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

Sahar Mohajerani [a] Robi Malik [b] Martin Fabian [c]

[a] *Vehicle Dynamics and Active Safety Center, Volvo Cars Corporation, Göteborg, Sweden*

[b] *Department of Computer Science, University of Waikato, Hamilton, New Zealand*

[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 [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 [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 [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* [8]. Petri net supervisors can be represented as *linear constraints* [7].

This paper proposes an improved version of the compositional method [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 [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 [8]. It is shown to be smaller than state machine-based supervisors [10], both by theoretic complexity analysis and by experiments. The new algorithm also performs better, because it permits nondeterminism, hiding, and transition removal [9], and avoids renaming.

The improved compositional synthesis algorithm has been implemented in the DES software tool Supremica [1], and its performance has been compared with the older algorithm [10]. Both algorithms successfully compute supervisors, even for systems with more than 10^{17} reachable states, within some seconds. In most cases, supervisor maps are calculated faster and require less memory than state machine-based supervisors.

Email addresses: sahar.mohajerani@volvocars.com (Sahar Mohajerani), robi@waikato.ac.nz (Robi Malik), fabian@chalmers.se (Martin Fabian).

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 $G = \langle \Sigma, Q, Q^\circ, Q^\omega, \to \rangle$, where Σ is a finite set of *events*, Q is a finite set of *states*, $Q^\circ \subseteq Q$ is the set of *initial states*, $Q^\omega \subseteq Q$ is the set of *accepting states*, and $\to \subseteq Q \times \Sigma \times Q$ 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 $X \xrightarrow{s} Y$ means $x \xrightarrow{s} y$ for some states $x \in X$ and $y \in Y$, and $X \to Y$ means $X \xrightarrow{s} Y$ for some $s \in \Sigma^*$. As events not in the event set of an FSM are assumed to be always enabled without state change, the transition relation is further extended by letting $x \xrightarrow{\sigma} x$ for all $x \in Q$ and $\sigma \notin \Sigma$.

For the purpose of control, the event set Σ is partitioned into the sets Σ_c of *controllable* events and Σ_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*

$$\Phi \colon Q \to 2^{\Sigma_\mathrm{c}} \tag{1}$$

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 $G_1 = \langle \Sigma, Q_1, Q_1^\circ, Q_1^\omega, \to_1 \rangle$ is a sub-FSM of $G = \langle \Sigma, Q, Q^\circ, Q^\omega, \to \rangle$, if $Q_1 \subseteq Q$, $Q_1^\circ \subseteq Q^\circ$, $Q_1^\omega \subseteq Q^\omega$, and $\to_1 \subseteq \to$. *Supervisory control theory* [13] is concerned with the question whether there exists a supervisor Φ 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 $\upsilon \in \Sigma_\mathrm{u}$ such that $x \xrightarrow{\upsilon} y$ in G, it also holds that $x \xrightarrow{\upsilon}_1 y$ in G_1 [3].
- An FSM $G = \langle \Sigma, Q, Q^\circ, Q^\omega, \to \rangle$ is *nonblocking*, if for every state $x \in Q$ such that $Q^\circ \to x$ it holds that $x \to Q^\omega$ [13].

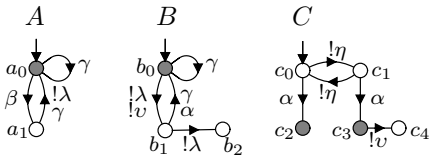

Fig. 1. Example system $G = A \parallel B \parallel C$. Uncontrollable events are prefixed with !, and accepting states are grey.

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 nonblocking. 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 $\sup\mathcal{C}(G)$, such that $\sup\mathcal{C}(G)$ is both controllable in G and nonblocking. Thus, $\sup\mathcal{C}(G)$ can be used to construct a *least restrictive* supervisor that controls G in a nonblocking way [3]. The process of computing $\sup\mathcal{C}(G)$ 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 $G_1 = \langle \Sigma_1, Q_1, Q_1^\circ, Q_1^\omega, \to_1 \rangle$ and $G_2 = \langle \Sigma_2, Q_2, Q_2^\circ, Q_2^\omega, \to_2 \rangle$ is $G_1 \parallel G_2 = \langle \Sigma_1 \cup \Sigma_2, Q_1 \times Q_2, Q_1^\circ \times Q_2^\circ, Q_1^\omega \times Q_2^\omega, \to \rangle$, where $(x_1, x_2) \xrightarrow{\sigma} (y_1, y_2)$ 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 Φ (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 Φ.

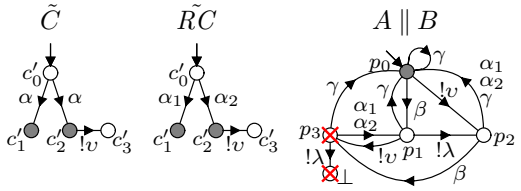

Fig. 2. Abstractions during compositional synthesis.

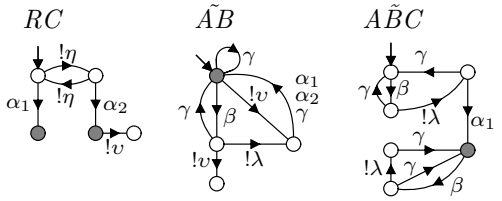

Fig. 3. FSM-based supervisor.

3.1 FSM-Based Supervisors

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 Φ 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 α-transitions from state c'_0, which may prevent the extraction of supervisor components in later steps. Therefore event α is replaced using *renaming* with two new events α_1 and α_2 [10], as shown in $\tilde{R}\tilde{C}$ in Fig. 2. At the same time, the α-transition in B is replaced by two transitions with α_1 and α_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 $A \parallel B$ in Fig. 2. Then events γ and $!\lambda$ are local, as they appear only in A and B, and therefore states $\perp$ and p_3, 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 *unsupervisabiliy removal* [15] to produce a second supervisor component $\tilde{A}B$ as shown in Fig. 3. The final composition $\tilde{A}B \parallel \tilde{R}\tilde{C}$ has ten states, four of which are removed by unsupervisabiliy removal to obtain the third supervisor component $A\tilde{B}\tilde{C}$ in Fig. 3. The FSM $A\tilde{B}\tilde{C}$ always disables the renamed version α_2 of the original event α.

The supervisor FSMs in synchronous composition with the renamed system produce the behaviour of the least

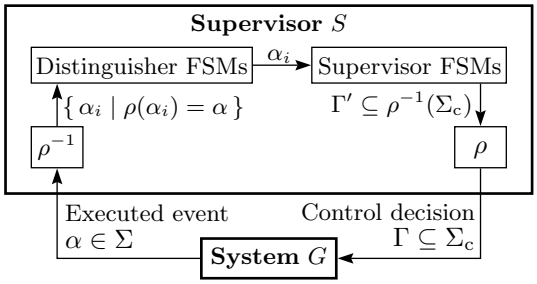

Fig. 4. FSM-based control architecture [10].

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 α, which has been been renamed and replaced by α_1 and α_2. When the supervisor receives the event α, it first uses the *inverse renaming* ρ^{-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 ($\tilde{A}B$ and $A\tilde{B}\tilde{C}$), which update their states and then form a control decision Γ' by collecting the enabled controllable events. This decision is based on renamed events, and the renaming ρ 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 $O(2(2n-1)|Q||\Sigma|) = O(n|Q||\Sigma|)$, 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 $G = A \parallel B \parallel C$ 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* μ_1 in Fig. 5, which links the concrete states c_i

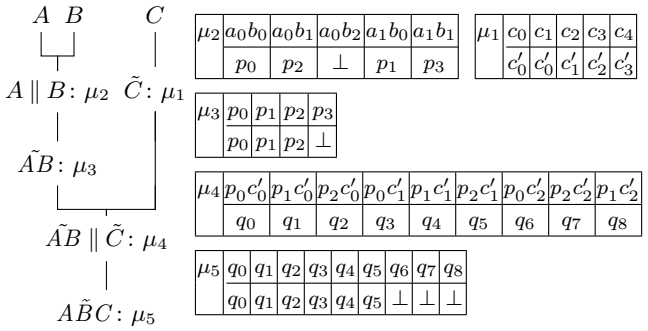

Fig. 5. State map-based supervisor.

of C to the abstract states c'_i of $\tilde{C}$. The map is used instead of the FSM to construct the supervisor, and the nondeterministic FSM $\tilde{C}$ is carried forward without renaming α.

The next step to compose $A\|B$ is represented by a second state map μ_2, which links pairs of states of A and B to composed states of $A\|B$. Then $A\|B$ is abstracted to $\tilde{AB}$, represented by state map μ_3. With map-based supervisors, $\tilde{AB}$ in Fig. 3 can be simplified further using *transition removal* [9] by deleting transitions $p_0 \xrightarrow{\gamma} p_0$ and $p_1 \xrightarrow{\gamma} p_0$. For example, since $!\lambda$ is a local uncontrollable event, $p_1 \xrightarrow{\gamma} p_0$ is implied by the sequence $p_1 \xrightarrow{!\lambda} p_2 \xrightarrow{\gamma} p_0$. Afterwards, the composition $\tilde{AB} \,\|\, \tilde{C}$ and its abstraction to $\tilde{ABC}$ is represented by two further maps μ_4 and μ_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_1, c_0). Controllable events α and β are possible in this state. Event α 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 μ_2, and p_0 is mapped to p_0 by map μ_3. Moreover, state c_2 is mapped to c'_1 by map μ_1. Next, the combination $p_0c'_1$ is mapped to q_3 by μ_4, which the final map μ_5 maps to q_3. This means that event α takes the system to a safe state in the final abstraction $\tilde{ABC}$, so the supervisor enables α. On the other hand, event β would lead the system to (a_1, b_1, c_0), which is mapped $\perp$ after μ_3. This is an unsafe state, so the supervisor disables β.

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 $\delta\colon Q \times \Sigma \to Q$. Then the new control decision is calculated by finding, for each controllable event $\gamma \in \Sigma_c$ 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 γ is enabled by including it

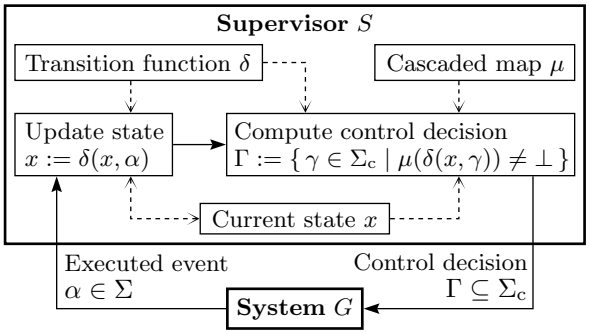

Fig. 6. State map-based control architecture.

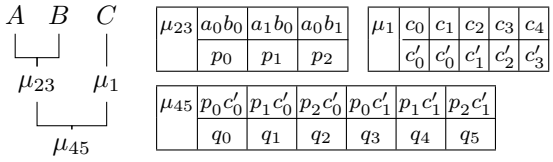

Fig. 7. Compacted state maps.

in the control decision Γ, otherwise γ is disabled.

The cascaded map representing the final supervisor can be compacted further. For example, in Fig. 5, the output of μ_2 can be fed directly into μ_3 to obtain a composed map $\mu_{23} = \mu_3 \circ \mu_2$ that replaces μ_2 and μ_3, and likewise μ_4 and μ_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.

Model		FSMs			State Maps		
Name	States	Time	Size	Trans	Time	Size	Trans
agv	$2.6 \cdot 10^7$	0.17 s	19729	28516	0.11 s	5585	18897
agvb	$2.3 \cdot 10^7$	0.30 s	27610	38979	0.09 s	4305	17013
aip0alps	$3.0 \cdot 10^8$	0.04 s	360	9142	14.52 s	209	197731
fencaiwon09b	$8.9 \cdot 10^7$	0.05 s	5097	8925	0.05 s	4185	6800
fencaiwon09s	$2.9 \cdot 10^8$	0.06 s	9626	7866	0.03 s	3615	3776
fms2003	$1.7 \cdot 10^7$	26.10 s	265556	405520	23.31 s	78675	140468
psl_big	$3.9 \cdot 10^7$	0.05 s	1997	5668	0.07 s	1721	4403
psl_restart	$3.9 \cdot 10^7$	0.26 s	74240	64045	31.19 s	37562	99763
psl_partleft	$7.7 \cdot 10^7$	42.07 s	603805	687899	2.12 s	111313	37846
tbed_hisc1	$2.9 \cdot 10^{17}$	3.90 s	68431	559409	3.17 s	112361	212142
tbed_noderailb	$3.2 \cdot 10^{12}$	9.08 s	498	1215023	4.54 s	1658	987372
tbed_uncont	$3.6 \cdot 10^{12}$	9.06 s	131469	1217637	3.48 s	132644	252952
verriegel3b	$1.3 \cdot 10^9$	0.42 s	34	93535	0.33 s	42	21590
verriegel4b	$6.2 \cdot 10^{10}$	0.63 s	34	146089	3.99 s	42	212035
6linka	$2.4 \cdot 10^{14}$	0.53 s	66016	114109	0.41 s	9045	7650
6linki	$2.7 \cdot 10^{14}$	0.26 s	98280	152248	0.13 s	13002	4960
6linkp	$4.2 \cdot 10^{14}$	0.55 s	91785	162335	0.41 s	8776	6849
6linkre	$6.2 \cdot 10^{14}$	0.11 s	7153	14114	0.10 s	2027	1900

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 μ_1 in Fig. 7, can be stored as an array. For example, μ_1 is stored as the sequence of states $\langle c'_0, c'_0, c'_1, c'_2, c'_3 \rangle$ of $\tilde{C}$, which are associated to the states c_i of 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 μ_{23} in Fig. 7 is the ordered sequence $\langle a_0, b_0, p_0, a_1, b_0, p_1, a_0, b_1, p_2 \rangle$. The abstracted state p_0 for the original state (a_0, b_0), e.g., can be found using binary search. Based on this, the array for map μ_1 uses 5 units of memory, while the associative maps for μ_{23} and μ_{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 σ the set of FSMs using σ as a possible candidate. This ensures that there is at least one local event, namely σ, 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 aip0alps 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

[1] Åkesson, K., Fabian, M., Flordal, H., Malik, R.: Supremica—an integrated environment for verification, synthesis and simulation of discrete event systems. In: Proc. 8th Int. Workshop on Discrete Event Systems, WODES '06. pp. 384–385. IEEE (Jul 2006)

[2] Feng, L., Wonham, W.M.: Supervisory control architecture for discrete-event systems. IEEE Trans. Autom. Control 53(6), 1449–1461 (Jul 2008)

[3] 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)

[4] Graf, S., Steffen, B.: Compositional minimization of finite state systems. In: Proc. 1990 Workshop on Computer-Aided Verification. LNCS, vol. 531, pp. 186–196. Springer (Jun 1990)

[5] Harel, D.: Statecharts: a visual formalism for complex systems. Sci. Comput. Programming 8(3), 231–274 (1987)

[6] Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall (1985)

[7] Luo, J., Nonami, K.: Approach for transforming linear constraints on Petri nets. IEEE Trans. Autom. Control 56(12), 2751–2765 (Dec 2011)

[8] Miremadi, S., Åkesson, K., Lennartson, B.: Symbolic computation of reduced guards in supervisory control. IEEE Trans. Autom. Sci. Eng. 8(4), 754–764 (Oct 2011)

[9] Mohajerani, S., Malik, R., Fabian, M.: Transition removal for compositional supervisor synthesis. In: Proc. 8th Int. Conf. Automation Science and Engineering, CASE 2012. pp. 690–695 (Aug 2012)

[10] 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)

[11] Mohajerani, S., Malik, R., Fabian, M.: Compositional supervisor synthesis with state merging and transition removal. Working Paper 02/2016, Dept. of Computer Science, University of Waikato, Hamilton, New Zealand (2016), `http://hdl.handle.net/10289/9889`

[12] Pilbrow, C., Malik, R.: An algorithm for compositional nonblocking verification using special events. Sci. Comput. Programming 113(2), 119–148 (Dec 2015)

[13] Ramadge, P.J.G., Wonham, W.M.: The control of discrete event systems. Proc. IEEE 77(1), 81–98 (Jan 1989)

[14] Schmidt, K., Breindl, C.: Maximally permissive hierarchical control of decentralized discrete event systems. IEEE Trans. Autom. Control 56(4), 723–737 (Apr 2011)

[15] Ware, S., Malik, R., Mohajerani, S., Fabian, M.: Certainly unsupervisable states. In: Proc. 2nd Int. Workshop on Formal Techniques for Safety-Critical Systems, FTSCS 2013. pp. 3–18 (2013)