Supervisor Reduction by Hiding Events

Robi Malik

Department of Computer Science, University of Waikato, Hamilton, New Zealand (e-mail: robi@waikato.ac.nz)

Abstract: This paper proposes a method to improve *supervisor reduction* for discrete event systems by first reducing the number of events. Supervisor reduction is a method to reduce the number of states of an automatically computed supervisor or controller in order to make it more manageable. This paper proposes to complement the most popular supervisor reduction algorithm currently in use by first reducing the supervisor's event set. Experimental results show that this does not only reduce the communication between the supervisor and plant, but also produces a simpler state machine that can be minimised more effectively.

1. INTRODUCTION

Given a finite-state model of a *plant* or a system to be controlled, *supervisory control theory* (Ramadge and Wonham, 1989) is concerned with means to restrict the plant's behaviour to enforce certain requirements. This can be achieved by *synthesising* or automatically computing another finite-state machine, called *supervisor*.

This supervisor is usually constructed by deleting transitions from the plant so as to avoid undesirable behaviour. Then it has the same state space as the plant, which can be very large—millions of states or more. This can pose problems for humans trying to understand the supervisor, or when implementing it on devices with limited memory. As a remedy, Vaz and Wonham (1986) propose *supervisor reduction*, which can replace a given supervisor by a smaller supervisor with equivalent behaviour.

Given a plant and supervisor, the goal of supervisor reduction is to find another supervisor which achieves the same controlled behaviour and has the fewest states possible. This problem is different from standard state machine minimisation problems (Hopcroft et al., 2001), because the supervisors can exploit the presence of the plant to use fewer states. Su and Wonham (2004) show that the problem to find a smallest equivalent supervisor is NP-hard.

Vaz and Wonham (1986) propose an algorithm to compute a reduced supervisor based on covers, but their method is exponential. Su and Wonham (2004) propose a polynomial algorithm that computes a reduced but not necessarily minimal supervisor. Their algorithm is the most popular supervisor reduction algorithm currently in use due to its trade-off between speed and reduction effectiveness. It will be referred to as the *SW Algorithm* in this paper. The SW Algorithm repeatedly merges pairs of states and is sensitive to the order in which state pairs are processed. Its result can be seen as arbitrary as the order often is.

This paper proposes to amend the arbitrariness by adding a preprocessing step that reduces the number of events. An automatically computed supervisor typically uses all events of the plant model, while only a part is needed to make the control decisions. By identifying a reasonable subset, it is not only possible to reduce the communication between the plant and supervisor, but also to produce a smaller and more reasonable supervisor that can be minimised more effectively with the SW Algorithm.

Cai and Wonham (2016) propose *supervisor localisation* among other improvements to the SW Algorithm. They also remove unnecessary events, but only after using the SW Algorithm. This paper proposes to remove events before invoking the SW Algorithm.

After introducing the relevant terminology of supervisory control theory in Section 2, Section 3 describes the proposed method, which consists of an algorithm to determine whether a given set of events is sufficient to make control decisions, and algorithms to find such event sets. Afterwards, Section 4 shows experimental results that demonstrate the effectiveness of the approach, and Section 5 adds concluding remarks.

2. PRELIMINARIES

2.1 Languages and Finite-State Machines

Event sequences and languages are a simple means to describe discrete system behaviours. Their basic building blocks are *events*, which are taken from a finite *alphabet* Σ. The *silent event* τ labels transitions that are only taken by the component under consideration. Σ^* denotes the set of all finite *traces* of the form $\sigma_1\sigma_2\cdots\sigma_n$ of events from Σ, including the *empty trace* ε. The *concatenation* of two traces $s, t \in \Sigma^*$ is written as st.

Definition 1. A *finite-state machine (FSM)* is a tuple $G = \langle \Sigma_G, Q_G, Q_G^\circ, \rightarrow_G \rangle$ where $\Sigma_G \subseteq \Sigma$ with $\tau \notin \Sigma_G$ is an event set, Q_G is a finite set of *states*, $Q_G^\circ \subseteq Q_G$ is the set of *initial states*, and $\rightarrow_G \subseteq Q_G \times (\Sigma_G \cup \{\tau\}) \times Q_G$ is the *transition relation*.

The transition relation is written in infix notation $x \xrightarrow{\sigma}_G y$ and extended to traces $s \in (\Sigma_G \cup \{\tau\})^*$ in the standard way. For a state $x \in Q_G$ and trace $s \in (\Sigma_G \cup \{\tau\})^*$, the notation $G \xrightarrow{s}_G x$ means $x_G^\circ \xrightarrow{s}_G x$ for some $x^\circ \in Q_G^\circ$, and $x \xrightarrow{s}_G$ means $x \xrightarrow{s}_G y$ for some $y \in Q_G$. Also, $x \not\xrightarrow{s}_G$ means that $x \xrightarrow{s}_G$ does not hold, $x \rightarrow y$ means $x \xrightarrow{s}_G y$ for some $s \in (\Sigma_G \cup \{\tau\})^*$, and a state x is *reachable* in G if $G \rightarrow_G x$.

Events not in the event set of an FSM are always enabled without state change, so the transition relation is further extended by $x \xrightarrow{\sigma}_G x$ for all $x \in Q$ and $\sigma \in (\Sigma \setminus \Sigma_G) \cup \{\tau\}$.

The FSM G is *deterministic* if it has exactly one initial state, $|Q_G^\circ| = 1$, no silent transitions, $\rightarrow_G \subseteq Q_G \times \Sigma_G \times Q_G$, and there is at most one transition from every state with a given event, i.e., $x \xrightarrow{\sigma}_G y$ and $x \xrightarrow{\sigma}_G z$ implies $y = z$.

2.2 Projection, Hiding, and Composition

Given a sub-alphabet $\Upsilon \subseteq \Sigma$, the behaviour of an FSM without the events in Υ is of interest. The *projection* $P_{\Sigma\to\Sigma\setminus\Upsilon}\colon \Sigma^* \to (\Sigma \setminus \Upsilon)^*$ is the operation that deletes all events in Υ from traces. Its inverse image map is

$$P^{-1}_{\Sigma\to\Sigma\setminus\Upsilon}\colon (\Sigma\setminus\Upsilon)^* \to 2^{\Sigma^*}; s \mapsto \{\, t \in \Sigma^* \mid P_{\Sigma\to\Sigma\setminus\Upsilon}(t) = s \,\}.$$

As a special case, the *natural projection* $P_\tau = P_{\Sigma\to\Sigma\setminus\{\tau\}}$ deletes silent (τ) events from traces. The FSM operation corresponding to projection is *hiding*.

Definition 2. Let $G = \langle \Sigma_G, Q_G, Q_G^\circ, \rightarrow_G\rangle$ be an FSM, and let $\Upsilon \subseteq \Sigma$. The result of *hiding* Υ from G is the FSM $G\setminus\Upsilon = \langle \Sigma_G \setminus \Upsilon, Q_G, Q_G^\circ, \rightarrow_{\Sigma\setminus\Upsilon}\rangle$, where $\rightarrow_{\Sigma\setminus\Upsilon}$ is obtained from $\rightarrow_G$ by replacing every transition $x \xrightarrow{\upsilon}_G y$ with $\upsilon \in \Upsilon$ by $x \xrightarrow{\tau}_{\Sigma\setminus\Upsilon} y$.

Hiding replaces transitions with events from Υ by silent τ-transitions, producing a nondeterministic FSM. To define the behaviour of such an FSM, another transition relation $\Rightarrow_G \subseteq Q_G \times \Sigma^* \times Q_G$ is introduced such that $x \xRightarrow{s}_G y$ if and only if there exists $t \in (\Sigma \cup \{\tau\})^*$ such that $P_\tau(t) = s$ and $x \xrightarrow{t}_G y$. Thus, $x \xRightarrow{s}_G y$ indicates a sequence of transitions from x to y using the events of s and possibly additional τ-transitions. Notation such as $G \xRightarrow{s}_G x$ is defined analogously to $\rightarrow$.

The *language* of an FSM $G = \langle \Sigma_G, Q_G, Q_G^\circ, \rightarrow_G\rangle$ is the set of traces starting from initial states without silent events,

$$\mathcal{L}(G) = \{\, s \in (\Sigma \setminus \{\tau\})^* \mid G \xRightarrow{s}_G \,\} \,. \tag{1}$$

This definition uses the extended definition of the transition relation, $x \xrightarrow{\sigma}_G x$ for $\sigma \in \Sigma \setminus \Sigma_G$. In this paper, the language $\mathcal{L}(G)$ is always defined over the same event set Σ, assumed fixed and including the event sets Σ_G of all FSMs under consideration.

A nondeterministic FSM such as $G\setminus\Upsilon$ can be transformed to a deterministic FSM that accepts the same language. In the following, $\det(G)$ denotes the minimal deterministic FSM that accepts the same language as G, i.e., $\det(G)$ is the deterministic FSM with the fewest states such that $\mathcal{L}(\det(G)) = \mathcal{L}(G)$. It can be computed by well-known algorithms (Hopcroft et al., 2001).

FSMs are synchronised in lock-step (Hoare, 1985).

Definition 3. The *synchronous composition* of two FSMs $G_1 = \langle \Sigma_1, Q_1, Q_1^\circ, \rightarrow_1\rangle$ and $G_2 = \langle \Sigma_2, Q_2, Q_2^\circ, \rightarrow_2\rangle$ is

$$G_1 \parallel G_2 \;=\; \langle \Sigma_1 \cup \Sigma_2, Q_1 \times Q_2, Q_1^\circ \times Q_2^\circ, \rightarrow\rangle \tag{2}$$

where

$$(x_1, x_2) \xrightarrow{\sigma} (y_1, y_2) \quad \text{if } \sigma \neq \tau,\ x_1 \xrightarrow{\sigma}_1 y_1,\ x_2 \xrightarrow{\sigma}_2 y_2\,; \tag{3}$$

$$(x_1, x_2) \xrightarrow{\tau} (y_1, x_2) \quad \text{if } x_1 \xrightarrow{\tau}_1 y_1\,; \tag{4}$$

$$(x_1, x_2) \xrightarrow{\tau} (x_1, y_2) \quad \text{if } x_2 \xrightarrow{\tau}_2 y_2\,. \tag{5}$$

The definition again uses the extended transition relation, $x \xrightarrow{\sigma}_i x$ for $\sigma \notin \Sigma_i$. Accordingly, shared events must be executed by both FSMs together, while events in only one FSM, including τ, are executed separately. The behaviour of two composed FSMs is obtained by language intersection, $\mathcal{L}(G_1 \parallel G_2) = \mathcal{L}(G_1) \cap \mathcal{L}(G_2)$.

2.3 Supervisors and Supervisor Reduction

Given an FSM representing a *plant*, *supervisory control theory* (Ramadge and Wonham, 1989) is concerned with means to restrict its behaviour. This can be achieved through a deterministic FSM S, called *supervisor*, which interacts with the plant G in synchronous composition, $S \parallel G$. This paper is concerned about whether two such supervisors result in the same behaviour.

Definition 4. Let G, S_1, and S_2 be deterministic FSMs. S_1 and S_2 are said to be *control equivalent* with respect to G, written $S_1 \simeq_G S_2$, if $\mathcal{L}(S_1 \parallel G) = \mathcal{L}(S_2 \parallel G)$.

If the supervisor S is computed automatically by *synthesis*, it is usually constructed by deleting certain transitions from the plant, resulting in a *sub-FSM*.

Definition 5. Let $G = \langle \Sigma_G, Q_G, Q_G^\circ, \rightarrow_G\rangle$ be an FSM. An FSM $S = \langle \Sigma_G, Q_S, Q_S^\circ, \rightarrow_S\rangle$ is a *sub-FSM* of G, written $S \sqsubseteq G$, if $Q_S \subseteq Q_G$ and $\rightarrow_S \subseteq \rightarrow_G$ and $Q_S^\circ \subseteq Q_G^\circ$.

Whether or not a supervisor is a sub-FSM of its plant, its control action is determined by what events it enables or disables.

Definition 6. (Su and Wonham, 2004) Let G and S be deterministic FSMs. The sets of events *enabled* or *disabled* by S at state $x \in Q_S$ are:

$$E_S(x) = \{\, \sigma \in \Sigma \mid x \xrightarrow{\sigma}_S \,\}\,; \tag{6}$$

$$D_{S,G}(x) = \{\, \sigma \in \Sigma \mid x \not\xrightarrow{\sigma}_S \text{ and there exists } s \in \Sigma^* \text{ such that } S \xrightarrow{s} x \text{ and } s\sigma \in \mathcal{L}(G) \,\}\,. \tag{7}$$

The set of enabled events $E_S(x)$ is the set of events enabled by the supervisor in its state x. The set of disabled events $D_{S,G}(x)$ is the set of events enabled by the plant, but disabled by the supervisor. These definitions reflect the observation that, if an event is not enabled by the plant, then it cannot occur regardless of whether or not it is enabled by the supervisor. In the special case where S is a sub-FSM of G, the set of disabled events can be written without the condition on reachability in G.

Lemma 1. Let G and S be deterministic FSMs such that $S \sqsubseteq G$. Then it holds for all reachable states $x \in Q_S$ that,

$$D_{S,G}(x) = \{\, \sigma \in \Sigma \mid x \xrightarrow{\sigma}_G \text{ and } x \not\xrightarrow{\sigma}_S \,\}\,. \tag{8}$$

Definition 7. (Su and Wonham, 2004) Let G and S be deterministic FSMs. The set of *control consistent* state pairs of S with respect to G is

$$\mathcal{R}_{S,G} = \{\, (x, y) \in Q_S \times Q_S \mid E_S(x) \cap D_{S,G}(y) = E_S(y) \cap D_{S,G}(x) = \emptyset \,\}\,. \tag{9}$$

Two states are control consistent if no event needs to be enabled in one state and disabled in the other. The SW Algorithm (Su and Wonham, 2004) computes a reduced supervisor after finding a relation that respects Def. 7 and is preserved under the transition relation. This paper uses the same relation to determine whether supervisor reduction can be achieved by hiding events.

3. SUPERVISOR REDUCTION BY EVENT HIDING

Given a plant G and supervisor S, the concern of this paper is to identify a set of events Υ that can be hidden from S without affecting control equivalence. As a first step, Section 3.1 proposes an algorithm to determine for a

given event set Υ whether or not its hiding affects control equivalence. Afterwards, Section 3.2 uses this algorithm to search for a suitable event set Υ.

3.1 Test for Feasible Event Set

Given a plant G, supervisor S, and event set Υ, the goal of this subsection is to determine whether S is control equivalent to $S \setminus \Upsilon$. If so, hiding Υ and the projection $P_{\Sigma \to \Sigma \setminus \Upsilon}$ are said to be *feasible* for supervisor reduction.

More precisely, the result $S \setminus \Upsilon$ of hiding must be converted to its deterministic equivalent $\det(S \setminus \Upsilon)$. As suggested by Cai and Wonham (2016), it is further modified by adding selfloops to retain control decisions for hidden events.

Definition 8. Let $G = \langle \Sigma_G, Q_G, Q_G^\circ, \to_G \rangle$ and $S = \langle \Sigma_S, Q_S, Q_S^\circ, \to_S \rangle$ be FSMs, and let $\Upsilon \subseteq \Sigma$. The *reduced supervisor* $\det(S, G, \Upsilon)$ for S with respect to G and Υ is obtained from $\det(S \setminus \Upsilon)$ by extending the event set to Σ_S and adding transitions $x \xrightarrow{\sigma} x$ for states x and events $\sigma \in \Sigma_S \cap \Upsilon$ if there exists $s \in \Sigma^*$ such that $\det(S \setminus \Upsilon) \xrightarrow{P_{\Sigma \to \Sigma \setminus \Upsilon}(s)} x$ and $s\sigma \in \mathcal{L}(S \parallel G)$.

The reduced supervisor $\det(S, G, \Upsilon)$ has the same structure as $\det(S \setminus \Upsilon)$, but it is defined over all events of S. Hidden events are enabled (without state change) whenever they would be enabled by G and S. Then its language includes that of G and S combined.

Lemma 2. Let G and S be FSMs, and let $\Upsilon \subseteq \Sigma$. Then $\mathcal{L}(S \parallel G) \subseteq \mathcal{L}(\det(S, G, \Upsilon))$.

Hidden events only appear as selfloops in $\det(S, G, \Upsilon)$, so the state only depends on the retained events, $\Sigma \setminus \Upsilon$. For a given trace $s \in (\Sigma \setminus \Upsilon)^*$, there may be more than one matching trace over Σ, and these may lead to different states in $S \setminus \Upsilon$. If these states are associated with different control decisions, then $\det(S, G, \Upsilon)$ does not produce the same control as S.

This suggests that it should be checked for all traces s and all paths $S \setminus \Upsilon \stackrel{s}{\Rightarrow} x$ and $S \setminus \Upsilon \stackrel{s}{\Rightarrow} y$, whether x and y are control consistent, i.e., whether $(x, y) \in \mathcal{R}_{S,G}$. This can be checked with *verifier* algorithms (Yoo and Lafortune, 2002; Pena et al., 2014) in polynomial time. These algorithms construct a *verifier* as the synchronous composition $(S \setminus \Upsilon) \parallel (S \setminus \Upsilon)$, whose reachable states are precisely the pairs (x, y) such that $S \setminus \Upsilon \stackrel{s}{\Rightarrow} x$ and $S \setminus \Upsilon \stackrel{s}{\Rightarrow} y$ for some s. While constructing the verifier, its state pairs are checked. Here, the check is for control consistency (Def. 7), i.e., it is checked whether the following condition holds:

$$\text{For all } x, y \in Q_S \text{ such that } (S \setminus \Upsilon) \parallel (S \setminus \Upsilon) \to (x, y) \text{ it holds that } (x, y) \in \mathcal{R}_{S,G}. \tag{10}$$

Algorithm 1 checks condition (10) for a given supervisor S and event set Υ. It uses the fact that pairs (x, x), for reachable states x of S, are reachable in the verifier and control consistent, and only their successors on hidden events can lead to pairs (x, y) with $x \neq y$. Line 4 calls the recursive procedure on line 6 for these successors (x, y), which checks their control consistency and explores further successors.

Line 7 assumes a pre-defined linear ordering $>$ on the states of S to exploit symmetry, as the pairs (x, y) and (y, x) have the same properties. Line 9 checks for control consistency, and stops the algorithm with a negative result as soon as an inconsistent pair is encountered. Otherwise, lines 11–19 explore the successors, avoiding pairs (x, x) and pairs that have already been checked. After all pairs have been explored without encountering a control inconsistent pair, the algorithm reaches line 5 and reports that hiding Υ preserves control equivalence.

Algorithm 1: Check for Feasibility of Projection

```
1  procedure check(S = ⟨Σ_S, Q_S, Q_S°, →_S⟩, Υ)
2      foreach reachable state x ∈ Q_S do
3          foreach transition x -υ→_S y with υ ∈ Υ do
4              check(x, y)
5      stop "S is control equivalent to S \ Υ."
6  procedure check(x ∈ Q_S, y ∈ Q_S)
7      if x > y then
8          check(y, x)
9      else if (x, y) ∉ R_{S,G} then
10         stop "S is not control equivalent to S \ Υ."
11     else if x ≠ y and (x, y) is not yet checked then
12         mark (x, y) as checked
13         foreach transition x -σ→_S x' with σ ∈ Σ_S \ Υ do
14             foreach transition y -σ→_S y' do
15                 check(x', y')
16         foreach transition x -υ→_S x' with υ ∈ Υ do
17             check(x', y)
18         foreach transition y -υ→_S y' with υ ∈ Υ do
19             check(x, y')
```

To estimate the time complexity of Algorithm 1, it is noted that each pair (x, y) is visited at most once, at most $|Q_S|^2$ pairs. This is the maximum number of times lines 12–19 can be executed. Assuming a deterministic FSM S, the loop on line 13 explores at most one transition for each event $\sigma \in \Sigma_S \setminus \Upsilon$, and the loops on lines 16 and 18 explore at most one transition for each event $\upsilon \in \Upsilon$. As all other operations can be completed in constant time, the worst-case time complexity is $O(|Q_S|^2 |\Sigma|)$.

The remainder of this section proves the correctness of Algorithm 1. To prove this, it is enough to show that condition (10) is equivalent to the control equivalence of S and $\det(S, G, \Upsilon)$. First, Prop. 3 proves that condition (10) implies this control equivalence, confirming that hiding is feasible whenever Algorithm 1 reports a positive result.

Proposition 3. Let G and S be deterministic FSMs, and let $\Upsilon \subseteq \Sigma$. If S satisfies (10), then $S \simeq_G \det(S, G, \Upsilon)$.

Proof. It follows from Lemma 2 that $\mathcal{L}(S \parallel G) \subseteq \mathcal{L}(\det(S, G, \Upsilon) \parallel G)$. By Def. 4, it remains to show that $\mathcal{L}(\det(S, G, \Upsilon) \parallel G) \subseteq \mathcal{L}(S \parallel G)$. Consider $s \in \mathcal{L}(\det(S, G, \Upsilon) \parallel G)$. It is shown by induction on the length of s that $s \in \mathcal{L}(S \parallel G)$.

Base case. It is clear by construction in Def. 8 that $\varepsilon \in \mathcal{L}(\det(S, G, \Upsilon) \parallel G) = \mathcal{L}(\det(S, G, \Upsilon)) \cap \mathcal{L}(G)$ implies $\varepsilon \in \mathcal{L}(S) \cap \mathcal{L}(G) = \mathcal{L}(S \parallel G)$.

Inductive step. Consider $s\sigma \in \mathcal{L}(\det(S, G, \Upsilon) \parallel G)$. Then there exists $x \in Q_S$ such that

$$S \setminus \Upsilon \stackrel{P(s)}{\Longrightarrow} x \xrightarrow{\sigma}, \tag{11}$$

where $P = P_{\Sigma\to\Sigma\setminus\Upsilon}$. Then $\sigma \in E_S(x)$ by (6). From $s\sigma \in \mathcal{L}(\det(S,G,\Upsilon)\,\|\,G)$, it follows that $s \in \mathcal{L}(\det(S,G,\Upsilon)\,\|\,G)$. Thus, $s \in \mathcal{L}(S\,\|\,G)$ by inductive assumption. Then $S \xrightarrow{s} y$ for some $y \in Q_S$, and by (10) and (11) it follows that

$$(S\setminus\Upsilon)\,\|\,(S\setminus\Upsilon) \overset{P(s)}{\Longrightarrow} (x,y) \in \mathcal{R}_{S,G}\ . \tag{12}$$

As $\sigma \in E_S(x)$, it follows by (9) that $\sigma \notin D_{S,G}(y)$. As $S \xrightarrow{s} y$ and $s\sigma \in \mathcal{L}(G)$, it follows from (7) that $y \not\xrightarrow{\sigma}_S$ cannot hold, i.e., $S \xrightarrow{s} y \xrightarrow{\sigma}$. Thus, $s\sigma \in \mathcal{L}(S) \cap \mathcal{L}(G) = \mathcal{L}(S\,\|\,G)$. □

Conversely it is of interest whether Algorithm 1 correctly identifies every feasible event set as such. This only holds under the additional assumption that the supervisor is a sub-FSM of its plant.

Proposition 4. Let G and S be deterministic FSMs such that $S \sqsubseteq G$, and let $\Upsilon \subseteq \Sigma$. If $S \simeq_G \det(S,G,\Upsilon)$, then S satisfies (10).

Proof. Assume that $S \simeq_G \det(S,G,\Upsilon)$, i.e., $\mathcal{L}(S\,\|\,G) = \mathcal{L}(\det(S,G,\Upsilon)\,\|\,G)$, and let $(S\setminus\Upsilon)\,\|\,(S\setminus\Upsilon) \to (x,y)$. It is to be shown that $(x,y) \in \mathcal{R}_{S,G}$, i.e., $E_S(x) \cap D_{S,G}(y) = E_S(y) \cap D_{S,G}(x) = \emptyset$ by (9). Because of symmetry, it is enough to show that $\sigma \in D_{S,G}(x)$ implies $\sigma \notin E_S(y)$.

So assume $\sigma \in D_{S,G}(x)$, i.e., $x \xrightarrow{\sigma}_G$ and $x \not\xrightarrow{\sigma}_S$ by (8). As $S\setminus\Upsilon \to x$ and $S \sqsubseteq G$, there exists $s \in \Sigma^*$ such that $G \xrightarrow{s} x \xrightarrow{\sigma}$ and $S \xrightarrow{s} x \not\xrightarrow{\sigma}$. Then $s\sigma \in \mathcal{L}(G)$ and $s\sigma \notin \mathcal{L}(S)$ as S is deterministic. It follows that $s\sigma \notin \mathcal{L}(S) \cap \mathcal{L}(G) = \mathcal{L}(S\,\|\,G) = \mathcal{L}(\det(S,G,\Upsilon)\,\|\,G)$, i.e., $s\sigma \notin \mathcal{L}(\det(S,G,\Upsilon))$. Also $s \in \mathcal{L}(S\|G) \subseteq \mathcal{L}(\det(S,G,\Upsilon))$ by Lemma 2, and thus $\det(S,G,\Upsilon) \xrightarrow{s} z \not\xrightarrow{\sigma}$. It follows from Def. 8 that $\sigma \notin \Upsilon$, and then $\det(S\setminus\Upsilon) \xrightarrow{s} z \not\xrightarrow{\sigma}$. By determinism it follows that $s\sigma \notin \mathcal{L}(\det(S\setminus\Upsilon)) \cap \mathcal{L}(G) = P^{-1}(P(\mathcal{L}(S))) \cap \mathcal{L}(G)$, where $P = P_{\Sigma\to\Sigma\setminus\Upsilon}$, which given $s\sigma \in \mathcal{L}(G)$ implies $P(s\sigma) \notin P(\mathcal{L}(S))$. As $(S\setminus\Upsilon)\,\|\,(S\setminus\Upsilon) \to (x,y)$ and $S \xrightarrow{s} x$, there exists $t \in \Sigma^*$ such that $P(s) = P(t)$ and $S \xrightarrow{t} y$. But then $y \not\xrightarrow{\sigma}_S$, since otherwise $P(s\sigma) = P(t\sigma) \in P(\mathcal{L}(S))$. It follows from (6) that $\sigma \notin E_S(y)$. □

According to Prop. 4, if a projection is feasible for supervisor reduction, then Algorithm 1 will terminate and report this fact—provided that $S \sqsubseteq G$. Fortunately, most synthesis algorithms construct the supervisor by removing transitions from the plant and ensure $S \sqsubseteq G$.

If the supervisor is obtained differently and $S \sqsubseteq G$ does not hold, then Algorithm 1 only tests a sufficient condition. This is because the control consistency relation $\mathcal{R}_{S,G}$ does not accurately characterise the equivalence of states in this case. Still, if Algorithm 1 reports that an event set is feasible, it can be used for supervisor reduction according to Prop. 4. It is only that some feasible event sets may be missed if $S \sqsubseteq G$ does not hold.

3.2 Search for Feasible Event Set

This subsection discusses ways to find, for a given supervisor S, an event set $\Upsilon \subseteq \Sigma$ that is feasible for supervisor reduction. The straightforward way to find such a set is to check all subsets $\Upsilon \subseteq \Sigma$ using Algorithm 1 and return the largest set that passes the test. Unfortunately, this naive approach has exponential complexity. The search can be simplified using the following proposition, which shows that the feasible event sets are closed under set inclusion.

Algorithm 2: Greedy Search for Projection

```
1  procedure greedySearch(S = ⟨Σ_S, Q_S, Q_S^∘, →_S⟩)
2      Υ ← ∅
3      foreach σ ∈ Σ_S do
4          if S ≃_G det(S \ (Υ ∪ {σ})) then
5              Υ ← Υ ∪ {σ}
6      return Υ
```

Algorithm 3: Exhaustive Search for Projection

```
1  procedure exhaustiveSearch(S = ⟨Σ_S, Q_S, Q_S^∘, →_S⟩)
2      return exhaustiveSearch(Σ_S, ∅, ∅)
3  procedure exhaustiveSearch(Σ', Υ, B ⊆ Σ)
4      if |Σ' ∪ Υ| > |B| then
5          Choose σ ∈ Σ'
6          if S ≃_G det(S \ (Υ ∪ {σ})) then
7              if |Υ ∪ {σ}| > |B| then
8                  B ← Υ ∪ {σ}
9              B ← exhaustiveSearch(Σ' \ {σ}, Υ ∪ {σ}, B)
10         B ← exhaustiveSearch(Σ' \ {σ}, Υ, B)
11     return B
```

Proposition 5. Let G and S be deterministic FSMs, and let $\Upsilon' \subseteq \Upsilon \subseteq \Sigma$. If $S \simeq_G \det(S,G,\Upsilon)$ then $S \simeq_G \det(S,G,\Upsilon')$.

Proof. $\mathcal{L}(S\|G) \subseteq \mathcal{L}(\det(S,G,\Upsilon')\|G)$ holds by Lemma 2. Conversely, $\Upsilon' \subseteq \Upsilon$ implies $\mathcal{L}(\det(S,G,\Upsilon')) \subseteq \mathcal{L}(\det(S,G,\Upsilon))$ by Def. 8. It follows that $\mathcal{L}(\det(S,G,\Upsilon')\,\|\,G) \subseteq \mathcal{L}(\det(S,G,\Upsilon)\,\|\,G) = \mathcal{L}(S\,\|\,G)$ because $S \simeq_G \det(S\setminus\Upsilon)$. It has been shown that $\mathcal{L}(S\,\|\,G) = \mathcal{L}(\det(S,G,\Upsilon')\,\|\,G)$, i.e., $S \simeq_G \det(S,G,\Upsilon')$ by Def. 4. □

According to Prop. 5, if hiding Υ is feasible, then this also holds for all subsets. This also means that, if some set Υ is *not* feasible, there is no need to check its supersets, as they also must be infeasible. This observation gives rise to a greedy search algorithm shown as Algorithm 2.

Algorithm 2 processes the events in a fixed order, trying to add each to the set Υ. If adding an event results in a feasible projection, it is included in Υ, otherwise it is skipped. By Prop. 5, it is clear that Algorithm 2 returns a maximal feasible event set, i.e., a feasible set Υ such that no proper superset $\Upsilon' \supset \Upsilon$ is feasible. Line 4 invokes Algorithm 1 to determine whether hiding Υ is feasible. As this invocation occurs once for each event in Σ, and the worst-case time complexity of Algorithm 1 is $O(|Q_S|^2|\Sigma|)$, it follows that the worst-case time complexity of Algorithm 2 is $O(|Q_S|^2|\Sigma|^2)$.

While the result of Algorithm 2 is maximal, there could be larger feasible event sets that are not supersets of the result. As an alternative, Algorithm 3 performs an exhaustive search to find a feasible event set Υ with the largest cardinality possible. The recursive procedure on line 3 is called with a set Σ' of events still available for inclusion in the set Υ of events being hidden, and a known best set B. If Σ' is not empty, line 5 chooses an event $\sigma \in \Sigma'$ to add to Υ. Line 6 checks whether this is feasible using Algorithm 1. If it is, and the extended set $\Upsilon \cup \{\sigma\}$ exceeds the known best B, it replaces B on line 8; and

line 9 tries to extend the set further. Additionally, line 10 searches for feasible events sets not including the selected event σ. The test on line 4 is a branch-and-bound condition to stop searching when it is clear that the remaining events in Σ' together with those already selected in Υ cannot improve the known best B.

Although the branch-and-bound condition together with Prop. 5 reduces the search space, many of the recursive calls in Algorithm 3 trigger two further calls. The worst-case time complexity of Algorithm 3 is $O(|Q_S|^2|\Sigma|2^{|\Sigma|})$.

The criterion to maximise the number of hidden events may be too naive. The implementation in Section 4 maximises the number of hidden non-selflooptransitions instead. This is achieved by changing the criterion for comparison on lines 4 and 6 in Algorithm 3, or by considering more desirable events first in Algorithm 2.

3.3 Enforcing the Observer Property

While the test for a feasible projection and the greedy search can be accomplished in polynomial time, the next step in supervisor reduction is to compute a deterministic FSM $\det(S \setminus \Upsilon)$. This is done with *subset construction* (Hopcroft et al., 2001). In most cases, the result will be smaller than the original supervisor S, but this is not guaranteed. The worst-case for the number of states of $\det(S \setminus \Upsilon)$ is exponential in the number of states of S.

The exponential worst case can be avoided by imposing an additional restriction, known as the *observer property*.

Definition 9. (Wong et al., 2000) Let $S = \langle \Sigma, Q, Q^\circ, \to \rangle$ be a deterministic FSM, and let $\Upsilon \subseteq \Sigma$. The projection $P = P_{\Sigma \to \Sigma \setminus \Upsilon}$ is an *observer projection* for S, if for all $s \in \mathcal{L}(S)$ and all $t \in (\Sigma \setminus \Upsilon)^*$ such that $P(s)t \in \mathcal{L}(S \setminus \Upsilon)$ there exists $t' \in \Sigma^*$ such that $P(t') = t$ and $st' \in \mathcal{L}(S)$.

The observer property ensures that, if two states are reached by traces with the same projection, i.e., $S \setminus \Upsilon \stackrel{s}{\Rightarrow} x$ and $S \setminus \Upsilon \stackrel{s}{\Rightarrow} y$, then all continuations from one state are also possible from the other, i.e., $x \stackrel{t}{\Rightarrow}$ if and only if $y \stackrel{t}{\Rightarrow}$ in $S \setminus \Upsilon$. Then hiding results in an observation equivalent FSM, which implies that $\det(S \setminus \Upsilon)$ cannot have more states than $S \setminus \Upsilon$ (Wong and Wonham, 2004).

The observer property can be checked with the *OP-Verifier* algorithm (Pena et al., 2014), which is similar to Algorithm 1: it checks for each verifier pair (x, y) whether it holds for all $\sigma \in \Sigma \setminus \Upsilon$ that $x \stackrel{\sigma}{\Rightarrow}$ if an only if $y \stackrel{\sigma}{\Rightarrow}$. This suggests to modify Algorithm 1 and check the observer property while checking whether the projection is feasible for supervisor reduction.

One complication arises because a result analogous to Prop. 5 does not hold for the observer property—if an event set fails the observer property, its supersets may still have it. Therefore the search for a feasible projection proceeds as per Algorithm 2 or 3, but discards any projections that do not have the observer property. Another complication exists, because the OP-Verifier uses an FSM without cycles of hidden events so that only verifier pairs without Υ-transitions need to be checked. Therefore, the FSM is preprocessed using *Tarjan's algorithm* (Tarjan, 1972) to find strongly connected components of Υ-transitions, check whether their states are control consistent, and if so merge each strongly connected component into a single state. This results in an FSM without cycles of Υ-transitions, which is passed to Algorithm 1 while also performing the checks of the OP-Verifier.

4. EXPERIMENTAL RESULTS

This section presents the results of some experiments to evaluate the effectiveness of event hiding as a preprocessing step in supervisor reduction. The hypothesis is that hiding results is a more structured input to the SW Algorithm (Su and Wonham, 2004), enabling it to produce better results.

The synthesis implementation of Supremica (Åkesson et al., 2006) was modified for the experiments in this section. After synthesis, supervisors are first minimised while preserving the language using the standard algorithm (Hopcroft et al., 2001). Then *supervisor localisation* (Cai and Wonham, 2016) is used to create a modular supervisor: for each controllable event, a copy of the supervisor is created which is responsible for only this event and leaves all other events always enabled. Each of these supervisors is reduced separately. This increases the number of test cases and the potential for supervisor reduction.

For each supervisor reduction attempt, Algorithm 2 or 3 is used to find a feasible event set with as many non-selfloop transitions as possible. If such a set is found, the natural projection is computed using subset construction (Hopcroft et al., 2001), and the result is again minimised. But if subset construction produces too many states (more than twice the size of the input), it is aborted and the original FSM is used instead. The result of this procedure is passed to the SW Algorithm for further minimisation.

Supervisors have been synthesised and reduced for 20 discrete event system models from industrial case studies and examples in other publications. Table 1 shows the name of each model, the number of events ($|\Sigma|$), the number of localised supervisors (Sup #), and the number of reachable states of the unreduced supervisor (Sup states). Under **SW**, the table shows time taken by the SW Algorithm alone and its number of reduced supervisor states, aggregated over all localised supervisors. Then **Greedy** and **Exhaustive** show results using Algorithm 2 and 3, respectively; **Greedy OP** and **Exhaustive OP** refer to the same algorithms when the observer property is also enforced. In each case, the projection rate (Proj rate) is a percentage of states removed after hiding (0% means no reduction). The projection time (Proj [s]) refers to the time needed to find a feasible event set plus the subset construction time to compute an FSM for the projection; the following columns show the time taken by the SW Algorithm applied to the projected FSM and the final number of supervisor states. The Timeout entries in the table mean that the run did not complete in 20 minutes. All experiments were conducted with a standard desktop PC using a 3.3 GHz microprocessor and not more than 4 GiB of RAM.

In most test cases, feasible event sets are found and many states are removed by hiding, often resulting in significantly smaller supervisors. In four cases, it is possible to compute reduced supervisors after hiding, while using only the SW Algorithm fails to complete within the time limit. Yet, no feasible event set is found for the **assembly53**

Table 1. Experimental results

Model				SW		Greedy				Exhaustive				Greedy OP				Exhaustive OP			
Name	|Σ|	Sup #	Sup states	Red [s]	Red States	Proj rate	Proj [s]	Red [s]	Red States	Proj rate	Proj [s]	Red [s]	Red States	Proj rate	Proj [s]	Red [s]	Red States	Proj rate	Proj [s]	Red [s]	Red States
aip0sub1p1	105	3	27165	3.9	18	67 %	2.0	4.5	111	67 %	17.2	4.6	111	54 %	2.1	7.7	114	Timeout			
assembly35r	32	17	68118	34.3	5019	29 %	8.0	26.6	3986	29 %	7.9	26.5	3986	15 %	8.9	35.0	4151	15 %	8.9	33.8	4151
assembly53tr	65	3	1219992	827.6	14818	0 %	3.9	828.7	14818	0 %	3.7	828.0	14818	0 %	3.9	819.9	14818	0 %	3.8	833.2	14818
assembly53vr	53	3	453183	536.1	8927	0 %	1.3	519.6	8927	0 %	1.3	508.9	8927	0 %	1.4	522.8	8927	0 %	1.4	522.7	8927
cm_unsup1 ⟨2, 3⟩	872	7	1998	3.4	2361	0 %	1.6	3.1	2555	0 %	1.5	3.2	2555	0 %	0.1	2.5	2361	0 %	0.1	2.6	2361
cm_unsup1 ⟨3, 2⟩	872	7	1998	2.9	2289	0 %	1.4	4.0	2733	0 %	1.5	4.1	2733	0 %	0.1	3.1	2289	0 %	0.1	3.3	2289
cm_unsup2 ⟨2, 2⟩	560	8	698	0.3	208	53 %	1.1	0.1	86	53 %	2.7	0.1	86	4 %	0.7	0.2	106	4 %	412.6	0.2	106
cm_unsup2 ⟨2, 3⟩	908	14	62836	Timeout		Timeout				Timeout				2 %	140.0	358.4	15159	Timeout			
cm_unsup2 ⟨3, 2⟩	908	14	62836	Timeout		0 %	511.9	456.9	20928	Timeout				2 %	141.1	300.4	14563	Timeout			
ctn ⟨5⟩	38	11	42964	50.4	879	92 %	16.7	0.8	186	92 %	16.4	0.8	186	92 %	15.9	0.8	186	92 %	16.3	0.9	186
ct6	46	14	447998	Timeout		95 %	624.5	15.7	800	95 %	616.2	15.8	800	95 %	619.7	15.7	800	95 %	620.2	15.5	800
fms2016	31	7	45504	4.8	14	96 %	21.2	0.2	14	96 %	21.3	0.2	14	96 %	21.0	0.2	14	96 %	20.9	0.2	14
ims_uncont	17	4	12960	0.8	8	88 %	1.1	0.1	8	88 %	1.0	0.1	8	88 %	1.1	0.1	8	88 %	1.0	0.1	8
ipc	16	8	9216	8.0	2034	93 %	1.4	0.1	145	93 %	1.4	0.1	145	93 %	1.5	0.1	145	93 %	1.5	0.1	145
ipc_cswitch	18	8	18432	24.7	5494	95 %	3.2	0.1	145	95 %	3.3	0.1	145	95 %	3.2	0.1	145	95 %	3.3	0.1	145
ipc_lswitch	20	5	4374	0.9	250	88 %	0.8	0.1	21	88 %	0.8	0.1	21	88 %	0.6	0.1	21	88 %	0.8	0.1	21
tictactoe	35	9	2394	3.3	1970	18 %	0.3	1.7	1245	18 %	0.4	1.6	1245	2 %	0.3	2.0	1450	12 %	6.7	1.4	1335
transferline_n ⟨3⟩	17	6	5992	0.9	18	89 %	19.0	0.1	18	89 %	19.3	0.1	18	42 %	16.3	0.3	18	84 %	117.9	0.1	18
zhennancase	31	10	485721	Timeout		41 %	42.6	244.7	19280	41 %	42.7	241.7	19280	35 %	43.2	308.5	21313	35 %	43.5	305.5	21313
2linkalt	26	3	6288	0.4	7	57 %	0.3	0.1	7	57 %	0.3	0.1	7	57 %	0.3	0.2	7	57 %	0.3	0.2	7

test cases, and while event sets are found for **aip0sub1p1** and **cm_unsup1**, the resulting supervisors are larger than those of the SW Algorithm alone. This suggests that the SW Algorithm usually benefits from the smaller input after hiding, but it retains the potential to find reductions that are not possible with hiding.

In the second and third **cm_unsup2** cases, state numbers increase massively after subset construction and the algorithm uses supervisors without projection. This is mitigated by imposing the observer property, and the **Greedy OP** method produces the best results in these cases. In most other cases, the observer property results in a small increase in supervisor size. Exhaustive search usually takes longer than greedy search, and finds a better result only in the case of **tictactoe** with the observer property.

5. CONCLUSIONS

It has been shown how events can be removed from a synthesised supervisor prior to supervisor reduction. This enables a more effective use of the existing SW Algorithm and often results in simpler and smaller supervisors. The main challenge with supervisor synthesis remains the exponentially large state number of automatically computed state machines, which are often too large for storage or processing by any algorithm. In some cases, smaller representations can be computed symbolically (Miremadi et al., 2011) or compositionally (Mohajerani et al., 2014). The method proposed here can also help with such supervisors.

REFERENCES

Åkesson, K., Fabian, M., Flordal, H., and Malik, R. (2006). Supremica—an integrated environment for verification, synthesis and simulation of discrete event systems. In *8th Int. Workshop on Discrete Event Systems, WODES '06*, 384–385. IEEE. doi:10.1109/WODES.2006.382401.

Cai, K. and Wonham, W.M. (2016). *Supervisor Localization: A Top-Down Approach to Distributed Control of Discrete-Event Systems*, volume 459 of *LNCS*. Springer.

Hoare, C.A.R. (1985). *Communicating Sequential Processes*. Prentice-Hall.

Hopcroft, J.E., Motwani, R., and Ullman, J.D. (2001). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Boston, MA, USA.

Miremadi, S., Åkesson, K., and Lennartson, B. (2011). Symbolic computation of reduced guards in supervisory control. *IEEE Trans. Autom. Sci. Eng.*, 8(4), 754–764. doi:10.1109/TASE.2011.2146249.

Mohajerani, S., Malik, R., and Fabian, M. (2014). A framework for compositional synthesis of modular nonblocking supervisors. *IEEE Trans. Autom. Control*, 59(1), 150–162. doi:10.1109/TAC.2013.2283109.

Pena, P.N., Bravo, H.J., da Cunha, A.E.C., Malik, R., Lafortune, S., and Cury, J.E.R. (2014). Verification of the observer property in discrete event systems. *IEEE Trans. Autom. Control*, 59(8), 2176–2181. doi:10.1109/TAC.2014.2298985.

Ramadge, P.J.G. and Wonham, W.M. (1989). The control of discrete event systems. *Proc. IEEE*, 77(1), 81–98. doi:10.1109/5.21072.

Su, R. and Wonham, W.M. (2004). Supervisor reduction for discrete-event systems. *Discrete Event Dyn. Syst.*, 14(1), 31–53. doi:10.1023/B:DISC.0000005009.40749.b6.

Tarjan, R. (1972). Depth first search and linear graph algorithms. *SIAM J. Computing*, 1(2), 146–160. doi:10.1137/0201010.

Vaz, A.F. and Wonham, W.M. (1986). On supervisor reduction in discrete-event systems. *Int. J. Control*, 44(2), 475–491. doi:10.1080/00207178608933613.

Wong, K.C., Thistle, J.G., Malhame, R.P., and Hoang, H.H. (2000). Supervisory control of distributed systems: Conflict resolution. *Discrete Event Dyn. Syst.*, 10, 131–186. doi:10.1023/A:1008391200517.

Wong, K.C. and Wonham, W.M. (2004). On the computation of observers in discrete-event systems. *Discrete Event Dyn. Syst.*, 14(1), 55–107. doi:10.1023/B:DISC.0000005010.55515.27.

Yoo, T.S. and Lafortune, S. (2002). Polynomial-time verification of diagnosability of partially observed discrete-event systems. *IEEE Trans. Autom. Control*, 47(9), 1491–1495. doi:10.1109/TAC.2002.802763.