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 supervisor, 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 1014 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 \( \Sigma \). For the purpose of supervisory control, this alphabet is partitioned into two disjoint subsets, the set \( \Sigma_c \) of controllable events and the set \( \Sigma_u \) 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 \( \Sigma^* = \Sigma \cup \{ \omega \} \).

\( \Sigma^* \) is the set of all finite traces of events from \( \Sigma \), including the empty trace \( \epsilon \). 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 \( s \preceq t \), if \( t = su \) for some \( u \in \Sigma^* \). For \( \Omega \subseteq \Sigma \), the natural projection \( \Pi_\Omega : \Sigma^* \to \Omega^* \) is the operation that removes from traces \( s \in \Sigma^* \) all events not in \( \Omega \).

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 \( G = (\Sigma, Q, \to, Q^0) \), where \( \Sigma \) is a finite set of events, \( Q \) is a finite set of states, \( \to \subseteq Q \times \Sigma \times Q \) is the state transition relation, and \( Q^0 \subseteq Q \) is the set of initial states. \( G \) is deterministic, if \( |Q^0| \leq 1 \), 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 \( \Sigma^* \) by letting \( x \xrightarrow{\sigma} x \) for all \( x \in Q \), and \( x \xrightarrow{\epsilon} z \) if \( y \xrightarrow{\sigma} y \) and \( y \xrightarrow{\epsilon} z \) for some \( y \in Q \). Furthermore, \( x \xrightarrow{\sigma} \) means that \( x \xrightarrow{\sigma} y \) for some \( y \in Q \), and \( x \xrightarrow{} \) means that \( x \xrightarrow{\sigma} y \) for some \( s \in \Sigma^* \), and \( x \xrightarrow{} \) does not hold. These notations also apply to state sets, \( X \xrightarrow{} \) for \( X \subseteq Q \) means that \( X \xrightarrow{} \) for some \( x \in X \), and to automata, \( G \xrightarrow{} \) means that \( Q^0 \xrightarrow{} \), etc. The language of an automaton \( G \) is \( L(G) = \{ s \in \Sigma^* | G \xrightarrow{} s \} \).

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 \( \omega \) do not have any outgoing transitions, i.e., if \( x \xrightarrow{} y \) then there does not exist \( \sigma \in \Sigma \) 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 \( Q^e = \{ x \in Q | x \xrightarrow{} \} \) in this notation. For graphical simplicity, states in \( Q^e \) are shown shaded in the figures of this paper instead of explicitly showing \( \omega \)-transitions.

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

Definition 2: Let \( G_1 = (\Sigma_1, Q_1, \to_1, Q^0_1) \) and \( G_2 = (\Sigma_2, Q_2, \to_2, Q^0_2) \) be two automata. The synchronous composition of \( G_1 \) and \( G_2 \) is defined as

\[
G_1 \parallel G_2 = (\Sigma_1 \cup \Sigma_2, Q_1 \times Q_2, \to_{12}, Q^0_1 \times Q^0_2)
\]  

(1)

where

\[
(x_1, x_2) \xrightarrow{\sigma} (y_1, y_2)
\]  

if \( \sigma \in (\Sigma_1 \cap \Sigma_2) \cup \{ \omega \} \),

\[
x_1 \xrightarrow{\sigma} y_1, \text{ and } x_2 \xrightarrow{\sigma} y_2;
\]

\[
(x_1, x_2) \xrightarrow{\epsilon} (y_1, y_2)
\]  

if \( \sigma \in \Sigma_1 \setminus \Sigma_2 \) and \( x_1 \xrightarrow{\sigma} y_1 \);

\[
(x_1, x_2) \xrightarrow{\epsilon} (y_1, y_2)
\]  

if \( \sigma \in \Sigma_2 \setminus \Sigma_1 \) and \( x_2 \xrightarrow{\sigma} y_2 \).

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 \( [x] = \{ x' \in Q | x \sim x' \} \), and \( X/\sim = \{ [x] | x \in X \} \) is the set of all equivalence classes modulo \( \sim \).

Definition 4: Let \( G = (\Sigma, Q, \to, Q^0) \) be an automaton and let \( \sim \subseteq Q \times Q \) be an equivalence relation. The quotient automaton of \( G \) modulo \( \sim \) is

\[
G/\sim = (\Sigma, Q/\sim, \to/\sim, Q^0/\sim),
\]

(2)

where \( \to/\sim = \{ ([x], [\sigma], [y]) | x \xrightarrow{\sigma} y \} \) and \( Q^0/\sim = \{ [x^0] | x^0 \in Q^0 \} \).
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 $\Sigma$, and let $\Gamma \subseteq \Sigma$. Then $K$ is said to be $\Gamma$-controllable with respect to $G$ if, for every state $x \in Q$, every state $x$ of $K$, and every event $\gamma \in \Gamma$ such that $K \xrightarrow{} x$ and $G \xrightarrow{\gamma}$, it holds that $x \xrightarrow{\gamma}$ in $K$.

When $\Gamma = \Sigma_u$ then $K$ is simply said to be controllable with respect to $G$.

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

For a deterministic plant $G$, it is well-known [1] that there exists a supremal controllable and nonblocking sublanguage of $L(G)$, 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] $G_1 = (\Sigma, Q_1, \rightarrow_1, Q^*_1)$ is a subautomaton of $G_2 = (\Sigma, Q_2, \rightarrow_2, Q^*_2)$, written $G_1 \subseteq G_2$, if $Q_1 \subseteq Q_2$, $\rightarrow_1 \subseteq \rightarrow_2$, and $Q^*_1 \subseteq Q^*_2$.

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

$$\text{supC}_\Gamma(G) = \sup \{ G' \subseteq G \mid G' \text{ is } \Gamma\text{-controllable with respect to } G \text{ and nonblocking} \}$$

Again, the subscript $\Gamma$ is omitted when $\Gamma = \Sigma_u$, i.e., $\text{supC}(G) = \text{supC}_{\Sigma_u}(G)$.

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, $\text{supC}(G)$ 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 $\Phi: \Sigma^* \rightarrow 2^\Sigma$ that assigns to each trace $s \in \Sigma^*$ a control decision $\Phi(s)$ such that $\Sigma_u \subseteq \Phi(s) \subseteq \Sigma$, 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

$$\Phi_S(s) = \Sigma_u \cup \{ \sigma \in \Sigma_c \mid s\sigma \in L(S) \}$$

If $S = \text{supC}(G)$, 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, $S = \{ S_1, \ldots, S_n \}$, consisting of a set of supervisor automata. The combined global supervisor can be constructed by applying the formal definition of synchronous composition,

$$||S|| = \bigcup_{i = 1}^n S_i$$

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 $\text{supC}$ 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 $K = (\Sigma, Q, \rightarrow, Q^*)$ be a specification. The complete plant automaton $K^\perp$ for $K$ is

$$K^\perp = (\Sigma, Q \cup \{ \bot \}, \rightarrow^\perp, Q^*)$$

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

$$\rightarrow^\perp = \rightarrow \cup \{ (x, \bot) \mid x \notin \Sigma_u \} \tag{7}$$

for some $s \in \Sigma^*$.

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 $\text{supC}(G \parallel K^\perp)$ [14].

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 $B_1$ and $B_2$.
and $B_2$ that can store one workpiece each. The first machine $M_1$ takes workpieces from outside the system (event $s_{11}$), processes them, and puts them into $B_1$ (event $f_1$). $M_1$ also takes workpieces from $B_2$ (event $s_{12}$), processes them, and outputs them from the system (event $l_1$). 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 $s_{us}$) and resume (event $s_{res}$) 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 $B_1'$ and $B_2'$ (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 $G_0 = \{W_1, W_2, M_1, M_2, B_1', B_2'\}$. In the first step of compositional synthesis, individual automata are abstracted if possible. Events $s_{us}$ and $s_{res}$ 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 $\hat{W}_1$ shown in Fig. 3. Automaton $\hat{W}_1$ is a self-loop-only automaton that always enables all its events, so it can be disregarded in the synthesis.

Similarly, events $s_{us}$ and $s_{res}$ 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 nondeterministic automaton $\hat{W}_2$ 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 $s_{res}$, and it is not straightforward to construct such a supervisor only from the abstraction $\hat{W}_2$.

To solve the nondeterminism problem, event $s_{res}$ in $\hat{W}_2$ is replaced by two new events $s_{21}$ and $s_{22}$. This procedure is referred to as renaming. Automaton $\hat{W}_2$ 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 $S$ of collected supervisors. It is the first component of the supervisor to be computed in the end.

Having replaced $s_{res}$ in $W_2$, automata $M_2$ and $B_2'$ need to be modified to use the new events $s_{21}$ and $s_{22}$. Therefore, $M_2$ and $B_2'$ are replaced by $M_2'$ and $B_2'$ shown in Fig. 3. These automata are constructed by replacing the $s_{21}$-transitions in $M_2$ and $B_2'$ by transitions labelled $s_{21}$ and $s_{22}$.

After this, events $s_{us}$ and $s_{res}$ 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 $l_0$ and $s_{11}$ 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 $q_1$ cannot be merged since $q_1$ can be a blocking state if $f_1$ is disabled by other components. Fig. 4 shows the abstracted automaton $\hat{M}_1$. Furthermore, event $l_0$ now only appears in a selfloop in the entire system and thus, the selfloop associated with this event can be removed from $\hat{M}_1$, resulting in the abstracted automaton $M_1'$ shown in Fig. 4.

At this point, the system has been simplified to $G = \{W_2'', M_1', M_2', B_1', B_2'\}$. None of these automata can be simplified further, so the next step is to compose some of them. Fig. 5
shows the composition of $M'_1$ and $B'_1$, which causes $\|f_1$ to become a local event. Clearly, the blocking state $\perp$ in $M'_1 \parallel B'_1$ 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 $MB''_1$ shown in Fig. 5. This is a special case of halfway synthesis [14], explained in more detail in Sect. V-B. The abstracted automaton $MB''_1$ is added to the set $S$ 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 $MB''_1$ 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 selfloops 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 $M'_2 \parallel B'_2$. Exploiting the local event $!f_2$ results in the abstracted automata $MB''_2$, $MB'_2$, and $MB'_3$ shown in Fig. 6.

After all these abstractions, the uncontrolled plant model is $\tilde{G} = \{W''_1, MB'_1, MB'_2\}$, and the collected supervisor set is $S = \{D, MB''_1, MB''_2\}$. The last two steps are to compose the automata in $\tilde{G}$, 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 $S$ results in the nonblocking modular supervisor

$$ S = \{D, MB''_1, MB''_2, S_1\}, $$

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 $G$. The largest component of the modular supervisor is $S_1$ with 4 states, and it has been computed by exploring the state space of $\tilde{G}$ 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,

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

(9)

The objective is to calculate a supervisor that constrains the behaviour of $G$ to its least restrictive nonblocking sub-behaviour, 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 $\Upsilon$, 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 $S$, 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 $S$, and the renaming is modified. Eventually, only one plant automaton is left, which is removed from $G$ and used to calculate the final supervisor to be added to $S$. Then $G$ becomes empty and the collected supervisors $S$, together with the renaming $\rho$, 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 $\tilde{G}$ to the events in the original plant, which works in the reverse direction compared to [21].

Definition 9: Let $\Sigma_1$ and $\Sigma_2$ be two sets of events. A renaming $\rho: \Sigma_2 \to \Sigma_1$ is a controllability-preserving map, i.e., a map such that $\rho(\sigma)$ is controllable if and only if $\sigma$ 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 $\rho$ is such that $\rho(s_{21}) = \rho(s_{22}) = s_2$ and $\rho(\sigma) = \sigma$ for all other events.

The definition of $\rho$ 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 $\Sigma_2$ by replacing all transitions $x \rightarrow y$ with $x \rho(\sigma) \rightarrow y$.

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 $G = (\Sigma, Q, \rightarrow, Q^0)$ differentiates event $\gamma_1$ from $\gamma_2$, if $\gamma_1 \notin \Sigma$ and $\gamma_2 \in \Sigma$ or there exists a transition $x \rightarrow y$ such that $x \delta_0 y$ does not hold.

**Definition 11:** Let $\rho: \Sigma_2 \rightarrow \Sigma_1$ be a renaming. An automaton $G_2$ with alphabet $\Sigma_2$ is a $\rho$-distinguishor if, for all traces $s, t \in L(G_2)$ 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 $\rho$-distinguishor 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 $G = \{G_1, \ldots, G_n\}$, the remaining automata $G_j$ with $j \neq i$ and the collected supervisors, $S$, need to be modified to use the new events.

**Definition 12:** Let $G = (\Sigma_1, Q, \rightarrow, Q^0)$ be an automaton, and let $\rho: \Sigma_2 \rightarrow \Sigma_1$ be a renaming. Then $\rho^{-1}(G) = (\Sigma_2, Q, \rho^{-1}(\rightarrow), Q^0)$ where $\rho^{-1}(\rightarrow) = \{(x, \sigma, y) \mid x \rho(\sigma) \rightarrow y\}$.

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 $B_1' = \rho^{-1}(B_1^2)$, which replace the original plants $M_2$ and $B_1^2$ 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 $\rho^{-1}$.

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

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

**C. Synthesis Triples**

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

- a set $G = \{G_1, \ldots, G_n\}$ of uncontrolled plant automata;
- a set $S = \{S_1, \ldots, S_m\}$ of collected supervisor automata;
- a renaming $\rho$ that maps the events of the automata in $G \cup S$ 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 $(G; S; \rho)$, where $G$ and $S$ are sets of deterministic automata and $\rho$ is a renaming, such that

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

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

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

---

**Fig. 8. Control architecture.** The graph shows the control architecture for a system with two supervisors. The original plant $G$ is composed of two supervisors $S_D$ and $S \setminus S_D$, with $\rho^{-1}$ and $\rho$ denoting the renaming and the inverse renaming, respectively. The control decision $\Phi'$ is made based on the renamed events $\{\gamma_1, \gamma_2\}$, which are then translated back to the original events $\{\gamma_1, \gamma_2\}$ using the inverse renaming $\rho^{-1}$. The supervisor $S_D$ enables only one of $\gamma_1$ or $\gamma_2$, and this decision is then used to update the states of the remaining supervisors.
supervisors is required to have the distinguishing property (ii) to ensure that it can be used with the control architecture in Fig. 8. Furthermore, if two events \( \gamma_1 \) and \( \gamma_2 \) 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 \( (\mathcal{G}; S; \rho) \) be a synthesis triple. Then

\[
\text{sup} \mathcal{C}(\mathcal{G}; S; \rho) = \rho(\text{sup} \mathcal{C}(\mathcal{G}) \parallel S) .
\]

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

**Example 1:** At the final step of the compositional synthesis in Sect. III, the abstracted uncontrolled system is \( \mathcal{G} = \{ W_{1}', MB_1', MB_2', \} \), the collected supervisor set is \( S = \{ D, MB_1^2, MB_2^2 \} \), and the renaming \( \rho \) is such that \( \rho(s_{21}) = s_2 \) and \( \rho(\sigma) = \sigma \) for \( \sigma \neq s_{21}, s_{22} \). This is represented by the synthesis triple \( (\mathcal{G}; S; \rho) = \{(W_{1}', MB_1', MB_2'); \{D, MB_1^2, MB_2^2 \}; \rho) \). The synthesis result for this triple is obtained by calculating a monolithic supervisor for the abstracted uncontrolled plant, \( S_1 = \text{sup} \mathcal{C}(W_{1}' \parallel MB_1' \parallel MB_2') \), which is added to the supervisor set, \( S \), and afterwards all components are renamed back. This gives \( \text{sup} \mathcal{C}(\mathcal{G}; S; \rho) = \rho(S_1 \parallel D \parallel MB_1^2 \parallel MB_2^2) \). 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 restrict extension of the original control problem. Every abstractions step must ensure that the synthesis result is the same as it would have been for the non-abstraction components. This property is called synthesis equivalence [16].

**Definition 15:** Two triples \( (\mathcal{G}_1; S_1; \rho_1) \) and \( (\mathcal{G}_2; S_2; \rho_2) \) are synthesis equivalent, \( (\mathcal{G}_1; S_1; \rho_1) \simeq \text{synth} (\mathcal{G}_2; S_2; \rho_2) \), if

\[
\mathcal{L}(\{\text{sup} \mathcal{C}(\mathcal{G}_1; S_1; \rho_1)\}) = \mathcal{L}(\{\text{sup} \mathcal{C}(\mathcal{G}_2; S_2; \rho_2)\}) .
\]

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 \( (\mathcal{G}; \mathcal{G}; \text{id}) \), where \( \text{id} : \Sigma \rightarrow \Sigma \) is the identity map, i.e., \( \text{id}(\sigma) = \sigma \) for all \( \sigma \in \Sigma \). Afterwards, the initial triple is abstracted repeatedly such that synthesis equivalence is preserved.

\[
(\mathcal{G}; \mathcal{G}; \text{id}) = (\mathcal{G}_0; \mathcal{G}_0; \rho_0) \simeq \text{synth} \cdots \simeq \text{synth} (\mathcal{G}_k; \mathcal{G}_k; \rho_k) .
\]

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 \( \rho_k \) 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 \( (\mathcal{G}; \mathcal{G}; \text{id}) \simeq \text{synth} (\emptyset; \mathcal{S}; \rho) \). Then \( \mathcal{L}(\rho(S)) = \mathcal{L}(\{\text{sup} \mathcal{C}(\mathcal{G}; \mathcal{S}; \rho)\}) \).

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 same automatic. 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 \( G_1 = \{G_1, \ldots, G_n\} \) and \( G_2 = \{G_1 | G_3, \ldots, G_n\} \), let \( \rho \) be a renaming, and let \( S \) be a \( \rho \)-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 \( (\mathcal{G}; \mathcal{S}; \rho) \simeq \text{synth} (\emptyset; \mathcal{S} \cup \{\text{sup} \mathcal{C}(\mathcal{G})\}, \rho) \).

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 \( \Gamma = (\Sigma, Q, \rightarrow, Q_o) \) and \( \Gamma \subseteq \Sigma_q \). The halfway synthesis result for \( \Gamma \) with respect to \( \Gamma \) is

\[
\text{hsup} \Gamma (G) = (\Sigma, Q \cup \{\perp\}, \rightarrow_{\text{hsup}}, Q_o^\perp) ,
\]

where \( \text{sup} \mathcal{C}(\Gamma) = (\Sigma, Q \cup \{\perp\}, \rightarrow_{\text{sup}}, Q_o^\perp) \), \( \perp \notin Q \), and

\[
\rightarrow_{\text{hsup}} = \rightarrow_{\text{sup}} \cup \{ (x, \sigma, \perp) \mid \sigma \in \Sigma_u \Gamma \}, \text{ and } x \sigma^\perp \text{ does not hold} \} .
\]

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 \( \perp \) instead.

**Example 2:** Consider automaton \( G \) in Fig. 9 with \( \Sigma_u = \{\lambda, \mu, \nu\} \) and \( \Upsilon = \{\gamma, 1\lambda\} \). State \( q_3 \) is blocking, so \( q_2 \) is
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 \(q_1 \xrightarrow{\nu} q_3\) and \(q_1 \xrightarrow{\mu} q_2\). State \(q_0\) may still be safe, because some other plant component may disable the shared events \(\nu\) and \(\mu\). The blocking state \(\bot\) is added and the \(\mu\)- and \(\nu\)-transitions are redirected to \(\bot\) in the halfway synthesis result \(\hsup\), see Fig. 9. This ensures that later synthesis is aware of the potential problem regarding \(\mu\) or \(\nu\).

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 \((G; \Sigma; \rho)\) be a synthesis triple with \(G = \{G_1, \ldots, G_n\}\), and let \(\Upsilon \subseteq \Sigma\) such that \((\Sigma_2 \cup \cdots \cup \Sigma_n) \cap \Upsilon = \emptyset\). Then \((G; \Sigma; \rho) \simeq_{\text{synth}} \{\hsup\}_{\Upsilon \subseteq \Sigma}(G_1, G_2, \ldots, G_n)\; \{\hsup\}_{\Upsilon \subseteq \Sigma}(G_1); \{\hsup\}_{\Upsilon \subseteq \Sigma}(G_1)\; \Upsilon \subseteq \Sigma; \rho\).

**Complexity:** Halfway synthesis can be achieved using a standard synthesis algorithm [1] and runs in time complexity \(O(|Q| \rightarrow |\rightarrow|)|\), where \(|Q|\) and \(|\rightarrow|\) 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 \(\rho\) such that \(\rho(H_1) = G_1\) and \(H_1\) is a \(\rho\)-distinguisher. Then \(H_1\) is added to the set \(\Sigma\) of supervisors as a distinguisher, and the renaming \(\rho\) is composed with the previous renamings.

**Theorem 6:** Let \((G_1; \Sigma; \rho_1)\) be a synthesis triple with \(G_1 = \{G_1, \ldots, G_n\}\), let \(\rho\) be a renaming, and let \(H_1\) be a \(\rho\)-distinguisher such that \(\rho(H_1) = G_1\) and \(H_2 = \{H_1, \rho^{-1}(G_2), \ldots, \rho^{-1}(G_n)\}\). Then \((G_1; \Sigma; \rho_1) \simeq_{\text{synth}} (G_2; \{H_1\} \cup \rho^{-1}(S); \rho_1 \circ \rho)\).

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 \(\lambda\) 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, \(\lambda\) can be removed from all automata. This step is called selfloop removal and formally described in Theorem 7.

**Definition 17:** An automaton \(G = (\Sigma, Q, \rightarrow, Q^c)\), 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 \(G = (\Sigma, Q, \rightarrow, Q^c)\) to \(\Sigma \subseteq \Sigma\) is \(G_{|\Sigma} = (\Sigma, Q_{|\Sigma}, \rightarrow_{|\Sigma}, Q_{|\Sigma}^c)\) where \(\rightarrow_{|\Sigma} = \{ (x, \sigma, y) \in \rightarrow | \sigma \in \Sigma \cup \{\omega\} \}\). The restriction of \(\Sigma\) is \(G_{|\Sigma} = \{G_1, \ldots, G_n|_\Sigma\}\).

**Theorem 7:** Let \((G; \Sigma; \rho)\) be a synthesis triple such that \(G\) is selfloop-only for \(\Lambda \subseteq \Sigma\). Then \((G; \Sigma; \rho) \simeq_{\text{synth}} (G_{|\Sigma \backslash \Lambda}; \Sigma; \rho)\).

**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 \(G = (\Sigma, Q, \rightarrow, Q^c)\) 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 \(\sigma \in \Sigma\), there exists \(y_2 \in Q\) such that \(x_2 \xrightarrow{\sigma} y_2\) and \(y_1 \sim y_2\).

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

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. However, better abstraction can be achieved by differentiating between local and shared events. This is the idea of the 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 \(G = (\Sigma, Q, \rightarrow, Q^c)\) be an automaton with \(\Sigma = \Omega \cup \Upsilon\). An equivalence relation \(\sim \subseteq Q \times Q\) is called an *observation equivalence* on \(G\) with respect to \(\Upsilon\), 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 \(s_1 \in S_\sigma\), then there exists \(y_2 \in Q\) and \(s_2 \in S_\sigma\) such that \(P_{\Upsilon \cup \{\omega\}}(s_1) = P_{\Upsilon \cup \{\omega\}}(s_2)\), \(x_2 \xrightarrow{\sigma} y_2\), 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 \(G'\), 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].
Example 4: Consider again the observation equivalent automata in Fig. 10, with $\Sigma_c = \{\alpha, \beta\}$ and $\Sigma_u = \{!v\}$. The triples $\{(\alpha)\}; \{G\}; \{!v\}$ and $\{(\beta)\}; \{G\}; \{!v\}$ are not synthesis equivalent. With $G$, a supervisor can disable the local controllable event $\alpha$ to prevent entering state $q_1$ and thus the occurrence of the undesirable uncontrollable $!v$, but this is not possible with $\tilde{G}$. It holds that $\omega \in \mathcal{L}(\text{sup}(G))$ while $\mathcal{L}(\text{sup}(\tilde{G})) = \emptyset$.

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 $\alpha$ and $\beta$ are uncontrollable. In fact, a result similar to Theorem 8 holds if observation equivalence is restricted to uncontrollable events [18]. With uncontrollable events, abstraction is also possible, but two other issues must be taken into account.

Example 5: Consider automaton $G$ in Fig. 11 with $\Sigma_u = \{!u, !v\}$ and $\Upsilon = \{!u, !v, !\mu\}$. 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 $\alpha$, possibly preceded and followed by the local event $!\mu$, the transition $q_4 \xrightarrow{!\mu} q_5$ must always be disabled to prevent blocking via the local uncontrollable event $!\nu$, while the transition $q_1 \xrightarrow{\alpha} q_5$ may be enabled. When used in a system that requires $\alpha$ 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 $T = \{(G, T)\}; \{G, T\}; \{!\mu\}$ and $\tilde{T} = \{(\tilde{G}, T)\}; \{G, T\}; \{!\mu\}$ are not synthesis equivalent as $\mathcal{L}(\text{sup}(T)) = \emptyset$ but $!\nu \in \mathcal{L}(\text{sup}(\tilde{T}))$.

Example 6: Consider automaton $G$ in Fig. 12 with $\Sigma_u = \{!v, !\mu\}$ and $\Upsilon = \{!\mu\}$. 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 avoided to prevent blocking in state $q_5$ via the local uncontrollable event $!\nu$. Thus, $\alpha$ should be disabled in $q_1$ and $q_2$, making $q_2$ a blocking state, while $q_1$ remains nonblocking due to the transition $q_1 \xrightarrow{!\mu} q_6$. The triples $\mathcal{T} = \{(G)\}; \{G\}; \{!\mu\}$ and $\tilde{\mathcal{T}} = \{(\tilde{G})\}; \{G\}; \{!\mu\}$ are not synthesis equivalent as $\mathcal{L}(\text{sup}(\mathcal{T})) = \emptyset$ but $!v \in \mathcal{L}(\text{sup}(\tilde{\mathcal{T}}))$. The problem in Example 5 is caused by considering the path $q_2 \xrightarrow{!\mu} q_1$ as a loop to quiesce states $q_1$ and $q_2$ to be merged. However, the path $q_2 \xrightarrow{!\mu} q_3$ passes through the unsafe state $q_6$, while $q_1 \xrightarrow{!\mu} q_6$ does not pass through any unsafe states. This situation can be avoided by only allowing local events before a controllable event. That is, for $x_1 \xrightarrow{\tau} y_1$ and $x_2 \xrightarrow{\tau} y_2$ it is required that there exists $t \in T^*$ such that $x_2 \xrightarrow{t \tau} y_2$ 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: Let $G = (\Sigma, Q, \rightarrow, Q^0)$ be an automaton with $\Sigma = \Omega \cup \Upsilon$. An equivalence relation $\sim \subseteq Q \times Q$ is a synthesis observation equivalence on $G$ with respect to $\Upsilon$, 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 $\sigma \in \Sigma_c \cup \{\omega\}$, then there exists a path $x_2 = x_2^0 \tau_2 \ldots \tau_2^k \rightarrow x_2$ such that $y_1 \xrightarrow{\omega} y_2$ and $\Upsilon, \ldots, \tau_2^k \in \Upsilon$, and if $\tau_i \in \Sigma_c$ then $x_1 \sim x_2$;

(ii) if $x_1 \xrightarrow{\gamma} y_1$ for $\gamma \in \Sigma_u$, then there exist $t_2, u_2 \in (\Upsilon \cap \Sigma_u)^*$ such that $x_2 \xrightarrow{t_2 \gamma u_2} y_2$ 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 $x_2 \xrightarrow{\gamma} x_2$, its target state $x_2$ 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 $P_3$ 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 $q_1 \sim q_2$ 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 $q_2 \xrightarrow{!\nu} q_5$ but $q_1 \xrightarrow{\alpha} q_7 \xrightarrow{!\nu} q_5$, and the local event $\beta$ occurs after the shared event $\alpha$ 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: Let $G = (\Sigma, Q, \rightarrow, Q^0)$ be an automaton with $\Sigma = \Omega \cup \Upsilon$. An equivalence relation $\sim \subseteq Q \times Q$ is
a weak synthesis observation equivalence on $G$ with respect to $\mathcal{T}$, if the following conditions hold for all $x_1, x_2 \in Q$.

(i) If $x_1 \xrightarrow{\sigma} y_1$ for $\sigma \in \Sigma_c \cup \{w\}$, then there exists a path $x_2 = x_2^0 \tau_3 \cdots \tau_3 x_2^0 \xrightarrow{P_{\Sigma_c}(\tau_3)} y_2^0 \tau_{n+1} \cdots \tau_{n} y_2^{n+1} = y_2$ such that $y_1 \sim y_2$ and $\tau_1, \ldots, \tau_m \in \mathcal{T}$ and,

a) if $\tau_i \in \Sigma_c$ for some $i \leq n$, then $x_1 \sim x_2$;

b) if $y_2^0 \xrightarrow{\sigma} z$ for some $u \in (\Sigma_u \cap \mathcal{T})^*$, then $z \sim y_2^0$ for some $n + 1 \leq j \leq m + 1$;

c) if $y_2^0 \xrightarrow{w} z$ for some $u \in \Sigma_u$ such that $P_D(u) \in \Sigma_u \setminus \mathcal{T}$, then there exists $u' \in \Sigma_u$ such that $P_D(u) = P_D(u')$ and $y_2 \xrightarrow{w'} z'$ for some $z' \sim z$.

(ii) If $x_1 \xrightarrow{v} y_1$ for $v \in \Sigma_u$, then there exist $t_2, u_2 \in (\mathcal{T} \cap \Sigma_u)^*$ such that $x_2 = x_2^0 \xrightarrow{t_2 P_D(v) u_2} y_2$ 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 $\mathcal{T} = \{\beta\}$. 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 $G$ can be avoided using a renaming $\rho: \{\alpha_1, \alpha_2, \beta, \gamma\} \rightarrow \{\alpha, \beta, \gamma\}$, 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 $(G; S; \rho)$ be a synthesis triple with $G = \{G_1, \ldots, G_n\}$ and $G_i = (\Sigma_i, Q_i, \rightarrow_i, Q^*_i)$. Let $\mathcal{T} \subseteq \Sigma$, such that $(\Sigma_2 \cup \cdots \cup \Sigma_n) \cap \mathcal{T} = \emptyset$. Let $\sim$ be a synthesis observation equivalence or a weak synthesis observation equivalence relation on $\rho(G_1)$ with respect to $\mathcal{T}$ such that $G_1/\sim$ is deterministic, and let $\tilde{G} = \{G_1/\sim, G_2, \ldots, G_n\}$. Then $(\tilde{G}; S; \rho) \simeq_{\text{synth}} G; S; \rho$.

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

VI. COMPOSITIONAL SYNTHESIS ALGORITHM

Given a set of plant automata $G$, the compositional synthesis algorithm repeatedly composes automata and applies abstraction rules. While doing so, it modifies a synthesis triple $(G; S; \rho)$, collecting supervisors in $S$ and updating the renaming $\rho$, 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 $G$ of plant automata. First, line 4 applies selfloop removal to the entire plant $G$ according to Theorem 7. This quick operation improves the performance of the following steps.

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

After identification and composition of a subsystem, the set $\mathcal{T}$ 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 $S$ of supervisors. If weak synthesis observation equivalence results in a deterministic abstracted automaton, this automaton is added back into the set $G$ of uncontrolled plants.

Weaken 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 makeDistinguisher() method in line 16 replaces the events of any transitions causing nondeterminism in the abstracted automaton $A$ 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 makeDistinguisher() method returns a renaming map $\rho_D$, the deterministic abstracted automaton $D$, and an appropriate distinguisher $D$. In line 17, the inverse renaming $\rho_D^{-1}$ is applied to the entire system $G$ and the collected supervisors $S$, the abstracted automaton $D$ and the distinguisher $D$ are added to the resultant automata sets, and the renaming $\rho$ is updated to include $\rho_D$. This is equivalent to the application of Theorem 6 followed by Theorem 9.

The loop terminates when the set $G$ 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 $S$, which in combination with the final renaming $\rho$ gives the least restrictive, controllable, and nonblocking supervisor for the original system $G$.

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 two-door model **verriegel3**, and a four-door model **verriegel4**. These models are derived from the KORSYS project [34].

6**link** 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 $10^3$ 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.), and the number of modular supervisors computed (Num.), and the number of states of the largest supervisory automaton (Largest). The table furthermore shows the number of events replaced by renaming (Ren.) and the number of events removed by self-loop 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 $\sigma$, a subsystem is formed by considering all automata with $\sigma$ in the alphabet, so $\sigma$ 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-
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 synchronize 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.

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 synchronize 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


[34] KORSYS Project. [Online]. Available: http://www4.in.tum.de/proj/korsys/


Sahar Mohajerani received the B.Tech. degree in electrical-control engineering from Khajeh 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, Göteborg, 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.