Compositional Synthesis of Supervisors in the Form of State Machines and State Maps

Sahar Mohajerani\textsuperscript{a} Robi Malik\textsuperscript{b} Martin Fabian\textsuperscript{c}

\textsuperscript{a}Vehicle Dynamics and Active Safety Center, Volvo Cars Corporation, Göteborg, Sweden
\textsuperscript{b}Department of Computer Science, University of Waikato, Hamilton, New Zealand
\textsuperscript{c}Department of Signals and Systems, Chalmers University of Technology, Göteborg, Sweden

Abstract

This paper investigates the compositional abstraction-based synthesis of least restrictive, controllable, and nonblocking supervisors for discrete event systems that are given as a large number of finite-state machines. It compares a previous algorithm that synthesises modular supervisors in the form of state machines, with an alternative that records state maps after each abstraction step and uses these to control the system. The state map-based algorithm supports all abstraction methods used previously, and in addition allows for nondeterminism, hiding, and transition removal. It has been implemented in the software tool Supremica and applied to several large industrial models. The experimental results and the complexity analysis show that state maps can be computed efficiently and in many cases require less memory than state machine-based supervisors.

Key words: Discrete event systems; Compositional synthesis; Controller constraints and structure; Algorithms and software; Computational issues.

1 Introduction

Supervisory control theory \cite{13} provides a general framework to automatically synthesise supervisors to control discrete event systems. Synthesis typically involves the calculation of a largest set of safe states, and then the supervisor is built to constrain the system within these states. This is a problem for systems with many components, as the number of states grows exponentially with the number of components. Then synthesis may fail to complete in reasonable time or run out of memory, or return large supervisors that are difficult to implement in limited-memory devices such as PLCs.

The complexity problems can be solved to some extent by compositional synthesis \cite{2,4,14}, which composes the system components gradually and uses abstraction to simplify each intermediate result. This method can compute least restrictive, controllable, and nonblocking supervisors in the form of several finite-state machines \cite{10}. Another approach to combat the complexity of synthesis uses symbolic representations. The set of safe states can be calculated symbolically and used to construct a supervisor in the form of guard formulas \cite{8}. Petri net supervisors can be represented as linear constraints \cite{7}.

This paper proposes an improved version of the compositional method \cite{10} that produces more memory-efficient supervisors. The set of safe states is represented as a cascaded map and used directly to control the system. This is similar to hierarchical groups of states as in Statecharts \cite{5}, except that the groupings are computed automatically as part of the synthesis process. The cascaded map is compact and easier to compute than guard formulas \cite{8}. It is shown to be smaller than state machine-based supervisors \cite{10}, both by theoretic complexity analysis and by experiments. The new algorithm also performs better, because it permits nondeterminism, hiding, and transition removal \cite{9}, and avoids renaming.

The improved compositional synthesis algorithm has been implemented in the DES software tool Supremica \cite{1}, and its performance has been compared with the older algorithm \cite{10}. Both algorithms successfully compute supervisors, even for systems with more than 10\textsuperscript{17} reachable states, within some seconds. In most cases, supervisor maps are calculated faster and require less memory than state machine-based supervisors.
In the following, Section 2 summarises the background of supervisory control theory. Section 3 describes and analyses the compositional synthesis of state machine and state map-based supervisors, and Section 4 shows experimental results to compare them. Section 5 adds concluding remarks. Further technical details of the state map-based method can be found in the working paper [11].

2 Preliminaries

A finite-state machine (FSM) is a tuple \( G = (\Sigma, Q, Q^o, Q^c, \rightarrow) \), where \( \Sigma \) is a finite set of events, \( Q \) is a finite set of states, \( Q^o \subseteq Q \) is the set of initial states, \( Q^c \subseteq Q \) is the set of accepting states, and \( \rightarrow \subseteq Q \times \Sigma \times Q \) is the transition relation.

The transition relation is written in infix notation, where \( x \xrightarrow{\sigma} y \) means the existence of a transition from state \( x \in Q \) to \( y \in Q \) with event \( \sigma \in \Sigma \). This notation is extended to traces \( s \in \Sigma^* \) in the standard way. Given state sets \( X, Y \subseteq Q \), the notation \( X \rightarrow Y \) means \( x \xrightarrow{\sigma} y \) for some states \( x \in X \) and \( y \in Y \), and \( X \rightarrow Y \) means \( x \xrightarrow{\sigma} y \) for some \( s \in \Sigma^* \). As events not in the event set \( \Sigma \) of an FSM are assumed to be always enabled without state change, the transition relation is further extended by letting \( y \xrightarrow{\sigma} x \) for all \( x \in Q \) and \( \sigma \notin \Sigma \).

For the purpose of control, the event set \( \Sigma \) is partitioned into the sets \( \Sigma_c \) of controllable events and \( \Sigma_u \) of uncontrollable events [13]. Control is exercised by a supervisor that observes the system, and depending on its state prevents the occurrence of some controllable events. This can be represented as a control function \( \Phi: Q \rightarrow 2^{\Sigma_c} \) (1) that assigns to each state \( x \in Q \) the set \( \Phi(x) \) of controllable events to be enabled in this state. The uncontrollable events cannot be prevented by the supervisor and always remain enabled.

Through supervision, the behaviour of an FSM is restricted to a so-called sub-FSM. An FSM \( G_1 = (\Sigma, Q_1, Q^o_1, Q^c_1, \rightarrow_1) \) is a sub-FSM of \( G = (\Sigma, Q, Q^o, Q^c, \rightarrow) \), if \( Q_1 \subseteq Q, Q^o_1 \subseteq Q^o, Q^c_1 \subseteq Q^c \), and \( \rightarrow_1 \subseteq \rightarrow \). Supervisory control theory [13] is concerned with the question whether there exists a supervisor \( \Phi \) that can constrain a system \( G \) to achieve the behaviour of a given sub-FSM \( G_1 \) of \( G \). The answer to this question is related to the concepts of controllability and nonblocking.

- A sub-FSM \( G_1 \) of \( G \) is said to be controllable in \( G \) if, for all states \( x \) of \( G_1 \) and \( y \) of \( G \), and for every uncontrollable event \( v \in \Sigma_u \) such that \( x \xrightarrow{v} y \) in \( G \), it also holds that \( x \xrightarrow{v} y \) in \( G_1 \) [3].
- An FSM \( G = (\Sigma, Q, Q^o, Q^c, \rightarrow) \) is nonblocking, if for every state \( x \in Q \) such that \( Q^c \rightarrow x \) it holds that \( x \xrightarrow{\sigma} Q^o \) [13].

Controllability reflects the supervisor’s inability to prevent uncontrollable events. It requires for all reachable states of the sub-FSM \( G_1 \), that all uncontrollable transitions present in \( G \) are also present in \( G_1 \). In addition, the supervised behaviour is typically required to be non-blocking. In a nonblocking FSM, it is always possible to reach an accepting state, thus ensuring the absence of livelocks and deadlocks.

Most standard questions of supervisory control theory can be expressed as a question of how a given system, which fails the nonblocking property, can be supervised to become nonblocking. Given an FSM \( G \), there exists a unique maximal sub-FSM, denoted \( \text{sup}(G) \), such that \( \text{sup}(G) \) is both controllable and nonblocking. Thus, \( \text{sup}(G) \) can be used to construct a least restrictive supervisor that controls \( G \) in a nonblocking way [3]. The process of computing \( \text{sup}(G) \) is called synthesis. It can be achieved by a standard fixpoint iteration, with time complexity polynomial in the number of states of \( G \).

This paper focuses on the case that the system \( G \) is given as the synchronous composition of several FSMs. The synchronous composition of two FSMs \( G_1 = (\Sigma_1, Q_1, Q^o_1, Q^c_1, \rightarrow_1) \) and \( G_2 = (\Sigma_2, Q_2, Q^o_2, Q^c_2, \rightarrow_2) \) is \( G_1 \parallel G_2 = (\Sigma_1 \cup \Sigma_2, Q_1 \times Q_2, Q^o_1 \times Q^o_2, Q^c_1 \times Q^c_2, \rightarrow) \), where \( (x_1, x_2) \xrightarrow{(\sigma_1, \sigma_2)} (y_1, y_2) \) if \( x_1 \xrightarrow{\sigma_1} y_1 \) and \( x_2 \xrightarrow{\sigma_2} y_2 \) [6]. The number of states in the synchronous composition grows exponentially with the number of FSMs composed, and so does the complexity of synthesis. This complexity is mitigated by compositional synthesis.

3 Compositional Synthesis

The input to compositional synthesis is a set of FSMs, whose synchronous composition represents the system to be controlled, and the objective is to compute a least restrictive control function \( \Phi \) (1). To avoid the full synchronous composition, the FSMs are composed step-wise and the intermediate results are simplified by abstraction after each step. As an example, consider the system \( G = A \parallel B \parallel C \) in Fig. 1. The following subsections describe two compositional approaches to synthesise the least restrictive supervisor for this system, which mainly differ in the representation of the control function \( \Phi \).
The compositional synthesis algorithm [10] computes a modular supervisor in the form of FSMs, which interact in synchronous composition with the system $G$. The control function $\Phi$ uses the supervisor FSMs to track the system state and disables any controllable events that are disabled by the supervisor FSMs. For this approach to work, the supervisor FSMs must be deterministic.

In the example system in Fig. 1, the event $!\eta$ appears only in the FSM $C$. Such an event is called local. With $!\eta$ being local, the FSM $C$ is simplified using weak synthesis observation equivalence [10] to the abstraction $\tilde{C}$ in Fig. 2. However, $\tilde{C}$ is nondeterministic because of its two $\alpha$-transitions from state $\alpha_0$, which may prevent the extraction of supervisor components in later steps. Therefore event $\alpha$ is replaced using renaming with two new events $\alpha_1$ and $\alpha_2$ [10], as shown in $RC$ in Fig. 2. At the same time, the $\alpha$-transition in $B$ is replaced by two transitions with $\alpha_1$ and $\alpha_2$. The decision which of these events is enabled is made by a so-called distinguisher, derived from $C$, which becomes the first component of the synthesised supervisor. It is shown as $RC$ in Fig. 3.

Next, $A$ and $B$ are replaced by their synchronous composition $A \| B$ in Fig. 2. Then events $\gamma$ and $!\lambda$ are local, as they appear only in $A$ and $B$, and therefore states $\perp$ and $p_1$, crossed out in the figure, must be avoided as they are blocking or lead to blocking by the local uncontrollable event $!\lambda$. These states are merged by unsupervisability removal [15] to produce a second supervisor component $AB$ as shown in Fig. 3. The final composition $AB \| RC$ has ten states, four of which are removed by unsupervisability removal to obtain the third supervisor component $ABC$ in Fig. 3. The FSM $ABC$ always disables the renamed version $\alpha_2$ of the original event $\alpha$.

The supervisor FSMs in synchronous composition with the renamed system produce the behaviour of the least restrictive supervisor. The control architecture in Fig. 4 shows how they are used to control the original system $G$ [10]. Assume the system generates event $\alpha$, which has been been renamed and replaced by $\alpha_1$ and $\alpha_2$. When the supervisor receives the event $\alpha$, it first uses the inverse renaming $\rho^{-1}$ to obtain the original events $\{\alpha_1, \alpha_2\}$. These are passed to the distinguisher ($RC$ in the example), which enables only one of them. That event is passed to the other supervisor components ($AB$ and $ABC$), which update their states and then form a control decision $\Gamma'$ by collecting the enabled controllable events. This decision is based on renamed events, and the renaming $\rho$ translates it to a control decision $\Gamma \subseteq \Sigma_c$ using the original events.

The space complexity of the FSM-based supervisor is determined by the number of FSMs and their number of transitions. If there are $n$ FSMs, this gives $n$ abstraction steps initially, plus $n-1$ abstraction steps after synchronous composition, as the number of FSMs decreases with each step. Each step may result in a distinguisher and a supervisor FSM, up to $2(2n-1)$ FSMs in total. Their size is determined by the number of transitions. A deterministic FSM with $|Q|$ states and $|\Sigma|$ events has up to $|Q| |\Sigma|$ transitions. This gives a worst-case space complexity of $O(2(2n-1)|Q| |\Sigma|) = \Theta(n |Q| |\Sigma|)$, which is linear in the number $n$ of FSMs, the number $|Q|$ of states of the largest FSM, and the number $|\Sigma|$ of events after renaming. However, the latter $|\Sigma|$ may be exponential in the number $n$ of FSMs.

### 3.2 State Map-Based Supervisors

Another approach to synthesis is to construct the control function (1) directly without the use of FSMs. The control function can be defined by adding conditions to transitions [8], or by calculating a formula representing the allowable states [3]. Similarly, the above compositional approach can be modified to construct a supervisor in the form of a state map as explained below. Technical details of this approach can be found in [11].

Consider again the example system $G = A \| B \| C$ in Fig. 1. In the first abstraction step, the FSM $C$ is abstracted to $\tilde{C}$ in Fig. 2. The abstraction is described by a state map $\mu_1$ in Fig. 5, which links the concrete states $c_i$
This is an unsafe state, so the supervisor disables make control decisions. For example, assume the FSMs labeled and nonblocking supervisor, and this is enough to set of states reached under the least restrictive control- 

The combination of the maps in Fig. 5 represents the

The next step to compose \( A \parallel B \) is represented by a second state map \( \mu_2 \), which links pairs of states of \( A \) and \( B \) to composed states of \( A|B \). Then \( A|B \) is abstracted to \( \hat{A}B \), represented by state map \( \mu_3 \). With map-based supervisors, \( \hat{A}B \) in Fig. 3 can be simplified further using transition removal [9] by deleting transitions \( p_0 \rightarrow p_0 \) and \( p_1 \rightarrow p_0 \). For example, since \( A \) is a local uncontrollable event, \( p_1 \rightarrow p_0 \) is implied by the sequence \( p_1 \rightarrow p_2 \rightarrow p_0 \). Afterwards, the composition \( A \parallel B \) and its abstraction to \( \hat{A}B \) is represented by two further maps \( \mu_4 \) and \( \mu_5 \) in Fig. 5.

The combination of the maps in Fig. 5 represents the set of states reached under the least restrictive controllable and nonblocking supervisor, and this is enough to make control decisions. For example, assume the FSMs in Fig. 1 are in states \((a_0, b_0, c_0)\). Controllable events \( \alpha \) and \( \beta \) are possible in this state. Event \( \alpha \) would lead the system to \((a_0, b_0, c_2)\). To look up this state, first the state pair \((a_0, b_0)\) is combined to \( p_0 \) according to map \( \mu_2 \), and \( p_0 \) is mapped to \( p_0 \) by \( \mu_3 \). Moreover, state \( c_2 \) is mapped to \( c_1' \) by \( \mu_1 \). Next, the combination \( p_0c_1' \) is mapped to \( q_3 \) by \( \mu_4 \), which is the final map \( \mu_5 \) maps to \( q_3 \). This means that event \( \alpha \) takes the system to a safe state in the final abstraction \( \hat{A}B \), so the supervisor enables \( \alpha \). On the other hand, event \( \beta \) would lead the system to \((a_1, b_1, c_0)\), which is mapped \( \perp \) after \( \mu_3 \). This is an unsafe state, so the supervisor disables \( \beta \).

The state map-based supervisor interacts with the system following the control architecture in Fig. 6. When the system executes an event \( \alpha \in \Sigma \), the supervisor first updates its state \( x \) using the transition function \( \delta : Q \times \Sigma \rightarrow Q \). Then the new control decision is calculated by finding, for each controllable event \( \gamma \in \Sigma \), the successor state \( y = \delta(x, \gamma) \) and, if it is defined, checking the maps to see whether it is a safe state. If the successor state \( y \) is defined and not mapped to the unsafe state \( \perp \), then the controllable event \( \gamma \) is enabled by including it in the control decision \( \Gamma \), otherwise \( \gamma \) is disabled.

The cascaded map representing the final supervisor can be compacted further. For example, in Fig. 5, the output of \( \mu_2 \) can be fed directly into \( \mu_3 \) to obtain a composed map \( \mu_23 = \mu_3 \circ \mu_2 \) that replaces \( \mu_2 \) and \( \mu_3 \), and likewise \( \mu_4 \) and \( \mu_5 \) can be replaced with \( \mu_45 = \mu_5 \circ \mu_4 \). Moreover, all unsafe state entries \( \perp \) can be removed, if the absence of an entry for a given state is interpreted as that state being unsafe. The compacted maps are shown in Fig. 7.

The space complexity of the state map-based supervisor is determined by the number of maps and their size. If there are \( n \) FSMs, this gives up to \( 2n - 1 \) abstraction steps as before, each of which produces one compacted map. The map size is determined by the number of states. The worst-case complexity to store all maps is \( O((2n - 1)|Q|) = O(n|Q|) \), where \( |Q| \) is the number of states of the largest FSM. This is linear in the numbers of FSMs and their states, but unlike FSM-based supervisors, space complexity does not depend on the number of events. This analysis suggests that state map-based supervisors are smaller than FSM-based supervisors. They are also easier to compute, because renaming is avoided and transition removal is possible.

### 4 Experimental Results

The FSM-based and state map-based compositional synthesis algorithms are implemented in the discrete event systems tool Supremica [1], and both implementations have been used to compute supervisors for several large discrete event system models. The test cases include complex industrial models and case studies from different application areas such as manufacturing systems and automotive body electronics, most of which are also used as benchmarks in [10].
Table 1. Experimental results.

<table>
<thead>
<tr>
<th>Model</th>
<th>FSMs</th>
<th>State Maps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>States</td>
<td>Time</td>
</tr>
<tr>
<td>agv</td>
<td>2.6 · 10^3</td>
<td>0.17 s</td>
</tr>
<tr>
<td>agvb</td>
<td>2.3 · 10^3</td>
<td>0.26 s</td>
</tr>
<tr>
<td>api0alps</td>
<td>3.0 · 10^3</td>
<td>0.04 s</td>
</tr>
<tr>
<td>fencaiwn09b</td>
<td>8.9 · 10^3</td>
<td>0.05 s</td>
</tr>
<tr>
<td>fencaiwn09a</td>
<td>2.9 · 10^3</td>
<td>0.06 s</td>
</tr>
<tr>
<td>fms2003</td>
<td>1.7 · 10^3</td>
<td>26.10 s</td>
</tr>
<tr>
<td>psli0g</td>
<td>3.9 · 10^3</td>
<td>0.05 s</td>
</tr>
<tr>
<td>psli_restart</td>
<td>3.9 · 10^3</td>
<td>0.26 s</td>
</tr>
<tr>
<td>psli_partleft</td>
<td>7.7 · 10^3</td>
<td>42.07 s</td>
</tr>
<tr>
<td>tbed_hisc1</td>
<td>2.9 · 10^3</td>
<td>3.90 s</td>
</tr>
<tr>
<td>tbed_noderailb</td>
<td>3.2 · 10^5</td>
<td>9.98 s</td>
</tr>
<tr>
<td>tbed_uncont</td>
<td>3.6 · 10^5</td>
<td>9.06 s</td>
</tr>
<tr>
<td>verriego13b</td>
<td>1.3 · 10^5</td>
<td>0.42 s</td>
</tr>
<tr>
<td>verriego14b</td>
<td>6.2 · 10^10</td>
<td>0.63 s</td>
</tr>
<tr>
<td>6linka</td>
<td>2.4 · 10^4</td>
<td>0.53 s</td>
</tr>
<tr>
<td>6linki</td>
<td>2.7 · 10^4</td>
<td>0.26 s</td>
</tr>
<tr>
<td>6linkp</td>
<td>4.2 · 10^4</td>
<td>0.55 s</td>
</tr>
<tr>
<td>6linkks</td>
<td>6.2 · 10^10</td>
<td>0.11 s</td>
</tr>
</tbody>
</table>

Table 1 shows the name of each test case (Name) and the number of reachable states of the uncontrolled system (States), followed by statistics about FSM-based and state map-based synthesis, namely the runtime (Time), an estimate of the memory needed to store the computed supervisor (Size), and the total number of transitions of all FSMs encountered by the algorithm (Trans).

The memory estimates (Size) are calculated as follows. For FSM-based supervisors, each state counts as one unit of memory, and each transition counts as two units of memory, estimating the amount of memory to form transition lists indexed by source states. For state map-based supervisors, two types of maps are distinguished. Maps resulting from the abstraction of a single FSM, such as $\mu_1$ in Fig. 7, can be stored as an array. For example, $\mu_1$ is stored as the sequence of states $\langle c_0, c_0', c_1, c_1', c_2', c_2 \rangle$ of $\hat{C}$, which are associated to the states $c_i$ of $\hat{C}$ by their index $i$. Differently, maps resulting from synchronous composition only cover the reachable states and are better stored as associative maps. A memory-efficient representation of map $\mu_{23}$ in Fig. 7 is the ordered sequence $\langle a_0, b_0, p_0, a_1, b_1, p_2 \rangle$. The abstracted state $p_0$ for the original state $\langle a_0, b_0 \rangle$, e.g., can be found using binary search. Based on this, the array for map $\mu_1$ uses 5 units of memory, while the associative maps for $\mu_{23}$ and $\mu_{45}$ use 9 and 18 units of memory respectively.

The performance of the compositional algorithms is sensitive to the order in which FSMs are composed and simplified. The experiments use a two-step heuristic approach. The first step computes a set of candidates, i.e., groups of FSMs to be considered for composition, and the second step selects a most promising candidate.

Four heuristics [12] are considered for the first step. MustL considers for every event $\sigma$ the set of FSMs using $\sigma$ as a possible candidate. This ensures that there is at least one local event, namely $\sigma$, after composition. Pairs considers as candidates all FSM pairs that have at least one common event. MinT considers as candidates all FSM pairs containing the FSM with the fewest transitions. MaxS considers as candidates all FSM pairs containing the FSM with the most states.

The second step of candidate selection attempts to identify the best candidate among the set of candidates from the first step. Six alternatives [12] are considered. MaxL chooses the candidate with the highest proportion of local events. MinE chooses the candidate with the minimum number of events. MaxC chooses the candidate with the highest proportion of events shared between FSMs of the candidate. MinS chooses the candidate with the smallest estimated number of states after abstraction. The estimate is obtained by multiplying the product of the state numbers of the FSMs forming the candidate with the ratio of the number of events the candidate shares with other FSMs over the total number of events of the candidate. MinSync computes the synchronous composition of the FSMs in each candidate and chooses the candidate with the fewest states in the synchronous composition. MinF chooses the candidate with the smallest number of other FSMs linked via events to the candidate’s FSMs, in an attempt to minimise event sharing between the candidate and the rest of the system.

In the experiments, synthesis was attempted for all 24 possible combinations of first and second step heuristics. Table 1 lists the result with the smallest supervisor in each case. All experiments were run on a standard desktop PC using a single 3.3 GHz microprocessor and not more than 2 GiB of RAM.

Table 1 shows that most state map-based supervisors use less memory than FSM-based supervisors, as expected. There are a few exceptions such as tbed_hisc1 and tbed_noderailb where the FSM-based supervisors are smaller. In these cases, closer analysis shows that the FSM-based supervisor consists of only a few FSMs that have been simplified substantially, whereas the state map-based supervisor has several large maps from intermediate steps.

State map-based synthesis usually runs faster and encounters substantially less transitions than FSM-based synthesis, which highlights the advantages of transition removal and not renaming. Yet again, there are examples such as api0alps where state-map based synthesis is slower. Usually, the avoidance of renaming results in fewer transitions and better performance, but occasionally the smaller event numbers seem to confuse the selection heuristics and trigger poor decisions.

5 Conclusions

The compositional synthesis of least restrictive, controllable, and nonblocking supervisors in the form of
modular finite-state machines and cascaded state maps has been investigated. Compositional synthesis usually completes in a few seconds, even for examples with $10^{17}$ states. This enables users to attempt synthesis with different parameters and use the best result for their application. While state machines have a standard structure that may help with further processing or optimisation, the complexity analysis and experiments in this paper suggest that state maps in many cases are smaller than state machine-based supervisors. This aspect is particularly important when the supervisor is implemented in a limited-memory device such as a PLC.

References


