_1 \simeq_{\text{synth}} G_2$ if, for all automata T,

$$\mathcal{M}^\uparrow(G_1 \parallel T) \;=\; \mathcal{M}^\uparrow(G_2 \parallel T) \; . \tag{10}$$

Two automata are synthesis equivalent if their synthesised languages are the same in all possible environments T. To justify that simplification and composition steps can be performed in arbitrary order, the equivalence must be a *congruence* with respect to synchronous composition. This is shown easily:

Proposition 2: Let G_1, G_2, and H be arbitrary automata. If $G_1 \simeq_{\text{synth}} G_2$, then $G_1 \parallel H \simeq_{\text{synth}} G_2 \parallel H$.

Proof: Let T be an automaton. Since $G_1 \simeq_{\text{synth}} G_2$ it follows that $\mathcal{M}^\uparrow((G_1 \parallel H) \parallel T) = \mathcal{M}^\uparrow(G_1 \parallel (H \parallel T)) = \mathcal{M}^\uparrow(G_2 \parallel (H \parallel T)) = \mathcal{M}^\uparrow((G_2 \parallel H) \parallel T)$, i.e., $G_1 \parallel H \simeq_{\text{synth}} G_2 \parallel H$. ■

A set of rules for calculating abstractions preserving synthesis equivalence can be constructed in a similar way as in [12]. Bisimulation [17] preserves synthesis equivalence, and most of the simplification rules given in [12] for supervision equivalence also apply to synthesis equivalence and are used in the example in Section IV below, without proof.

In the end of the first pass, all automata are composed, producing a single automaton with only local events. After hiding the last events, only two final results are possible: either the empty automaton is returned, indicating that the original synthesis problem (9) has no solution, or a one-state automaton accepting the language $\{\varepsilon\}$ is returned. This final abstraction is only used to determine whether a solution exists—it is too abstract to produce a useful supervisor.

A supervisor is calculated in the *second pass*, during which the final result is passed back through all steps of the first pass. At each step, a modular supervisor component is obtained using the following result.

Theorem 2: Let $G = \langle Q_G, \Sigma_G, \rightarrow_G, Q_G^i, Q_G^m \rangle$ be an automaton, and $T = \langle Q_T, \Sigma_T, \rightarrow_T, Q_T^i, Q_T^m \rangle$ be a deterministic automaton. Let $\Sigma_G \cap \Sigma_T \subseteq \Omega \subseteq \Sigma_G \cup \Sigma_T$, and

write $\Upsilon_G = \Sigma_G \setminus \Omega$ and $\Upsilon_T = \Sigma_T \setminus \Omega$. Furthermore let G' and T' be automata such that

$$G' \quad \simeq_{\text{synth}} \quad G \setminus_! \Upsilon_G \ ; \tag{11}$$

$$T' \quad \simeq_{\text{synth}} \quad \mathcal{M}^{\uparrow}(G' \parallel T \setminus_! \Upsilon_T) \ . \tag{12}$$

Then

$$\mathcal{M}^{\uparrow}(G \parallel T) \quad \subseteq \quad \mathcal{M}^{\uparrow}(G' \parallel T) \parallel \mathcal{M}^{\uparrow}(G \parallel T') \ . \tag{13}$$

Proof: See [14]. ■

This result is used as follows. Assume component G_1 in (9) has been replaced by $G_1' \simeq_{\text{synth}} G_1 \setminus_! \Upsilon_1$, and a supervisor has been obtained for the abstracted system $G_1' \parallel T$ where $T = G_2 \parallel \cdots \parallel G_n$. This supervisor can be simplified after hiding events local to T, yielding $T' \simeq_{\text{synth}} \mathcal{M}^{\uparrow}(G_1' \parallel T \setminus_! \Upsilon_T)$, and used together with G_1 to compute a new supervisor component $\mathcal{M}^{\uparrow}(G_1 \parallel T')$.

Theorem 2 does not guarantee equality of languages. In general, the behaviour achieved by the modular supervisors is an over-approximation of the monolithic synthesis result, and an additional nonblocking check is needed to ensure equality. Using methods of [16], this check can be done without explicitly constructing the synchronous product, and if it fails, weaker abstractions can be attempted.

It is also necessary in Theorem 2 that the automaton T, representing the remainder of the system, is deterministic. Initially, this is not a problem, since the input (9) for the synthesis procedure is assumed to consist of deterministic automata. To iterate the method, it is advisable to allow only deterministic abstractions while simplifying. Yet G, unlike T, may be nondeterministic in Theorem 2, so nondeterministic abstractions can be part of the subsystem G, i.e., the system considered for further simplification.

IV. Example

In this section, the proposed synthesis procedure is applied to a part of the "Flexible Manufacturing System" (FMS) [18]. The model consists of a robot R, a conveyor C, a painting device PD, an assembly machine AM, and two buffers B_7 and B_8. Workpieces move from the robot R through B_7, C, and B_8 to the painting device PD, and back through B_8, C, and B_7 to the assembly machine AM. Fig. 1 shows the "plants-only" version of the synthesis problem. Two specifications in the original example have been transformed into plants $B_7^{\perp}$ and $B_8^{\perp}$ according to Proposition 1. In the figures, uncontrollable events are prefixed by exclamation marks, !, and local events have parentheses, (), around them.

Note that all states except $\perp$ are marked in the buffer plants $B_7^{\perp}$ and $B_8^{\perp}$. This permits deadlock in the system with a workpiece in B_7 (en route to PD) and another workpiece in B_8 (en route to AM). To eliminate this fault, only states $\mathbf{b_e}$ should be marked, but the model in Fig. 1 poses a more challenging synthesis problem.

A. First Pass

First of all, events s_r, s_a, s_1, f_1, and f_2 in Fig. 1 are local, which may enable some simplifications. These events occur in R, which cannot be simplified, and in AM, which

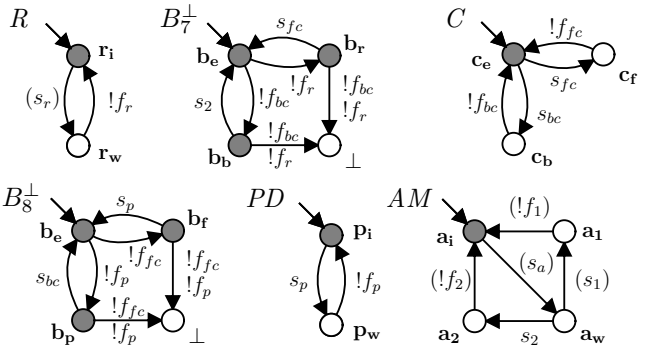

Fig. 1. The automata in the FMS example.

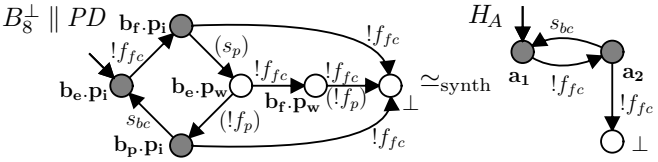

Fig. 2. The composition $B_8^{\perp} \parallel PD$ and its simplification $H_A \simeq_{\text{synth}} (B_8^{\perp} \parallel PD) \setminus_! \{s_p, f_p\}$.

can be simplified significantly. The only event by which AM interacts with other components is s_2. Since s_2 is controllable and AM can always silently reach both a state where s_2 can occur and a marked state, AM can be reduced to an automaton with a single marked state and a selfloop on s_2. This makes event s_2 entirely superfluous—in the perspective of $B_7^{\perp}$, AM acts just like an infinite output buffer. In other words, based on the fact that

$$AM \setminus_! \{s_1, s_a, f_1, f_2\} \simeq_{\text{synth}} \quad , \tag{14}$$

AM can be dropped. This, in turn, means that s_2 is now a local event in $B_7^{\perp}$, but no simplification can be made there.

At this point, no more simplification can be made, so some automata need to be composed. A reasonable starting point is to compose $B_8^{\perp}$ and PD. This makes events s_p and f_p local. The result of this composition is shown to the left in Fig. 2; to the right is the simplification H_A.

Next, R and $B_7^{\perp}$ are composed, causing f_r to become local. The result of this composition is shown in Fig. 3 along with a simplification H_B. Fig. 4 shows the composition of H_B and C, making s_{fc} and f_{bc} local, and a simplification H_C of the result. Finally, H_A and H_C are composed and

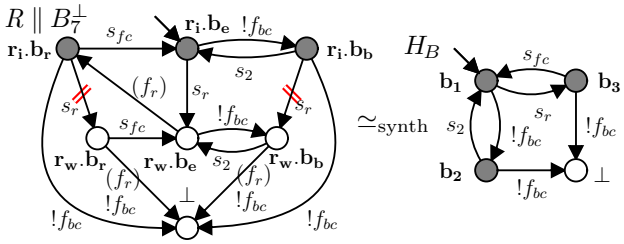

Fig. 3. The composition $R \parallel B_7^{\perp}$ and its simplification $H_B \simeq_{\text{synth}} (R \parallel B_7^{\perp}) \setminus_! \{f_r\}$. Two transitions must be disabled by synthesis and are crossed out in the figure.

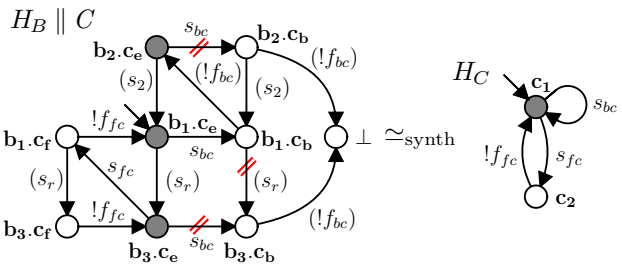

Fig. 4. The composition $H_B \parallel C$ and its simplification $H_C \simeq_{\text{synth}} (H_B \parallel C) \setminus_! \{f_{bc}, s_2, s_r\}$.

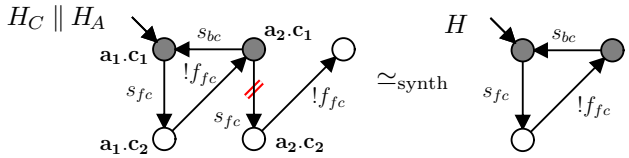

Fig. 5. The composition $H_C \parallel H_A$ and its supervisor $H = \mathcal{M}^{\uparrow}(H_C \parallel H_A)$.

simplified, see Fig. 5. At this point, all events are local and can be hidden. This results in a nonempty language, showing that a supervisor exists.

In summary, the system in Fig. 1 is simplified in the following steps. At each step, the automata in brackets () are composed and simplified, possibly after hiding.

1) $R \parallel B_7^{\perp} \parallel C \parallel B_8^{\perp} \parallel PD \parallel (AM)$;
2) $R \parallel B_7^{\perp} \parallel C \parallel (B_8^{\perp} \parallel PD)$;
3) $(R \parallel B_7^{\perp}) \parallel C \parallel H_A$;
4) $(H_B \parallel C) \parallel H_A$;
5) $(H_C \parallel H_A)$;
6) H.

B. Second Pass

In the second pass, Theorem 2 is applied to each step of the first pass, potentially producing a supervisor component for each simplification step. The starting point is the final result H of all the simplification steps, shown in Fig. 5, which can be considered as the first supervisor component. In this case, it achieves least restrictive nonblocking supervision of the last composition, since

$$H \;=\; \mathcal{M}^{\uparrow}(H_C \parallel H_A)\;. \tag{15}$$

To find a supervisor component for the previous step 4), where $H_B \parallel C$ is simplified, events not in $H_B \parallel C$ can be hidden from H. However, all events in H are shared and no simplification is possible. Using $H_C \simeq_{\text{synth}} (H_B \parallel C) \setminus_! \{f_{bc}, s_2, s_r\}$ and (15) in Theorem 2, it follows that

$$\begin{aligned} &\mathcal{M}^{\uparrow}((H_B \parallel C) \parallel H_A) \\ &\subseteq \mathcal{M}^{\uparrow}(H_C \parallel H_A) \parallel \mathcal{M}^{\uparrow}(H_B \parallel C \parallel H) \\ &= H \parallel \mathcal{M}^{\uparrow}(H_B \parallel C \parallel H)\;. \end{aligned} \tag{16}$$

The supervisor computed at this stage

$$S_1 \;=\; \mathcal{M}^{\uparrow}(H_B \parallel C \parallel H) \tag{17}$$

is shown in Fig. 6. Since no events have been hidden, it holds that $H \parallel S_1 = S_1$, and the new supervisor S_1 includes

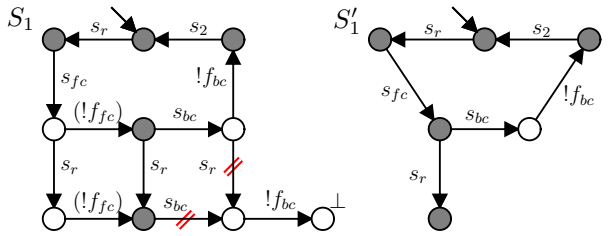

Fig. 6. The supervisor $S_1 = \mathcal{M}^{\uparrow}(H_B \parallel C \parallel H)$ and its abstraction $S_1' \simeq_{\text{synth}} S_1 \setminus_! \{f_{fc}\}$.

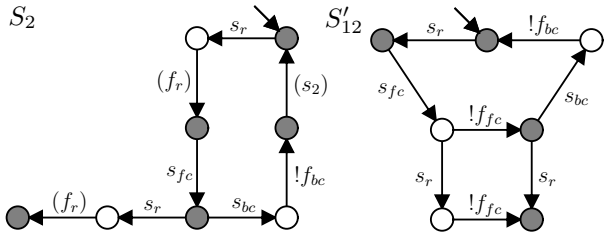

Fig. 7. The supervisor $S_2 = \mathcal{M}^{\uparrow}(R \parallel B_7^{\perp} \parallel S_1')$ and the abstraction $S_{12}' \simeq_{\text{synth}} (S_1 \parallel S_2) \setminus_! \{s_2, f_r\}$.

the previous supervisor H. Thus, H can be dropped. A nonblocking check reveals that equality holds in (16), i.e.,

$$\mathcal{M}^{\uparrow}(H_B \parallel C \parallel H_A) = H \parallel \mathcal{M}^{\uparrow}(H_B \parallel C \parallel H) = H \parallel S_1 = S_1\;.$$

The supervisor S_1 is passed back to the previous simplification step 3), where $R \parallel B_7^{\perp}$ is simplified. Using the fact that event f_{fc} is not used in $R \parallel B_7^{\perp}$, it is possible to simplify S_1 preserving synthesis equivalence to

$$S_1' \simeq_{\text{synth}} S_1 \setminus_! \{f_{fc}\}\;. \tag{18}$$

This automaton is also shown in Fig. 6. Using $H_B \simeq_{\text{synth}} (R \parallel B_7^{\perp}) \setminus_! \{f_r\}$ and $S_1' \simeq_{\text{synth}} S_1 \setminus_! \{f_{fc}\} = \mathcal{M}^{\uparrow}(H_B \parallel (C \parallel H_A) \setminus_! \{f_{fc}\})$ in Theorem 2, it follows that

$$\begin{aligned} &\mathcal{M}^{\uparrow}((R \parallel B_7^{\perp}) \parallel (C \parallel H_A)) \\ &\subseteq \mathcal{M}^{\uparrow}(H_B \parallel C \parallel H_A) \parallel \mathcal{M}^{\uparrow}(R \parallel B_7^{\perp} \parallel S_1') \\ &= S_1 \parallel \mathcal{M}^{\uparrow}(R \parallel B_7^{\perp} \parallel S_1') \end{aligned} \tag{19}$$

The new supervisor component

$$S_2 \;=\; \mathcal{M}^{\uparrow}(R \parallel B_7^{\perp} \parallel S_1') \tag{20}$$

is shown in Fig. 7. So far, two modular supervisors have been computed, S_1 and S_2, and their composed behaviour needs to be considered for the back-processing of the remaining simplification steps. Since (19) also is nonblocking,

$$\mathcal{M}^{\uparrow}(R \parallel B_7^{\perp} \parallel C \parallel H_A) = S_1 \parallel \mathcal{M}^{\uparrow}(R \parallel B_7^{\perp} \parallel S_1') = S_1 \parallel S_2\;.$$

In the preceding step 2), the composition $B_8^{\perp} \parallel PD$ has been simplified. This automaton does not use the supervisor's events s_2 and f_r, so a simplified automaton S_{12}', shown in Fig. 7, can be used in this step. Using $H_A \simeq_{\text{synth}} (B_8^{\perp} \parallel PD) \setminus_! \{s_p, f_p\}$ and

$$\begin{aligned} S_{12}' &\simeq_{\text{synth}} (S_1 \parallel S_2) \setminus_! \{s_2, f_r\} \\ &= \mathcal{M}^{\uparrow}(R \parallel B_7^{\perp} \parallel C \parallel H_A) \setminus_! \{s_2, f_r\} \\ &= \mathcal{M}^{\uparrow}(H_A \parallel (R \parallel B_7^{\perp} \parallel C) \setminus_! \{s_2, f_r\}) \end{aligned} \tag{21}$$

in Theorem 2, it follows that

$$\begin{aligned}&\mathcal{M}^{\uparrow}((B_8^{\perp} \parallel PD) \parallel (R \parallel B_7^{\perp} \parallel C))\\ &\subseteq \mathcal{M}^{\uparrow}(H_A \parallel R \parallel B_7^{\perp} \parallel C) \parallel \mathcal{M}^{\uparrow}(B_8^{\perp} \parallel PD \parallel S'_{12})\\ &= S_1 \parallel S_2 \parallel \mathcal{M}^{\uparrow}(B_8^{\perp} \parallel PD \parallel S'_{12}) \ . \end{aligned} \tag{22}$$

It turns out that $\mathcal{M}^{\uparrow}(B_8^{\perp} \parallel PD \parallel S'_{12}) = B_8^{\perp} \parallel PD \parallel S'_{12}$ (11 states) and $S_1 \parallel S_2 \parallel S'_{12} = S_1 \parallel S_2$, i.e., no additional supervision is needed in this step. A nonblocking check of (22) ensures equality, and thus

$$\begin{aligned}&\mathcal{M}^{\uparrow}(B_8^{\perp} \parallel PD \parallel R \parallel B_7^{\perp} \parallel C)\\ &= S_1 \parallel S_2 \parallel \mathcal{M}^{\uparrow}(B_8^{\perp} \parallel PD \parallel S'_{12})\\ &= S_1 \parallel S_2 \parallel B_8^{\perp} \parallel PD \parallel S'_{12}\\ &= S_1 \parallel S_2 \parallel B_8^{\perp} \parallel PD \ . \end{aligned} \tag{23}$$

In the final step to be back-processed, 1), AM has been simplified according to (14). All events except s_2 are local and can be hidden from the supervisor $S_1 \| S_2 \| B_8^{\perp} \| PD$, producing a three-state abstraction S'. Using (14) and $S' \simeq_{\text{synth}} (S_1 \parallel S_2 \parallel B_8^{\perp} \parallel PD) \setminus_! \Upsilon = \mathcal{M}^{\uparrow}(B_8^{\perp} \parallel PD \parallel R \parallel B_7^{\perp} \parallel C) \setminus_! \Upsilon$, where $\Upsilon = \Sigma \setminus \{s_2\}$, in Theorem 2, it follows that

$$\begin{aligned}&\mathcal{M}^{\uparrow}(AM \parallel (B_8^{\perp} \parallel PD \parallel R \parallel B_7^{\perp} \parallel C))\\ &\subseteq \mathcal{M}^{\uparrow}(B_8^{\perp} \parallel PD \parallel R \parallel B_7^{\perp} \parallel C) \parallel \mathcal{M}^{\uparrow}(AM \parallel S')\\ &= S_1 \parallel S_2 \parallel B_8^{\perp} \parallel PD \parallel \mathcal{M}^{\uparrow}(AM \parallel S') \ . \end{aligned} \tag{24}$$

Again, it turns out that no additional supervision is needed because $\mathcal{M}^{\uparrow}(AM \parallel S') = AM \parallel S'$ (12 states) and $S_1 \parallel S_2 \parallel S' = S_1 \parallel S_2$, and the system is nonblocking. Thus,

$$\begin{aligned}&\mathcal{M}^{\uparrow}(AM \parallel B_8^{\perp} \parallel PD \parallel R \parallel B_7^{\perp} \parallel C)\\ &= S_1 \parallel S_2 \parallel B_8^{\perp} \parallel PD \parallel \mathcal{M}^{\uparrow}(AM \parallel S')\\ &= S_1 \parallel S_2 \parallel B_8^{\perp} \parallel PD \parallel AM \parallel S'\\ &= S_1 \parallel S_2 \parallel B_8^{\perp} \parallel PD \parallel AM \ . \end{aligned} \tag{25}$$

Therefore, adding the modular supervisor components S_1 and S_2 to the FMS system produces the least restrictive nonblocking behaviour. This result has been obtained without ever considering automata larger than twelve states, although there are 184 reachable states in the synchronous product of the six automata in Fig. 1.

V. Conclusions

A two-pass procedure for compositional synthesis of modular supervisors for discrete event systems has been presented. The strength of this procedure lies in that, at each step of the second pass, the method accesses both *local* information—given by the intermediate result visited—and *global* information—given by the abstraction of the monolithic behaviour passed back. This allows for the synthesis of specialised supervisor modules for individual synthesis problems, found locally, using knowledge about the global system to ensure least restrictiveness.

While the algorithm can accurately determine whether a supervisory control problem is solvable without constructing the full synchronous product, the supervisor returned may be an over-approximation of the least restrictive solution that is not automatically nonblocking. A nonblocking check is needed to confirm correctness of the result, and if this check fails, the procedure needs to be restarted using weaker abstractions. It is yet an open question how information from the failed nonblocking check can be used to guide the search for more appropriate abstractions.

The framework of synthesis equivalence has the potential to overcome several weaknesses of previous approaches to compositional synthesis: there is no need for state labels [12], making bisimulation-based simplifications possible; there is the possibility to hide controllable and uncontrollable events; and the use of nondeterministic automata paves the way for better abstractions than projection-based methods [6], [11].

References

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

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

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

[4] R. Song and R. J. Leduc, "Symbolic synthesis and verification of hierarchical interface-based supervisory control," in *Proc. 8th Int. Workshop Discrete Event Systems, WODES '06*, Ann Arbor, MI, USA, July 2006, pp. 419–426.

[5] K. Åkesson, H. Flordal, and M. Fabian, "Exploiting modularity for synthesis and verification of supervisors," in *Proc. 15th IFAC World Congress*, Barcelona, Spain, July 2002.

[6] R. C. Hill and D. M. Tilbury, "Modular supervisory control of discrete-event systems with abstraction and incremental hierarchical construction," in *Proc. 8th Int. Workshop Discrete Event Systems, WODES '06*, Ann Arbor, MI, USA, July 2006, pp. 399–406.

[7] 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.

[8] M. H. d. Queiroz and J. E. R. Cury, "Modular supervisory control of large scale discrete event systems," in *Discrete Event Systems, Analysis and Control*, R. Boel and G. Stremersch, Eds. Kluwer, 2000, pp. 103–110.

[9] P. Malik, R. Malik, D. Streader, and S. Reeves, "Modular synthesis of discrete controllers," in *Proc. 12th IEEE Int. Conf. Engineering of Complex Computer Systems, ICECCS '07*, Auckland, New Zealand, 2007, pp. 25–34.

[10] R. Su and W. M. Wonham, "Supervisor reduction for discrete-event systems," *Discrete Event Dynamic Systems*, vol. 14, no. 1, pp. 31–53, Jan. 2004.

[11] L. Feng and W. M. Wonham, "Computationally efficient supervisor design: Abstraction and modularity," in *Proc. 8th Int. Workshop Discrete Event Systems, WODES '06*, Ann Arbor, MI, USA, July 2006, pp. 3–8.

[12] H. Flordal, R. Malik, M. Fabian, and K. Åkesson, "Compositional synthesis of maximally permissive supervisors using supervision equivalence," *Discrete Event Dynamic Systems*, vol. 17, no. 4, pp. 475–504, 2007.

[13] H. Flordal, "Compositional approaches in supervisory control—with application to automatic generation of robot interlocking policies," Ph.D. dissertation, Dept. Signals and Systems, Chalmers Univ. Technol., Göteborg, Sweden, Oct. 2006.

[14] R. Malik and H. Flordal, "Compositional synthesis of discrete event systems via synthesis equivalence," Working Paper 05/2008, Dept. Computer Science, Univ. Waikato, Hamilton, New Zealand, 2008.

[15] C. A. R. Hoare, *Communicating sequential processes*, ser. Series in Computer Science. Prentice-Hall, 1985.

[16] H. Flordal and R. Malik, "Modular nonblocking verification using conflict equivalence," in *Proc. 8th Int. Workshop Discrete Event Systems, WODES '06*, Ann Arbor, MI, USA, July 2006, pp. 100–106.

[17] R. Milner, *Communication and concurrency*, ser. Series in Computer Science. Prentice-Hall, 1989.

[18] M. H. d. Queiroz, J. E. R. Cury, and W. M. Wonham, "Multitasking supervisory control of discrete-event systems," *Discrete Event Dynamic Systems*, vol. 15, no. 4, pp. 375–395, 2005.