A Framework for Compositional Synthesis of Modular Nonblocking Supervisors

Sahar Mohajerani Robi Malik Martin Fabian

***Abstract*—This paper describes a framework for *compositional supervisor synthesis*, which is applicable to all discrete event systems modelled as a set of deterministic automata. Compositional synthesis exploits the modular structure of the input model, and therefore works best for models consisting of a large number of small automata. The state-space explosion is mitigated by the use of abstraction to simplify individual components, and the property of *synthesis equivalence* guarantees that the final synthesis result is the same as it would have been for the non-abstracted model. The paper describes synthesis equivalent abstractions and shows their use in an algorithm to efficiently compute supervisors. The algorithm has been implemented in the DES software tool Supremica and successfully computes nonblocking modular supervisors, even for systems with more than 10^{14} reachable states, in less than 30 seconds.**

***Index Terms*—Finite-state automata, abstraction, synthesis, supervisory control theory.**

I. Introduction

The *supervisory control theory* [1], [2] provides a general framework for the synthesis of reactive control functions. Given a model of the system, the *plant*, to be controlled, and a *specification* of the desired behaviour, it is possible to automatically compute, i.e. *synthesise*, a *supervisor* that restricts the plant behaviour while satisfying the specification.

Commonly, a supervisor is required to be *controllable* and *nonblocking*, i.e., it should not disable uncontrollable events, and the controlled system should always be able to complete some desired task [1]. In addition, it is typically required of a supervisor to achieve some minimum functionality. Most synthesis algorithms ensure this by producing the *least restrictive* supervisor, which restricts the system as little as possible while still being controllable and nonblocking [1]. Alternatives to least restrictiveness have been investigated [3]–[5]. They require additional analysis to guarantee minimum functionality, particularly when supervisors are synthesised automatically.

It is known [1] that for a given plant and specification, a unique least restrictive, controllable, and nonblocking supervisor exists. Straightforward synthesis algorithms explore the complete *monolithic* state space of the considered system, and are therefore limited by the well-known *state-space explosion* problem. The sheer size of the supervisor also makes it humanly incomprehensible, which hinders acceptance of the synthesis approach in industrial settings.

Various approaches for *modular* and *compositional* synthesis have been proposed to overcome these problems. Some of these approaches [3], [6] rely on structure provided by users and hence are hard to automate. Other early methods [7], [8] only consider the synthesis of a least restrictive controllable supervisors, ignoring nonblocking. *Supervisor reduction* [9] and *supervisor localisation* [10] greatly help to reduce synthesised supervisors in size, yet rely on a supervisor to be constructed first and thus remain limited by its size.

Compositional methods [11] use *abstraction* to remove states and transitions that are superfluous for the purpose of synthesis. The most common abstraction method is *natural projection* which, when combined with the *observer property*, produces a nonblocking but not necessarily least restrictive supervisor [3]. If *output control consistency* is added as an additional requirement, least restrictiveness can be ensured [12]. Output control consistency can be replaced by a weaker condition called *local control consistency* [13].

Conflict-preserving abstractions [4] and *weak observation equivalence* [5] can be used as abstractions for the synthesis of nonblocking supervisors. In these works it is assumed that, when an event is abstracted, supervisor components synthesised at a later stage cannot observe or disable that event. This makes abstracted events *unobservable* and removes some possibilities of control.

Halfway synthesis [14] and *local supervisors* [5] are different strategies to avoid uncontrollable transitions to blocking states. Local supervisors [5] remove the source states of these transitions by disabling some controllable events. This may cause unnecessary disablements as it may be discovered later that some uncontrollable transitions are disabled by other plants. Halfway synthesis [14] defers the decision to remove states and retains uncontrollable transitions until it is clear that they cannot be disabled by any other component.

In [5], [14], [15], synthesis is considered in a nondeterministic setting, which leads to some problems when interpreting results and ensuring least restrictiveness. These problems are overcome to some extent by *synthesis abstraction* [16]–[19]. Several compositional synthesis methods require all automata and their abstraction results to be deterministic, which makes some desirable abstractions impossible. Following ideas from [20]–[22], *renaming* is used in [16] to avoid nondeterminism after abstraction.

This paper presents a compositional synthesis approach with abstraction methods that guarantee the preservation of the final synthesis result. A data structure called *synthesis triple* is intro-

This work was supported by the Swedish Research Council.

Sahar Mohajerani and Martin Fabian are with the Department of Signals and Systems, Chalmers University of Technology, Gothenburg, Sweden, {mohajera,fabian}@chalmers.se

Robi Malik is with the Department of Computer Science, University of Waikato, Hamilton, New Zealand; robi@waikato.ac.nz

duced to combine abstraction methods [14], [16]–[19] together with renaming. This is a general framework intended for use with a variety of present and future means of abstraction. The implementation presented in this paper uses halfway synthesis, which is adapted from [14] and observation equivalence-based abstractions [18], [19], which have higher abstraction potential than methods based on natural projection [18]. These methods allow for the abstraction of observable events in such a way that abstracted events can still be used by supervisor components synthesised at a later stage. Nondeterminism after abstraction is avoided using renaming [20]–[22] as proposed in [16].

These results are combined in a general framework for compositional synthesis, and an algorithm is proposed to compute modular supervisors that are least restrictive, controllable, and nonblocking. This is a completely automatic synthesis method, applicable to general discrete event systems, provided that they are represented as a set of deterministic finite-state automata. The algorithm has been implemented in the DES software tool Supremica [23] and applied to compute modular supervisors for several large industrial models. It successfully computes modular supervisors, even for systems with more than 10^{14} reachable states, within 30 seconds and using no more than 640 MB of memory.

In the following, Sect. II briefly introduces the background of supervisory control theory, and Sect. III gives a motivating example to informally illustrate compositional synthesis and abstraction. Next, Sect. IV explains compositional synthesis and the idea of synthesis equivalence underlying the compositional algorithm. Then, Sect. V presents different ways of computing abstractions that preserve synthesis equivalence. The algorithm for the proposed compositional synthesis procedure is described in Sect. VI, and Sect. VII applies the algorithm to several benchmark examples. Some concluding remarks are drawn in Sect. VIII. Formal proofs of technical results are omitted from this paper and can be found in [24].

II. Preliminaries

A. Events and Languages

The behaviour of discrete event systems can be described using events and languages. *Events* represent incidents that cause transitions from one state to another and are taken from a finite alphabet Σ. For the purpose of supervisory control, this alphabet is partitioned into two disjoint subsets, the set Σ_c of *controllable* events and the set Σ_u of *uncontrollable* events. Controllable events can be disabled by a supervisor, while uncontrollable events may not be disabled by a supervisor. In addition, the special *termination event* $\omega \notin \Sigma$ is used, with the notation $\Sigma_\omega = \Sigma \cup \{\omega\}$.

Σ^* is the set of all finite traces of events from Σ, including the *empty trace* ε. A subset $L \subseteq \Sigma^*$ is called a *language*. The concatenation of two traces $s, t \in \Sigma^*$ is written as st. A trace $s \in \Sigma^*$ is called a *prefix* of $t \in \Sigma^*$, written $s \sqsubseteq t$, if $t = su$ for some $u \in \Sigma^*$. For $\Omega \subseteq \Sigma$, the *natural projection* $P_\Omega \colon \Sigma^* \to \Omega^*$ is the operation that removes from traces $s \in \Sigma^*$ all events not in Ω.

B. Finite-State Automata

Discrete system behaviours are typically modelled by deterministic automata, but in this paper nondeterministic automata may arise as intermediate results during abstraction.

Definition 1: A finite-state automaton is a tuple $G = \langle \Sigma, Q, \to, Q^\circ \rangle$, where Σ is a finite set of events, Q is a finite set of states, $\to \subseteq Q \times \Sigma_\omega \times Q$ is the *state transition relation*, and $Q^\circ \subseteq Q$ is the set of *initial states*. G is *deterministic*, if $|Q^\circ| \leq 1$, and $x \xrightarrow{\sigma} y_1$ and $x \xrightarrow{\sigma} y_2$ always implies $y_1 = y_2$.

The transition relation is written in infix notation $x \xrightarrow{\sigma} y$, and is extended to traces in Σ_ω^* by letting $x \xrightarrow{\varepsilon} x$ for all $x \in Q$, and $x \xrightarrow{s\sigma} z$ if $x \xrightarrow{s} y$ and $y \xrightarrow{\sigma} z$ for some $y \in Q$. Furthermore, $x \xrightarrow{s}$ means that $x \xrightarrow{s} y$ for some $y \in Q$, and $x \to y$ means that $x \xrightarrow{s} y$ for some $s \in \Sigma_\omega^*$, and $x \not\xrightarrow{\sigma}$ means $x \xrightarrow{\sigma}$ does not hold. These notations also apply to state sets, $X \xrightarrow{s}$ for $X \subseteq Q$ means that $x \xrightarrow{s}$ for some $x \in X$, and to automata, $G \xrightarrow{s}$ means that $Q^\circ \xrightarrow{s}$, etc. The *language* of an automaton G is $\mathcal{L}(G) = \{\, s \in \Sigma^* \mid G \xrightarrow{s} \,\}$.

The termination event $\omega \notin \Sigma$ denotes completion of a task and does not appear anywhere else but to mark such completions. It is required that states reached by ω do not have any outgoing transitions, i.e., if $x \xrightarrow{\omega} y$ then there does not exist $\sigma \in \Sigma_\omega$ such that $y \xrightarrow{\sigma}$. This ensures that the termination event, if it occurs, is always the final event of any trace. The traditional set of marked states is $Q^\omega = \{\, x \in Q \mid x \xrightarrow{\omega} \,\}$ in this notation. For graphical simplicity, states in Q^ω are shown shaded in the figures of this paper instead of explicitly showing ω-transitions.

When two or more automata are brought together to interact, lock-step synchronisation in the style of [25] is used.

Definition 2: Let $G_1 = \langle \Sigma_1, Q_1, \to_1, Q_1^\circ \rangle$ and $G_2 = \langle \Sigma_2, Q_2, \to_2, Q_2^\circ \rangle$ be two automata. The *synchronous composition* of G_1 and G_2 is defined as

$$G_1 \parallel G_2 = \langle \Sigma_1 \cup \Sigma_2, Q_1 \times Q_2, \to, Q_1^\circ \times Q_2^\circ \rangle \qquad (1)$$

where

$$\begin{aligned}
(x_1, x_2) &\xrightarrow{\sigma} (y_1, y_2) \quad \text{if } \sigma \in (\Sigma_1 \cap \Sigma_2) \cup \{\omega\}, \\
&\qquad\qquad x_1 \xrightarrow{\sigma}_1 y_1, \text{ and } x_2 \xrightarrow{\sigma}_2 y_2\,; \\
(x_1, x_2) &\xrightarrow{\sigma} (y_1, x_2) \quad \text{if } \sigma \in \Sigma_1 \setminus \Sigma_2 \text{ and } x_1 \xrightarrow{\sigma}_1 y_1\,; \\
(x_1, x_2) &\xrightarrow{\sigma} (x_1, y_2) \quad \text{if } \sigma \in \Sigma_2 \setminus \Sigma_1 \text{ and } x_2 \xrightarrow{\sigma}_2 y_2\,.
\end{aligned}$$

Synchronous composition is associative, that is, $G_1 \parallel (G_2 \parallel G_3) = (G_1 \parallel G_2) \parallel G_3 = G_1 \parallel G_2 \parallel G_3$.

Another common automaton operation is the *quotient* modulo an equivalence relation on the state set.

Definition 3: Let X be a set. A relation $\sim \subseteq X \times X$ is called an *equivalence relation* on X if it is reflexive, symmetric, and transitive. Given an equivalence relation $\sim$ on X, the *equivalence class* of $x \in X$ is $[x] = \{\, x' \in Q \mid x \sim x' \,\}$, and $X/{\sim} = \{\, [x] \mid x \in X \,\}$ is the set of all equivalence classes modulo $\sim$.

Definition 4: Let $G = \langle \Sigma, Q, \to, Q^\circ \rangle$ be an automaton and let $\sim \subseteq Q \times Q$ be an equivalence relation. The *quotient automaton* of G modulo $\sim$ is

$$G/{\sim} = \langle \Sigma, Q/{\sim}, \to/{\sim}, \tilde{Q}^\circ \rangle\ , \qquad (2)$$

where $\to/{\sim} = \{\, ([x], \sigma, [y]) \mid x \xrightarrow{\sigma} y \,\}$ and $\tilde{Q}^\circ = \{\, [x^\circ] \mid x^\circ \in Q^\circ \,\}$.

C. Supervisory Control Theory

Given a *plant* automaton G and a *specification* automaton K, a *supervisor* is a controlling agent that restricts the behaviour of the plant such that the specification is always fulfilled. *Supervisory control theory* [1] provides a method to synthesise a supervisor. Two common requirements for the supervisor are *controllability* and *nonblocking*.

Definition 5: Let G and K be two automata using the same alphabet Σ, and let $\Gamma \subseteq \Sigma$. Then K is said to be Γ-*controllable* with respect to G if, for every trace $s \in \Sigma^*$, every state x of K, and every event $\gamma \in \Gamma$ such that $K \xrightarrow{s} x$ and $G \xrightarrow{s\gamma}$, it holds that $x \xrightarrow{\gamma}$ in K.

When $\Gamma = \Sigma_\mathrm{u}$ then K is simply said to be *controllable* with respect to G.

Definition 6: An automaton $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ is *nonblocking*, if for every state $x \in Q$ and every trace $s \in \Sigma^*$ such that $G \xrightarrow{s} x$ there exists $t \in \Sigma^*$ such that $x \xrightarrow{t\omega}$.

For a deterministic plant G, it is well-known [1] that there exists a supremal controllable and nonblocking sublanguage of $\mathcal{L}(G)$, which represents the *least restrictive* feasible supervisor. Algorithmically, it is more convenient to perform synthesis on the automaton G instead of this language, or more precisely on the lattice of *subautomata* of G [26].

Definition 7: [15] $G_1 = \langle \Sigma, Q_1, \rightarrow_1, Q_1^\circ \rangle$ is a *subautomaton* of $G_2 = \langle \Sigma, Q_2, \rightarrow_2, Q_2^\circ \rangle$, written $G_1 \subseteq G_2$, if $Q_1 \subseteq Q_2$, $\rightarrow_1 \subseteq \rightarrow_2$, and $Q_1^\circ \subseteq Q_2^\circ$.

Theorem 1: [14] Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a deterministic automaton and $\Gamma \subseteq \Sigma$. Then there exists a supremal Γ-controllable and nonblocking subautomaton,

$$\sup\mathcal{C}_\Gamma(G) = \sup\{\, G' \subseteq G \mid G' \text{ is } \Gamma\text{-controllable with respect to } G \text{ and nonblocking} \,\} \ . \tag{3}$$

Again, the subscript Γ is omitted when $\Gamma = \Sigma_\mathrm{u}$, i.e., $\sup\mathcal{C}(G) = \sup\mathcal{C}_{\Sigma_\mathrm{u}}(G)$.

The supremal element is defined based on the subautomaton relationship (Def. 7). The result is equivalent to that of traditional supervisory control theory [1]. That is, $\sup\mathcal{C}(G)$ represents the behaviour of the least restrictive supervisor that disables only controllable events in G such that nonblocking is ensured.

The supervisor can be represented as a map $\Phi\colon \Sigma^* \to 2^\Sigma$ that assigns to each trace $s \in \Sigma^*$ a *control decision* $\Phi(s)$ such that $\Sigma_\mathrm{u} \subseteq \Phi(s) \subseteq \Sigma$, consisting of the events to be enabled after observing the trace s. A supervisor can only disable controllable events and leaves all uncontrollable events enabled [1], [22]. An automaton S can also implement a supervisor map, using

$$\Phi_S(s) = \Sigma_\mathrm{u} \cup \{\, \sigma \in \Sigma_\mathrm{c} \mid s\sigma \in \mathcal{L}(S) \,\} \ . \tag{4}$$

If $S = \sup\mathcal{C}(G)$, then controllability and nonblocking are ensured. The supervisor automaton can be computed using a fixpoint algorithm, which iteratively identifies and removes from G blocking states and states leading to blocking states via uncontrollable events [14].

In this paper, the supervisor has a modular structure, $\mathcal{S} = \{S_1, \ldots, S_n\}$, consisting of a set of supervisor automata. The combined global supervisor can be constructed by applying the formal definition of synchronous composition,

$$\|\mathcal{S} = \bigparallel_{i=1}^{n} S_i \ . \tag{5}$$

In practice, the supervisor can be represented in its modular form, and synchronisation is performed on-line, tracking the component states as the system evolves. In this way, explicit synchronous product computation and state-space explosion are avoided. Based on this, supervisors are identified with automata or sets of automata in the following.

The operator $\sup\mathcal{C}$ only defines the synthesis result for a plant automaton G. In order to apply this synthesis to control problems that also involve specifications, the transformation proposed in [14] is used. Specification automata are transformed into plants by adding, for every uncontrollable event that is not enabled in a state, a transition to a new blocking state $\bot$. This essentially transforms all potential controllability problems into potential blocking problems.

Definition 8: [14] Let $K = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a specification. The *complete plant automaton* $K^\bot$ for K is

$$K^\bot \ = \ \langle \Sigma, Q \cup \{\bot\}, \rightarrow^\bot, Q^\circ \rangle \tag{6}$$

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

$$\rightarrow^\bot \ = \ \rightarrow \cup \{\, (x, \upsilon, \bot) \mid \upsilon \in \Sigma_\mathrm{u} \text{ and } K \xrightarrow{s} x \not\xrightarrow{\upsilon} \text{ for some } s \in \Sigma^* \,\} \ . \tag{7}$$

In general, synthesis of the least restrictive nonblocking and controllable behaviour allowed by a specification K with respect to a plant G is achieved by computing $\sup\mathcal{C}(G \parallel K^\bot)$ [14].

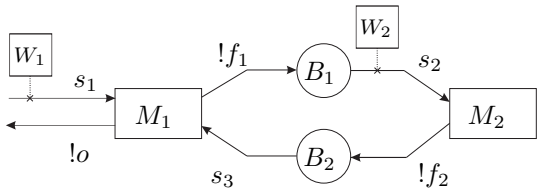

Fig. 1. Manufacturing system overview.

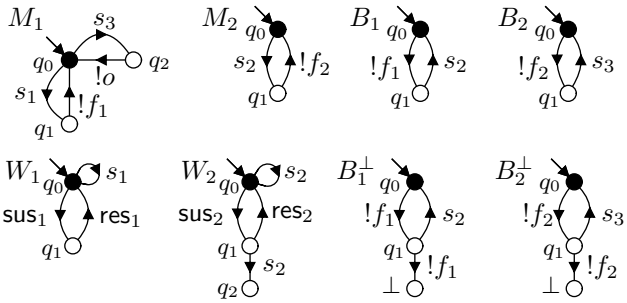

Fig. 2. Automata models of the manufacturing system.

III. Motivating Example

This section demonstrates compositional synthesis using the example of a simple manufacturing system shown in Fig. 1. Two machines M_1 and M_2 are linked by two buffers B_1

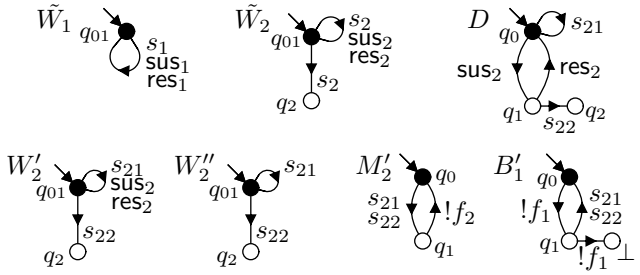

Fig. 3. Abstraction results for switches in the manufacturing system example.

and B_2 that can store one workpiece each. The first machine M_1 takes workpieces from outside the system (event s_1), processes them, and puts them into B_1 (event $!f_1$). M_1 also takes workpieces from B_2 (event s_3), processes them, and outputs them from the system (event $!o$). Machine M_2 takes workpieces from B_1 (event s_2), processes them, and puts them into B_2 (event $!f_2$). Using switches W_1 and W_2, the user can suspend (event sus_i) and resume (event res_i) production of M_1 or M_2, respectively.

Fig. 2 shows an automata model of the system. All events are observable, and uncontrollable events are prefixed by an exclamation mark (!). Automata M_1, M_2, W_1, and W_2 are plants. For illustration, the two switches are not identical. W_2 models a requirement for the synthesised supervisor to prevent starting of M_2 in suspend mode, while W_1 models a plant where it is physically impossible to start M_1 in suspend mode. Automata B_1 and B_2 are specifications to avoid buffer overflow and underflow, which are transformed into complete plant automata $B_1^{\perp}$ and $B_2^{\perp}$ (Def. 8). To satisfy these specifications, a supervisor must be synthesised for the system.

The compositional synthesis procedure is a sequence of small steps. At each step, automata are simplified and replaced by abstracted versions such that the supervisor synthesised from the abstracted system yields the same language when controlling the system as would the supervisor synthesised from the original system. Synchronous composition is computed step by step on the abstracted automata. In addition to synchronisation and abstractions, a supervisor component may also be produced at each step. In the end, the procedure results in a single abstracted automaton, which is simpler than the original system, and standard synthesis is applied to this abstracted automaton.

Initially, the system is $\mathcal{G}_0 = \{W_1, W_2, M_1, M_2, B_1^{\perp}, B_2^{\perp}\}$. In the first step of compositional synthesis, individual automata are abstracted if possible. Events sus_1 and res_1 only appear in automaton W_1, and such events are referred to as *local events*. Exploiting local events, states q_0 and q_1 in W_1 can be merged, as synthesis will always remove either none or both of these states. Automaton W_1 can then be replaced by a *synthesis equivalent* automaton $\tilde{W}_1$ shown in Fig. 3. Automaton $\tilde{W}_1$ is a selfloop-only automaton that always enables all its events, so it can be disregarded in the synthesis.

Similarly, events sus_2 and res_2 are local to automaton W_2, so the same abstraction method can be applied. However, an attempt to compute an abstraction as before results in the

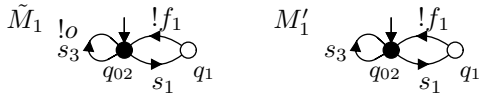

Fig. 4. Abstracted automata of M_1.

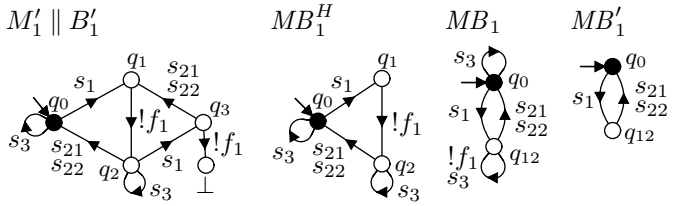

Fig. 5. $M_1' \parallel B_1'$ and its abstraction result.

nondeterministic automaton $\tilde{W}_2$ shown in Fig. 3. A correct supervisor needs to be aware of the states of W_2 in order to decide whether or not to enable controllable event s_2, and it is not straightforward to construct such a supervisor only from the abstraction $\tilde{W}_2$.

To solve the nondeterminism problem, event s_2 in $\tilde{W}_2$ is replaced by two new events s_{21} and s_{22}. This procedure is referred to as *renaming*. Automaton $\tilde{W}_2$ is replaced by the renamed deterministic automaton W_2' shown in Fig. 3, and automaton D, which is the renamed version of W_2, is stored as a *distinguisher* in a set $\mathcal{S}$ of *collected supervisors*. It is the first component of the supervisor to be computed in the end.

Having replaced s_2 in W_2, automata M_2 and $B_1^{\perp}$ need to be modified to use the new events s_{21} and s_{22}. Therefore, M_2 and $B_1^{\perp}$ are replaced by M_2' and B_1' shown in Fig. 3. These automata are constructed by replacing the s_2-transitions in M_2 and $B_1^{\perp}$ by transitions labelled s_{21} and s_{22}.

After this, events sus_2 and res_2 only appear in selfloops in the entire system, and as a result no state change is possible by executing these events. Thus, the selfloops associated with these events can be removed, which results in the abstracted automaton W_2'' shown in Fig. 3.

Next, events $!o$ and s_1 are local events in M_1. States q_0 and q_2 can be merged. However, since $!f_1$ is not a local event, q_0 and q_1 can not be merged since q_1 can be a blocking state if $!f_1$ is disabled by other components. Fig. 4 shows the abstracted automaton $\tilde{M}_1$. Furthermore, event $!o$ now only appears in a selfloop in the entire system and thus, the selfloop associated with this event can be removed from $\tilde{M}_1$, resulting in the abstracted automaton M_1' shown in Fig. 4.

At this point, the system has been simplified to $\mathcal{G} = \{W_2'', M_1', M_2', B_1', B_2^{\perp}\}$. None of these automata can be simplified further, so the next step is to compose some of them. Fig. 5

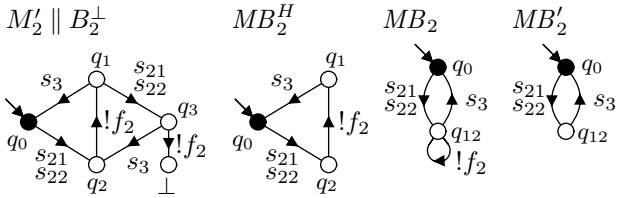

Fig. 6. $M_2' \parallel B_2^{\perp}$ and its abstraction result.

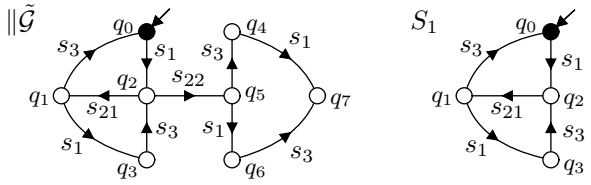

Fig. 7. The final abstracted system and the calculated supervisor for $\|\tilde{\mathcal{G}}$.

shows the composition of M_1' and B_1', which causes $!f_1$ to become a local event. Clearly, the blocking state $\perp$ in $M_1' \| B_1'$ must be avoided, and since the uncontrollable event $!f_1$ only appears in this automaton, this means that state q_3 also must be avoided. Then controllable event s_1 must be disabled in q_2. Therefore, automaton $M_1' \parallel B_1'$ is replaced by the synthesis equivalent abstraction MB_1^H shown in Fig. 5. This is a special case of *halfway synthesis* [14], explained in more detail in Sect. V-B. The abstracted automaton MB_1^H is added to the set $\mathcal{S}$ of collected supervisors to enable the final supervisor to make the control decision for s_1. Furthermore, since $!f_1$ is a local uncontrollable event, states q_1 and q_2 in MB_1^H can be merged, which results in the synthesis equivalent automaton MB_1 shown in Fig. 5. Then events s_3 is always enabled in MB_1, and only appears on selfloop transitions, and $!f_1$ only appears on selfloops in the entire model. Thus, these events can be removed, resulting in MB_1' shown in Fig. 5.

A similar procedure is applied to $M_2' \parallel B_2^{\perp}$. Exploiting the local event $!f_2$ results in the abstracted automata MB_2^H, MB_2, and MB_2' shown in Fig. 6.

After all these abstractions, the uncontrolled plant model is $\tilde{\mathcal{G}} = \{W_2'', MB_1', MB_2'\}$, and the collected supervisor set is $\mathcal{S} = \{D, MB_{1\_}^H, MB_2^H\}$. The last two steps are to compose the automata in $\tilde{\mathcal{G}}$, resulting in the 8-state automaton shown in Fig. 7, and to calculate a supervisor for this automaton. This supervisor is S_1 in Fig. 7 and has 4 states. Adding it to the set $\mathcal{S}$ results in the nonblocking modular supervisor

$$\mathcal{S} = \{D, MB_1^H, MB_2^H, S_1\} \ , \tag{8}$$

which is the least restrictive, controllable and nonblocking supervisor, and produces the exact same controlled behaviour as would a monolithic supervisor calculated for the original system $\mathcal{G}$. The largest component of the modular supervisor is S_1 with 4 states, and it has been computed by exploring the state space of $\|\tilde{\mathcal{G}}$ with 8 states. In contrast, standard monolithic synthesis explores a state space of 138 states and produces a single supervisor with 52 states.

The example demonstrates how compositional synthesis works. In the sequel, Sect. IV explains the concepts formally and shows how the renamed supervisor can control the unrenamed plant, and Sect. V describes the individual abstraction methods.

IV. Compositional Synthesis

This section describes the compositional synthesis framework. The data structure of *synthesis triples* is introduced, which represents partially solved synthesis problems in the algorithm. Based on this, a *control architecture* is presented to implement the computed supervisors after renamings.

A. Basic Idea

The input to compositional synthesis is an arbitrary set of deterministic automata representing the plant to be controlled,

$$\mathcal{G} = \{G_1, G_2, \ldots, G_n\} \ . \tag{9}$$

The objective is to calculate a supervisor that constrains the behaviour of $\mathcal{G}$ to its least restrictive nonblocking subbehaviour, by disabling only controllable events.

Compositional synthesis works by repeated abstraction of system components G_i based on *local events*; events that appear in G_i and in no other automata G_j with $j \neq i$ are *local* to G_i, and they are crucial to abstraction. In the following, the set of local events is denoted by Υ, and $\Omega = \Sigma \setminus \Upsilon$ denotes the set of non-local or *shared* events.

Using abstraction, some components G_i in (9) are replaced by simpler versions G_i'. If this is no longer possible, some components in (9) are selected and composed, i.e., replaced by their synchronous composition. This typically leads to new local events, making further abstraction possible.

When an abstraction G_i' is computed, this may lead to the discovery of new supervisor decisions. For example, if G_i contains a controllable transition leading to a blocking state, it is clear that this transition must be disabled by every supervisor. Therefore, abstraction may produce a supervisor component S_i in addition to the abstracted automaton G_i'. The algorithm collects these supervisor components in a set $\mathcal{S}$, called the set of *collected supervisors*. Abstraction may also result in nondeterminism, which is avoided by applying a renaming.

Thus, compositional synthesis starts with the set of plant automata (9), no collected supervisors, and no renaming. At each step, plant automata are replaced by the result of abstraction or synchronous composition, supervisors are added to $\mathcal{S}$, and the renaming is modified. Eventually, only one plant automaton is left, which is removed from $\mathcal{G}$ and used to calculate the final supervisor to be added to $\mathcal{S}$. Then $\mathcal{G}$ becomes empty and the collected supervisors $\mathcal{S}$, together with the renaming ρ, form a least restrictive supervisor for the original synthesis problem.

B. Renaming

Nondeterminism is avoided in the compositional synthesis algorithm, because it is not straightforward to compute supervisors from nondeterministic abstractions. If an abstraction step results in a nondeterministic automaton, a *renaming* is applied first, introducing new events to disambiguate nondeterministic branching.

The use of renaming to disambiguate abstractions is proposed in [21]. In the following, a renaming is a map that relates the events of the current abstracted system $\mathcal{G}$ to the events in the original plant, which works in the reverse direction compared to [21].

Definition 9: Let Σ_1 and Σ_2 be two sets of events. A *renaming* $\rho\colon \Sigma_2 \to \Sigma_1$ is a controllability-preserving map, i.e., a map such that $\rho(\sigma)$ is controllable if and only if σ is controllable.

For example, when event s_2 is disambiguated into s_{21} and s_{22} in automaton $\tilde{W}_2$ in Fig. 3 in the introductory example, the renaming ρ is such that $\rho(s_{21}) = \rho(s_{22}) = s_2$ and $\rho(\sigma) = \sigma$ for all other events.

The definition of ρ is extended to cover the termination event by letting $\rho(\omega) = \omega$. Renamings are extended to traces $s \in \Sigma_2^*$ by applying them to each event, and to languages $L \subseteq \Sigma_2^*$ by applying them to all traces. They are also extended to automata with alphabet Σ_2 by replacing all transitions $x \xrightarrow{\sigma} y$ with $x \xrightarrow{\rho(\sigma)} y$.

When new events are introduced, the compositional synthesis algorithm continues to operate using the new events and thus produces a supervisor based on an alphabet different from that of the original plant. To communicate correctly with the original plant, the supervisor needs to determine which of the new events (s_{21} or s_{22}) is to be executed when the plant generates one of its original events (s_2). This is achieved by adding a so-called *distinguisher* [20] to the synthesis result.

Definition 10: An automaton $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ *differentiates* event γ_1 from γ_2, if $\gamma_1 \notin \Sigma$ and $\gamma_2 \in \Sigma$ or there exists a transition $x \xrightarrow{\gamma_1} y$ such that $x \xrightarrow{\gamma_2} y$ does not hold.

Definition 11: Let $\rho\colon \Sigma_2 \to \Sigma_1$ be a renaming. An automaton G_2 with alphabet Σ_2 is a *ρ-distinguisher* if, for all traces $s, t \in \mathcal{L}(G_2)$ such that $\rho(s) = \rho(t)$, it holds that $s = t$.

Based on Def. 11, a distinguisher differentiates between the renamed events. Furthermore, two traces accepted by a distinguisher never differ only in the renamed events. This guarantees that only one of the renamed events is enabled at each state. In the introductory example, automaton D in Fig. 3 is a ρ-distinguisher that differentiates s_{21} from s_{22}. This is because D enables at most one of the events s_{21} and s_{22} in each state, so it can always make a choice between these two events.

Another operation is necessary in combination with renaming. After applying a renaming to an automaton G_i in a system $\mathcal{G} = \{G_1, \ldots, G_n\}$, the remaining automata G_j with $j \neq i$ and the collected supervisors, $\mathcal{S}$, need to be modified to use the new events.

Definition 12: Let $G = \langle \Sigma_1, Q, \rightarrow, Q^\circ \rangle$ be an automaton, and let $\rho\colon \Sigma_2 \to \Sigma_1$ be a renaming. Then $\rho^{-1}(G) = \langle \Sigma_2, Q, \rho^{-1}(\rightarrow), Q^\circ \rangle$ where $\rho^{-1}(\rightarrow) = \{\, (x, \sigma, y) \mid x \xrightarrow{\rho(\sigma)} y \,\}$.

Automaton $\rho^{-1}(G)$ is obtained by replacing transitions labelled with the original event by new transitions labelled with each of the new events. For example, Fig. 3 in the introductory example shows $M_2' = \rho^{-1}(M_2)$ and $B_1' = \rho^{-1}(B_1^\perp)$, which replace the original plants M_2 and $B_1^\perp$ after the renaming. When a renaming is applied, the distinguisher is the only automaton that differentiates between the renamed events, all others are transformed by ρ^{-1}.

The compositional synthesis algorithm proposed in the following repeatedly applies renamings and eventually produces a supervisor $\mathcal{S}$ using a modified alphabet $\Sigma_\mathcal{S}$, and a renaming $\rho\colon \Sigma_\mathcal{S} \to \Sigma$ that maps the renamed events back to the events of the original plant. The control architecture in Fig. 8 enables the renamed supervisor $\mathcal{S}$ to interact with the original unrenamed plant $\mathcal{G}$.

Assume that, after execution of a trace t, an event γ occurs in the plant, and γ has been renamed and replaced by γ_1 and γ_2. Being unaware of the renaming, the plant will just communicate the occurrence of γ to the supervisor. When this happens, first the function ρ^{-1} replaces γ by the set $\{\gamma_1, \gamma_2\}$, sending both possibilities to the distinguisher $\mathcal{S}_D$, which is part of the set $\mathcal{S}$ of collected supervisors. Following Def. 11, $\mathcal{S}_D$ enables only one of γ_1 or γ_2. The selected event γ', either γ_1 or γ_2, is passed to the remaining components of the supervisor, $\mathcal{S} \setminus \mathcal{S}_D$, to update their states and issue a new control decision $\Phi'(t'\gamma') \subseteq \Sigma_\mathcal{S}$. Here, t' is the renamed version of the history t. The control decision is based on the renamed model and therefore contains renamed events, so the renaming ρ is applied to translate it back to a control decision $\Phi(t\gamma) \subseteq \Sigma$ using the original plant events.

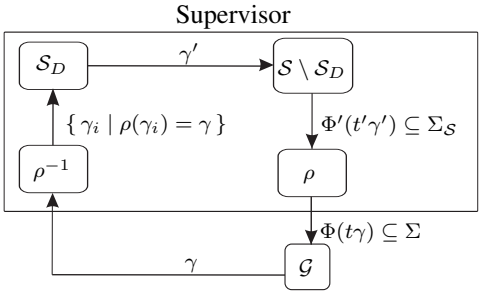

Fig. 8. Control architecture. $\mathcal{G}$ is the original plant, $\mathcal{S}$ are the computed modular supervisors, and $\mathcal{S}_D \subseteq \mathcal{S}$ are the distinguishers.

C. Synthesis Triples

The compositional synthesis algorithm keeps track of three pieces of information:

- a set $\mathcal{G} = \{G_1, \ldots, G_n\}$ of uncontrolled plant automata;
- a set $\mathcal{S} = \{S_1, \ldots, S_m\}$ of collected supervisor automata;
- a renaming ρ that maps the events of the automata in $\mathcal{G} \cup \mathcal{S}$ back to events of the original plant.

This information is combined in a *synthesis triple*, which is the main data structure manipulated by the compositional synthesis algorithm.

Definition 13: A *synthesis triple* is a triple $(\mathcal{G}; \mathcal{S}; \rho)$, where $\mathcal{G}$ and $\mathcal{S}$ are sets of deterministic automata and ρ is a renaming, such that

(i) $\mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}(\mathcal{G})$;
(ii) $\mathcal{S}$ is a ρ-distinguisher;
(iii) for all events γ_1, γ_2 such that $\rho(\gamma_1) = \rho(\gamma_2)$, there exists at most one automaton $G_j \in \mathcal{G}$ that differentiates γ_1 from γ_2.

Here and in the following, sets $\mathcal{G}$ and $\mathcal{S}$ are also used to denote the synchronous composition of their elements, like $\|\mathcal{G} = G_1 \parallel \cdots \parallel G_n$. When $\mathcal{G} = \emptyset$ then $\|\mathcal{G}$ is the universal automaton that accepts the language Σ^*.

A synthesis triple represents a partially solved control problem at an intermediate step of compositional synthesis. The set $\mathcal{G}$ contains an abstracted plant model, and $\mathcal{S}$ contains the supervisors collected so far, which must constrain the behaviour of the plant (i). The renaming ρ maps the events found in the abstracted plant or collected supervisors back to events in the original plant. The synchronous composition of the

supervisors is required to have the distinguisher property (ii) to ensure that it can be used with the control architecture in Fig. 8. Furthermore, if two events γ_1 and γ_2 are renamed to the same event, then there can be at most one automaton in the set $\mathcal{G}$ that differentiates between these events (iii).

The following notation associates with each synthesis triple a synthesis result.

Definition 14: Let $(\mathcal{G};\mathcal{S};\rho)$ be a synthesis triple. Then

$$\sup\mathcal{C}(\mathcal{G};\mathcal{S};\rho) = \rho(\sup\mathcal{C}(\mathcal{G}) \parallel \mathcal{S}) \ . \tag{10}$$

The synthesis result for the partially solved control problem $(\mathcal{G};\mathcal{S};\rho)$ is obtained by composing the monolithic supervisor for the remaining plants, $\sup\mathcal{C}(\mathcal{G})$, with the supervisors collected so far, $\mathcal{S}$, and afterwards renaming.

Example 1: At the final step of the compositional synthesis in Sect. III, the abstracted uncontrolled system is $\tilde{\mathcal{G}} = \{W_2'', MB_1', MB_2'\}$, the collected supervisor set is $\mathcal{S} = \{D, MB_1^H, MB_2^H\}$, and the renaming ρ is such that $\rho(s_{21}) = \rho(s_{22}) = s_2$ and $\rho(\sigma) = \sigma$ for $\sigma \notin \{s_{21}, s_{22}\}$. This is represented by the synthesis triple $(\tilde{\mathcal{G}};\mathcal{S};\rho) = (\{W_2'', MB_1', MB_2'\};\{D, MB_1^H, MB_2^H\};\rho)$. The synthesis result for this triple is obtained by calculating a monolithic supervisor for the abstracted uncontrolled plant, $S_1 = \sup\mathcal{C}(W_2'' \parallel MB_1' \parallel MB_2')$, which is added to the supervisor set, $\mathcal{S}$; and afterwards all components are renamed back. This gives $\sup\mathcal{C}(\tilde{\mathcal{G}};\mathcal{S};\rho) = \rho(S_1 \parallel D \parallel MB_1^H \parallel MB_2^H)$. As explained in Sect. IV-B, the synchronous composition never has to be computed explicitly as it can be represented in its modular form.

While manipulating synthesis triples, the compositional synthesis algorithm maintains the invariant that all generated triples have the same synthesis result, which is equivalent to the least restrictive solution of the original control problem. Every abstraction step must ensure that the synthesis result is the same as it would have been for the non-abstracted components. This property is called *synthesis equivalence* [16].

Definition 15: Two triples $(\mathcal{G}_1;\mathcal{S}_1;\rho_1)$ and $(\mathcal{G}_2;\mathcal{S}_2;\rho_2)$ are *synthesis equivalent*, $(\mathcal{G}_1;\mathcal{S}_1;\rho_1) \simeq_{\text{synth}} (\mathcal{G}_2;\mathcal{S}_2;\rho_2)$, if

$$\mathcal{L}(\sup\mathcal{C}(\mathcal{G}_1;\mathcal{S}_1;\rho_1)) = \mathcal{L}(\sup\mathcal{C}(\mathcal{G}_2;\mathcal{S}_2;\rho_2)) \ . \tag{11}$$

The compositional synthesis algorithm calculates a modular supervisor for a modular system $\mathcal{G} = \mathcal{G}_0$. Initially no renaming has been applied and no supervisor or distinguisher has been collected. Thus, this input is converted to the initial synthesis triple $(\mathcal{G};\mathcal{G};\mathrm{id})$, where $\mathrm{id}\colon \Sigma \to \Sigma$ is the identity map, i.e., $\mathrm{id}(\sigma) = \sigma$ for all $\sigma \in \Sigma$. Afterwards, the initial triple is abstracted repeatedly such that synthesis equivalence is preserved,

$$(\mathcal{G};\mathcal{G};\mathrm{id}) = (\mathcal{G}_0;\mathcal{S}_0;\rho_0) \simeq_{\text{synth}} \cdots \simeq_{\text{synth}} (\mathcal{G}_k;\mathcal{S}_k;\rho_k) \ . \tag{12}$$

Some of these steps replace an automaton in $\mathcal{G}_k$ by an abstraction, others reduce the number of automata in $\mathcal{G}_k$ by synchronous composition or by replacing an automaton in $\mathcal{G}_k$ with a supervisor in $\mathcal{S}_{k+1}$. The algorithm terminates when $\mathcal{G}_k = \emptyset$, at which point $\mathcal{S}_k$ together with ρ_k forms the modular supervisor. The following result, which follows directly from Def. 14 and 15, confirms that this approach gives the same supervised behaviour as a monolithic supervisor for the original system.

Theorem 2: Let $\mathcal{G} = \{G_1, \ldots, G_n\}$ be a set of automata, and let $(\mathcal{G};\mathcal{G};\mathrm{id}) \simeq_{\text{synth}} (\emptyset;\mathcal{S};\rho)$. Then $\mathcal{L}(\rho(\mathcal{S})) = \mathcal{L}(\sup\mathcal{C}(\emptyset;\mathcal{S};\rho)) = \mathcal{L}(\sup\mathcal{C}(\mathcal{G}))$.

V. Synthesis Triple Abstraction Operations

The idea of compositional synthesis is to continuously rewrite synthesis triples such that synthesis equivalence is preserved. This section gives an overview of different ways to simplify automata that can be used in the framework of this paper. Sect. V-A and V-B present abstraction methods from [1], [14], which here are adapted to synthesis triples, and Sect. V-C and V-D describe methods proposed by the authors in [16], [18]. Further details and formal proofs of correctness can be found in [24].

A. Basic Rewrite Operations

The simplest methods to rewrite synthesis triples are *synchronous composition* and *monolithic synthesis*. It is always possible to compose two automata in the set $\mathcal{G}$ of uncontrolled plants, or to place their monolithic synthesis result into the set $\mathcal{S}$ of supervisors. These basic methods are included here for the sake of completeness. They do not contribute to simplification, and are only needed when no other abstraction is possible.

Theorem 3: Let $\mathcal{G}_1 = \{G_1, \ldots, G_n\}$ and $\mathcal{G}_2 = \{G_1 \parallel G_2, G_3, \ldots, G_n\}$, let ρ be a renaming, and let $\mathcal{S}$ be a ρ-distinguisher. Then $(\mathcal{G}_1;\mathcal{S};\rho) \simeq_{\text{synth}} (\mathcal{G}_2;\mathcal{S};\rho)$.

Theorem 4: Let $(\mathcal{G};\mathcal{S};\rho)$ be a synthesis triple. Then $(\mathcal{G};\mathcal{S};\rho) \simeq_{\text{synth}} (\emptyset;\mathcal{S} \cup \{\sup\mathcal{C}(\mathcal{G})\},\rho)$.

B. Halfway Synthesis

Halfway synthesis is an abstraction method that works well in compositional synthesis [14]. Sometimes it is clear that certain states in an automaton must be removed in synthesis, no matter what the behaviour of the rest of the system is. Clearly, blocking states can never become nonblocking. Moreover, local uncontrollable transitions to blocking states must be removed, because no other component nor the supervisor can disable a local uncontrollable transition.

Definition 16: Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ and $\Gamma \subseteq \Sigma_\mathrm{u}$. The *halfway synthesis result* for G with respect to Γ is

$$\mathrm{hsup}\mathcal{C}_\Gamma(G) = \langle \Sigma, Q \cup \{\bot\}, \rightarrow_\mathrm{hsup}, Q^\circ \rangle \ , \tag{13}$$

where $\sup\mathcal{C}_\Gamma(G) = \langle \Sigma, Q, \rightarrow_\mathrm{sup}, Q^\circ \rangle$, $\bot \notin Q$, and

$$\rightarrow_\mathrm{hsup} = \rightarrow_\mathrm{sup} \cup \{\, (x,\sigma,\bot) \mid \sigma \in \Sigma_\mathrm{u} \backslash \Gamma,\ x \xrightarrow{\sigma}, \text{ and } x \xrightarrow{\sigma}_\mathrm{sup} \text{ does not hold} \,\} \ . \tag{14}$$

Halfway synthesis is calculated like ordinary synthesis, but considering only local events as uncontrollable. Shared uncontrollable transitions to blocking states do not necessarily cause blocking, as some other plant component may yet disable them. Therefore, these transitions are retained and redirected to the blocking state $\bot$ instead.

Example 2: Consider automaton G in Fig. 9 with $\Sigma_\mathrm{u} = \{!\lambda, !\mu, !\upsilon\}$ and $\Upsilon = \{\gamma, !\lambda\}$. State q_3 is blocking, so q_2 is

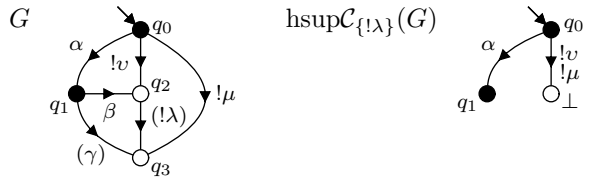

Fig. 9. Example of halfway synthesis. Uncontrollable events are prefixed with !, and local events have parentheses around them.

also considered as unsafe, because the local uncontrollable $!\lambda$-transition cannot be disabled by the supervisor nor by any other plant component. Every nonblocking supervisor can and will disable the controllable transitions $q_1 \xrightarrow{\gamma} q_3$ and $q_1 \xrightarrow{\beta} q_2$. State q_0 may still be safe, because some other plant component may disable the shared events $!\mu$ and $!\upsilon$. The blocking state $\bot$ is added and the $!\mu$- and $!\upsilon$-transitions are redirected to $\bot$ in the halfway synthesis result $\mathrm{hsup}\mathcal{C}_{\{!\lambda\}}(G)$, see Fig. 9. This ensures that later synthesis is aware of the potential problem regarding $!\mu$ or $!\upsilon$.

The following theorem extends a result about halfway synthesis for supervision equivalence using state labels [14] to the more general framework of synthesis triples.

Theorem 5: Let $(\mathcal{G};\mathcal{S};\rho)$ be a synthesis triple with $\mathcal{G} = \{G_1,\ldots,G_n\}$, and let $\Upsilon \subseteq \Sigma_1$ such that $(\Sigma_2 \cup \cdots \cup \Sigma_n) \cap \Upsilon = \emptyset$. Then $(\mathcal{G};\mathcal{S};\rho) \simeq_{\mathrm{synth}} (\{\mathrm{hsup}\mathcal{C}_{\Upsilon\cap\Sigma_\mathrm{u}}(G_1), G_2,\ldots,G_n\}; \{\mathrm{hsup}\mathcal{C}_{\Upsilon\cap\Sigma_\mathrm{u}}(G_1)\} \cup \mathcal{S};\rho)$.

Complexity: Halfway synthesis can be achieved using a standard synthesis algorithm [1] and runs in time complexity $O(|Q||{\to}|)$, where $|Q|$ and $|{\to}|$ are the numbers of states and transitions of the input automaton.

C. Renaming and Selfloop Removal

Another way of rewriting a synthesis triple is by renaming. As explained in Sect. IV, an automaton G_1 can be rewritten into H_1 using a renaming ρ such that $\rho(H_1) = G_1$ and H_1 is a ρ-distinguisher. Then H_1 is added to the set $\mathcal{S}$ of supervisors as a distinguisher, and the renaming ρ is composed with the previous renamings.

Theorem 6: Let $(\mathcal{G}_1;\mathcal{S};\rho_1)$ be a synthesis triple with $\mathcal{G}_1 = \{G_1,\ldots,G_n\}$, let ρ be a renaming, and let H_1 be a ρ-distinguisher such that $\rho(H_1) = G_1$ and $\mathcal{G}_2 = \{H_1, \rho^{-1}(G_2),\ldots,\rho^{-1}(G_n)\}$. Then $(\mathcal{G}_1;\mathcal{S};\rho_1) \simeq_{\mathrm{synth}} (\mathcal{G}_2; \{H_1\} \cup \rho^{-1}(\mathcal{S}); \rho_1 \circ \rho)$.

In compositional verification, events used in only one automaton can immediately be removed from the model [11]. This is not always possible in compositional synthesis. Even if no other automata use an event, the synthesised supervisor may still need to use it for control decisions that are not yet apparent. Therefore, events can only be removed if it is clear that no further supervisor decision depends on them.

An event λ can be removed from a synthesis triple, if it causes no state change, which means that it appears only on selfloop transitions in the automata model. In this case, λ can be removed from all automata. This step is called *selfloop removal* and formally described in Theorem 7.

Definition 17: An automaton $G = \langle \Sigma, Q, \to, Q^\circ\rangle$, is *selfloop-only* for $\lambda \in \Sigma$ if $x \xrightarrow{\lambda} y$ implies $x = y$. Automaton G is selfloop-only for $\Lambda \subseteq \Sigma$ if G is selfloop-only for each $\lambda \in \Lambda$.

Definition 18: The *restriction* of $G = \langle \Sigma, Q, \to, Q^\circ\rangle$ to $\Xi \subseteq \Sigma$ is $G_{|\Xi} = \langle \Xi, Q, \to_{|\Xi}, Q^\circ\rangle$ where $\to_{|\Xi} = \{\, (x,\sigma,y) \in \to \mid \sigma \in \Xi \cup \{\omega\} \,\}$. The restriction of $\mathcal{G} = \{G_1,\ldots,G_n\}$ to Ξ is $\mathcal{G}_{|\Xi} = \{G_{1|\Xi},\ldots,G_{n|\Xi}\}$.

Theorem 7: Let $(\mathcal{G};\mathcal{S};\rho)$ be a synthesis triple such that $\mathcal{G}$ is selfloop-only for $\Lambda \subseteq \Sigma$. Then $(\mathcal{G};\mathcal{S};\rho) \simeq_{\mathrm{synth}} (\mathcal{G}_{|\Sigma\setminus\Lambda};\mathcal{S};\rho)$.

D. Abstraction Based on Observation Equivalence

This section gives an overview of previous results on observation equivalence-based abstractions for synthesis purposes. *Bisimulation* and *observation equivalence* [27] provide well-known abstraction methods that work well in compositional verification [11]. Both can be implemented efficiently [28]. They are known to preserve all temporal logic properties [29], but unfortunately this does not help for synthesis [18]. Synthesis equivalence is preserved when an automaton is replaced by a bisimilar automaton, while observation equivalence must be strengthened to achieve the same result. This can be achieved by *synthesis observation equivalence* [18] and *weak synthesis observation equivalence* [19].

Definition 19: [27] Let $G = \langle \Sigma, Q, \to, Q^\circ\rangle$ be an automaton. An equivalence relation $\sim\ \subseteq Q \times Q$ is called a *bisimulation* on G, if the following holds for all $x_1, x_2 \in Q$ such that $x_1 \sim x_2$: if $x_1 \xrightarrow{\sigma} y_1$ for some $\sigma \in \Sigma_\omega$, then there exists $y_2 \in Q$ such that $x_2 \xrightarrow{\sigma} y_2$ and $y_1 \sim y_2$.

Theorem 8: [18] Let $(\mathcal{G};\mathcal{S};\rho)$ be a synthesis triple with $\mathcal{G} = \{G_1,\ldots,G_n\}$, let $\sim$ be a bisimulation on G_1, and let $\tilde{\mathcal{G}} = \{G_1/{\sim}, G_2,\ldots,G_n\}$. Then it holds that $(\mathcal{G};\mathcal{S};\rho) \simeq_{\mathrm{synth}} (\tilde{\mathcal{G}};\mathcal{S};\rho)$.

Bisimulation is the strongest of the branching process equivalences. Two states are treated as equivalent if they have exactly the same outgoing transitions to the same or equivalent states. Theorem 8 confirms that it is possible to merge bisimilar states in a plant automaton in a synthesis triple while preserving synthesis equivalence.

Bisimulation does not consider local events for abstraction. However, better abstraction can be achieved by differentiating between local and shared events. This is the idea of observation equivalence, which considers two states as equivalent if they can reach equivalent states by the same sequences of shared events.

Definition 20: [27] Let $G = \langle \Sigma, Q, \to, Q^\circ\rangle$ be an automaton with $\Sigma = \Omega \,\dot\cup\, \Upsilon$. An equivalence relation $\sim\ \subseteq Q \times Q$ is called an *observation equivalence* on G with respect to Υ, if the following holds for all $x_1, x_2 \in Q$ such that $x_1 \sim x_2$: if $x_1 \xrightarrow{s_1} y_1$ for some $s_1 \in \Sigma_\omega^*$, then there exist $y_2 \in Q$ and $s_2 \in \Sigma_\omega^*$ such that $P_{\Omega\cup\{\omega\}}(s_1) = P_{\Omega\cup\{\omega\}}(s_2)$, $x_2 \xrightarrow{s_2} y_2$, and $y_1 \sim y_2$.

Example 3: In automaton G in Fig. 10, states q_0 and q_1 can be considered as observation equivalent with respect to $\Upsilon = \{\alpha,\beta\}$. Merging these states results in $\tilde{G}$, also shown in Fig. 10.

Unfortunately, observation equivalence in general does not imply synthesis equivalence, so Theorem 8 cannot be generalised for observation equivalence [18].

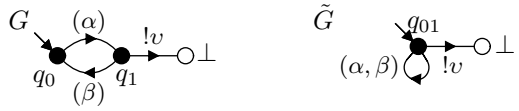

Fig. 10. Example to demonstrate observation equivalence.

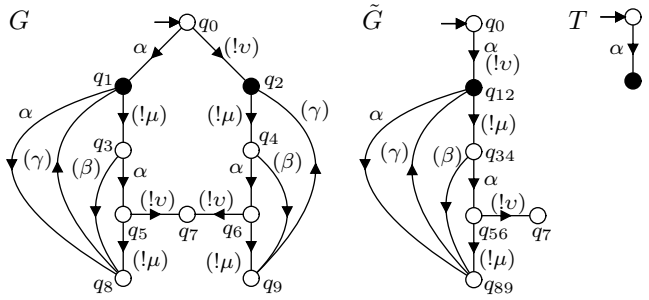

Fig. 11. Observation equivalent automata that are not synthesis equivalent.

Example 4: Consider again the observation equivalent automata in Fig. 10, with $\Sigma_{\mathrm{c}} = \{\alpha, \beta\}$ and $\Sigma_{\mathrm{u}} = \{!\upsilon\}$. The triples $(\{G\}; \{G\}; \mathrm{id})$ and $(\{\tilde{G}\}; \{G\}; \mathrm{id})$ are not synthesis equivalent. With G, a supervisor can disable the local controllable event α to prevent entering state q_1 and thus the occurrence of the undesirable uncontrollable $!\upsilon$, but this is not possible with $\tilde{G}$. It holds that $\omega \in \mathcal{L}(\sup\mathcal{C}(G))$ while $\mathcal{L}(\sup\mathcal{C}(\tilde{G})) = \emptyset$.

There are different ways to restrict observation equivalence so that it can be used in compositional synthesis. The problem in Example 4 does not arise if the local events α and β are uncontrollable. In fact, a result similar to Theorem 8 holds if observation equivalence is restricted to uncontrollable events [18]. With controllable events, abstraction is also possible, but two other issues must be taken into account.

Example 5: Consider automaton G in Fig. 11 with $\Sigma_{\mathrm{u}} = \{!\mu, !\upsilon\}$ and $\Upsilon = \{\beta, \gamma, !\mu, !\upsilon\}$. Merging of observation equivalent states results in $\tilde{G}$, but states q_1 and q_2 should not be merged for synthesis purposes. Although both states can reach the same states via the controllable event α, possibly preceded and followed by the local event $!\mu$, the transition $q_4 \xrightarrow{\alpha} q_6$ must always be disabled to prevent blocking via the local uncontrollable event $!\upsilon$, while the transition $q_1 \xrightarrow{\alpha} q_8$ may be enabled. When used in a system that requires α to occur for correct behaviour, such as T in Fig. 11, state q_1 is retained in synthesis while q_2 is removed. The triples $\mathcal{T} = (\{G, T\}; \{G, T\}; \mathrm{id})$ and $\tilde{\mathcal{T}} = (\{\tilde{G}, T\}; \{G, T\}; \mathrm{id})$ are not synthesis equivalent as $\mathcal{L}(\sup\mathcal{C}(\mathcal{T})) = \emptyset$ but $!\upsilon \in \mathcal{L}(\sup\mathcal{C}(\tilde{\mathcal{T}}))$.

Example 6: Consider automaton G in Fig. 12 with $\Sigma_{\mathrm{u}} = \{!\upsilon, !\mu\}$ and $\Upsilon = \{\alpha, \beta\}$. Merging of observation equivalent states results in $\tilde{G}$, but states q_1 and q_2 should not be merged for synthesis purposes. In G, states q_3 and q_4 should be

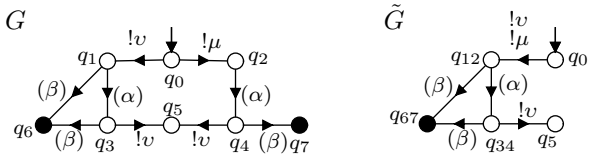

Fig. 12. Observation equivalent automata that are not synthesis equivalent.

avoided to prevent blocking in state q_5 via the local uncontrollable event $!\upsilon$. Thus, α should be disabled in q_1 and q_2, making q_2 a blocking state, while q_1 remains nonblocking due to the transition $q_1 \xrightarrow{\beta} q_6$. The triples $\mathcal{T} = (\{G\}; \{G\}; \mathrm{id})$ and $\tilde{\mathcal{T}} = (\{\tilde{G}\}; \{G\}; \mathrm{id})$ are not synthesis equivalent as $\mathcal{L}(\sup\mathcal{C}(\mathcal{T})) = \emptyset$ but $!\mu \in \mathcal{L}(\sup\mathcal{C}(\tilde{\mathcal{T}}))$.

The problem in Example 5 is caused by considering the path $q_2 \xrightarrow{!\mu\alpha!\mu} q_9$ as equivalent to $q_1 \xrightarrow{\alpha} q_8$ to justify states q_1 and q_2 to be merged. However, the path $q_2 \xrightarrow{!\mu\alpha!\mu} q_9$ passes through the unsafe state q_6, while $q_1 \xrightarrow{\alpha} q_8$ does not pass through any unsafe states. This situation can be avoided by only allowing local events before a controllable event. That is, for $x_1 \xrightarrow{\sigma} y_1$ and $x_1 \sim x_2$ it is required that there exists $t \in \Upsilon^*$ such that $x_2 \xrightarrow{t\sigma} y_2$ and $y_1 \sim y_2$. In Example 5, the local events in t are all uncontrollable. Controllable events can lead to the problem in Example 6. They can be allowed under the additional condition that their target states are equivalent to the start state of the path.

Imposing such conditions on observation equivalence results in *synthesis observation equivalence*, which preserves synthesis results in a way similar to Theorem 8 [18].

Definition 21: [18] Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an automaton with $\Sigma = \Omega \,\dot\cup\, \Upsilon$. An equivalence relation $\sim \subseteq Q \times Q$ is a *synthesis observation equivalence* on G with respect to Υ, if the following conditions hold for all $x_1, x_2 \in Q$ such that $x_1 \sim x_2$:

(i) if $x_1 \xrightarrow{\sigma} y_1$ for $\sigma \in \Sigma_{\mathrm{c}} \cup \{\omega\}$, then there exists a path $x_2 = x_2^0 \xrightarrow{\tau_1} \cdots \xrightarrow{\tau_n} x_2^n \xrightarrow{P_{\Omega\cup\{\omega\}}(\sigma)} y_2$ such that $y_1 \sim y_2$ and $\tau_1, \ldots, \tau_n \in \Upsilon$, and if $\tau_i \in \Sigma_{\mathrm{c}}$ then $x_1 \sim x_2^i$;

(ii) if $x_1 \xrightarrow{\upsilon} y_1$ for $\upsilon \in \Sigma_{\mathrm{u}}$, then there exist $t_2, u_2 \in (\Upsilon \cap \Sigma_{\mathrm{u}})^*$ such that $x_2 \xrightarrow{t_2 P_\Omega(\upsilon) u_2} y_2$ and $y_1 \sim y_2$.

Condition (i) allows for a state x_1 with an outgoing controllable event to be equivalent to another state x_2, if that state allows the same controllable event, possibly after a sequence of local events. If that sequence includes a controllable transition $x_2^{i-1} \rightarrow x_2^i$, its target state x_2^i must be equivalent to the start states $x_1 \sim x_2$. Condition (ii) is similar to observation equivalence, but restricted to uncontrollable events. The projection P_Ω is used in the definition to ensure that the conditions (i) and (ii) apply to both local and shared events.

Example 7: Consider automaton G in Fig. 13, with all events controllable and $\Upsilon = \{\beta\}$. An equivalence relation with $q_1 \sim q_3$ and $q_4 \sim q_7$ is a synthesis observation equivalence on G. Merging the equivalent states results in the deterministic automaton G' shown in Fig. 13. Note that q_1 and q_2 in G are not synthesis observation equivalent, because $q_2 \xrightarrow{\alpha} q_6$ but $q_1 \xrightarrow{\alpha} q_7 \xrightarrow{\beta} q_6$, and the local event β occurs after the shared event α on the path.

Synthesis observation equivalence does not allow local events *after* a controllable event. This condition can be further relaxed, allowing local events after controllable events, provided that it can be guaranteed that the states reached by the local transitions after a controllable event are all present in the synthesis result.

Definition 22: [19] Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an automaton with $\Sigma = \Omega \,\dot\cup\, \Upsilon$. An equivalence relation $\sim \subseteq Q \times Q$ is

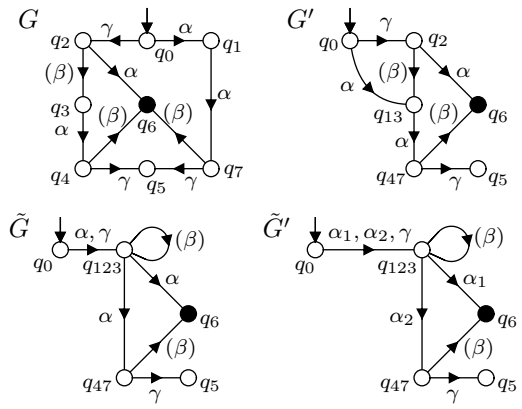

Fig. 13. Example of synthesis observation equivalence and weak synthesis observation equivalence.

a *weak synthesis observation equivalence* on G with respect to Υ, if the following conditions hold for all $x_1, x_2 \in Q$.

(i) If $x_1 \xrightarrow{\sigma} y_1$ for $\sigma \in \Sigma_\mathrm{c} \cup \{\omega\}$, then there exists a path $x_2 = x_2^0 \xrightarrow{\tau_1} \cdots \xrightarrow{\tau_n} x_2^n \xrightarrow{P_{\Omega\cup\{\omega\}}(\sigma)} y_2^{n+1} \xrightarrow{\tau_{n+1}} \cdots \xrightarrow{\tau_m} y_2^{m+1} = y_2$ such that $y_1 \sim y_2$ and $\tau_1, \ldots, \tau_m \in \Upsilon$ and,
   a) if $\tau_i \in \Sigma_\mathrm{c}$ for some $i \leq n$, then $x_1 \sim x_2^i$;
   b) if $y_2^i \xrightarrow{u} z$ for some $u \in (\Sigma_\mathrm{u} \cap \Upsilon)^*$, then $z \sim y_2^j$ for some $n+1 \leq j \leq m+1$;
   c) if $y_2^i \xrightarrow{u} z$ for some $u \in \Sigma_\mathrm{u}^*$ such that $P_\Omega(u) \in \Sigma_\mathrm{u} \setminus \Upsilon$, then there exists $u' \in \Sigma_\mathrm{u}^*$ such that $P_\Omega(u) = P_\Omega(u')$ and $y_2 \xrightarrow{u'} z'$ for some $z' \sim z$.

(ii) If $x_1 \xrightarrow{\upsilon} y_1$ for $\upsilon \in \Sigma_\mathrm{u}$, then there exist $t_2, u_2 \in (\Upsilon \cap \Sigma_\mathrm{u})^*$ such that $x_2 \xrightarrow{t_2 P_\Omega(\upsilon) u_2} y_2$ and $y_1 \sim y_2$.

Condition (i) weakens the condition in Def. 21 for controllable events in that it allows for a path of local events after a controllable event, if local uncontrollable transitions outgoing from the path lead to a state equivalent to a state on the path, and shared uncontrollable transitions are possible in the end state of the path. Condition (ii) is the same as for synthesis observation equivalence.

Example 8: Consider again automaton G in Fig. 13, with all events controllable and $\Upsilon = \{\beta\}$. An equivalence relation with $q_1 \sim q_2 \sim q_3$ and $q_4 \sim q_7$ is a weak synthesis observation equivalence on G, producing the abstraction $\tilde{G} = G/\!\sim$. For example, states q_1 and q_2 can be equivalent as $q_2 \xrightarrow{\alpha} q_6$ and $q_1 \xrightarrow{\alpha} q_7 \xrightarrow{\beta} q_6$, with no uncontrollable transitions from these paths. The nondeterminism in $\tilde{G}$ can be avoided using a renaming $\rho\colon \{\alpha_1, \alpha_2, \beta, \gamma\} \to \{\alpha, \beta, \gamma\}$, which leads to the deterministic automaton $\tilde{G}'$ in Fig. 13.

Both synthesis observation equivalence and weak synthesis observation equivalence can be used for abstraction steps in compositional synthesis. After computing an appropriate equivalence relation $\sim$ on a renamed automaton $\rho(G_1)$, the automaton G_1 can be replaced by its quotient $G_1/\!\sim$.

Theorem 9: [19] Let $(\mathcal{G}; \mathcal{S}; \rho)$ be a synthesis triple with $\mathcal{G} = \{G_1, \ldots, G_n\}$ and $G_i = \langle \Sigma_i, Q_i, \to_i, Q_i^\circ \rangle$. Let $\Upsilon \subseteq \Sigma_1$ such that $(\Sigma_2 \cup \cdots \cup \Sigma_n) \cap \Upsilon = \emptyset$. Let $\sim$ be a synthesis observation equivalence or a weak synthesis observation equivalence relation on $\rho(G_1)$ with respect to Υ such that $G_1/\!\sim$ is deterministic, and let $\tilde{\mathcal{G}} = \{G_1/\!\sim, G_2, \ldots, G_n\}$. Then $(\mathcal{G}; \mathcal{S}; \rho) \simeq_\mathrm{synth} (\tilde{\mathcal{G}}; \mathcal{S}; \rho)$.

Complexity: Observation equivalence-based abstractions can be computed in polynomial time. The time complexity to compute a bisimulation is $O(|\to| \log |Q|)$ [28]. Synthesis observation equivalence and weak synthesis observation equivalence are computed by a modified version of the same algorithm in $O(|\to||Q|^4)$ and $O(|\to||Q|^5)$ time, respectively [19].

Algorithm 1 Compositional synthesis

```
1:  input 𝒢 = {G_1, G_2, ..., G_n}
2:  𝒮 ← 𝒢, ρ ← id
3:  while |𝒢| > 1 do
4:      𝒢 ← selfloopRemoval(𝒢)
5:      subsys ← selectSubSystem(𝒢)
6:      𝒢 ← 𝒢 \ subsys
7:      A ← synchronousComposition(subsys)
8:      Υ ← Σ_A \ Σ_𝒢
9:      A ← hsup𝒞_{Υ∩Σ_u}(A)
10:     𝒮 ← 𝒮 ∪ {A}
11:     A ← bisimulation(A)
12:     Ã ← WSOE_Υ(A)
13:     if Ã is deterministic then
14:         𝒢 ← 𝒢 ∪ {Ã}
15:     else
16:         ⟨ρ_D, D̃, D⟩ ← makeDistinguisher(Ã, A)
17:         𝒢 ← ρ_D^{-1}(𝒢) ∪ {D̃}, 𝒮 ← ρ_D^{-1}(𝒮) ∪ {D}, ρ ← ρ ∘ ρ_D
18:     end if
19: end while
20: 𝒮 ← 𝒮 ∪ {sup𝒞(𝒢)}
```

VI. Compositional Synthesis Algorithm

Given a set of plant automata $\mathcal{G}$, the compositional synthesis algorithm repeatedly composes automata and applies abstraction rules. While doing so, it modifies a synthesis triple $(\mathcal{G}; \mathcal{S}; \rho)$, collecting supervisors in $\mathcal{S}$ and updating the renaming ρ, and continues until only one automaton that cannot be further abstracted is left. Then a standard synthesis algorithm is used to compute a final supervisor from the remaining automaton. This principle, which is justified by Theorems 2 and 4, is shown in Algorithm 1.

During each iteration of the main loop, a series of steps is applied to simplify the set $\mathcal{G}$ of plant automata. First, line 4 applies selfloop removal to the entire plant $\mathcal{G}$ according to Theorem 7. This quick operation improves the performance of the following steps.

The next step is to choose a subsystem of $\mathcal{G}$ for simplification. If no automaton can be simplified individually, a group of automata is selected for composition. The $\mathrm{selectSubSystem}()$ method in line 5 selects an appropriate subsystem, which is then removed from $\mathcal{G}$ and composed. Different methods to select this subsystem are available in the implementation.

After identification and composition of a subsystem, the set Υ of local events is formed in line 8, which contains the events used only in the subsystem to be simplified. Based on the local events, the abstraction rules given in Theorems 5, 8,

and 9 are applied in lines 9–12. Rules of lower complexity are applied first, so halfway synthesis is followed by bisimulation and weak synthesis observation equivalence, which are implemented according to [28] and [19], respectively. If halfway synthesis produces a new supervisor, it is added to the set $\mathcal{S}$ of supervisors. If weak synthesis observation equivalence results in a deterministic abstracted automaton, this automaton is added back into the set $\mathcal{G}$ of uncontrolled plants.

Weak synthesis observation equivalence may also result in nondeterminism, if some states in an equivalence class have successor states reached by the same event, but belonging to different equivalence classes. In this case, a renaming is introduced. The $\mathrm{makeDistinguisher}()$ method in line 16 replaces the events of any transitions causing nondeterminism in the abstracted automaton $\tilde{A}$ by new events and records the target states of these transitions. Using the recorded target states, the same modification to the corresponding transitions is applied to the original automaton A. The $\mathrm{makeDistinguisher}()$ method returns a renaming map ρ_D, the deterministic abstracted automaton $\tilde{D}$, and an appropriate distinguisher D. In line 17, the inverse renaming ρ_D^{-1} is applied to the entire system $\mathcal{G}$ and the collected supervisors $\mathcal{S}$, the abstracted automaton $\tilde{D}$ and the distinguisher D are added to the resultant automata sets, and the renaming ρ is updated to include ρ_D. This is equivalent to the application of Theorem 6 followed by Theorem 9.

The loop terminates when the set $\mathcal{G}$ of uncontrolled plants contains only a single automaton, which is passed to standard synthesis [1] in line 20. According to Theorem 4, the result is added to the set $\mathcal{S}$, which in combination with the final renaming ρ gives the least restrictive, controllable, and nonblocking supervisor for the original system $\mathcal{G}$.

VII. Experimental Results

The compositional synthesis algorithm has been implemented in the DES software tool *Supremica* [23]. The algorithm is completely automatic and does not use any prior knowledge about the structure of the system. The implementation has successfully computed supervisors for several large discrete event systems models. The test cases include the following complex industrial models and case studies, which are taken from different application areas such as manufacturing systems and automotive body electronics:

agv Automated guided vehicle coordination based on the Petri net model in [30]. To make the example blocking in addition to uncontrollable, there is also a variant, **agvb**, with an additional zone added at the input station.

aip Automated manufacturing system of the Atelier Inter-établissement de Productique [31].

fencaiwon09 Model of a production cell in a metal-processing plant from [32].

tbed Model of a toy railroad system based on [33]. Two versions present different control objectives.

verriegel Models of the central locking system of a BMW car. There are two variants, a three-door model **verriegel3**, and a four-door model **verriegel4**. These models are derived from the KorSys project [34].

6link Models of a cluster tool for wafer processing previously studied for synthesis in [5].

tline Parametrised model of a manufacturing transfer line [2] with different numbers of serially connected cells.

All the test cases considered have at least 10^7 reachable states in their synchronous composition and are either uncontrollable, blocking, or both. Algorithm 1 has been used to compute supervisors for each of these models. The algorithm is controlled by a state limit of 5000 states: if the synchronous composition of a subsystem in line 7 of Algorithm 1 exceeds 5000 states, that subsystem is discarded and another subsystem is chosen instead. All experiments have been run on a standard desktop PC using a single 2.66 GHz microprocessor.

The results of the experiments are shown in Table I. For each model, the table shows the number of automata (Aut), the number of reachable states (Size), and whether the model is nonblocking (Nonb.) or controllable (Cont.). Next, the table shows the size of the largest synchronous composition encountered during abstraction (Peak States), the total runtime (Time), the total amount of memory used (Mem.), the number of modular supervisors computed (Num.), and the number of states of the largest supervisor automaton (Largest). The table furthermore shows the number of events replaced by renaming (Ren.) and the number of events removed by selfloop removal (SR), and finally the number of states removed by halfway synthesis (HS), bisimulation (Bis.), and weak synthesis observation equivalence (WSOE).

All examples have been solved successfully in a few seconds or minutes, never using more than 1 GB of memory.

To select a subsystem in line 5 of Algorithm 1, a strategy known as **MustL** [11] is used, which facilitates the exploitation of local events. For each event σ, a subsystem is formed by considering all automata with σ in the alphabet, so σ becomes a local event after composing the subsystem. This gives several candidate subsystems, one for each event, so a second step applies a strategy called **MinSync**, which chooses the subsystem with the smallest number of states in its synchronous composition. It is worth mentioning that other methods [11], [35] for selecting subsystems give smaller supervisors for the **agv**, **tbed**, and **tline** examples. However, persistently good results can be achieved for all the examples in this test with the **MustL/MinSync** strategy.

Fig. 14 shows some data concerning the performance of the abstraction rules. For each example, it shows the ratio of the number of states removed by each rule over the total number of states removed, and the ratio of the runtime consumed by each rule over the total runtime of all abstraction rules. The **tline** bars show the average of these ratios for models with 100–1000 cells. Particularly for large models, halfway synthesis and bisimulation run much faster than weak synthesis observation equivalence, as is expected from the higher complexity class. However, weak synthesis observation equivalence also has the highest percentage of states removed and typically contributes most of the states removed by abstraction. The data suggests a correlation between the percentage of runtime and the percentage of states removed by each rule. By this measure, the three abstraction rules have similar performance in practice.

Fig. 15 shows the runtimes and supervisor sizes for in-

TABLE I
EXPERIMENTAL RESULTS

Model	Aut	Size	Nonb.	Cont.	Peak States	Time [s]	Mem. [MB]	Supervisor Num.	Supervisor Largest	Events Ren.	Events SR	Abstraction HS	Abstraction Bis.	Abstraction WSOE
agv	16	$2.6{\cdot}10^{7}$	true	false	856	3.11	27.9	6	12339	0	30	208	0	671
agvb	17	$2.3{\cdot}10^{7}$	false	false	562	0.81	61.3	7	9380	0	30	187	0	464
aip0alps	35	$3.0{\cdot}10^{8}$	false	true	502	0.43	84.3	3	17	2	53	3	8	576
fencaiwon09b	29	$8.9{\cdot}10^{7}$	false	true	182	0.27	118.4	6	917	4	56	57	3	328
fencaiwon09s	29	$2.9{\cdot}10^{8}$	false	false	525	0.44	150.2	11	436	5	59	186	2	500
tbed-noderailb	84	$3.1{\cdot}10^{12}$	false	true	4989	6.22	265.2	17	4982	0	12	158	112	1086
tbed-uncont	84	$3.6{\cdot}10^{12}$	true	false	4479	5.34	491.6	10	19737	1	1	190	73	189
verriegel3b	52	$1.3{\cdot}10^{9}$	false	true	1367	1.80	218.2	1	4	77	64	1	390	1796
verriegel4b	64	$6.2{\cdot}10^{10}$	false	true	1382	4.86	250.5	1	4	21	71	189	622	950
6linka	53	$2.4{\cdot}10^{14}$	false	true	3614	19.52	515.3	13	2073	15	48	1754	0	2103
6linki	53	$2.7{\cdot}10^{14}$	false	true	2925	13.72	635.4	12	4017	12	49	1205	0	1897
6linkp	48	$4.2{\cdot}10^{14}$	false	true	3614	26.62	538.3	17	2073	25	45	1731	0	2107
6linkre	59	$6.2{\cdot}10^{14}$	false	true	240	1.01	584.9	19	375	10	51	221	0	279
tline100	401	$6.5{\cdot}10^{150}$	true	false	50	3.44	252.4	201	79	0	495	1192	0	4126
tline1000	4001	$2.8{\cdot}10^{1505}$	true	false	50	336.46	864.1	2001	79	0	4995	11992	0	41926

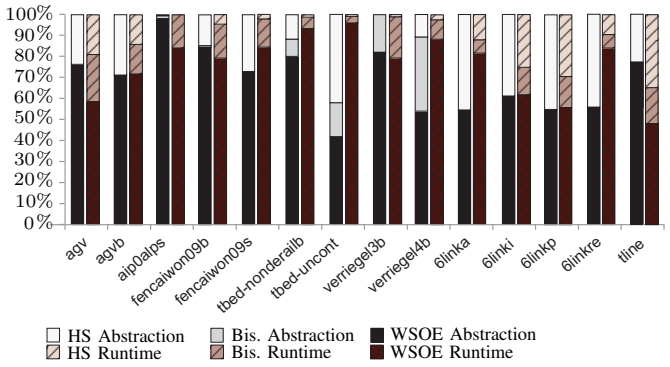

Fig. 14. Share of states removed and runtime for different abstraction rules.

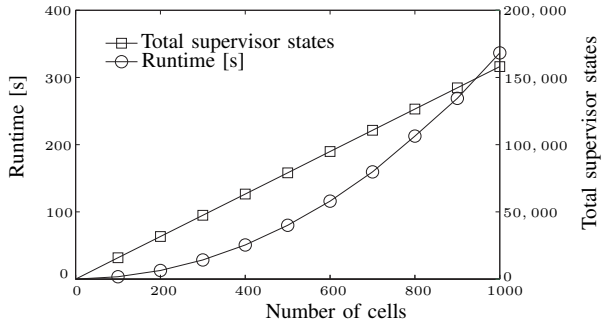

Fig. 15. Experimental results for transfer line example.

stances of the *transfer line* example [2] with 100–1000 serially connected cells. Although the state space for these models grows exponentially, the cells are identical and the practical complexity of the system is small. Even with no knowledge of the symmetry of the model, the compositional synthesis algorithm successfully computes modular supervisors for transfer lines with up to 1000 serially connected cells. Table I shows that the algorithm never constructs a supervisor component with more than 79 states. Fig. 15 shows a linear relation between the number of connected cells and the total number of supervisor states. Moreover, the relation between the number of cells and the execution time is quadratic. This behaviour is due to the complexity of evaluating and choosing subsystems from growing lists. This experiment shows that the compositional synthesis algorithm automatically discovers that the cells are identical and produces identical supervisors accordingly.

VIII. CONCLUSIONS

A general framework for compositional synthesis in supervisory control has been presented, which supports the synthesis of least restrictive, controllable, and nonblocking supervisors for large models consisting of several automata that synchronise in lock-step synchronisation. The framework supports compositional reasoning using different kinds of abstractions that are guaranteed to preserve the final synthesis result, even when applied to individual components. Nondeterminism is avoided by renaming, which solves problems in previous related work. The computed supervisor has a modular structure in that it consists of several interacting components, which makes it easy to understand and implement. The algorithm has been implemented, and experimental results show that the method successfully computes nonblocking modular supervisors for a set of large industrial models.

In future work, the authors would like to extend the compositional synthesis algorithm to use the symmetric structure of parametrised system automatically in such a way that an abstraction computed for a single module can be reused. Furthermore, finite-state machines augmented with bounded discrete variables show good modelling potential, and it is of interest to adapt the described compositional synthesis approach to work directly with this type of modelling formalism.

REFERENCES

[1] 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.

[2] W. M. Wonham, "Supervisory control of discrete-event systems," 2007. [Online]. Available: http://www.control.utoronto.edu/

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

[4] 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*, 2007, pp. 25–34.

[5] R. Su, J. H. van Schuppen, and J. E. Rooda, "Aggregative synthesis of distributed supervisors based on automaton abstraction," *IEEE Trans. Autom. Control*, vol. 55, no. 7, pp. 1267–1640, July 2010.
[6] R. Song and R. J. Leduc, "Symbolic synthesis and verification of hierarchical interface-based supervisory control," in *Proc. 8th Int. Workshop on Discrete Event Systems, WODES '06*. IEEE, July 2006, pp. 419–426.
[7] K. Åkesson, H. Flordal, and M. Fabian, "Exploiting modularity for synthesis and verification of supervisors," in *Proc. 15th IFAC World Congress on Automatic Control*, 2002.
[8] B. A. Brandin, R. Malik, and P. Malik, "Incremental verification and synthesis of discrete-event systems guided by counter-examples," *IEEE Trans. Control Syst. Technol.*, vol. 12, no. 3, pp. 387–401, May 2004.
[9] R. Su and W. M. Wonham, "Supervisor reduction for discrete-event systems," *Discrete Event Dyn. Syst.*, vol. 14, no. 1, pp. 31–53, Jan. 2004.
[10] K. Cai and W. M. Wonham, "Supervisor localization: A top-down approach to distributed control of discrete-event systems," *IEEE Trans. Autom. Control*, vol. 55, no. 3, pp. 605–618, Mar. 2010.
[11] H. Flordal and R. Malik, "Compositional verification in supervisory control," *SIAM J. Control Optim.*, vol. 48, no. 3, pp. 1914–1938, 2009.
[12] L. Feng and W. M. Wonham, "Supervisory control architecture for discrete-event systems," *IEEE Trans. Autom. Control*, vol. 53, no. 6, pp. 1449–1461, July 2008.
[13] K. Schmidt and C. Breindl, "Maximally permissive hierarchical control of decentralized discrete event systems," *IEEE Trans. Autom. Control*, vol. 56, no. 4, pp. 723–737, Apr. 2011.
[14] H. Flordal, R. Malik, M. Fabian, and K. Åkesson, "Compositional synthesis of maximally permissive supervisors using supervision equivalence," *Discrete Event Dyn. Syst.*, vol. 17, no. 4, pp. 475–504, 2007.
[15] R. Malik and H. Flordal, "Yet another approach to compositional synthesis of discrete event systems," in *Proc. 9th Int. Workshop on Discrete Event Systems, WODES '08*. IEEE, May 2008, pp. 16–21.
[16] S. Mohajerani, R. Malik, and M. Fabian, "Nondeterminism avoidance in compositional synthesis of discrete event systems," in *Proc. 7th Int. Conf. Automation Science and Engineering, CASE 2011*, 2011, pp. 19–24.
[17] S. Mohajerani, R. Malik, S. Ware, and M. Fabian, "Compositional synthesis of discrete event systems using synthesis abstraction," in *Proc. 23rd Chinese Control and Decision Conf., CCDC 2011*, 2011, pp. 1549–1554.
[18] ——, "On the use of observation equivalence in synthesis abstraction," in *Proc. 3rd IFAC Workshop on Dependable Control of Discrete Systems, DCDS 2011*, 2011, pp. 84–89.
[19] S. Mohajerani, R. Malik, and M. Fabian, "An algorithm for weak synthesis observation equivalence for compositional supervisor synthesis," in *Proc. 11th Int. Workshop on Discrete Event Systems, WODES '12*. IFAC, Oct. 2012, pp. 239–244.
[20] G. Bouzon, M. H. de Queiroz, and J. E. R. Cury, "Exploiting distinguishing sensors in supervisory control of DES," in *Proc. 7th IEEE Int. Conf. Control and Automation, ICCA '09*, Dec. 2009, pp. 442–447.
[21] K. C. Wong and W. M. Wonham, "On the computation of observers in discrete-event systems," *Discrete Event Dyn. Syst.*, vol. 14, no. 1, pp. 55–107, 2004.
[22] K. Schmidt and T. Moor, "Marked-string accepting observers for the hierarchical and decentralized control of discrete event systems," in *Proc. 8th Int. Workshop on Discrete Event Systems, WODES '06*. IEEE, July 2006, pp. 413–418.
[23] K. Åkesson, M. Fabian, H. Flordal, and R. Malik, "Supremica—an integrated environment for verification, synthesis and simulation of discrete event systems," in *Proc. 8th Int. Workshop on Discrete Event Systems, WODES '06*. IEEE, July 2006, pp. 384–385.
[24] S. Mohajerani, R. Malik, and M. Fabian, "Synthesis equivalence of triples," Working Paper 04/2012, Dept. of Computer Science, University of Waikato, Hamilton, New Zealand, 2012. [Online]. Available: http://hdl.handle.net/10289/7162
[25] C. A. R. Hoare, *Communicating Sequential Processes*. Prentice-Hall, 1985.
[26] M. Fabian, "On object oriented nondeterministic supervisory control," Ph.D. dissertation, Chalmers University of Technology, Göteborg, Sweden, 1995. [Online]. Available: https://publications.lib.chalmers.se/cpl/record/index.xsql?pubid=1126
[27] R. Milner, *Communication and concurrency*, ser. Series in Computer Science. Prentice-Hall, 1989.
[28] J.-C. Fernandez, "An implementation of an efficient algorithm for bisimulation equivalence," *Sci. Comput. Programming*, vol. 13, pp. 219–236, 1990.
[29] S. D. Brookes and W. C. Rounds, "Behavioural equivalence relations induced by programming logics," in *Proc. 16th Int. Colloquium on Automata, Languages, and Programming, ICALP '83*, ser. LNCS, vol. 154. Springer, 1983, pp. 97–108.
[30] J. O. Moody and P. J. Antsaklis, *Supervisory Control of Discrete Event Systems Using Petri Nets*. Kluwer, 1998.
[31] B. Brandin and F. Charbonnier, "The supervisory control of the automated manufacturing system of the AIP," in *Proc. Rensselaer's 4th Int. Conf. Computer Integrated Manufacturing and Automation Technology*. IEEE Computer Society Press, 1994, pp. 319–324.
[32] L. Feng, K. Cai, and W. M. Wonham, "A structural approach to the non-blocking supervisory control of discrete-event systems," *Int. J. Adv. Manuf. Technol.*, vol. 41, pp. 1152–1168, 2009.
[33] R. J. Leduc, "PLC implementation of a DES supervisor for a manufacturing testbed: An implementation perspective," Master's thesis, Dept. of Electrical Engineering, University of Toronto, ON, Canada, 1996. [Online]. Available: http://www.cas.mcmaster.ca/~leduc
[34] KORSYS Project. [Online]. Available: http://www4.in.tum.de/proj/korsys/
[35] R. Francis, "An implementation of a compositional approach for verifying generalised nonblocking," Working Paper 04/2011, Dept. of Computer Science, University of Waikato, Hamilton, New Zealand, 2011. [Online]. Available: http://hdl.handle.net/10289/5312

Sahar Mohajerani received the B.Tech. degree in electrical-control engineering from Khaje Nasir Toosi University of Technology, Tehran, Iran in 2005 and the M.S. degree in systems, control, and mechatronics from Chalmers University of Technology, Gothenburg, Sweden in 2009. She is currently working towards the Ph.D. degree in automation with the Department of Signals and Systems at Chalmers. Her current research interests are in the area of formal methods for verification and synthesis of large discrete event systems.

Robi Malik received the M.S. and Ph.D. degree in computer science from the University of Kaiserslautern, Germany, in 1993 and 1997, respectively. From 1998 to 2002, he worked in a research and development group at Siemens Corporate Research in Munich, Germany, where he was involved in the development and application of modelling and analysis software for discrete event systems. Since 2003, he is lecturing at the Department of Computer Science at the University of Waikato in Hamilton, New Zealand. He is participating in the development of the Supremica software for modelling and analysis of discrete event systems. His current research interests are in the area of model checking and synthesis of large discrete event systems and other finite-state machine models.

Martin Fabian was born in Gothenburg, Sweden, in 1960. He received his Ph.D. degree in control engineering in 1995 from Chalmers University of Technology, Gteborg, Sweden. He is currently a Professor with the Department of Signals and Systems, Chalmers University of Technology. His research interests involve modeling and supervisory control of discrete-event systems, modular and compositional methods for complex systems, and generic architectures for flexible production systems.