Transition Removal for Compositional Supervisor Synthesis

Sahar Mohajerani Robi Malik Martin Fabian

***Abstract*— This paper investigates under which conditions *transitions* can be removed from an automaton while preserving important synthesis properties. The work is part of a framework for *compositional synthesis* of least restrictive controllable and nonblocking supervisors for modular discrete event systems. The method for transition removal complements previous results, which are largely focused on state merging. Issues concerning transition removal in synthesis are discussed, and *redirection maps* are introduced to enable a supervisor to process an event, even though the corresponding transition is no longer present in the model. Based on the results, different techniques are proposed to remove controllable and uncontrollable transitions, and an example shows the potential of the method for practical problems.**

I. Introduction

Supervisory control theory [1] provides a general framework to compute least restrictive strategies to control a given *plant* such that its behaviour satisfies a given *specification*. Synthesis for systems with a large number of components is impeded by an inherent complexity problem known as *state-space explosion*. A lot of research has been devoted to overcome the state-space explosion problem, and also to find more comprehensible supervisors [1]–[3]. *Compositional* methods seek to avoid large state spaces using *abstraction*, and have been used in verification [4], [5] and synthesis [3], [6], [7]. In a system with a large number of components, it is often possible to simplify individual components before composing them with the rest of the system, achieving significant performance improvements. Several ways to simplify components have been investigated in recent years.

Natural projection is a standard and effective way to compute abstractions, although strong restrictions need to be imposed to ensure the preservation of synthesis results [8], [9]. *Observation equivalence* [10] and *conflict equivalence* [11] are well-known abstraction methods for nonblocking verification [5], but for synthesis these abstractions can only be applied in combination with unobservable events [12], [13], which limits their applicability.

Recently, frameworks for compositional synthesis based on abstractions of nondeterministic automata have been proposed [3], [6], [7], in some cases showing substantial reduction of the number of states encountered during synthesis. This paper seeks to enhance these methods by providing means to remove *transitions*. This is important, because for large systems, the number of transitions may exceed the number of states by several orders of magnitude.

Compositional verification often uses observation equivalence for abstraction, which allows for transition removal using the transitive reduction [14], but observation equivalence does not necessarily preserve synthesis results [6]. *Supervision equivalence* [3] allows for transition removal, but relies on additional state labels that make some desirable abstractions impossible. The methods [6], [7] avoid *event hiding* that may cause problems in synthesis abstraction, but these approaches make it difficult to remove transitions.

This paper proposes some concrete means to identify transitions that are redundant for the purpose of synthesis. These methods are based on observation equivalence [10], but are more restrictive because of the need to preserve synthesis results. It is also shown how to restore the removed transitions to enable a synthesised supervisor to make control decisions based on a model with removed transitions.

This paper is organised as follows. After the preliminaries in Sect. II, a framework to support transition removal in compositional synthesis is presented in Sect. III. In Sect. IV, a sufficient condition for transition-removing abstraction is described, and in Sect. V, concrete methods to remove transitions are given. Finally, Sect. VI demonstrates transition removal using a practical example, and Sect. VII adds some concluding remarks. Formal correctness proofs are omitted for lack of space in this paper and can be found in [15].

II. Preliminaries

A. Events and Languages

The behaviour of discrete event systems is described using events and languages. *Events* represent incidents that cause transitions from one state to another and are taken from a finite alphabet Σ. For the purpose of supervisory control, this alphabet is partitioned into the set Σ_c of *controllable* events and the set Σ_u of *uncontrollable* events. Controllable events can be disabled by a supervisor, while uncontrollable events occur spontaneously, and are prefixed by an exclamation mark (!) in this paper. The special *termination event* $\omega \in \Sigma_c$ denotes completion of a task, and does not appear anywhere else but to mark such completions.

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

This work was supported by the Swedish Research Council.

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

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

B. Finite-State Automata

Discrete event systems are typically modelled as deterministic automata, but nondeterministic automata may be obtained as intermediate results from abstraction.

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

The transition relation is written in infix notation $x \xrightarrow{\sigma} y$, and is extended to traces in Σ^* by letting $x \xrightarrow{\varepsilon} x$ for all $x \in Q$, and $x \xrightarrow{s\sigma} z$ if $x \xrightarrow{s} y$ and $y \xrightarrow{\sigma} z$ for some $y \in Q$. Furthermore, $x \xrightarrow{s}$ means $x \xrightarrow{s} y$ for some $y \in Q$, and $x \rightarrow y$ means $x \xrightarrow{s} y$ for some $s \in \Sigma^*$. For an alphabet $\Omega \subseteq \Sigma$, the notation $x \xrightarrow{\Omega} y$ means $x \xrightarrow{\sigma} y$ for some $\sigma \in \Omega$, and $G \xrightarrow{s} x$ means $q^\circ \xrightarrow{s} x$ for some $q^\circ \in Q^\circ$. The *language* of automaton G is $\mathcal{L}(G) = \{\, s \in \Sigma^* \mid G \xrightarrow{s} \,\}$. Finally, G is *deterministic*, if $|Q^\circ| \leq 1$, and $x \xrightarrow{\sigma} y_1$ and $x \xrightarrow{\sigma} y_2$ always implies $y_1 = y_2$.

A special requirement is that states reached by the termination event ω do not have any outgoing transitions, i.e., if $x \xrightarrow{\omega} y$ 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 $Q^\omega = \{\, x \in Q \mid x \xrightarrow{\omega} \,\}$ in this notation. For graphical simplicity, states in Q^ω are shown shaded in the figures of this paper instead of explicitly showing ω-transitions.

When multiple automata are brought together to interact, lock-step synchronisation in the style of [16] is used.

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

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

where

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

C. Supervisory Control Theory

Given *plant* and *specification* automata, *supervisory control theory* [1] provides a method to *synthesise* a *supervisor* that restricts the behaviour of the plant such that the specification is always fulfilled. Two common requirements for this supervisor are *controllability* and *nonblocking*.

Definition 3: Specification $K = \langle \Sigma, Q_K, \rightarrow_K, Q_K^\circ \rangle$ is *controllable* with respect to plant $G = \langle \Sigma, Q_G, \rightarrow_G, Q_G^\circ \rangle$ if, for every trace $s \in \Sigma^*$, every state $x \in Q_K$, and every uncontrollable event $\upsilon \in \Sigma_u$ such that $K \xrightarrow{s} x$ and $G \xrightarrow{s\upsilon}$, it holds that $x \xrightarrow{\upsilon}_K$.

Definition 4: An automaton $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ is *nonblocking* if, for every state $x \in Q$ and every trace $s \in (\Sigma \setminus \{\omega\})^*$ such that $Q^\circ \xrightarrow{s} x$, there exists a trace $t \in \Sigma^*$ such that $x \xrightarrow{t\omega}$. Two automata G_1 and G_2 are *nonconflicting* if $G_1 \parallel G_2$ is nonblocking.

For a plant G and specification K, it is shown in [1] that there exists a *least restrictive* controllable sublanguage

$$\sup\mathcal{C}_G(\mathcal{L}(K)) \subseteq \mathcal{L}(K) \qquad (2)$$

such that $\sup\mathcal{C}_G(\mathcal{L}(K))$ is controllable with respect to G and nonblocking, and this language can be computed using a fixpoint iteration. This result can be reformulated in automata form, using an iteration on the state set. The synthesis result for an automaton G is obtained by restricting G to a maximal set of controllable and nonblocking states.

Definition 5: The *restriction* of $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ to $X \subseteq Q$ is $G_{|X} = \langle \Sigma, Q, \rightarrow_{|X}, Q^\circ \cap X \rangle$ where $\rightarrow_{|X} = \{\, (x,\sigma,y) \in \rightarrow \mid x, y \in X \,\}$.

Definition 6: [17] Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an automaton. The *synthesis step* operator $\Theta_G \colon 2^Q \rightarrow 2^Q$ for G is defined by $\Theta_G(X) = \Theta_G^{\mathrm{cont}}(X) \cap \Theta_G^{\mathrm{nonb}}(X)$, where

$$\Theta_G^{\mathrm{cont}}(X) = \{\, x \in X \mid x \xrightarrow{\Sigma_u} y \text{ implies } y \in X \,\} ; \qquad (3)$$

$$\Theta_G^{\mathrm{nonb}}(X) = \{\, x \in X \mid x \xrightarrow{t\omega}_{|X} \text{ for some } t \in \Sigma^* \,\} . \qquad (4)$$

Theorem 1: [17] Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$. The synthesis step operator Θ_G has a greatest fixpoint $\mathrm{gfp}\Theta_G = \hat{\Theta}_G \subseteq Q$. If the state set Q is finite, then the sequence $X^0 = Q$, $X^{i+1} = \Theta_G(X^i)$ reaches this fixpoint in a finite number of steps, i.e., $\hat{\Theta}_G = X^n$ for some $n \geq 0$.

Definition 7: The *synthesis result* for $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ is $\sup\mathcal{CN}(G) = G_{|\hat{\Theta}_G}$.

Theorem 2: Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be a deterministic automaton. $\sup\mathcal{CN}(G)$ is the least restrictive subautomaton of G that is controllable with respect to G and nonblocking.

The synthesis operator $\sup\mathcal{CN}$ performs synthesis for a plant automaton G. A simple transformation [3] exists to transform problems that also involve specifications into the plant-only control problems considered in this paper.

The result of synthesis is an automaton $\sup\mathcal{CN}(G)$ or a language $\mathcal{L}(\sup\mathcal{CN}(G))$, which describes the behaviour of a controlled system. In practice this is implemented as a *supervisor* that decides which controllable events are to be enabled or disabled in a given state. In this paper, a supervisor is a map

$$\mathcal{S} \colon \Sigma^* \rightarrow \{0,1\} . \qquad (5)$$

If $\mathcal{S}(s\sigma) = 0$ for some $s \in \Sigma^*$ and $\sigma \in \Sigma_c$ then the supervisor disables the controllable event σ after observing trace s, otherwise it enables σ. This results in the following *closed-loop behaviour* $\mathcal{L}(\mathcal{S}/G)$ of the plant G under the control of supervisor $\mathcal{S}$:

$$\mathcal{L}(\mathcal{S}/G) = \{\, s \in \mathcal{L}(G) \mid \mathcal{S}(s) = 1 \,\} . \qquad (6)$$

A supervisor can be constructed naturally from a language $L \subseteq \Sigma^*$, by letting $\mathcal{S}_L(s) = 1$ if and only if $s \in L$. For such a supervisor to be feasible, L must be controllable [1].

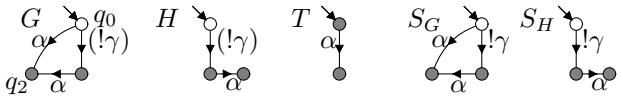

Fig. 1. Example of transition removal.

III. Compositional Synthesis

Many supervisory control problems can be presented as a set of interacting components. Then the synthesis problem consists of finding the least restrictive controllable and nonblocking supervisor for a set of plants,

$$\mathcal{G} = \{G_1, G_2, \ldots, G_n\} \ . \tag{7}$$

Compositional synthesis exploits the modularity of such systems and avoids building the complete synchronous product. Individual components G_i are simplified and replaced by smaller abstractions H_i. Synchronous composition is computed step by step, abstracting again the intermediate results. Eventually the abstractions result in a single automaton H, the abstract description of the system (7). Once found, H is used instead of the original system to calculate a synthesis result that leads to a solution for the original synthesis problem (7).

Individual components G_i typically contain events that do not appear in any other component G_j with $j \neq i$. These events are called *local events*. In the following, the set of local events is denoted by Υ, and $\Omega = \Sigma \setminus \Upsilon$ denotes the non-local or *shared* events. Local events are helpful to find abstractions and are parenthesised in the figures.

This paper focuses on abstractions that remove transitions from an automaton. This leads to a problem, because it is no longer obvious how to construct a supervisor from such an abstraction. After removal of transitions it is not clear how a supervisor can enact control over the events labelling the removed transitions.

Example 1: Consider automata G and T in Fig. 1 with $\Sigma_u = \Upsilon = \{!\gamma\}$. Automaton H is obtained by removing $q_0 \xrightarrow{\alpha} q_2$. Although H is an appropriate abstraction of G, as explained below in Example 2, the supervisor $S_H = \sup\mathcal{CN}(H \parallel T)$ disables event α in the initial state, and therefore is not a least restrictive supervisor for $G \parallel T$.

To solve this problem, the models (7) are augmented by a *redirection map* that contains the information needed to finally implement a supervisor.

Definition 8: A *synthesis pair* is a pair $(\mathcal{G}; \mathcal{D})$, where

- $\mathcal{G} = \{G_1, G_2, \ldots, G_n\}$ is a set of uncontrolled plant automata;
- $\mathcal{D}\colon \Sigma^* \to \Sigma^*$ is a prefix-preserving *redirection map*, i.e., a map such that $s \sqsubseteq t$ implies $\mathcal{D}(s) \sqsubseteq \mathcal{D}(t)$.

The compositional synthesis algorithm manipulates synthesis pairs. Each pair represents a partially solved synthesis problem, consisting of the plant model $\mathcal{G}$ to be controlled and the redirection map $\mathcal{D}$, which maps each input trace s accepted by the original plant before all abstractions, to a trace accepted by the current abstracted plant $\mathcal{G}$. A solution to the abstracted synthesis problem $\mathcal{G}$ can be interpreted as a supervisor for the original plant by taking the redirection map into account.

Definition 9: For every synthesis pair $(\mathcal{G}; \mathcal{D})$, define the represented supervisor map $\mathcal{S}_{(\mathcal{G};\mathcal{D})}\colon \Sigma^* \to \{0, 1\}$ as follows:

$$\mathcal{S}_{(\mathcal{G};\mathcal{D})}(s) = \begin{cases} 1, & \text{if } \mathcal{D}(s) \in \mathcal{L}(\sup\mathcal{CN}(\mathcal{G})); \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

Compositional synthesis starts by converting a control problem such as (7) into a synthesis pair $(\mathcal{G}_0; \mathrm{id})$ where $\mathcal{G}_0 = \{G_1, G_2, \ldots, G_n\}$ and $\mathrm{id}\colon \Sigma^* \to \Sigma^*$ is the identity map, i.e, $\mathrm{id}(s) = s$ for all $s \in \Sigma^*$. This initial synthesis pair is repeatedly abstracted such that the supervisor obtained from the abstraction remains a solution for the original problem. To ensure this property, each new synthesis pair needs to be *synthesis equivalent* to the previous pair.

Definition 10: Two synthesis pairs $(\mathcal{G}_1; \mathcal{D}_1)$ and $(\mathcal{G}_2; \mathcal{D}_2)$ are called *synthesis equivalent* with respect to plant G, written $(\mathcal{G}_2; \mathcal{D}_2) \simeq_{\mathrm{synth},G} (\mathcal{G}_1; \mathcal{D}_1)$, if $\mathcal{L}(\mathcal{S}_{(\mathcal{G}_1;\mathcal{D}_1)}/G) = \mathcal{L}(\mathcal{S}_{(\mathcal{G}_2;\mathcal{D}_2)}/G)$. Furthermore, $(\mathcal{G}_1; \mathcal{D}_1)$ and $(\mathcal{G}_2; \mathcal{D}_2)$ are synthesis equivalent, written $(\mathcal{G}_2; \mathcal{D}_2) \simeq_{\mathrm{synth}} (\mathcal{G}_1; \mathcal{D}_1)$, if $(\mathcal{G}_2; \mathcal{D}_2) \simeq_{\mathrm{synth},G} (\mathcal{G}_1; \mathcal{D}_1)$ for every automaton G.

Compositional synthesis terminates once $\mathcal{G} = \{H\}$ consists of a single automaton representing the abstracted system description. The following result, proved in [15], confirms that the closed-loop behaviour obtained in the end is equal to a solution for the original synthesis problem.

Proposition 3: Let $\mathcal{G}_0 = \{G_1, \ldots, G_n\}$ be a set of automata, and let $(\mathcal{G}_k; \mathcal{D}_k)$ be a synthesis pair such that $(\mathcal{G}_0; \mathrm{id}) \simeq_{\mathrm{synth},\mathcal{G}_0} (\mathcal{G}_k; \mathcal{D}_k)$. Then

$$\mathcal{L}(\mathcal{S}_{(\mathcal{G}_k;\mathcal{D}_k)}/\mathcal{G}_0) = \mathcal{L}(\sup\mathcal{CN}(\mathcal{G}_0)) \ . \tag{9}$$

IV. Transition-Wise Synthesis Equivalence

Several methods are known to abstract synthesis pairs such that the number of states is reduced [3], [6]. The abstractions are performed by manipulating the states and transitions of individual automata, such that synthesis equivalence is preserved. To allow for transition removal, state-wise synthesis abstraction, which is a special case of a definition from [6], is augmented by a transition-based concept in Def. 12.

Definition 11: Let $G = \langle \Sigma, Q, \rightarrow_G, Q^\circ \rangle$ and $H = \langle \Sigma, Q, \rightarrow_H, Q^\circ \rangle$ be two automata. H is a *state-wise synthesis abstraction* of G with respect to $\Upsilon \subseteq \Sigma$, if it holds for all automata T with $\Sigma_T \cap \Upsilon = \emptyset$ that $\hat{\Theta}_{G\parallel T} \subseteq \hat{\Theta}_{H\parallel T}$.

Definition 12: Let $G = \langle \Sigma, Q, \rightarrow_G, Q^\circ \rangle$ and $H = \langle \Sigma, Q, \rightarrow_H, Q^\circ \rangle$ be two automata. H is a *transition-wise synthesis abstraction* of G with respect to $\Upsilon \subseteq \Sigma$ if for every transition $x \xrightarrow{\sigma}_G y$ there exist $t, u \in \Upsilon^*$ such that:

(i) $x \xrightarrow{tP_\Omega(\sigma)u}_H y$;

(ii) for all automata T such that $\Sigma_T \cap \Upsilon = \emptyset$ and all transitions $(x, x_T) \xrightarrow{\sigma}_{|\hat{\Theta}_{G\parallel T}} (y, y_T)$ of $\sup\mathcal{CN}(G \parallel T)$ it holds that $(x, x_T) \xrightarrow{tP_\Omega(\sigma)u}_{|\hat{\Theta}_{H\parallel T}} (y, y_T)$.

Definition 13: Two automata G and H are state-wise (or transition-wise) *synthesis equivalent* with respect to Υ, if G is a state-wise (or transition-wise) synthesis abstraction of H with respect to Υ and H is a state-wise (or transition-wise) synthesis abstraction of G with respect to Υ.

To preserve transition-wise synthesis equivalence after removal of a transition, Def. 12 requires the existence of a so-called *redirection path* that links the source and target states of the removed transition. A redirection path for transition $x \xrightarrow{\sigma} y$ with respect to Υ is a path $x \xrightarrow{tP_\Omega(\sigma)u} y$ such that $t, u \in \Upsilon^*$. Using these paths, the redirection map is constructed to replace the removed transitions by the matching redirection paths. This enables the supervisor to make control decisions about the removed transitions.

Example 2: Consider again the automata in Fig. 1. Transition $q_0 \xrightarrow{\alpha} q_2$ can be removed from G, producing the state-wise and transition-wise synthesis equivalent automaton H. From this abstraction, a redirection map $\mathcal{D}\colon \Sigma^* \to \Sigma^*$ is constructed where $\mathcal{D}(\alpha s) = !\gamma\alpha s$ for all $s \in \Sigma^*$ and $\mathcal{D}(s) = s$ for all s such that α is not a prefix of s.

If G in Fig. 1 is placed in a larger system, say $\mathcal{G} = \{G, T\}$, then the synthesis pair $(\mathcal{G}; \mathrm{id})$ is synthesis equivalent to $(\mathcal{H}; \mathcal{D})$ where $\mathcal{H} = \{H, T\}$. Although the supervisor $S_H = \sup\mathcal{CN}(H \parallel T)$ obtained for $\mathcal{H}$ cannot directly be used to control the original plant $\mathcal{G}$, this becomes possible in combination with the redirection map $\mathcal{D}$. As $\mathcal{D}(\alpha) = !\gamma\alpha \in \mathcal{L}(\sup\mathcal{CN}(H \parallel T))$, the supervisor computed for $(\mathcal{H}, \mathcal{D})$ will enable the controllable event α in the initial state, in the same way as a supervisor computed for the original system $\mathcal{G}$.

The following result confirms that a redirection map as shown in Example 2 can be constructed in all cases where transition removal applied to a component results in a state-wise and transition-wise synthesis equivalent abstraction.

Theorem 4: [15] Let $\mathcal{G} = \{G_1, \ldots, G_n\}$ and $\mathcal{H} = \{H_1, G_2, \ldots, G_n\}$ such that G_1 and H_1 are state-wise and transition-wise synthesis equivalent with respect to $\Upsilon \subseteq \Sigma_1$ such that $\Upsilon \cap \Sigma_2 = \cdots = \Upsilon \cap \Sigma_n = \emptyset$ and $\to_{H_1} \subseteq \to_{G_1}$. Then there exists a redirection map $\mathcal{D}_1\colon \Sigma^* \to \Sigma^*$ such that $(\mathcal{G}; \mathcal{D}) \simeq_{\mathrm{synth}} (\mathcal{H}; \mathcal{D}_1 \circ \mathcal{D})$.

V. Transition Removal Abstraction

According to Theorem 4, synthesis results are preserved if transition removal in a component results in a state-wise and transition-wise synthesis equivalent abstraction. This section proposes some concrete methods to construct such abstractions, based on the idea of observation equivalence.

A. Observation Equivalence

Observation equivalence or *weak bisimilarity* is a well-known general abstraction method for nondeterministic automata [10]. It can be implemented by simple algorithms, and its application in compositional verification can substantially reduce the state space [5]. Observation equivalence is tested based on the transitive closure of the local event transitions [18]. The number of transitions can be substantially reduced by considering only the transitive reduction. More precisely, a transition $x \xrightarrow{\sigma} y$ is *observation equivalence redundant* and can be removed [14] if the automaton contains a matching redirection path.

Definition 14: Let $G = \langle \Sigma, Q, \to_G, Q^\circ \rangle$ and $H = \langle \Sigma, Q, \to_H, Q^\circ \rangle$ be two automata with $\Sigma = \Omega \dot\cup \Upsilon$ and $\to_H \subseteq \to_G$. Automaton H is a result of *observation equivalence redundant transition removal* from G with respect to Υ, if for all transitions $x \xrightarrow{\sigma}_G y$ there exist $t, u \in \Upsilon^*$ such that $x \xrightarrow{tP_\Omega(\sigma)u}_H y$.

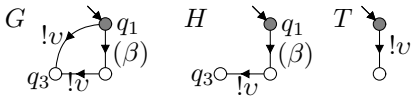

Fig. 2. H is observation equivalent to G, but not a synthesis abstraction.

Observation equivalence redundant transitions can be removed while preserving observation equivalence, which in turn ensures preservation of most temporal logic properties [10], [14]. Unfortunately, this does not include synthesis equivalence [6].

Example 3: Consider automata G, H, and T in Fig. 2. The uncontrollable transition $q_1 \xrightarrow{!\upsilon} q_3$ is observation equivalence redundant with respect to $\Upsilon = \{\beta\}$. Removing it produces H. In G and H, the uncontrollable event $!\upsilon$ leads to the blocking state q_3. With H, blocking can be prevented by disabling β, leaving only the initial state. But with G, the uncontrollable transition $q_1 \xrightarrow{!\upsilon} q_3$ produces an empty synthesis result. The test T demonstrates that G and H are not state-wise synthesis equivalent since G is not a state-wise synthesis abstraction of H.

This counterexample shows that in general synthesis equivalence is not preserved by removing observation equivalence redundant transitions, so extra restrictions need to be imposed.

B. Uncontrollable Redundant Transitions

In Example 3, if the local event β was uncontrollable, then the resultant abstraction H would be a transition-wise synthesis abstraction of G. This suggests to interpret an uncontrollable transition as redundant if the local transitions used in the redirection path are also uncontrollable.

Definition 15: Let $G = \langle \Sigma, Q, \to_G, Q^\circ \rangle$ and $H = \langle \Sigma, Q, \to_H, Q^\circ \rangle$ be two automata with $\Sigma = \Omega \dot\cup \Upsilon$ and $\to_H \subseteq \to_G$. Automaton H is a result of *uncontrollable redundant transition removal* from G with respect to Υ, if the following conditions hold for all transitions $x \xrightarrow{\sigma}_G y$.

(i) If $\sigma \in \Sigma_c$ then $x \xrightarrow{\sigma}_H y$.
(ii) If $\sigma \in \Sigma_u$ then there exist $t, u \in (\Upsilon \cap \Sigma_u)^*$ such that $x \xrightarrow{tP_\Omega(\sigma)u}_H y$.

The transitions present in $\to_G$ but not in $\to_H$ in Def. 15 are called *uncontrollable redundant* transitions. These transitions can be removed while producing a synthesis equivalent abstraction.

Theorem 5: [15] Let $H = \langle \Sigma, Q, \to_H, Q^\circ \rangle$ be a result of uncontrollable redundant transition removal from $G = \langle \Sigma, Q, \to_G, Q^\circ \rangle$ with respect to $\Upsilon \subseteq \Sigma$. Then G and H are state-wise and transition-wise synthesis equivalent with respect to Υ.

C. Controllable Redundant Transitions

For uncontrollable events, an uncontrollable redirection path guarantees transition-wise synthesis equivalence. For

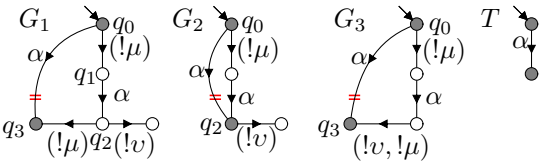

Fig. 3. Different redirection paths after the event of a removed transition. The transitions to be removed are marked by double-line strike-through.

controllable events, all events on a redirection path except for the event of the removed transition should be uncontrollable. However, the following counterexample reveals that one more condition is needed to guarantee a correct abstraction.

Example 4: Consider automaton G_1 in Fig. 3 where $\Sigma_u = \Upsilon = \{!\mu, !\upsilon\}$. Transition $q_0 \xrightarrow{\alpha} q_3$ is observation equivalence redundant because $q_0 \xrightarrow{!\mu\alpha!\mu} q_3$. Let H_1 be the result of removing the transition $q_0 \xrightarrow{\alpha} q_3$. In both G_1 and H_1, the controllable transition $q_1 \xrightarrow{\alpha} q_2$ must be disabled to avert blocking via the uncontrollable event $!\upsilon$. Removing this transition makes q_3 unreachable in $\sup\mathcal{CN}(H_1 \| T)$, but it remains reachable in $\sup\mathcal{CN}(G_1 \| T)$. The test T demonstrates that G_1 and H_1 are not transition-wise synthesis equivalent as G_1 is not a transition-wise synthesis abstraction of H_1.

Example 4 shows that there is a problem with uncontrollable local events *after* the event of a removed transition on a redirection path. The problem disappears if there are no further events after the removed event, as in automaton G_2 in Fig. 3. This leads to the idea of *controllable prefix-redundant* transition removal, which can be shown to imply both state-wise and transition-wise synthesis abstraction.

Definition 16: Let $G = \langle \Sigma, Q, \rightarrow_G, Q^\circ \rangle$ and $H = \langle \Sigma, Q, \rightarrow_H, Q^\circ \rangle$ be two automata with $\Sigma = \Omega \dot\cup \Upsilon$ and $\rightarrow_H \subseteq \rightarrow_G$. Automaton H is a result of *controllable prefix-redundant transition removal* from G with respect to Υ, if the following conditions hold for all transitions $x \xrightarrow{\sigma}_G y$.

(i) If $\sigma \in \Sigma_u$ then $x \xrightarrow{\sigma}_H y$.
(ii) If $\sigma \in \Sigma_c$ then there exists $t \in (\Upsilon \cap \Sigma_u)^*$ such that $x \xrightarrow{tP_\Omega(\sigma)}_H y$.

Controllable prefix-redundant transition removal only allows for local events *before* the event of a removed transition. Local events after this event can also be considered by adding additional requirements.

Example 5: As shown in Example 4, removal of the transition $q_0 \xrightarrow{\alpha} q_3$ in G_1 in Fig. 3 does not ensure synthesis abstraction because of the uncontrollable $!\upsilon$-transition in state q_2. Automaton G_3 also has the observation equivalence redundant transition $q_0 \xrightarrow{\alpha} q_3$ and an $!\upsilon$-transition enabled after α on the redirection path $q_0 \xrightarrow{!\mu\alpha!\mu} q_3$. Yet, in this case, the $!\upsilon$-transition does not lead to a blocking state, and the removal of $q_0 \xrightarrow{\alpha} q_3$ results in a state-wise and transition-wise synthesis equivalent automaton.

Automata G_1 and G_3 in Fig. 3 differ in the target state of $q_2 \xrightarrow{!\upsilon}$. This suggests to allow uncontrollable events in the second part of a redirection provided that they are local and lead to a target state on the redirection path.

Definition 17: Let $G = \langle \Sigma, Q, \rightarrow, Q^\circ \rangle$ be an automaton and $\Upsilon \subseteq \Sigma$. A path

$$x_0 \xrightarrow{\sigma_1} x_1 \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_k} x_k \tag{10}$$

is a *weakly controllable* Υ*-path* if $\sigma_1, \ldots, \sigma_k \in \Upsilon$ and for all uncontrollable transitions $x_l \xrightarrow{\upsilon} y$ with $0 \leq l < k$ and $\upsilon \in \Sigma_u$ it holds that $\upsilon \in \Upsilon$ and $y = x_j$ for some $0 \leq j \leq k$.

A weakly controllable path consists of only local transitions, and furthermore all uncontrollable transitions enabled along this path must use local events and lead to states along the path. Imposing this condition on the redirection path gives the condition for a *controllable suffix-redundant transition*, which is sufficient for synthesis equivalence.

Definition 18: Let $G = \langle \Sigma, Q, \rightarrow_G, Q^\circ \rangle$ and $H = \langle \Sigma, Q, \rightarrow_H, Q^\circ \rangle$ be two automata with $\Sigma = \Omega \dot\cup \Upsilon$ and $\rightarrow_H \subseteq \rightarrow_G$. Automaton H is a result of *controllable suffix-redundant transition removal* from G with respect to Υ, if the following conditions hold for all transitions $x \xrightarrow{\sigma}_G y$.

(i) If $\sigma \in \Sigma_u$ then $x \xrightarrow{\sigma}_H y$.
(ii) If $\sigma \in \Sigma_c$ then there exists $u \in \Upsilon^*$ such that $x \xrightarrow{P_\Omega(\sigma)}_H z \xrightarrow{u}_H y$, and $z \xrightarrow{u}_G y$ is a weakly controllable Υ-path.

Both controllable prefix-redundant and suffix-redundant transition removal preserve synthesis equivalence. These conditions can be combined to allow sequences of local events before *and* after a removed transition.

Definition 19: Let $G = \langle \Sigma, Q, \rightarrow_G, Q^\circ \rangle$ and $H = \langle \Sigma, Q, \rightarrow_H, Q^\circ \rangle$ be two automata with $\Sigma = \Omega \dot\cup \Upsilon$ and $\rightarrow_H \subseteq \rightarrow_G$. Automaton H is a result of *controllable redundant transition removal* from G with respect to Υ, if the following conditions hold for all transitions $x \xrightarrow{\sigma}_G y$.

(i) If $\sigma \in \Sigma_u$ then $x \xrightarrow{\sigma}_H y$.
(ii) If $\sigma \in \Sigma_c$ then there exist $t \in (\Upsilon \cap \Sigma_u)^*$ and $u \in \Upsilon^*$ such that $x \xrightarrow{tP_\Omega(\sigma)}_H z \xrightarrow{u}_H y$, and $z \xrightarrow{u}_G y$ is a weakly controllable Υ-path.

Theorem 6: [15] Let $H = \langle \Sigma, Q, \rightarrow_H, Q^\circ \rangle$ be a result of controllable redundant transition removal from $G = \langle \Sigma, Q, \rightarrow_G, Q^\circ \rangle$ with respect to $\Upsilon \subseteq \Sigma$. Then G and H are state-wise and transition-wise synthesis equivalent with respect to Υ.

VI. Example

In this section, the proposed synthesis procedure is applied to a manufacturing system. The model consists of four machines M_1, M_2, M_3, and M_4, linked by two buffers B_1 and B_2. Workpieces are first processed by M_1 (s_1) and then placed into B_1 ($!f_1$), then they go to M_2 (s_2) and are placed into B_2 ($!f_2$). From B_2, the workpieces either go to M_3 for final processing (s_3) or to M_4 (s_4) for additional processing. However, M_4 has a fault that occasionally sends a workpiece back to B_1 ($!re$). At any time, M_1 and B_1 can be reset by the controllable event *rs*. Fig. 4 shows the system layout and the automata model. Events $!f_1$, $!f_2$, $!f_3$, $!f_4$ and $!re$ are uncontrollable, all other events are controllable.

Compositional synthesis starts with the pair $(\mathcal{G}_0; \mathrm{id})$ where $\mathcal{G}_0 = \{M_1, M_2, M_3, M_4, B_1, B_2\}$. The first step is to calculate the composition $B_1 \parallel M_1$ shown in Fig. 5. Now $!f_1$, rs, and s_1 are local events, which makes $q_0 \xrightarrow{rs} q_0$ and $q_2 \xrightarrow{rs}$

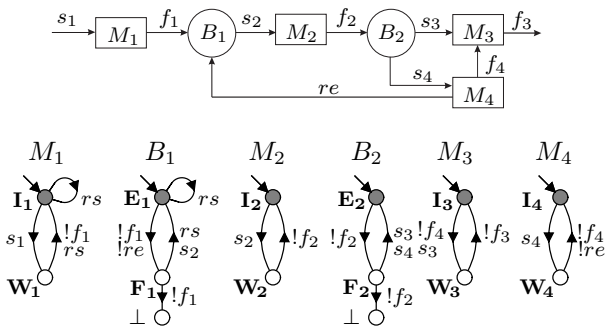

Fig. 4. Manufacturing system example.

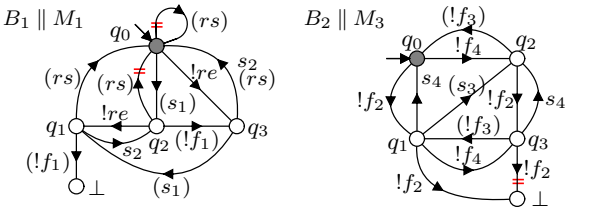

Fig. 5. Some subsystems of the manufacturing example. The transitions to be removed are marked by double-line strike-through.

q_0 controllable prefix-redundant transitions with redirection paths $q_0 \xrightarrow{\varepsilon} q_0$ and $q_2 \xrightarrow{!f_1} q_3 \xrightarrow{rs} q_0$ respectively. Removal of these transitions results in H_1. The modified synthesis pair is $(\mathcal{G}_1; \mathcal{D}_1)$ where $\mathcal{G}_1 = \{H_1, M_2, M_3, M_4, B_2\}$ and $\mathcal{D}_1$ redirects $q_2 \xrightarrow{rs} q_0$ and $q_0 \xrightarrow{rs} q_0$ via $q_2 \xrightarrow{!f_1} q_3 \xrightarrow{rs} q_0$ and $q_0 \xrightarrow{\varepsilon} q_0$, respectively.

Next, $B_2 \parallel M_3$ is computed, shown in Fig. 5. This makes $!f_3$ and s_3 local events, and $q_3 \xrightarrow{!f_2} \perp$ becomes an uncontrollable redundant transition with redirection path $q_3 \xrightarrow{!f_3} q_1 \xrightarrow{!f_2} \perp$. The new synthesis pair is $(\mathcal{G}_2; \mathcal{D}_2 \circ \mathcal{D}_1)$ where $\mathcal{G}_2 = \{H_1, M_2, M_4, H_2\}$ and $\mathcal{D}_2$ redirects $q_3 \xrightarrow{!f_2} \perp$ via $q_3 \xrightarrow{!f_3} q_1 \xrightarrow{!f_2} \perp$.

The final synthesis step to compute $\sup\mathcal{CN}(\mathcal{G}_2)$ explores the state space of $\mathcal{G}_2$ which has 100 states and 290 transitions. This is in contrast to standard monolithic synthesis, which explores the same state space using 340 transitions. Both the final monolithic and compositional supervisor have 26 states. However, the compositional supervisor has 63 transitions, while the monolithic supervisor has 81 transitions.

These improvements have been achieved by removing just three transitions from the model. More savings are likely in larger contexts, particularly in combination with state-removing abstraction rules.

VII. Conclusions

It has been shown under which conditions transitions can be removed from an automaton while preserving compositional synthesis results. Different techniques to remove controllable and uncontrollable transitions have been presented, and a practical example has demonstrated how the number of transitions is reduced. The methods proposed in this paper are not intended to be used in isolation, but they will be combined with other synthesis-preserving abstraction methods. In the future, the authors plan to develop a framework for compositional synthesis that combines abstractions that remove states [3], [6] and transitions, as well as renaming [7] to remove nondeterminism.

References

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

[2] R. J. Leduc, B. A. Brandin, M. Lawford, and W. M. Wonham, "Hierarchical interface-based supervisory control—part I: Serial case," *IEEE Trans. Autom. Control*, vol. 50, no. 9, pp. 1322–1335, Sept. 2005.

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

[4] E. M. Clarke, D. E. Long, and K. L. McMillan, "Compositional model checking," in *Proc. 5th IEEE Symp. Logic in Computer Science*, 1989, pp. 353–362.

[5] H. Flordal and R. Malik, "Compositional verification in supervisory control," *SIAM J. Control and Optimization*, vol. 48, no. 3, pp. 1914–1938, 2009.

[6] S. Mohajerani, R. Malik, S. Ware, and M. Fabian, "On the use of observation equivalence in synthesis abstraction," in *Proc. 3rd IFAC Workshop on Dependable Control of Discrete Systems, DCDS 2011*, Saarbrücken, Germany, 2011, pp. 84–89.

[7] S. Mohajerani, R. Malik, and M. Fabian, "Nondeterminism avoidance in compositional synthesis of discrete event systems," in *Proc. 7th Int. Conf. Automation Science and Engineering, CASE 2011*, Trieste, Italy, 2011, pp. 19–24.

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

[9] K. Schmidt and C. Breindl, "On maximal permissiveness of hierarchical and modular supervisory control approaches for discrete event systems," in *Proc. 9th Int. Workshop on Discrete Event Systems, WODES '08*, Göteborg, Sweden, May 2008, pp. 462–467.

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

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

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

[13] R. Su, J. H. van Schuppen, and J. E. Rooda, "Model abstraction of nondeterministic finite-state automata in supervisor synthesis," *IEEE Trans. Autom. Control*, vol. 55, no. 11, pp. 2527–2541, Nov. 2010.

[14] J. Eloranta, "Minimizing the number of transitions with respect to observation equivalence," *BIT*, vol. 31, no. 4, pp. 397–419, 1991.

[15] S. Mohajerani, R. Malik, and M. Fabian, "Five abstraction rules to remove transitions while preserving compositional synthesis results," Working Paper 01/2012, Dept. of Computer Science, University of Waikato, Hamilton, New Zealand, 2012.

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

[17] R. Malik and H. Flordal, "Yet another approach to compositional synthesis of discrete event systems," in *Proc. 9th Int. Workshop on Discrete Event Systems, WODES '08*, Göteborg, Sweden, May 2008, pp. 16–21.

[18] T. Bolognesi and S. A. Smolka, "Fundamental results for the verification of observational equivalence: a survey," in *Protocol Specification, Testing and Verification VII: Proc. IFIP WG6.1 7th Int. Conf. Protocol Specification, Testing and Verification*, H. Rudin and C. H. West, Eds. Amsterdam, The Netherlands: North Holland, 1987, pp. 165–179.