Certainly Unsupervisable States

Simon Ware[1], Robi Malik[1], Sahar Mohajerani[2], and Martin Fabian[2]

[1] Department of Computer Science,
University of Waikato, Hamilton, New Zealand
{siw4,robi}@waikato.ac.nz
[2] Department of Signals and Systems,
Chalmers University of Technology, Gothenburg, Sweden
{mohajera,fabian}@chalmers.se

Abstract. This paper proposes an abstraction method for compositional synthesis. *Synthesis* is a method to automatically compute a *control program* or *supervisor* that restricts the behaviour of a given system to ensure safety and liveness. *Compositional synthesis* uses repeated abstraction and simplification to combat the state-space explosion problem for large systems. The abstraction method proposed in this paper finds and removes the so-called *certainly unsupervisable* states. By removing these states at an early stage, the final state space can be reduced substantially. The paper describes an algorithm with cubic time complexity to compute the largest possible set of removable states. A practical example demonstrates the feasibility of the method to solve real-world problems.

1 Introduction

Reactive systems are used extensively to control safety-critical applications, where a small error can result in huge financial or human losses. With their size and complexity continuously increasing, there is an increasing demand for formal modelling and analysis. *Model checking* [4] has been used successfully to automatically detect errors in reactive systems. In some cases, it is possible to go further and *synthesise*, i.e., automatically compute a controlling agent that removes certain kinds of errors from a system.

The controller synthesis problem has been studied by several researchers in computing and control. The synthesis of a stand-alone controller from a temporal logic specification is studied in [7, 19]. Synthesis has been generalised to the extraction of an environment to interact with a given software *interface* [1], and to the construction controllers interacting with a given *environment* or *plant* [2,5]. *Supervisory control theory* [21] of discrete event systems provides a framework to synthesise a *supervisor* that restricts the behaviour of a given plant as little as possible while ensuring the safety and liveness properties of *controllability* and *nonblocking*.

Straightforward synthesis algorithms explore the complete *monolithic* state space of the 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. These problems are addressed by *compositional* methods [3,8]. If a temporal logic specification is the conjunction of several requirements, it is possible to synthesise separate controller components for each requirement [5, 7]. Compositional approaches in supervisory control [9, 16] exploit the structure of the model of the plant to be controlled, which typically consists of several interacting components. These approaches avoid constructing the full state space by first simplifying individual components, then applying synchronous composition step by step, and simplifying the intermediate results again.

This kind of compositional synthesis requires specific abstraction methods to guarantee a least restrictive, controllable, and nonblocking final synthesis result. *Supervision equivalence* [9] and *synthesis abstraction* [16] have been proposed for this purpose, and several abstraction methods to simplify automata preserving these properties are known.

This paper proposes another abstraction method that can be used in compositional synthesis frameworks such as [9, 16]. The proposed method finds all the states that will certainly be removed by any supervisor. Removing these so-called *certainly unsupervisable states* at an early stage reduces the state space substantially. Previously, *halfway synthesis* [9] was used for this purpose, which approximates the removable states. The set of certainly unsupervisable states is the largest possible set of removable states, and it can be computed in the same cubic complexity as halfway synthesis.

This paper is organised as follows. Section 2 introduces the terminology of supervisory control theory [21] and the framework of compositional synthesis [9, 16]. Next, Section 3 explains the ideas of compositional synthesis with certainly unsupervisable states using the example of a manufacturing system. Section 4 presents the results of this paper: it defines the set of certainly unsupervisable states, gives an algorithm to compute it, performs complexity analysis, and compares certainly unsupervisable states to halfway synthesis. Finally, Section 5 adds some concluding remarks.

2 Preliminaries

2.1 Events and Languages

Discrete event systems [21] are modelled 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, the 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 supervising agent, while uncontrollable events occur spontaneously. In addition, the *silent controllable* event $\tau_{\mathrm{c}} \in \Sigma_{\mathrm{c}}$ and the *silent uncontrollable* event $\tau_{\mathrm{u}} \in \Sigma_{\mathrm{u}}$ denote transitions that are not taken by any component other than the one being considered. The set of all finite *traces* of events from Σ, including the *empty trace* ε, is denoted by Σ^*. A subset $L \subseteq \Sigma^*$ is called a *language*. The *concatenation* of two traces $s, t \in \Sigma^*$ is written as st.

2.2 Nondeterministic Automata

System behaviours are typically modelled by deterministic automata, but nondeterministic automata may arise as intermediate results during abstraction.

Definition 1. A (nondeterministic) finite automaton is a tuple $G = \langle \Sigma, Q, \to, Q^\circ, Q^\omega \rangle$, where Σ is a finite set of events, Q is a finite set of *states*, $\to \subseteq Q \times (\Sigma \cup \{\tau_{\mathrm{u}}, \tau_{\mathrm{c}}\}) \times Q$

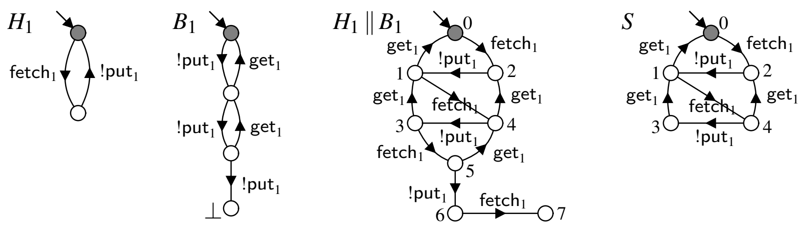

Fig. 1. Simple manufacturing system. Events fetch$_1$ and get$_1$ are controllable, while !put$_1$ is uncontrollable.

is the *state transition relation*, $Q^\circ \subseteq Q$ is the set of *initial states*, and $Q^\omega \subseteq Q$ is the set of *accepting states*.

The transition relation is written in infix notation $x \xrightarrow{\sigma} y$, and is extended to traces and languages in the standard way. For example, $x \xrightarrow{\tau_u^*\sigma} y$ means that there exists a possibly empty sequence of τ_u-transitions followed by a σ-transition that leads from state x to y. Furthermore, $x \xrightarrow{s}$ means $x \xrightarrow{s} y$ for some $y \in Q$, and $x \rightarrow y$ means $x \xrightarrow{s} y$ for some $s \in \Sigma^*$. These notations also apply to state sets and to automata: $X \xrightarrow{s} Y$ for $X, Y \subseteq Q$ means $x \xrightarrow{s} y$ for some $x \in X$ and $y \in Y$, and $G \xrightarrow{s} x$ means $Q^\circ \xrightarrow{s} x$.

Example 1. Fig. 1 shows an automata model of a simple manufacturing system consisting of a handler H_1 and a buffer B_1. The handler fetches a workpiece (fetch$_1$) and then puts it into the buffer (!put$_1$). The event !put$_1$ also increases the number of workpieces in the buffer by 1. Afterwards the buffer can release the workpiece (get$_1$), reducing the number of workpieces in the buffer by 1. The buffer can store only two workpieces, adding more workpieces causes overflow as represented by the state $\perp$.

Definition 2. Let $G_1 = \langle \Sigma_1, Q_1, \rightarrow_1, Q_1^\circ, Q_1^\omega \rangle$ and $G_2 = \langle \Sigma_2, Q_2, \rightarrow_2, Q_2^\circ, Q_2^\omega \rangle$ be two automata. The *synchronous composition* of G_1 and G_2 is

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

where

$$(x_1, x_2) \xrightarrow{\sigma} (y_1, y_2), \quad \text{if } \sigma \in (\Sigma_1 \cap \Sigma_2) \setminus \{\tau_u, \tau_c\},\ x_1 \xrightarrow{\sigma}_1 y_1, \text{ and } x_2 \xrightarrow{\sigma}_2 y_2\ ; \qquad (2)$$

$$(x_1, x_2) \xrightarrow{\sigma} (y_1, x_2), \quad \text{if } \sigma \in (\Sigma_1 \setminus \Sigma_2) \cup \{\tau_u, \tau_c\} \text{ and } x_1 \xrightarrow{\sigma}_1 y_1\ ; \qquad (3)$$

$$(x_1, x_2) \xrightarrow{\sigma} (x_1, y_2), \quad \text{if } \sigma \in (\Sigma_2 \setminus \Sigma_1) \cup \{\tau_u, \tau_c\} \text{ and } x_2 \xrightarrow{\sigma}_2 y_2\ . \qquad (4)$$

Automata are synchronised in lock-step synchronisation [11]. Shared events must be executed by all automata together, while events used by only one automaton (and the silent events τ_u and τ_c) are executed by only that automaton. Fig. 1 shows the synchronous composition $H_1 \,\|\, B_1$ of the automata mentioned in Example 1.

Another common operation in compositional synthesis is *hiding*, which removes the identity of certain events and in general produces a nondeterministic automaton.

Definition 3. Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$ be an automaton and $\Upsilon \subseteq \Sigma$. The result of *controllability preserving hiding* of Υ from G is $G \setminus_! \Upsilon = \langle \Sigma \setminus \Upsilon, Q, \rightarrow_!, Q^\circ, Q^\omega \rangle$, where $\rightarrow_!$ is obtained from $\rightarrow$ by replacing each transition $x \xrightarrow{\sigma} y$ such that $\sigma \in \Upsilon$ by $x \xrightarrow{\tau_c} y$ if $\sigma \in \Sigma_c$ or by $x \xrightarrow{\tau_u} y$ if $\sigma \in \Sigma_u$.

2.3 Supervisory Control Theory

Supervisory control theory [21] provides a means to automatically compute a so-called *supervisor* that controls a given system to perform some desired functionality. Given an automata model of the possible behaviour of a physical system, called the *plant*, a supervisor is sought to restrict the behaviour in such a way that only a certain subset of the state space is reachable. The supervisor is implemented as a *control function* [21]

$$\Phi \colon Q \rightarrow 2^{\Sigma \times Q} \tag{5}$$

that assigns to each state $x \in Q$ the set $\Phi(x)$ of transitions to be enabled in this state. That is, a transition $x \xrightarrow{\sigma} y$ with $\sigma \in \Sigma_c$ will only be possible under the control of supervisor Φ if $(\sigma, y) \in \Phi(x)$. Uncontrollable events cannot be disabled, so it is required that $\Sigma_u \times Q \subseteq \Phi(x)$ for all $x \in Q$. Controllable transitions can be disabled individually, i.e., if a nondeterministic system contains multiple outgoing controllable transitions from a state x, then the supervisor may disable some of them while leaving others enabled [9]. If the plant is modelled by a nondeterministic automaton, then such a supervisor can be represented as a *subautomaton*.

Definition 4. [9] Let $G = \langle \Sigma, Q_G, \rightarrow_G, Q_G^\circ, Q_G^\omega \rangle$ and $K = \langle \Sigma, Q_K, \rightarrow_K, Q_K^\circ, Q_K^\omega \rangle$ be two automata. K is a *subautomaton* of G, written $K \subseteq G$, if $Q_K \subseteq Q_G$, $\rightarrow_K \subseteq \rightarrow_G$, $Q_K^\circ \subseteq Q_G^\circ$, and $Q_K^\omega \subseteq Q_G^\omega$.

A subautomaton K of G contains a subset of the states and transitions of G. It represents a supervisor that enables only those transitions present in K, i.e., it implements the control function

$$\Phi_K(x) = (\Sigma_u \times Q) \cup \{\, (\sigma, y) \in \Sigma_c \times Q \mid x \xrightarrow{\sigma}_K y \,\} \ . \tag{6}$$

As uncontrollable events cannot be disabled, the control function includes all possible uncontrollable transitions. Not every subautomaton of G can be implemented through control—the property of *controllability* [21] characterises those behaviours than can be implemented.

Definition 5. [9] Let $G = \langle \Sigma, Q_G, \rightarrow_G, Q_G^\circ, Q_G^\omega \rangle$ and $K = \langle \Sigma, Q_K, \rightarrow_K, Q_K^\circ, Q_K^\omega \rangle$ such that $K \subseteq G$. Then K is called *controllable* in G if, for all states $x \in Q_K$ and $y \in Q_G$ and for every uncontrollable event $\upsilon \in \Sigma_u$ such that $x \xrightarrow{\upsilon}_G y$, it also holds that $x \xrightarrow{\upsilon}_K y$.

If a subautomaton K is controllable in G, then every uncontrollable transition possible in G is also contained in K. In Fig. 1, automaton S is controllable in $H_1 \parallel B_1$. However, if state 5 was to be included in S, then because of the uncontrollable transition

$5 \xrightarrow{!\mathsf{put}_1} 6$, state 6 would also have to be included for S to be controllable. Controllability ensures that the control function (6) can be implemented without disabling any uncontrollable events.

In addition to controllability, the supervised behaviour is typically required to be *nonblocking*.

Definition 6. [15] Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$ be an automaton. G is called *nonblocking* if for every state $x \in Q$ such that $Q^\circ \rightarrow x$ it holds that $x \rightarrow Q^\omega$.

In a nonblocking automaton, termination is possible from every reachable state. The nonblocking property, also referred to as *weak termination* [17], ensures the absence of livelocks and deadlocks. Combined with controllability, the requirement to be nonblocking can express arbitrary safety properties [9]. For example, the buffer model B_1 in Fig. 1 contains the $!\mathsf{put}_1$-transition to the blocking state $\perp$ to specify a supervised behaviour that does not allow a third workpiece to be placed into the buffer when it already contains two workpieces, i.e., it requests a supervisor that prevents buffer overflow.

Given a plant automaton G, the objective of *supervisor synthesis* [21] is to compute a subautomaton $K \subseteq G$, which is controllable and nonblocking and restricts the behaviour of G as little as possible. The set of subautomata of G forms a lattice [6], and the upper bound of a set of controllable and nonblocking subautomata in this lattice is again controllable and nonblocking.

Theorem 1. [9] Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$ be an automaton. There exists a unique subautomaton $\sup\mathscr{C}(G) \subseteq G$ such that $\sup\mathscr{C}(G)$ is nonblocking and controllable in G, and such that for every subautomaton $S \subseteq G$ that is also nonblocking and controllable in G, it holds that $S \subseteq \sup\mathscr{C}(G)$.

The subautomaton $\sup\mathscr{C}(G)$ is the unique *least restrictive* sub-behaviour of G that can be achieved by any possible supervisor. It can be computed using a fixpoint iteration [9], by iteratively removing blocking states and states leading to blocking states via uncontrollable events, until a fixpoint is reached.

Definition 7. [9] Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$ be an automaton. The *restriction* of G to $X \subseteq Q$ is $G_{|X} = \langle \Sigma, X, \rightarrow_{|X}, Q^\circ \cap X, Q^\omega \cap X \rangle$, where $\rightarrow_{|X} = \{\, (x, \sigma, y) \in \rightarrow \mid x, y \in X \,\}$.

Definition 8. [9] Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$ be an automaton. The *synthesis step* operator $\Theta_G \colon 2^Q \rightarrow 2^Q$ for G is defined as $\Theta_G(X) = \Theta_G^{\mathrm{cont}}(X) \cap \Theta_G^{\mathrm{cont}}(X)$, where

$$\Theta_G^{\mathrm{cont}}(X) = \{\, x \in X \mid \text{for all transitions } x \xrightarrow{\upsilon} y \text{ with } \upsilon \in \Sigma_{\mathrm{u}} \text{ it holds that } y \in X \,\} \; ; \quad (7)$$

$$\Theta_G^{\mathrm{nonb}}(X) = \{\, x \in X \mid x \rightarrow_{|X} Q^\omega \,\} \; . \quad (8)$$

Given a state set $X \subseteq Q$, the operator Θ_G^{cont} removes from X any states that have an uncontrollable successor not contained in X, and Θ_G^{nonb} removes any states from where it is not possible to reach an accepting state via transitions contained in X. Thus, Θ_G^{cont} captures controllability and Θ_G^{nonb} captures nonblocking. Both operators and their combination Θ_G are monotonic, and it follows by the Knaster-Tarski theorem [20] that they have greatest fixpoints. The least restrictive synthesis result $\sup\mathscr{C}(G)$ is obtained by restricting G to the greatest fixpoint of Θ_G.

Theorem 2. [9] Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$. The synthesis step operator Θ_G has a greatest fixpoint $\mathrm{gfp}\Theta_G = \hat{\Theta}_G \subseteq Q$, such that $G_{|\hat{\Theta}_G}$ is the greatest subautomaton of G that is both controllable in G and nonblocking, i.e.,

$$\sup\mathscr{C}(G) = G_{|\hat{\Theta}_G} . \tag{9}$$

Example 2. The automaton $H_1 \,\|\, B_1$ in Fig. 1 is blocking, because the trace $\mathsf{fetch}_1 \,!\mathsf{put}_1$ $\mathsf{fetch}_1 \,!\mathsf{put}_1 \mathsf{fetch}_1 \,!\mathsf{put}_1$ leads to state 6, from where no accepting state is reachable. To prevent this blocking situation, event $!\mathsf{put}_1$ needs to be disabled in state 5. However, $!\mathsf{put}_1$ is an uncontrollable event that cannot be disabled by the supervisor, so the best feasible solution is to disable the controllable event fetch_1 in state 3. Fig. 1 shows the least restrictive supervisor $S = \sup\mathscr{C}(H_1 \,\|\, B_1)$.

In the finite-state case, the state set of the least restrictive supervisor can be calculated as the limit of the sequence $X^0 = Q$, $X^{i+1} = \Theta_G(X^i)$. This iteration converges in at most $|Q|$ iterations, and the worst-case time complexity is $O(|Q||{\rightarrow}|) = O(|\Sigma||Q|^3)$, where $|\Sigma|$, $|Q|$, and $|{\rightarrow}|$ are the numbers of events, states, and transitions of the plant automaton G. However, often the behaviour the system is specified by a large number of synchronised automata, and when measured by the number of components, the synthesis problem is NP-complete [10].

2.4 Compositional Synthesis

Many discrete event systems are *modular* in that they consist of a large number of interacting components. This modularity allows to simplify individual components before composing them, in many cases avoiding state-space explosion. This idea has been used successfully for verification [8] and synthesis [9, 16] of large discrete event systems.

Given a system of concurrent plant automata

$$\mathscr{G} = G_1 \,\|\, G_2 \,\|\, \cdots \,\|\, G_n , \tag{10}$$

the objective of synthesis is to find a least restrictive supervisor, which ensures nonblocking without disabling uncontrollable events. The standard solution [21] to this problem is to calculate a finite-state representation of the synchronous composition (10) and use a synthesis iteration to calculate $\sup\mathscr{C}(\mathscr{G}) = \sup\mathscr{C}(G_1 \,\|\, \cdots \,\|\, G_n)$.

A compositional algorithm tries to find the same result without explicitly calculating the synchronous composition (10). It seeks to abstract individual automata G_i by removing some states or transitions, and replace them by abstracted versions $\tilde{G}_i$. If no more abstraction is possible, synchronous composition is computed step by step, abstracting the intermediate results again.

The individual automata G_i typically contain some events that do not appear in any other automata G_j. These events are called *local* events, denoted by the set Υ in the following. After hiding the local events, the automaton G_i is replaced by $G_i \setminus_! \Upsilon$, which increases the possibility of further abstraction.

Eventually, the procedure leads to a single automaton $\tilde{G}$, the abstract description of the system $\mathscr{G}$. After abstraction, the automaton of $\tilde{G}$ has less states and transitions

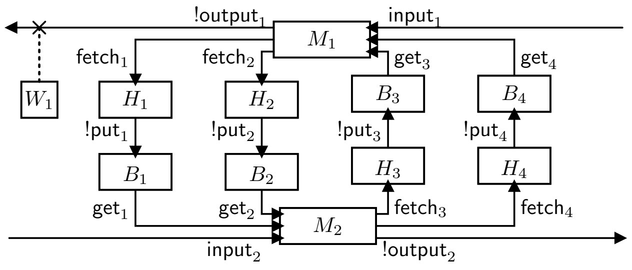

Fig. 2. Manufacturing system overview.

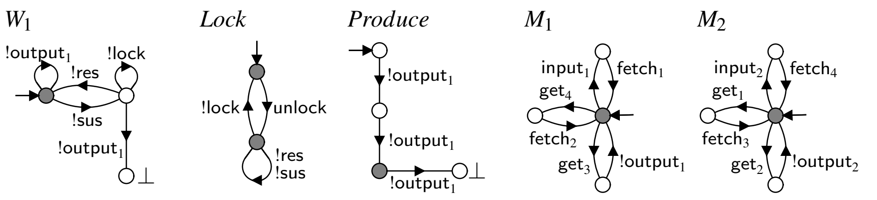

Fig. 3. Automata for manufacturing system model. Uncontrollable events are prefixed by !.

compared to (10). Once $\tilde{G}$ is found, the final step is to use it instead of the original system, to obtain a synthesis result $\sup\mathscr{C}(\tilde{G}) = \sup\mathscr{C}(\mathscr{G})$.

The abstraction steps to simplify the individual automata G_i must satisfy certain conditions to guarantee that the synthesis result obtained from the final abstraction is a correct supervisor for the original system.

Definition 9. Let G and H be two automata with alphabet Σ. Then G is *synthesis equivalent* to H, written $G \simeq_{\text{synth}} H$ if, for every automaton T, it holds that $\sup\mathscr{C}(G \parallel T) = \sup\mathscr{C}(H \parallel T)$.

Def. 9 is a special case of synthesis abstraction [16]. Synthesis equivalence requires that the abstracted automaton H yields the same supervisor as the original automaton G, no matter what the remainder of the system T is.

3 Manufacturing System Example

This section demonstrates compositional synthesis using a modified version of a manufacturing system previously studied in [13]. The manufacturing system consists of two machines (M_1 and M_2) and four pairs of handlers (H_i) and buffers (B_i) for transferring workpieces between the machines. Fig. 2 gives an overview of the system.

The manufacturing system can produce two types of workpieces. Type I workpieces are first processed by machine M_1 (input$_1$). Then they are fetched by handler H_1 (fetch$_1$) and placed into buffer B_1 (!put$_1$). Next, they are processed by M_2 (get$_1$), fetched by H_4 (fetch$_4$) and placed into B_4 (!put$_4$). Finally, they are processed by M_1 once more (get$_4$), and released (!output$_1$). Using a switch W_1, users can request to suspend (!sus) or resume (!res) production of M_1, provided that the switch has been unlocked (unlock) by

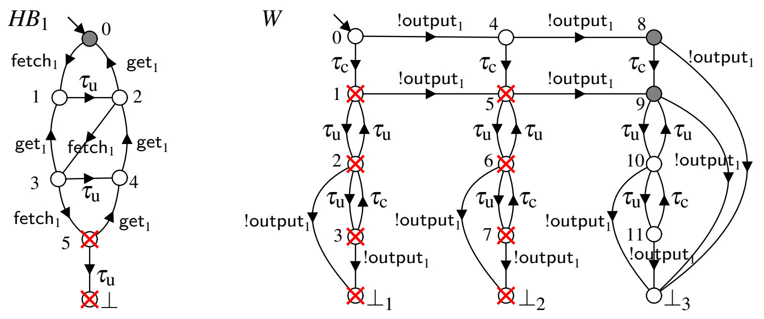

Fig. 4. Automata encountered during compositional synthesis of manufacturing system example.

the system. Type II workpieces are first processed by M_2, passed through H_3 and B_3, further processed by M_1, passed through H_2 and B_2, processed a second time by M_2, and released. The handlers and buffers are modelled as in Fig. 1, and Fig. 3 shows the rest of the automata model of the system. Automata W_1 and *Produce* use the blocking states $\perp$ to model requirements for the synthesised supervisor to prevent output from M_1 in suspend mode and to produce exactly two Type I workpieces.

In the following, compositional synthesis is used to synthesise a supervisor subject to these requirements. Initially, the system is

$$\mathscr{G} = M_1 \parallel M_2 \parallel W_1 \parallel Lock \parallel Produce \parallel H_1 \parallel B_1 \parallel \cdots \parallel H_4 \parallel B_4 \ . \tag{11}$$

In the first step, H_1 and B_1 are composed, so that event $\mathsf{!put}_1$ becomes an uncontrollable local event and can be hidden. Thus, H_1 and B_1 are replaced by $HB_1 = (H_1 \parallel B_1) \setminus_! \{\mathsf{!put}_1\}$ shown in Fig. 4, where for graphical simplicity the two blocking states from Fig. 1 are replaced by the state $\perp$. Clearly, such blocking states must be avoided, and since the silent uncontrollable transition $5 \xrightarrow{\tau_u} \perp$ cannot be disabled by the supervisor or by any plant, state 5 must also be avoided. States 5 and $\perp$ are *certainly unsupervisable states* and are crossed out in Fig. 4. Automaton HB_1 is replaced by the synthesis equivalent abstraction $\tilde{HB}_1$ with 5 states, which is obtained by deleting states 5 and $\perp$. The same abstraction is applied to the other buffers and handlers.

After composition of W_1, *Produce*, and *Lock*, events $\mathsf{!sus}$, $\mathsf{!res}$, $\mathsf{!lock}$, and unlock are local and can be hidden. Fig. 4 shows the result $W = (W_1 \parallel Produce \parallel Lock) \setminus_! \{\mathsf{!sus}, \mathsf{!res}, \mathsf{!lock}, \mathsf{unlock}\}$. Clearly, states $\perp_1$ and $\perp_2$ are blocking states. Moreover, the only way to reach an accepting state from state 1 is via the transition $1 \xrightarrow{\mathsf{!output}_1} 5$. However, $1 \xrightarrow{\tau_u} 2 \xrightarrow{\mathsf{!output}_1} \perp_1$, and since neither the supervisor nor any other plant can disable τ_u, a supervisor that enables event $\mathsf{!output}_1$ in state 1, inevitably permits the blocking state $\perp_1$. State 1 is a certainly unsupervisable state, and similar arguments hold for states 2, 3, 5, 6, and 7. Deleting these states from W results in the synthesis equivalent automaton $\tilde{W}$. Next, M_1 and $\tilde{W}$ are composed, which results in $\mathsf{!output}_1$ becoming a local event. The composed automaton, MW, has 28 states. Applying certain unsupervisability results in $\tilde{MW}$ with 20 states. Replacing W_1, *Produce*, and *Lock* by $\tilde{MW}$ gives the final abstracted system $\tilde{\mathscr{G}} = \tilde{MW} \parallel M_2 \parallel \tilde{HB}_1 \parallel \tilde{HB}_2 \parallel \tilde{HB}_3 \parallel \tilde{HB}_4$.

Finally, the components of $\tilde{\mathscr{G}}$ are composed to calculate a supervisor. This requires the exploration of the synchronous composition $\tilde{\mathscr{G}}$ with 48400 states, in contrast to the state space of the original system $\mathscr{G}$ with 1.3×10^6 states. The final supervisors calculated from $\mathscr{G}$ and $\tilde{\mathscr{G}}$ are identical and have 4374 states.

4 Certain Unsupervisability

4.1 Certainly Unsupervisable States and Transitions

The above example shows that some states of an automaton G must be avoided by synthesis in every possible context. That is, no matter what other automata are later composed with G, it is clear that these states are unsafe. Blocking states are examples of such states, but there are more states with this property.

Definition 10. Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$ be an automaton. The *certainly unsupervisable state set* of G is

$$\hat{U}(G) = \{\, x \in Q \mid \text{for every automaton } T = \langle \Sigma, Q_T, \rightarrow_T, Q_T^\circ, Q_T^\omega \rangle \text{ and every state } x^T \in Q_T \text{ it holds that } (x, x^T) \notin \hat{\Theta}_{G\|T} \,\} \,. \tag{12}$$

A state x of G is certainly unsupervisable, if there exists no other automaton T such that the state x is present in the least restrictive synthesis result $\hat{\Theta}_{G\|T}$. If a state is certainly unsupervisable, it is known that this state will be removed by every synthesis. If such states are encountered in an automaton during compositional synthesis, they can be removed before composing this automaton further.

Example 3. Consider again automaton HB_1 in Fig. 4. Clearly, the blocking state $\perp$ is certainly unsupervisable. In addition, state 5 is also certainly unsupervisable, because of the local uncontrollable transition $5 \xrightarrow{\tau_u} \perp$. As this transition is silent, no other component disables it, and as it is uncontrollable, the supervisor cannot disable it. Therefore, if the automaton ever enters state 5, blocking is unavoidable. It holds that $\hat{U}(HB_1) = \{5, \perp\}$.

In addition to states, it is worth considering transitions as certainly unsupervisable. If an uncontrollable event υ can take a state x to a certainly unsupervisable state, then all υ-transitions from x are certainly unsupervisable. Such transitions can be removed because it is clear that no supervisor will allow state x to be entered while υ is possible in the plant.

Definition 11. Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$ be an automaton. A transition $x \xrightarrow{\upsilon} y$ with $\upsilon \in \Sigma_u$ is a *certainly unsupervisable transition* if $x \xrightarrow{\tau_u^* \upsilon} \hat{U}(G)$.

Example 4. Consider automaton W in Fig. 4. States $\perp_1$, $\perp_2$, and $\perp_3$ are blocking and therefore certainly unsupervisable. The transition $5 \xrightarrow{\mathsf{!output_1}} 9$ is certainly unsupervisable, because $\mathsf{!output_1}$ is uncontrollable and $5 \xrightarrow{\tau_u} 6 \xrightarrow{\mathsf{!output_1}} \perp_2 \in \hat{U}(W)$. The uncontrollable event $\mathsf{!output_1}$ cannot be allowed in state 5, because if it was possible, blocking in state $\perp_2$ would be unavoidable.

Further, as every path from state 5 to an accepting state must take the certainly unsupervisable transition, it follows that state 5 is certainly unsupervisable. By similar arguments, it is established that $\hat{U}(W) = \{1,2,3,5,6,7,\perp_1,\perp_2,\perp_3\}$.

If the certainly unsupervisable states and transitions are known, they can be used to simplify an automaton to form a synthesis equivalent abstraction.

Definition 12. Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$ be an automaton. The result of *unsupervisability removal* from G is the automaton

$$\text{unsup}\mathscr{C}(G) = \langle \Sigma, Q, \rightarrow_{\text{unsup}}, Q^\circ \setminus \hat{U}(G), Q^\omega \setminus \hat{U}(G) \rangle \ , \tag{13}$$

where

$$\begin{aligned}
\rightarrow_{\text{unsup}} = \{\, (x,\sigma,y) \in \rightarrow \mid \sigma \in \Sigma_\text{c} \text{ and } x,y \notin \hat{U}(G) \,\} \cup \quad & (14) \\
\{\, (x,\upsilon,y) \in \rightarrow \mid \upsilon \in \Sigma_\text{u},\ x \notin \hat{U}(G),\ \text{and } y \in \hat{U}(G) \,\} \cup \quad & (15) \\
\{\, (x,\upsilon,y) \in \rightarrow \mid \upsilon \in \Sigma_\text{u},\ x \notin \hat{U}(G),\ \text{and } x \xrightarrow{\tau_\text{u}^*\upsilon} \hat{U}(G) \text{ does not hold} \,\} \ . \quad & (16)
\end{aligned}$$

The automaton resulting from unsupervisability removal has the same state set as the original automaton G, only the initial and accepting state sets are reduced by removing certainly unsupervisable states. All controllable transitions to certainly unsupervisable states are removed (14), as these transitions can always be disabled by the supervisor and therefore never appear in the final synthesis result. Uncontrollable transitions to certainly unsupervisable states, however, are retained (15), because they are needed to inform future synthesis steps. If another component disables these events, they may disappear in synchronous composition with that component, otherwise the source state may have to be removed in synthesis. Uncontrollable transitions to other states are deleted if they are certainly unsupervisable (16).

Example 5. When applied to automaton W in Fig. 4, unsupervisability removal deletes all transitions linked to the crossed out states. While state $\perp_3$ is certainly unsupervisable, the shared uncontrollable $!\texttt{output}_1$-transitions to this state are retained. They are needed in the following steps of compositional synthesis. If some other component disables $!\texttt{output}_1$ while in state 10 or 11, then these states may be retained, otherwise they will be removed at a later stage.

The following theorem confirms that unsupervisability removal results in a synthesis equivalent automaton. Therefore, the abstraction can be used to replace an automaton during compositional synthesis without affecting the final synthesis result.

Theorem 3. Let G be an automaton. Then $G \simeq_{\text{synth}} \text{unsup}\mathscr{C}(G)$.

Unsupervisability removal by definition only removes transitions and no states. Yet, states may become unreachable as a result of transition removal, and unreachable states can always be removed. Furthermore, it is possible to combine all remaining unsupervisable states, which have no outgoing transitions, into a single state [16].

4.2 Iterative Characterisation

The following definition provides an alternative characterisation of the certainly unsupervisable states through an iteration. It forms the basis for an algorithm to compute the set of certainly unsupervisable states.

Definition 13. Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$ be an automaton. Define the set $U(G)$ inductively as follows.

$$U^0(G) = \emptyset \; ; \tag{17}$$

$$U^{k+1}(G) = \{\, x \in Q \mid \text{for all paths } x = x_0 \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_n} x_n \in Q^\omega \text{ there exists } i = 0, \ldots, n \text{ such that } x_i \xrightarrow{\tau_u^*} U^k(G) \text{ or } i > 0 \text{ and } \sigma_i \in \Sigma_u \text{ and } x_{i-1} \xrightarrow{\tau_u^* \sigma_i \tau_u^*} U^k(G) \,\} \; ; \tag{18}$$

$$U(G) = \bigcup_{k \geq 0} U^k(G) \; . \tag{19}$$

The set $U^k(G)$ contains unsupervisable states of *level k*. There are no unsupervisable states of level 0, and the unsupervisable states of level 1 are the blocking states, i.e., those states from where it is not possible to ever reach an accepting state. Unsupervisable states at a higher level are states from where every path to an accepting state is known to pass through an unsupervisable state or an unsupervisable transition of a lower level.

Example 6. Consider automaton W in Fig. 4. It holds that $U^0(W) = \emptyset$, and $U^1(W) = \{\bot_1, \bot_2, \bot_3\}$ contains the three blocking states. Next, it can be seen that $1 \in U^2(W)$, because every path from 1 to an accepting state includes the transition $1 \xrightarrow{\mathsf{!output_1}} 5$ with $\mathsf{!output_1} \in \Sigma_u$ and $1 \xrightarrow{\tau_u} 2 \xrightarrow{\mathsf{!output_1}} \bot_1 \in U^1(W)$. Likewise, it holds that $2, 3, 5, 6, 7 \in U^2(W)$. No further states are contained in $U^2(W)$ or in $U^k(W)$ for $k > 2$, so that $U(W) = U^2(W) = \{1, 2, 3, 5, 6, 7, \bot_1, \bot_2, \bot_3\} = \hat{U}(W)$.

The following Theorem 4 confirms that the iteration $U^k(G)$ reaches the set of certainly unsupervisable states.

Theorem 4. Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$ be an automaton. Then $U(G) = \hat{U}(G)$.

To determine whether some state x is contained in the set $U^{k+1}(G)$ of unsupervisable states of a new level, the definition (18) considers all paths from state x to an accepting state. Such a condition is difficult to implement directly. It is more feasible to search backwards from the accepting states using the following secondary iteration.

Definition 14. Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$ be an automaton. Define the sets of *supervisable states* $S^k(G)$ for $k \geq 1$ inductively as follows.

$$S_0^{k+1}(G) = \{\, x \in Q^\omega \mid x \xrightarrow{\tau_u^*} U^k(G) \text{ does not hold} \,\} \; ; \tag{20}$$

$$S_{j+1}^{k+1}(G) = \{\, x \in Q \mid x \xrightarrow{\sigma} S_j^{k+1}(G), \text{ and } x \xrightarrow{\tau_u^*} U^k(G) \text{ does not hold, and if } \sigma \in \Sigma_u \text{ then } x \xrightarrow{\tau_u^* \sigma \tau_u^*} U^k(G) \text{ does not hold} \,\} \; ; \tag{21}$$

$$S^{k+1}(G) = \bigcup_{j \geq 0} S_j^{k+1}(G) \; . \tag{22}$$

Given the set $U^k(G)$ of unsupervisable states at level k, the iteration $S_j^{k+1}(G)$ computes a set of supervisable states, i.e., states from where a supervisor can reach an accepting state while avoiding the unsupervisable states in $U^k(G)$. The process starts as a backwards search from those accepting states from where it is not possible to reach a known unsupervisable state using only τ_u-transitions (20). Then transitions leading to the states already found are explored backwards (21). However, source states x that can reach a known unsupervisable state using only τ_u-transitions ($x \xrightarrow{\tau_\mathrm{u}^*} U^k(G)$), and known unsupervisable transitions ($x \xrightarrow{\tau_\mathrm{u}^*\sigma\tau_\mathrm{u}^*} U^k(G)$) are excluded.

Example 7. As shown in Example 6, the first iteration for unsupervisable states of automaton W in Fig. 4 gives the blocking states, $U^1(W) = \{\bot_1, \bot_2, \bot_3\}$. Then the first set of supervisable states for the next level contains the two accepting states, $S_0^2(W) = \{8, 9\}$ according to (20). Then $4 \xrightarrow{\texttt{!output}_1} 8 \in S_0^2(W)$ and $8 \xrightarrow{\tau_\mathrm{u}} 9 \in S_0^2(W)$ and $10 \xrightarrow{\tau_\mathrm{u}} 9 \in S_0^2(W)$, and it does not hold that $4 \xrightarrow{\tau_\mathrm{u}^*} U^1(W)$ or $4 \xrightarrow{\tau_\mathrm{u}^*\texttt{!output}_1\tau_\mathrm{u}^*} U^1(W)$ or $8 \xrightarrow{\tau_\mathrm{u}^*} U^1(W)$ or $10 \xrightarrow{\tau_\mathrm{u}^*} U^1(W)$. Therefore, $S_1^2(W) = \{4, 8, 10\}$ according to (21). Note that $5 \notin S_1^2(W)$ because despite the transition $5 \xrightarrow{\texttt{!output}_1} 9$ it holds that $5 \xrightarrow{\tau_\mathrm{u}} 6 \xrightarrow{\texttt{!output}_1} \bot_2 \in U^1(W)$. The next iteration gives $S_2^2(W) = \{0, 4, 9, 11\}$, and following iterations do not add any further states. The result is $S^2(W) = \{0, 4, 8, 9, 10, 11\} = Q \setminus U^2(W)$.

The following theorem confirms that the iteration $S_j^{k+1}(G)$ converges against the complement of the next level of unsupervisable states, $U^{k+1}(G)$.

Theorem 5. Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$ be an automaton. For all $k \geq 1$ it holds that $S^k(G) = Q \setminus U^k(G)$.

4.3 Algorithm

Algorithm 1 is an implementation of the iterations in Def. 13 and 14 to compute the set of certainly unsupervisable states for a given automaton G. First, the sets of certainly unsupervisable states U and certainly unsupervisable transitions UT are initialised in lines 2 and 3. Then the loop in lines 4–28 performs the iterations for $U^k(G)$.

The first step is to compute the supervisable states $S^{k+1}(G)$, which are stored in S. In line 5, this variable is initialised to the set $S_0^{k+1}(G)$ containing the accepting states that are not yet known to be unsupervisable. Then the loop in lines 7–15 uses a *stack* to perform a backwards search over the transition relation, avoiding known unsupervisable source states and known unsupervisable transitions. Upon termination, the variable S contains the set $S^{k+1}(G)$ of supervisable state for the next level.

Then the loop in lines 17–27 updates the sets U and UT. For every state that was not added to S, it explores the predecessor states reachable by sequences of τ_u-transitions, and adds any states found to U, if not yet included. By adding the τ_u-predecessors to the set U immediately, the reachability tests in (20) and (21) can be replaced by the direct membership tests in line 10. Next, for any new unsupervisable state x, the loop in lines 21–25, searches for possible uncontrollable transitions followed by sequences of τ_u and

Algorithm 1 Calculate $U(G)$

```
 1: input $G = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$
 2: $U \leftarrow \emptyset$
 3: $UT \leftarrow \emptyset$
 4: repeat
 5:     $S \leftarrow \{ x \in Q^\omega \mid x \notin U \}$
 6:     $stack$.init($S$)
 7:     while $stack$ not empty do
 8:         $x \leftarrow stack$.pop()
 9:         for all $w \xrightarrow{\sigma} x$ do
10:             if $w \notin S$ and $w \notin U$ and $(w, \sigma) \notin UT$ then
11:                 $S \leftarrow S \cup \{w\}$
12:                 $stack$.push($w$)
13:             end if
14:         end for
15:     end while
16:     $done \leftarrow$ true
17:     for all $x \xrightarrow{\tau_\mathrm{u}^*} Q \setminus S$ do
18:         if $x \notin U$ then
19:             $U \leftarrow U \cup \{x\}$
20:             $done \leftarrow$ false
21:             for all $\upsilon \in \Sigma_\mathrm{u} \setminus \{\tau_\mathrm{u}\}$ do
22:                 for all $w \xrightarrow{\tau_\mathrm{u}^* \upsilon} x$ do
23:                     $UT \leftarrow UT \cup \{(w, \upsilon)\}$
24:                 end for
25:             end for
26:         end if
27:     end for
28: until $done$
29: return $U$
```

adds such combinations of source states and uncontrollable events to the set certainly unsupervisable transitions UT.

The algorithm terminates if no new unsupervisable states are found during execution of the loop in lines 17–27, in which case the flag *done* retains its true value. At this point, the set U contains all certainly unsupervisable states.

4.4 Complexity

This section gives an estimate for the time complexity of Algorithm 1. Each iteration of the main loop in lines 4–28, except the last, adds at least one state to U, which gives at most $|Q| + 1$ iterations. During each of these iterations, the loop in lines 7–15 visits each transition at most once, giving up to $|{\rightarrow}|$ iterations, and the loop in lines 17–27 visits up to $|Q|$ predecessors of each state, which gives another $|Q|^2$ iterations. Assuming that the transitive closure of τ_u-transitions is calculated in advance, these iterations can be executed without overhead. The inner loop in lines 21–25 has another $|Q|^2$ iterations, again assuming that the closure of τ_u-transitions is calculated in advance. However, the inner loop is not executed more than once per state during the entire algorithm. The complexity to compute the τ_u-closure in advance is $O(|Q|^3)$ [18].

Summing up these computation costs, the worst-case time complexity of Algorithm 1 is found to be:

$$O((|Q|+1)\cdot(|{\rightarrow}|+|Q|^2)+|Q|\cdot|Q|^2+|Q|^3) \;=\; O(|\Sigma||Q|^3)\,. \tag{23}$$

Thus, the set of certainly unsupervisable states can be computed in polynomial time. This is surprising given the nondeterministic nature of similar problems, which require *subset construction* [12]. For example, the *set of certain conflicts* [14], which is the equivalent of the set of certainly unsupervisable states in nonblocking verification, can only be computed in exponential time. In synthesis, the assumption of a supervisor with the capability of full observation of the plant makes it possible to distinguish states and avoid subset construction.

4.5 Halfway Synthesis

This section introduces *halfway synthesis* [9], which has been used previously [9, 16] to remove unsupervisable states in compositional synthesis, and compares it with the set of certainly unsupervisable states. It is shown that in general more states can be removed by taking certain unsupervisability into account.

Definition 15. Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ, Q^\omega \rangle$, and let $\hat{\Theta}_{G,\tau_u}$ be the greatest fixpoint of the synthesis step operator according to Def. 8, but computed under the assumption that $\Sigma_u = \{\tau_u\}$. The *halfway synthesis result* for G is

$$\mathrm{hsup}\mathscr{C}(G) = \langle \Sigma, Q, \rightarrow_{\mathrm{hsup}}, Q^\circ \cap \hat{\Theta}_{G,\tau_u}, Q^\omega \cap \hat{\Theta}_{G,\tau_u} \rangle\,, \tag{24}$$

where

$$\rightarrow_{\mathrm{hsup}} = \{\,(x,\sigma,y) \in \rightarrow \mid x,y \in \hat{\Theta}_{G,\tau_u}\,\} \cup \tag{25}$$

$$\{\,(x,\upsilon,y) \in \rightarrow \mid x \in \hat{\Theta}_{G,\tau_u},\ \upsilon \in \Sigma_u \setminus \{\tau_u\},\ \text{and } y \notin \hat{\Theta}_{G,\tau_u}\,\} \tag{26}$$

The idea of halfway synthesis is to use standard synthesis, but treating only the silent uncontrollable event τ_u as uncontrollable. All other events are assumed to be controllable, because other plant components may yet disable shared uncontrollable events, so it is not guaranteed that these events cause controllability problems [9]. After computing the synthesis fixpoint $\hat{\Theta}_{G,\tau_u}$, the abstraction is obtained by removing controllable transitions to states not contained in $\hat{\Theta}_{G,\tau_u}$, while uncontrollable transitions are retained for the same reasons as in Def. 12.

Theorem 6. Let G be an automaton. Then $\mathrm{unsup}\mathscr{C}(G) \subseteq \mathrm{hsup}\mathscr{C}(G)$.

Example 8. When applied to automaton $H_1 \parallel B_1$ in Fig. 4, halfway synthesis removes the crossed out states and produces the same result as unsupervisability removal. However, it only considers states $\perp_1$, $\perp_2$, and $\perp_3$ of W in Fig. 4 as unsupervisable, because the shared uncontrollable event $!\mathtt{output}_1$ is treated as a controllable event. This automaton is left unchanged by halfway synthesis.

Halfway synthesis only removes those unsupervisable states that can reach a blocking state via local uncontrollable τ_u-transitions, but it does not take into account certainly unsupervisable transitions. Theorem 6 confirms that unsupervisability removal achieves all the simplification achieved by halfway synthesis, and Example 8 shows that there are cases where unsupervisability removal can do more. On the other hand, the complexity of halfway synthesis is the same as for standard synthesis, $O(|\Sigma||Q|^3)$, which is the same as found above for certain unsupervisability (23).

5 Conclusions

The set of *certainly unsupervisable states* of an automaton comprises all the states that must be avoided during synthesis of a controllable and nonblocking supervisor, in every possible context. In compositional synthesis, the removal of certainly unsupervisable states gives rise to a better abstraction than the previously used halfway synthesis, while maintaining the same cubic complexity.

The results of this paper are not intended to be used in isolation. In future work, the authors will integrate the removal of certainly unsupervisable states with their compositional synthesis framework [16]. It will be investigated in what order to apply unsupervisability removal and other abstraction methods, and how to group automata together for best performance.

Certainly unsupervisable states are also of crucial importance to determine whether two states of an automaton can be treated as equivalent for synthesis purposes. The results of this paper can be extended to develop abstraction methods that identify and merge equivalent states in compositional synthesis.

References

1. de Alfaro, L., Henzinger, T.A.: Interface automata. In: Proc. 9th ACM SIGSOFT Int. Symp. on Foundations of Software Engineering 2001. pp. 109–120. Vienna, Austria (2001)
2. Asarin, E., Maler, O., Pnueli, A.: Symbolic controller synthesis for discrete and timed systems. In: Hybrid Systems II, LNCS, vol. 999, pp. 1–20. Springer (1995)
3. Aziz, A., Singhal, V., Swamy, G.M., Brayton, R.K.: Minimizing interacting finite state machines: A compositional approach to language containment. In: Proc. IEEE Int. Conf. Computer Design: VLSI in Computers and Processors, ICCD '94. pp. 255–261 (1994)
4. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press (2008)
5. Baier, C., Klein, J., Klüppelholz, S.: A compositional framework for controller synthesis. In: Katoen, J.P., König, B. (eds.) Proc. 22nd Int. Conf. Concurrency Theory, CONCUR 2011. LNCS, vol. 6901, pp. 512–527. Springer, Aachen, Germany (Sep 2011)
6. Fabian, M.: On Object Oriented Nondeterministic Supervisory Control. Ph.D. thesis, Chalmers University of Technology, Göteborg, Sweden (1995), `https://publications.lib.chalmers.se/cpl/record/index.xsql?pubid=1126`
7. Filiot, E., Jin, N., Raskin, J.F.: Compositional algorithms for LTL synthesis. In: Chin, W.N. (ed.) Proc. 8th Int. Symp. Automated Technology for Verification and Analysis, ATVA 2010. LNCS, vol. 6252, pp. 112–127. Springer, Singapore, Singapore (Sep 2010)
8. Flordal, H., Malik, R.: Compositional verification in supervisory control. SIAM J. Control and Optimization 48(3), 1914–1938 (2009)

9. Flordal, H., Malik, R., Fabian, M., Åkesson, K.: Compositional synthesis of maximally permissive supervisors using supervision equivalence. Discrete Event Dyn. Syst. 17(4), 475–504 (2007)
10. Gohari, P., Wonham, W.M.: On the complexity of supervisory control design in the RW framework. IEEE Trans. Syst., Man, Cybern. 30(5), 643–652 (Oct 2000)
11. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall (1985)
12. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley (2001)
13. Lin, F., Wonham, W.M.: Decentralized control and coordination of discrete-event systems with partial observation. IEEE Trans. Autom. Control 35(12), 1330–1337 (Dec 1990)
14. Malik, R.: The language of certain conflicts of a nondeterministic process. Working Paper 05/2010, Dept. of Computer Science, University of Waikato, Hamilton, New Zealand (2010)
15. Malik, R., Streader, D., Reeves, S.: Conflicts and fair testing. Int. J. Found. Comput. Sci. 17(4), 797–813 (2006)
16. Mohajerani, S., Malik, R., Fabian, M.: A framework for compositional synthesis of modular nonblocking supervisors. IEEE Trans. Autom. Control 59(1), 150–162 (Jan 2014)
17. Mooij, A.J., Stahl, C., Voorhoeve, M.: Relating fair testing and accordance for service replaceability. J. Logic and Algebraic Programming 79(3–5), 233–244 (Apr–Jul 2010)
18. Nuutila, E.: Efficient Transitive Closure Compuation in Large Digraphs, Acta Polytechnica Scandinavica, Mathematics and Computing in Engineering Series, vol. 74. Finnish Academy of Technology, Helsinki, Finland (1995)
19. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: Proc. 16th ACM Symp. Principles of Programming Languages. pp. 179–190 (1989)
20. Tarski, A.: A lattice-theoretical fixpoint theorem and its applications. Pacific J. Math. 5(2), 285–309 (1955)
21. Wonham, W.M.: On the control of discrete-event systems. In: Nijmeijer, H., Schumacher, J.M. (eds.) Three Decades of Mathematical System Theory. LNCIS, vol. 135, pp. 542–562. Springer (1989)