Modular Supervisor Synthesis for Extended Finite-State Machines Subject to Controllability

Robi Malik   Marcelo Teixeira

***Abstract*— This paper proposes an algorithm for the synthesis of modular supervisors using extended finite-state machines, i.e., state machines with variables and guards on the transitions. Synthesis is performed by iteratively selecting components from a synchronous composition until a least restrictive controllable solution is obtained. This method is usually faster and produces smaller supervisors than standard monolithic synthesis, while offering the modelling benefits of variables. An example of manufacturing system control illustrates the approach.**

I. Introduction

Supervisory Control Theory [1] provides a framework for the *synthesis*, i.e., automatic computation, of *supervisors* for discrete event systems. The theory has been generalised for *Extended Finite-state Machines* (EFSMs) [2], [3], which include *variables* and improve modelling capabilities for systems with data dependency or software. Several synthesis algorithms for EFSMs have been proposed [4]–[6]. The straightforward synthesis algorithms explore the full system state space, including all possible combinations of variable values, and this can result in prohibitively many states. This complexity can be avoided to some extent using *symbolic* representation [5] or *abstraction* [7], [8].

This paper focuses on the synthesis of *least restrictive* and *controllable* supervisors, i.e., considering only prefix-closed behaviours, in a *modular* setting, where the system model consists of several interacting EFSM components. In this case, efficient solutions for systems without variables are known [9]–[12]. These methods decompose the system into appropriate subsystems, and synthesise local supervisors that, in combination, achieve the least restrictive controllable behaviour of the entire system.

These solutions do not generalise directly to EFSMs, because the composition with EFSM components may introduce new variable assignments and increase behaviour. Therefore, this paper proposes the use of *abstractions* that capture the possible variable changes outside of the considered subsystem. This results in an incremental procedure that identifies small groups of EFSMs that are sufficient to ensure a controllable and maximally permissive supervisor.

In the following, Section II introduces an improved version of the algorithm [9] for modular synthesis of ordinary finite-state machines, and Section III generalises this algorithm for EFSMs. Afterwards, Section IV illustrates the approach with an example, and Section V presents some conclusions. Formal proofs of the technical results can be found in [13].

II. Finite-State Machines

A. Definitions

A *finite-state machine (FSM)* is a tuple $F = \langle \Sigma, Q, Q^\circ, \rightarrow\rangle$, 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*, 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. Furthermore, 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 $X \rightarrow Y$ means $X \xrightarrow{s} Y$ for some $s \in \Sigma^*$, and $X \xrightarrow{s}$ means $X \xrightarrow{s} Y$ for some Y, and $F \xrightarrow{s} X$ means $Q^\circ \xrightarrow{s} X$. A trace $s \in \Sigma^*$ is *accepted* by the FSM if $F \xrightarrow{s}$, and the *language* or *behaviour* of F is the set of all traces it accepts, $\mathcal{L}(F) = \{\, s \in \Sigma^* \mid F \xrightarrow{s} \,\}$.

In this paper, FSMs do not have accepting states, and all languages are prefix-closed: trace $s \in \Sigma^*$ is a *prefix* of $t \in \Sigma^*$ if $t = su$ for some $u \in \Sigma^*$. Language $L \subseteq \Sigma^*$ is *prefix-closed*, if all prefixes of traces $t \in L$ are contained in L.

FSMs executing in parallel are synchronised in lock-step [14]. The *synchronous composition* of two FSMs $F_1 = \langle \Sigma, Q_1, Q_1^\circ, \rightarrow_1\rangle$ and $F_2 = \langle \Sigma, Q_2, Q_2^\circ, \rightarrow_2\rangle$ with the same event set Σ is $F_1 \parallel F_2 = \langle \Sigma, Q_1 \times Q_2, Q_1^\circ \times Q_2^\circ, \rightarrow\rangle$, where $(x_1, x_2) \xrightarrow{\sigma} (y_1, y_2)$ if and only if $x_1 \xrightarrow{\sigma} x_2$ and $y_1 \xrightarrow{\sigma} y_2$. FSMs with different event sets can be composed after the *selfloop* operation [15], adding selfloop transitions $x \xrightarrow{\sigma} x$ with the missing events to all states of an FSM; this is not considered in this section for the sake of brevity, and all FSMs are assumed to have the same event set Σ.

For the purpose of control, the event set is partitioned into the sets Σ_c of *controllable* events and Σ_u of *uncontrollable* events. Controllable events can be disabled by a controlling agent, while uncontrollable events occur spontaneously. A prefix-closed *specification* language $K \subseteq \Sigma^*$ is Σ_u-controllable [1] with respect to (w.r.t.) a prefix-closed *plant* language $L \subseteq \Sigma^*$ if $K\Sigma_\mathrm{u} \cap L \subseteq K$, i.e., if every uncontrollable event continuation possible in L is also possible in K.

If a language K is not controllable, the task of *synthesis* is to find a controllable sublanguage $K' \subseteq K$. It is a classical result of supervisory control theory [1] that the union of controllable languages is again controllable, and there exists a unique *supremal controllable sublanguage* of any given

Robi Malik is with the Department of Computer Science, The University of Waikato, Hamilton, New Zealand (robi@waikato.ac.nz).

Marcelo Teixeira is with the Academic Department of Informatics, Federal University of Technology Paraná, Pato Branco, Brazil (marceloteixeira@utfpr.edu.br).

Algorithm 1: Modular FSM synthesis

```
Input: plants $\mathcal{G} = \{G_1, \ldots, G_m\}$; specification $E$;
       uncontrollable events $\Sigma_\mathrm{u}$;
1 $\mathcal{G}^0 \leftarrow \emptyset$; $S^0 \leftarrow E$; $\Sigma_\mathrm{u}^0 \leftarrow \emptyset$; $i \leftarrow 0$;
2 while $\mathcal{L}(S^i)$ is not $\Sigma_\mathrm{u}$-controllable w.r.t. $\mathcal{L}(\|(\mathcal{G}^i))$ do
3     $\Sigma_\mathrm{u}^{i+1} \leftarrow \Sigma_\mathrm{u}^i \cup \{\, \mu \in \Sigma_\mathrm{u} \mid \|(\mathcal{G}^i) \parallel S^i \to (x_G, x_S)$ and
          $x_G \xrightarrow{\mu}$ and $x_S \overset{\mu}{\not\to} \}$;
4     $\mathcal{G}^{i+1} \leftarrow \{\, G' \in \mathcal{G} \mid G' \to x_G \overset{\mu}{\not\to}$ for some $\mu \in \Sigma_\mathrm{u}^{i+1} \,\}$;
5     $S^{i+1} \leftarrow \sup\mathcal{C}(\|(\mathcal{G}^{i+1}), E, \Sigma_\mathrm{u}^{i+1})$;
6     $i \leftarrow i + 1$;
7 end
8 return $S^i$
```

language,

$$\sup\mathcal{C}(L, K, \Sigma_\mathrm{u}) = \bigcup\{\, K' \subseteq L \cap K \mid K' \text{ is } \Sigma_\mathrm{u}\text{-controllable w.r.t. } L \,\} \ . \qquad (1)$$

It is common to require that the result of synthesis is contained in the plant language L, which is enforced by the intersection $L \cap K$ in (1). If the plant and specification are given by FSMs G and E, a standard algorithm [1] with time complexity polynomial in the number of transitions of $G \parallel E$ can construct an FSM that accepts (1). This FSM is denoted $\sup\mathcal{C}(G, E, \Sigma_\mathrm{u})$. It can be used as a so-called *supervisor*, which restricts the plant through synchronous composition, enforcing the specification by disabling only controllable events in the least restrictive way possible.

B. Modular Synthesis Algorithm

In the following, it is assumed that the plant is given by several FSMs, $G = G_1 \parallel \cdots \parallel G_n$, and the specification is given by a single FSM E. In this case, the complexity to compute $\sup\mathcal{C}(G, E, \Sigma_\mathrm{u})$ is exponential in the number n of plant components due to the exponential number of states in the synchronous composition. It has been proposed [9], [10] to mitigate this complexity by identifying an appropriate subset of the plants to perform synthesis with.

Algorithm 1 shows such an approach. Here and in the following, for a set $\mathcal{G}$ of state machines, $\|(\mathcal{G})$ denotes the synchronous composition of all elements of $\mathcal{G}$, and $\|(\emptyset) = \langle \Sigma, \{x^\circ\}, \{x^\circ\}, \{x^\circ\} \times \Sigma \times \{x^\circ\} \rangle$ is the neutral element of synchronous composition, a state machine that accepts all events without state change.

The idea of Algorithm 1 is to gradually increase the set of plants and uncontrollable events considered in synthesis. At the beginning, the loop entry condition in line 2 checks whether the specification $S^0 = E$ is controllable by itself, in which case E is returned as the least restrictive solution. This may succeed if, for example, E has only controllable events. Otherwise the loop is entered and performs synthesis w.r.t. selected subsets $\mathcal{G}^{i+1}$ of plants and Σ_u^{i+1} of uncontrollable events (line 5). Inside the loop, line 2 ensures that the current result S^i is not controllable w.r.t. the full set Σ_u of uncontrollable events. Thus, S^i disables some event $\mu \in \Sigma_\mathrm{u} \setminus \Sigma_\mathrm{u}^i$, which really is uncontrollable but was assumed controllable in synthesis. By including these events in Σ_u^{i+1}, they are treated as uncontrollable in the next iteration (line 3). To ensure the least restrictive result, all plants that in some state may disable one of these uncontrollable events are also included (line 4).

The procedure continues until a Σ_u-controllable solution is found. Termination is guaranteed, because the set Σ_u^i of included uncontrollable events increases with every iteration. As the result is Σ_u-controllable w.r.t. a subset of plants, it is also Σ_u-controllable w.r.t. the full plant [10]. The approach [9] ensures least restrictiveness by including all plants that share an uncontrollable event with the specification, or with a plant already included. Algorithm 1 improves on this in line 3, by considering only uncontrollable events that cause a controllability problem, and in line 4, by only adding plants that disable an uncontrollable event, as opposed to plants that have it in their event set.

Prop. 1 confirms that the result S^i of Algorithm 1 implements the least restrictive supervisor. As synthesis within the loop only includes a part of the plant, the remaining plants have to be composed with the result to get the exact supremal controllable sublanguage. A proof is given in [13].

Proposition 1: Upon termination of Algorithm 1, it holds that $\mathcal{L}(\|(\mathcal{G}) \parallel S^i) = \sup\mathcal{C}(\mathcal{L}(\|(\mathcal{G})), \mathcal{L}(E), \Sigma_\mathrm{u})$.

III. Extended Finite-State Machines

Extended finite-state machines (EFSMs) add to FSMs *variables* and the ability to read and update these variables on the occurrence of transitions [2], [3]. Examples of such state machines are shown in Fig. 3.

A. Variables and Updates

An *update* is a formula constructed from variables, integer constants, Boolean literals, and the usual arithmetic and logic connectives. The set of all update formulas is denoted by Π.

A *variable* v is an entity associated with a finite discrete *domain* $\mathrm{dom}(v)$ and an initial value $\hat{v}^\circ \in \mathrm{dom}(v)$. Let $V = \{v_0, \ldots, v_n\}$ be the set of variables with combined domain $\mathrm{dom}(V) = \mathrm{dom}(v_0) \times \cdots \times \mathrm{dom}(v_n)$. An element $\hat{v}$ of $\mathrm{dom}(V)$ is also considered as a *valuation* that assigns to each variable $v \in V$ a value $\hat{v}(v) \in \mathrm{dom}(v)$, and by extension a truth value to each update. The *initial valuation* is $\hat{v}^\circ \in \mathrm{dom}(V)$ with $\hat{v}^\circ(v) = \hat{v}^\circ$ for each $v \in V$. An update is *satisfiable* if it is true for at least one valuation, and *valid* if it is true for all valuations of its variables. The *restriction* of a valuation $\hat{v} \in \mathrm{dom}(V)$ to $W \subseteq V$ is $\hat{v}_{|W} \in \mathrm{dom}(W)$ with $\hat{v}_{|W}(v) = \hat{v}(v)$ for all $v \in W$.

A second set of variables, called *next-state* variables and denoted $V' = \{\, v' \mid v \in V \,\}$ is used to describe the values of the variables after a transition. The next-state variable v' has the same domain as its *current-state* variable v. Given $\hat{v} \in \mathrm{dom}(V)$, the valuation $\hat{v}' \in \mathrm{dom}(V')$ is defined by $\hat{v}'(v') = \hat{v}(v)$ for all $v \in V$. For an update $p \in \Pi$, the term $\mathrm{vars}(p)$ denotes the set of all variables that occur as current-state or next-state variable in p, and $\mathrm{vars}'(p)$ denotes the set of all variables whose corresponding next-state variables occur in p. For example, if p is the update $x' = y + 1$, then $\mathrm{vars}(p) = \{x, y\}$ and $\mathrm{vars}'(p) = \{x\}$.

B. EFSM Definition and Operations

Definition 1: An *Extended finite-state machine (EFSM)* is a tuple $F = \langle \Sigma, Q, Q^\circ, \rightarrow \rangle$, where Σ is a finite set of events, Q is a finite set of *locations*, $Q^\circ \subseteq Q$ is the set of *initial locations*, and $\rightarrow \subseteq Q \times \Sigma \times \Pi \times Q$ is the *extended transition relation*.

A transition between locations $x, y \in Q$ with event $\sigma \in \Sigma$ and update $p \in \Pi$ is written $x \overset{\sigma:p}{\to} y$. It can occur if F is in location x and the update p evaluates to true, and when it occurs, F changes its location to y while updating the variables in $\mathrm{vars}'(p)$ in accordance with p; variables not in $\mathrm{vars}'(p)$ remain unchanged. For example, let x be a variable with domain $\mathrm{dom}(x) = \{0, \ldots, 5\}$. A transition with update $x' = x+1$ changes the variable x by adding 1 to its current value, if it currently is less than 5. Otherwise (if $x = 5$) the transition is disabled. The update $x = 3$ disables a transition unless $x = 3$ in the current state, and the value of x in the next state is unchanged. Differently, the update $x' = 3$ always enables its transition, and the value of x in the next state is forced to be 3.

Given an EFSM F and event $\sigma \in \Sigma$, the *referenced variable set* is $\mathrm{vars}(F, \sigma) = \bigcup \{ \mathrm{vars}(p) \mid x \overset{\sigma:p}{\to} y \}$, and $\mathrm{vars}(F) = \bigcup_{\sigma \in \Sigma} \mathrm{vars}(F, \sigma)$. Furthermore, for a set $\mathcal{F}$ of EFSMs, $\mathrm{vars}(\mathcal{F}, \sigma) = \bigcup_{F' \in \mathcal{F}} \mathrm{vars}(F', \sigma)$ and $\mathrm{vars}(\mathcal{F}) = \bigcup_{F' \in \mathcal{F}} \mathrm{vars}(F')$. Analogous notation is defined for vars'.

This paper imposes some restrictions on system models, which are needed for the modularity results.

Definition 2: Let $F = \langle \Sigma, Q, Q^\circ, \rightarrow \rangle$ be an EFSM.

- F is *normalised*, if for any two transitions $x_1 \xrightarrow{\sigma:p_1} y_1$ and $x_2 \xrightarrow{\sigma:p_2} y_2$ with the same event $\sigma \in \Sigma$, it holds that $\mathrm{vars}'(p_1) = \mathrm{vars}'(p_2)$.
- F is *pure* if $\mathrm{vars}'(F) = \emptyset$.
- F is *state-deterministic* if $|Q^\circ| \leq 1$, and for all transitions $x \xrightarrow{\sigma:p_1} y_1$ and $x \xrightarrow{\sigma:p_2} y_2$ such that $p_1 \wedge p_2$ is satisfiable, it holds that $y_1 = y_2$.

In a normalised EFSM, the set of variables changed by an event is the same on all transitions. This assumption helps to recognise the implicitly unchanged variables after synchronous composition. Every EFSM can be transformed into a normalised EFSM by a process of renaming similar to normalisation [3]. As a stronger condition, a pure EFSM cannot assign any variables, it only restricts events. State-determinism ensures that the target locations are uniquely determined from the source location, event, and variable assignment. It is needed for supervisors to track the location of the plant by the observation of events and variable values.

In the following, plants are modelled by normalised state-deterministic EFSMs, while specifications are pure state-deterministic EFSMs. The synthesised supervisor is not subject to these requirements and can restrict variable assignments.

An EFSM $F = \langle \Sigma, Q, Q^\circ, \rightarrow \rangle$ can be *unfolded* [3], [8] and interpreted as an FSM with state set $Q \times \mathrm{dom}(\mathrm{vars}(F))$. The states $(x, \hat{v})$ consist of a location $x \in Q$ and a valuation $\hat{v} \in \mathrm{dom}(\mathrm{vars}(F))$. A transition between two states, written $(x, \hat{v}) \overset{\sigma}{\to} (y, \hat{w})$, exists if F contains a transition $x \overset{\sigma:p}{\to} y$ such that the update p is true if the current-state variables are interpreted according to $\hat{v}$ and the next-state variables according to $\hat{w}$, and all variables that do not appear as next-state variables in the update are unchanged between $\hat{v}$ and $\hat{w}$. This transition relation is also defined for variables not in $\mathrm{vars}(F)$, which remain unchanged, and for events not in the EFSM's event set Σ, which are always enabled without changing the EFSM's location or any variables. The $\rightarrow$ notation is extended to traces, state sets, and state machines in the same way as for FSMs. Based on this, the set of *accessible states* of an EFSM F is $Q^{\mathrm{acc}}(F) = \{ (x, \hat{v}) \in Q \times \mathrm{dom}(\mathrm{vars}(F)) \mid F \rightarrow (x, \hat{v}) \}$.

When comparing EFSMs, variables must be considered in addition to events, so the following notion of behavioural inclusion replaces language inclusion as used for FSMs.

Definition 3: An EFSM F_1 is *behaviourally included* in another EFSM F_2, written $F_1 \subseteq_{\mathrm{v}} F_2$, if for every path

$$(x_0, \hat{v}_0) \overset{\sigma_1}{\to} (x_1, \hat{v}_1) \overset{\sigma_2}{\to} \cdots \overset{\sigma_n}{\to} (x_n, \hat{v}_n) \tag{2}$$

in F_1, with $\hat{v}_i \in \mathrm{dom}(\mathrm{vars}(F_1))$, there exists a path

$$(y_0, \hat{w}_0) \overset{\sigma_1}{\to} (y_1, \hat{w}_1) \overset{\sigma_2}{\to} \cdots \overset{\sigma_n}{\to} (y_n, \hat{w}_n) \tag{3}$$

in F_2 such that $\hat{w}_i \in \mathrm{dom}(\mathrm{vars}(F_2))$ and $\hat{v}_{i\,|V_{12}} = \hat{w}_{i\,|V_{12}}$ for $1 \leq i \leq n$, where $V_{12} = \mathrm{vars}(F_1) \cap \mathrm{vars}(F_2)$.

If F_1 is behaviourally included in F_2 then every path in F_1 corresponds to a path in F_2 with the same events and variable assignments. The two EFSMs typically use the same variables, but if not, only the common variables are required to match.

Definition 4: The *synchronous composition* of two EFSMs $F_1 = \langle \Sigma_1, Q_1, Q_1^\circ, \rightarrow_1 \rangle$ and $F_2 = \langle \Sigma_2, Q_2, Q_2^\circ, \rightarrow_2 \rangle$ is $F_1 \parallel F_2 = \langle \Sigma_1 \cup \Sigma_2, Q_1 \times Q_2, Q_1^\circ \times Q_2^\circ, \rightarrow \rangle$, where $(x_1, x_2) \xrightarrow{\sigma:p_1 \wedge p_2} (y_1, y_2)$ if $x_1 \xrightarrow{\sigma:p_1}_1 y_1$ and $x_2 \xrightarrow{\sigma:p_2}_2 y_2$.

This definition captures EFSMs with different event sets through the extended definition of the transition relation above. Updates in synchronous composition are combined by conjunction. They may cancel each other out, e.g., if $x_1 \xrightarrow{\sigma:x'=0}_1 y_1$ in F_1 and $x_2 \xrightarrow{\sigma:x'=1}_2 y_2$ in F_2, then the conjunction $x' = 0 \wedge x' = 1$ is logically false, or equivalently there is no such transition in $F_1 \parallel F_2$. Synchronous composition can override the assumption of implicitly unchanged variables in an EFSM. If $x_1 \xrightarrow{\sigma:x=0}_1 y_1$ and $x_2 \xrightarrow{\sigma:x'=x+1}_2 y_2$, e.g., then $(x_1, x_2) \xrightarrow{\sigma:x=0 \wedge x'=x+1} (y_1, y_2)$. So the value of x changes from 0 to 1 in $F_1 \parallel F_2$ although implicitly unchanged in F_1.

C. EFSM Controllability and Synthesis

For the supervisory control of EFSM systems, this paper assumes that all variables are controlled by the plant. The plant is modelled by a set of normalised EFSMs that represent the possible system behaviour including all possible variable changes. The specification is typically modelled by one or more pure EFSMs, which only restrict the occurrence of events. The supervisor can also restrict variable changes associated with controllable events. The following definition of controllability covers specifications and supervisors.

Definition 5: Let $G = \langle \Sigma_G, Q_G, Q_G^\circ, \rightarrow_G \rangle$ and $E = \langle \Sigma_E, Q_E, Q_E^\circ, \rightarrow_E \rangle$ be two EFSMs, and let Σ_{u} be a set of events. E is Σ_{u}-*controllable* w.r.t. G, if for all valuations $\hat{v}, \hat{w} \in \mathrm{dom}(\mathrm{vars}(G) \cup \mathrm{vars}(E))$, all states $(x_G, x_E, \hat{v}) \in Q^{\mathrm{acc}}(G \parallel E)$, and all transitions $(x_G, \hat{v}) \xrightarrow{\mu} (y_G, \hat{w})$ in G such that $\mu \in \Sigma_E \cap \Sigma_{\mathrm{u}}$, there exists a location y_E of E such that $(x_G, x_E, \hat{v}) \xrightarrow{\mu} (y_G, y_E, \hat{w})$ in $G \parallel E$.

Σ_{u}-controllability means that, from any accessible state in the synchronous composition of the plant G and specification E, if an *uncontrollable* event is eligible in the plant, then it is also eligible in the specification. In addition, the specification must allow any assignment to next-state variables prescribed by the plant. The condition $(x_G, x_E, \hat{v}) \xrightarrow{\mu} (y_G, y_E, \hat{w})$ is applied to the synchronous composition $G \parallel E$, so it requires the plant and specification to be able to take the transition together. This allows a pure specification to follow the plant's assignments to next-state variables.

If a specification is not controllable, *synthesis* is used to find a *supervisor* in the form of an EFSM composed with the plant. Unlike the specification, the supervisor may include next-state variables on its updates. Thus, the supervisor can disable (controllable) events completely or under certain circumstances, and it can remove some of the plant's variable assignments from a controllable transition.

Definition 6: Let G and E be two EFSMs, and let Σ_{u} be a set of events. A *supremal supervisor* for E w.r.t. G and Σ_{u} is an EFSM S such that

(i) $\mathrm{vars}(S) \subseteq \mathrm{vars}(G) \cup \mathrm{vars}(E)$;
(ii) $G \parallel S \subseteq_{\mathrm{v}} G \parallel E$;
(iii) S is Σ_{u}-controllable w.r.t. G;
(iv) For any EFSM S' that satisfies (ii) and (iii), it holds that $G \parallel S' \subseteq_{\mathrm{v}} G \parallel S$.

Def. 6 characterises the possible synthesis results for a plant G and specification E. A correct supervisor can only use variables that appear in the plant or specification (i). This syntactic condition is convenient but not essential, as any extra variables can be unfolded into locations to construct an equivalent supervisor. A correct supervisor must satisfy the specification through behavioural inclusion after composition with the plant (ii), and it must be controllable (iii). It also must be *least restrictive* or *supremal*, i.e., any other supervisor that controllably satisfies the specification has less possible behaviour, again in composition with the plant (iv).

A supervisor satisfying these four conditions can be computed by means of a standard fixpoint iteration [13] on the unfolded state set of $G \parallel E$. The result of this procedure is denoted $\sup\mathcal{C}(G, E, \Sigma_{\mathrm{u}})$ in the following.

D. Modular Synthesis Algorithm

The idea of modular synthesis (Algorithm 1) is to identify a suitable subsystem to perform synthesis and use the result to control the entire system. This is based on modularity properties, according to which synchronous composition of a state machine with another only ever restricts the behaviour [10]. Then controllability w.r.t. a part of the plant implies controllability w.r.t. the entire plant.

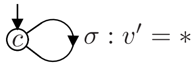

Fig. 1. The EFSM $\mathrm{chaos}(\sigma, v)$.

EFSMs do not have this property. When EFSMs are combined in synchronous composition, new next-state variables can be added to transitions, possibly changing variables that were implicitly unchanged. To enable modular synthesis with EFSMs, one solution is to replace the parts of the system not considered in a synthesis attempt by an *abstraction* that includes all possible variable changes. This abstraction is called *chaos EFSM*.

Definition 7: Given an event σ and a variable v, the *chaos EFSM* for σ and v is defined as

$$\mathrm{chaos}(\sigma, v) = \langle \{\sigma\}, \{c\}, \{c\}, \{(c, \sigma, v' = *, c)\} \rangle \ . \qquad (4)$$

The EFSM $\mathrm{chaos}(\sigma, v)$ is shown in Fig. 1. The update $v' = *$ means that the variable v can assume any value from its domain in the next state. Formally, this update is true for all valuations, but it includes the next-state variable v' so that v is no longer implicitly unchanged.

In the synchronous composition $F_1 \parallel F_2$ of two EFSMs, some variables in F_1 may be changed by transitions in F_2. A variable v can be changed after composition of a transition in F_1 that does not mention v' with a transition in F_2 that mentions v'; or by a transition with an event that only appears in F_2. By inspection of the next-state variables on the transitions of F_2, it can be determined that certain variables are not changed in F_2, or are only changed on the occurrence of certain events. The following Lemma 2 shows how to identify the specific chaos EFSMs to capture possible variable changes in another EFSM.

Lemma 2: Let F_1 and F_2 be two EFSMs, and let

$$C = \big\|(\{\, \mathrm{chaos}(\sigma, v) \mid v \in \mathrm{vars}(F_1) \cap \mathrm{vars}'(F_2, \sigma) \,\}) \ . \qquad (5)$$

If $(x_1, x_2, \hat{v}) \xrightarrow{\sigma} (y_1, y_2, \hat{w})$ in $F_1 \parallel F_2$ then $(x_1, c, \hat{v}_{|\mathrm{vars}(F_1)}) \xrightarrow{\sigma} (y_1, c, \hat{w}_{|\mathrm{vars}(F_1)})$ in $F_1 \parallel C$, where c is the single location of C.

In a synchronous composition $F_1 \parallel F_2$, Lemma 2 allows F_2 to be replaced by chaos EFSMs C for variables in F_1 and events with transitions assigning to these variables in F_2. Then all transitions in $F_1 \parallel F_2$ are also possible in $F_1 \parallel C$.

Algorithm 2 uses this result to extend Algorithm 1 for EFSM plants $\mathcal{G}$ and specification E. As before, the algorithm seeks to identify suitable subsets $\mathcal{G}^i \subseteq \mathcal{G}$ of the plants and $\Sigma_{\mathrm{u}}^i \subseteq \Sigma_{\mathrm{u}}$ of the uncontrollable events, starting without uncontrollable events or plants, and trying to use the specification E as supervisor. Differently from Algorithm 1, the plants $\bar{\mathcal{G}}^i = \mathcal{G} \setminus \mathcal{G}^i$ not included at each step are replaced by chaos EFSMs $\mathcal{C}^i$ based on Lemma 2. If the supervisor S^i found in the i-th iteration is controllable w.r.t. the abstraction $\big\|(\mathcal{G}^i) \parallel \big\|(\mathcal{C}^i)$ of the plant and all uncontrollable events, then it is returned as the result.

Otherwise line 3 extends the set Σ_{u}^i by including uncontrollable events that cause S^i to violate controllability as it is done in Algorithm 1. Then line 4 extends the plant $\mathcal{G}^i$, which

Algorithm 2: Modular EFSM synthesis

```
   Input: normalised state-deterministic plants $\mathcal{G} = \{G_1, \dots, G_m\}$; pure state-deterministic specification $E$;
          uncontrollable events $\Sigma_\mathrm{u}$.
1  $\Sigma_\mathrm{u}^0 \leftarrow \emptyset$;  $\mathcal{G}^0 \leftarrow \emptyset$;  $\bar{\mathcal{G}}^0 \leftarrow \mathcal{G}$;  $V^0 \leftarrow \mathrm{vars}(E)$;  $\mathcal{C}^0 \leftarrow \{\, \mathrm{chaos}(\sigma, v) \mid v \in \mathrm{vars}(E) \cap \mathrm{vars}'(\mathcal{G}, \sigma) \,\}$;  $S^0 \leftarrow E$;  $i \leftarrow 0$;
2  while $S^i$ is not $\Sigma_\mathrm{u}$-controllable w.r.t. $\Vert(\mathcal{G}^i) \parallel \Vert(\mathcal{C}^i)$ do
3      $\Sigma_\mathrm{u}^{i+1} \leftarrow \Sigma_\mathrm{u}^i \cup \{\, \mu \in \Sigma_\mathrm{u} \mid$ there exist $\hat{v}, \hat{w} \in \mathrm{dom}(V^i)$, $(x_G, c, x_S, \hat{v}) \in Q^\mathrm{acc}(\Vert(\mathcal{G}^i) \parallel \Vert(\mathcal{C}^i) \parallel S^i)$, and $(x_G, c, \hat{v}) \xrightarrow{\mu} (y_G, c, \hat{w})$
               in $\Vert(\mathcal{G}^i) \parallel \Vert(\mathcal{C}^i)$, but no location $y_S$ of $S^i$ such that $(x_G, c, x_S, \hat{v}) \xrightarrow{\mu} (y_G, c, y_S, \hat{w})$ in $\Vert(\mathcal{G}^i) \parallel \Vert(\mathcal{C}^i) \parallel S^i \,\}$;
4      $\mathcal{G}^{i+1} \leftarrow \{\, G' \in \mathcal{G} \mid$ there exists $\mu \in \Sigma_\mathrm{u}^{i+1}$ such that $\mu$ is not always enabled in $G' \,\}$;
5      $\bar{\mathcal{G}}^{i+1} \leftarrow \mathcal{G} \setminus \mathcal{G}^{i+1}$;
6      $V^{i+1} \leftarrow \mathrm{vars}(\mathcal{G}^{i+1}) \cup \mathrm{vars}(E)$;
7      $\mathcal{C}^{i+1} \leftarrow \{\, \mathrm{chaos}(\sigma, v) \mid v \in V^{i+1} \cap \mathrm{vars}'(\bar{\mathcal{G}}^{i+1}, \sigma) \,\}$;
8      $S^{i+1} \leftarrow \mathrm{sup}\mathcal{C}(\, \Vert(\mathcal{G}^{i+1}) \parallel \Vert(\mathcal{C}^{i+1}), E, \Sigma_\mathrm{u}^{i+1})$;
9      $i \leftarrow i + 1$;
10 end
11 return $S^i$
```

like in Algorithm 1 must include all components that may disable some uncontrollable event, however here variables must be taken into account.

Definition 8: Let $F = \langle \Sigma, Q, Q^\circ, \rightarrow \rangle$ be an EFSM. An event $\sigma \in \Sigma$ is *always enabled* at location $x \in Q$, if the disjunction $\bigvee_{i=1}^n p_i$ is valid, where $x \xrightarrow{\sigma : p_i} y_i$ for $1 \le i \le n$ are all the σ-transitions originating from x. Event σ is always enabled in F, if it is always enabled at every location $x \in Q$.

An event is always enabled in an EFSM if it can be taken at any location, independently of variable updates. Plants not satisfying this condition for some uncontrollable event in Σ_u^{i+1} are included in the next iteration following line 4 of Algorithm 2. Having determined the plants and uncontrollable events, line 8 performs synthesis, which can be done by a brute-force explicit algorithm or using more advanced symbolic methods [5].

The algorithm terminates because the set Σ_u^i increases with each iteration, and the following Prop. 3 confirms that it returns a least restrictive supervisor. A proof is given in [13].

Proposition 3: Upon termination of Algorithm 2, S^i is a supremal supervisor for E w.r.t. $\Vert(\mathcal{G})$ and Σ_u.

E. Multiple Specifications

If an FSM system contains more than one specification, then controllability can be verified for each specification separately, and it is also known that a least restrictive supervisor can be obtained by combining the least restrictive supervisors obtained for the individual specifications [10]. Under the assumption of pure specifications, these results extend directly to EFSMs.

Proposition 4: Let G be a state-deterministic normalised EFSM, let E_1 and E_2 be pure EFSMs, and let Σ_u be a set of events. Let S_1 and S_2 be supremal supervisors for E_1 and E_2, respectively, w.r.t. G and Σ_u. Then $S_1 \parallel S_2$ is a supremal supervisor for $E_1 \parallel E_2$ w.r.t. G and Σ_u.

Given sets of plant EFSMs $\mathcal{G}$ and specifications EFSMs $\mathcal{E}$, by Props. 3 and 4, synthesis can be performed separately for each specification $E_j \in \mathcal{E}$ using Algorithm 2, and the synchronous composition of the resulting supervisors $S_j = \mathrm{sup}\mathcal{C}(\Vert(\mathcal{G}), E_j, \Sigma_\mathrm{u})$ gives the least restrictive supervisor for the combined specification $\Vert(\mathcal{E})$ and plant $\Vert(\mathcal{G})$.

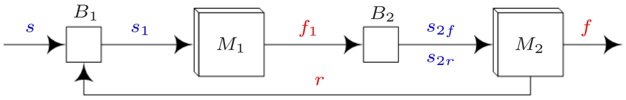

Fig. 2. Manufacturing system with material feedback [8].

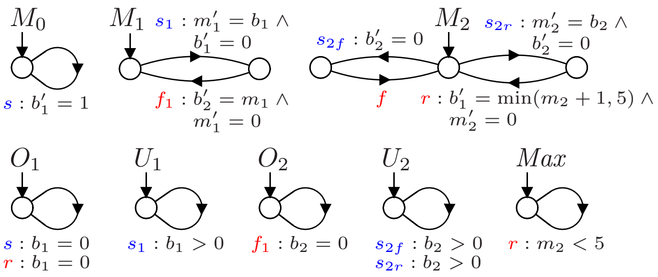

Fig. 3. EFSM model of manufacturing system [8].

IV. Manufacturing System Example

Fig. 2 shows a manufacturing system with material feedback [8]. The system consists of two machines, M_1 and M_2, linked by one-place buffers B_1 and B_2. Buffer B_1 receives external workpieces by event s. Machine M_1 removes workpieces from B_1 (event s_1), manufactures and puts them in B_2 (event f_1), where a quality test determines the operation to be performed by M_2 (s_{2f} or s_{2r}), which leads to a release of the workpiece (event f) or its return to B_1 for rework (event r). Events f_1, f, and r are uncontrollable, the others are controllable. The control objective is to avoid overflow and underflow of the buffers, and to ensure that workpieces pass through the system at most five times.

Fig. 3 shows an EFSM model of this system. The variables b_1, m_1, b_2, and m_2 with domain $\{0, \dots, 5\}$ and initial value 0 record the contents of the machines and buffers. A value of 0 indicates that the machine or buffer is empty, while a value of $k > 0$ indicates the presence of a workpiece that has been placed k times into buffer B_1.

Plants M_0, M_1, and M_2 model the behaviour of the machines and buffers. A first unload to B_1 (transition s in M_0) sets the variable b_1 to 1, indicating presence of a workpiece in its first cycle. Loading a workpiece from B_1

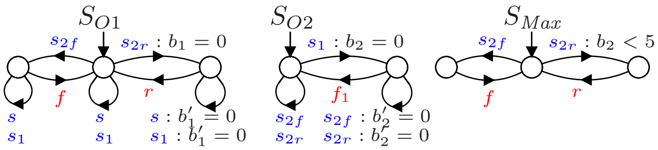

Fig. 4. Synthesised supervisors for manufacturing system.

to M_1 (transition s_1 in M_1) transfers the value from b_1 to m_1 and resets b_1 to 0, as B_1 is again empty. When a workpiece is returned to B_1 on event r, the condition $b_1' = \min(m_2+1, 5)$ in M_2 assigns to b_1 the value that identifies the next work cycle. The transition is also defined when $m_2 = 5$, i.e., the workpiece is already in its last cycle, but this case is ruled out by specification Max.

Specifications O_1, U_1, O_2, and U_2 model the avoidance of overflow and underflow. For example, O_1 specifies that a workpiece can only be added to buffer B_1 on event s or r when B_1 is empty, $b_1 = 0$. Specification Max disallows the return of workpieces in their fifth cycle to B_1 by event r.

To obtain a modular supervisor, Algorithm 2 is invoked for each specification separately. Specifications U_1 and U_2 have only controllable events, so they are naturally controllable and Algorithm 2 terminates without entering the loop. These specifications can serve as supervisors directly, e.g., U_1 prevents loading of a workpiece from B_1 when B_1 is empty.

For specification O_1, the first iteration of Algorithm 2 without plants, $\mathcal{G}^0 = \emptyset$, uses chaos EFSMs for variable b_1, which appears in O_1 and is changed by the plant on events s, s_1, and r, i.e., $\mathcal{C}^0 = \{\mathrm{chaos}(s, b_1), \mathrm{chaos}(s_1, b_1), \mathrm{chaos}(r, b_1)\}$. It turns out that O_1 is not Σ_{u}-controllable w.r.t. $\|(\mathcal{C}^0)$, because O_1 prevents the uncontrollable event r when $b_1 \neq 0$. Therefore, r is considered as uncontrollable in the next iteration, $\Sigma_{\mathrm{u}}^1 = \{r\}$, and plant M_2 that can disable r is added, $\mathcal{G}^1 = \{M_2\}$. Among the variables in M_2 and O_1, $V^1 = \{b_1, b_2, m_2\}$, only b_1 and b_2 are ever changed by the remaining plants M_0 and M_1, so that $\mathcal{C}^1 = \{\mathrm{chaos}(s, b_1), \mathrm{chaos}(s_1, b_1), \mathrm{chaos}(f_1, b_2)\}$. Synthesis gives the supervisor $S_{O1} = \sup\mathcal{C}(M_2 \parallel \|(\mathcal{C}^1), O_1, \{r\})$ shown in Fig. 4, which is Σ_{u}-controllable w.r.t. $M_2 \parallel \|(\mathcal{C}^1)$. Algorithm 2 stops here.

The figure only shows updates added during synthesis, a computed supervisor may contain more updates. The update on s_{2r} in S_{O1} avoids overflow of B_1 by preventing M_2 from loading a workpiece to be reworked unless B_1 is empty. The update on s contradicts plant M_0 and thereby disables s, i.e., the loading of a new workpiece to B_1, while another is being returned. The update on s_1 is redundant due to plant M_1, but appears here as M_1 was not included in the partial synthesis.

Synthesis for specifications O_2 and Max also terminates after the first iteration, producing the supervisors S_{O2} and S_{Max} in Fig. 4. Overall, modular synthesis never composes more than two of the EFSMs from Fig. 3 and gives five supervisor components with 1–3 locations each. The largest compositions are encountered at the end of synthesis for O_1 and Max and have 288 unfolded states each. In contrast, the standard algorithm to construct a supervisor for all plants and specifications together explores a state space of 912 states and produces a single supervisor with six locations.

V. Conclusions

The paper presents an algorithm that calculates modular controllable supervisors that control a system in the least restrictive way. The proposed incremental approach is simpler and produces smaller supervisors than the usual methods of monolithic synthesis in the literature. It improves on the authors' previous work [8] by completely removing some components and variables from the subsystems subject to synthesis at each step of the algorithm.

In future work, the authors would like to investigate the use of *variable abstraction* [8] to further reduce the subsystems. It would also be interesting to extend the approach to consider the synthesis of nonblocking supervisors.

References

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

[2] Y. Chen and F. Lin, "Modeling of discrete event systems using finite state machines with parameters," in *Proc. 2000 IEEE Int. Conf. Control Applications (CCA)*, 2000, pp. 941–946.

[3] S. Mohajerani, R. Malik, and M. Fabian, "A framework for compositional nonblocking verification of extended finite-state machines," *Discrete Event Dyn. Syst.*, 2015.

[4] T. Le Gall, B. Jeannet, and H. Marchand, "Supervisory control of infinite symbolic systems using abstract interpretation," in *Proc. 46th IEEE Conf. Decision and Control, CDC '05*, Dec. 2005, pp. 30–35.

[5] S. Miremadi, K. Åkesson, and B. Lennartson, "Symbolic computation of reduced guards in supervisory control," *IEEE Trans. Autom. Sci. Eng.*, vol. 8, no. 4, pp. 754–764, Oct. 2011.

[6] L. Ouedraogo, R. Kumar, R. Malik, and K. Åkesson, "Nonblocking and safe control of discrete-event systems modeled as extended finite automata," *IEEE Trans. Autom. Sci. Eng.*, vol. 8, no. 3, pp. 560–569, July 2011.

[7] M. R. Shoaei, L. Feng, and B. Lennartson, "Abstractions for nonblocking supervisory control of extended finite automata," in *Proc. 8th Int. Conf. Automation Science and Engineering, CASE 2012*, Aug. 2012, pp. 364–370.

[8] M. Teixeira, R. Malik, J. E. R. Cury, and M. H. de Queiroz, "Supervisory control of DES with extended finite-state machines and variable abstraction," *IEEE Trans. Autom. Control*, vol. 60, no. 1, pp. 118–129, Jan. 2015.

[9] K. Åkesson, H. Flordal, and M. Fabian, "Exploiting modularity for synthesis and verification of supervisors," in *Proc. 15th IFAC World Congress on Automatic Control*, 2002.

[10] B. A. Brandin, R. Malik, and P. Malik, "Incremental verification and synthesis of discrete-event systems guided by counter-examples," *IEEE Trans. Control Syst. Technol.*, vol. 12, no. 3, pp. 387–401, May 2004.

[11] M. H. de Queiroz and J. E. R. Cury, "Modular supervisory control of large scale discrete event systems," in *Proc. 5th Int. Workshop on Discrete Event Systems, WODES '00*, Aug. 2000, pp. 103–110.

[12] J. Komenda and J. H. van Schuppen, "Control of discrete-event systems with modular or distributed structure," *Theoretical Comput. Sci.*, vol. 388, pp. 199–226, 2007.

[13] R. Malik and M. Teixeira, "An algorithm for the synthesis of least restrictive controllable supervisors for extended finite-state machines," Working Paper 01/2016, Dept. of Computer Science, University of Waikato, Hamilton, New Zealand, 2016. [Online]. Available: http://hdl.handle.net/10289/9841

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

[15] W. M. Wonham, "Supervisory control of discrete-event systems," Systems Control Group, Dept. of Electrical Engineering, University of Toronto, ON, Canada; at http://www.control.utoronto.edu/DES/, 2009.