Discrete Event Dynamic Systems manuscript No.
(will be inserted by the editor)

Synthesis of Least Restrictive Controllable Supervisors for Extended Finite-State Machines with Variable Abstraction

Robi Malik · Marcelo Teixeira

the date of receipt and acceptance should be inserted later

Abstract This paper presents an algorithm that combines modular synthesis for extended finite-state machines (EFSM) with abstraction of variables by symbolic manipulation, in order to compute least restrictive controllable supervisors. Given a modular EFSM system consisting of several components, the proposed algorithm synthesises a separate supervisor for each specification component. To synthesise each supervisor, the algorithm iteratively selects components (plants and variables) from a synchronous composition until a least restrictive controllable solution is obtained. This improves on previous results of the authors where abstraction is only performed by the selection of components and not variables. The paper explains the theory of EFSM synthesis and abstraction and its algorithms. An example of a flexible manufacturing system illustrates how the proposed algorithm works to compute a modular supervisor.

Keywords Supervisory control, Discrete event systems, Extended finite-state machines.

1 Introduction

Supervisory Control Theory [4,19] provides a general framework for the synthesis of reactive control functions. Given a model of the system, the *plant*, to be controlled, and a *specification* of the desired behaviour, it is possible to automatically compute, i.e. *synthesise*, a *supervisor* that restricts the plant behaviour while satisfying the specification. Originally, the theory is grounded on the *Finite-state Machines* formalism and several approaches have been developed to make synthesis more efficient [1, 15, 27, 29].

Robi Malik
University of Waikato, Hamilton, New Zealand
E-mail: robi@waikato.ac.nz

Marcelo Teixeira
Federal University of Technology Paraná, Pato Branco, Brazil
E-mail: marceloteixeira@utfpr.edu.br

In recent years, Supervisory Control Theory has been generalised for *Extended Finite-state Machines* (EFSM) [5, 16, 25, 28], which include *variables* and improve modelling capabilities for systems with data dependency or software. Variables are more general than approaches with similar purpose [9, 18], including *event distinguishers* [6, 20, 27], and can simplify the modelling task for various discrete event systems. However, they require more sophisticated tools and methods for synthesis. Several synthesis algorithms for EFSMs have been proposed [10, 14, 17, 28], which explore the full system state space, including all possible combinations of variable values. The resulting complexity can be avoided to some extent using *symbolic* representation [14] or *abstraction* [22, 23, 28].

Recently, a modular approach for the synthesis of *least restrictive* and *controllable* supervisors from plants modelled with EFSMs has been proposed [12], which generalises earlier work on modular synthesis without variables [1,2]. The approach considers only prefix-closed behaviours, and the system model consists of several interacting plant and specification components. In this case, synthesis can be performed separately for each specification EFSM, and the results can be combined to form a modular supervisor. For each specification, the algorithm [12] iteratively selects plant components to be included in synthesis until a least restrictive controllable solution is found. The obtained modular supervisors, in combination, achieve the least restrictive controllable behaviour for the entire system.

This paper extends the approach of [12] by including the idea of *existential abstraction* [28] of variables. The algorithm of [12] performs abstraction only by selecting EFSM components and always includes all variables of the selected components. Existential abstraction improves on this, because it allows for abstractions to be formed by selecting components and some of their variables. Other variables are quantified out and do not contribute to the state space when synthesis is performed.

In the following, Section 2 introduces the ideas of modular synthesis by presenting an improved version of the modular synthesis algorithm [1] for ordinary finite-state machines. Then Section 3 introduces EFSMs and outlines frameworks for their synthesis and abstraction. Afterwards, Section 4 presents the algorithms for modular synthesis with abstraction in the EFSM framework. The proposed method is illustrated by an example in Section 5, and finally Section 6 adds concluding remarks. Formal proofs of the technical results can be found in [13].

2 Finite-State Machines

2.1 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, because termination or the nonblocking property are not considered. As a result, all languages are prefix-closed: trace $s \in \Sigma^*$ is a *prefix* of $t \in \Sigma^*$ if $t = su$ for some $u \in \Sigma^*$, and a 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 [7]. 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

$$F_1 \parallel F_2 = \langle \Sigma, Q_1 \times Q_2, Q_1^\circ \times Q_2^\circ, \rightarrow \rangle \ , \tag{1}$$

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 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. In this case, it is clear that synchronous composition of FSMs results in the intersection of their languages, $\mathcal{L}(F_1 \parallel F_2) = \mathcal{L}(F_1) \cap \mathcal{L}(F_2)$.

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 cannot be prevented from occurring. A prefix-closed *specification* language $K \subseteq \Sigma^*$ is Σ_u-controllable 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 [19].

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 [19] that the union of controllable languages is again controllable, and there exists a unique *supremal controllable sublanguage* of any given language,

$$\sup\mathcal{C}(L, K, \Sigma_\mathrm{u}) = \bigcup \{\, K' \subseteq L \cap K \mid K' \text{ is } \Sigma_\mathrm{u}\text{-controllable w.r.t. } L \,\} \ . \tag{2}$$

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 (2). If the plant and specification are given by FSMs G and E, a standard algorithm [19] with time complexity polynomial in the number of transitions of $G \parallel E$ can construct an FSM that accepts the supremal controllable sublanguage $\sup\mathcal{C}(L, K, \Sigma_\mathrm{u})$. 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.

2.2 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 time complexity to compute $\sup\mathcal{C}(G, E, \Sigma_\mathrm{u})$ is in the worst case exponential in the number n of plant components because the number of states in the synchronous composition could be exponential. It has been proposed [1, 2] to mitigate this complexity by identifying an appropriate subset of the plants to perform synthesis with.

Algorithm 1: Modular FSM synthesis

Input: plants $\mathcal{G} = \{G_1, \ldots, G_m\}$; specification E; uncontrollable events Σ_u;
Output: Least restrictive supervisor S^k;

1 $\mathcal{G}^0 \leftarrow \emptyset$;
2 $S^0 \leftarrow E$;
3 $\Sigma_\mathrm{u}^0 \leftarrow \emptyset$;
4 $i \leftarrow 0$;
5 **while** $\mathcal{L}(S^i)$ is not Σ_u-controllable w.r.t. $\mathcal{L}(\|(\mathcal{G}^i))$ **do**
6 $\quad \Sigma_\mathrm{u}^{i+1} \leftarrow \Sigma_\mathrm{u}^i \cup \{\, \mu \in \Sigma_\mathrm{u} \mid \|(\mathcal{G}^i) \parallel S^i \to (x_G, x_S) \text{ and } x_G \xrightarrow{\mu} \text{ and } x_S \not\xrightarrow{\mu} \,\}$;
7 $\quad \mathcal{G}^{i+1} \leftarrow \{\, G' \in \mathcal{G} \mid G' \to x_G \not\xrightarrow{\mu} \text{ for some } \mu \in \Sigma_\mathrm{u}^{i+1} \,\}$;
8 $\quad S^{i+1} \leftarrow \sup\mathcal{C}(\|(\mathcal{G}^{i+1}), E, \Sigma_\mathrm{u}^{i+1})$;
9 $\quad i \leftarrow i + 1$;
10 **end**
11 **return** S^i;

Algorithm 1 shows such an approach. This algorithm is the basis for the extended finite-state machine synthesis algorithm in the following section. Here, for a set $\mathcal{G}$ of FSMs, $\|(\mathcal{G})$ 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 algorithm starts without plants, $\mathcal{G}^0 = \emptyset$. Then the loop entry condition on line 5 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 8). Inside the loop, line 5 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 6). To ensure the least restrictive result, all plants that in some state may disable one of these uncontrollable events are also included (line 7).

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 and is bounded by the finite set Σ_u. 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 [2]. The approach [1] 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 on line 6, by considering only uncontrollable events that cause a controllability problem, and on line 7, 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 includes only a part of the plant, the

remaining plants have to be composed with the result to get the exact supremal controllable sublanguage.

Proposition 1 [11] Algorithm 1 terminates, and upon termination it holds that

$$\mathcal{L}(\|(\mathcal{G}) \parallel S^i) = \sup\mathcal{C}(\mathcal{L}(\|(\mathcal{G})), \mathcal{L}(E), \Sigma_{\mathrm{u}}) \ . \tag{3}$$

3 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 [5, 16]. In this section, the concept of update formulas is introduced, followed by a formal definition of EFSMs. Then the FSM concepts of synchronous composition, behavioural inclusion, and synthesis are generalised to the EFSM setting. Finally, notions of EFSM abstraction are introduced.

3.1 Variables and Updates

An *update* is a first-order logic formula [8] constructed from variables, integer constants, Boolean literals, the existential and universal quantifiers ($\exists$ and $\forall$), and the usual arithmetic and logical operators. Variables can be *bound* to quantifiers or occur *free* in an update. For example, in the update $\exists y\, x > y + 2$, the variable y is bound to the existential quantifier, while x is a free variable. The set of all update formulas is denoted by $\mathit{\Pi}$.

In this paper, formulas are interpreted over *finite* domains. This is a common assumption in supervisory control, as synthesis algorithms are not guaranteed to terminate for infinite state spaces. Nevertheless, finiteness is not essential for the results on *abstraction*. The results in this paper can also be proven for infinite domains, but the algorithms may fail to terminate if the underlying synthesis algorithms without abstraction do not terminate.

Thus, every *variable* z is associated with a finite discrete *domain* $\mathrm{dom}(z)$ and an initial value $z^\circ \in \mathrm{dom}(z)$. Let $V = \{z_0, \ldots, z_n\}$ be the set of variables with combined domain $\mathrm{dom}(V) = \mathrm{dom}(z_0) \times \cdots \times \mathrm{dom}(z_n)$. An element $\hat{v}$ of $\mathrm{dom}(V)$ is also considered as a *valuation* that assigns to each variable $z \in V$ a value $\hat{v}(z) \in \mathrm{dom}(z)$, and by extension a truth value to each update. The *initial valuation* is $V^\circ \in \mathrm{dom}(V)$ with $V^\circ(z) = z^\circ$ for each $z \in V$.

A second set of variables, called *next-state* variables and denoted $V' = \{\, z' \mid z \in V \,\}$ is used to describe the values of the variables after a transition. Variables in V are also referred to as *current-state* variables to differentiate them from the next-state variables in V'. The next-state variable z' has the same domain as its current-state variable z. Given $\hat{v} \in \mathrm{dom}(V)$, the valuation $\hat{v}' \in \mathrm{dom}(V')$ is defined by $\hat{v}'(z') = \hat{v}(z)$ for all $z \in V$. For an update $p \in \mathit{\Pi}$, the term $\mathrm{vars}(p)$ denotes the set of all variables with a free occurrence as current-state or next-state variable in p, and $\mathrm{vars}'(p)$ denotes the set of all variables whose corresponding next-state variables have a free occurrence in p. For example, if $p \equiv \exists z\, x' = y + z + 1$, then $\mathrm{vars}(p) = \{x, y\}$ and $\mathrm{vars}'(p) = \{x\}$. Here and in the following, the relation $\equiv$ denotes syntactic identity of updates to avoid ambiguity when an update contains the equality symbol $=$.

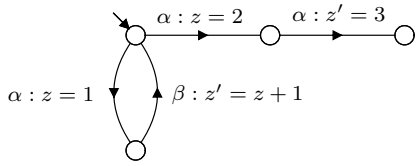

Fig. 1 An EFSM with a variable z, where $\mathrm{dom}(z) = \{0, \ldots, 5\}$ and $z^\circ = 0$.

An update $p \in \Pi$ is *satisfiable* if it is true for at least one valuation of its variables, i.e., if there is a valuation $\hat{v} \in \mathrm{dom}(V \cup V')$ such that $\hat{v}(p) = \mathrm{true}$. Otherwise the update p is *unsatisfiable*. An update p is *valid* if it is true for all valuations of its variables, i.e., if $\hat{v}(p) = \mathrm{true}$ for every valuation $\hat{v}$. The *restriction* of a valuation $\hat{v} \in \mathrm{dom}(V)$ to $W \subseteq V$ is $\hat{v}{\restriction_W} \in \mathrm{dom}(W)$ with $\hat{v}{\restriction_W}(z) = \hat{v}(z)$ for all $z \in W$. Two valuations $\hat{v} \in \mathrm{dom}(V)$ and $\hat{w} \in \mathrm{dom}(W)$ can be combined by *functional override* to give $\hat{v} \lhd \hat{w} \in \mathrm{dom}(V \cup W)$ where $(\hat{v} \lhd \hat{w})(z) = \hat{v}(z)$ for $z \in V \setminus W$ and $(\hat{v} \lhd \hat{w})(z) = \hat{w}(z)$ for $z \in W$.

3.2 EFSM Definition

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 $x \xrightarrow{\sigma:p} y$. The behaviour of an EFSM does not only involve changes of locations, but the EFSM also maintains a current valuation of all the variables that appear on its transitions, starting from their initial values. A transition $x \xrightarrow{\sigma:p} y$ can occur if the EFSM is in location x and the update p evaluates to true under the current valuation. When it occurs, the EFSM changes its location to y and the variables in $\mathrm{vars}'(p)$ are updated in accordance with p, while variables not in $\mathrm{vars}'(p)$ remain unchanged. More precisely, the transition $x \xrightarrow{\sigma:p} y$ can occur if there exists a valuation of the variables in $\mathrm{vars}'(p)$ such that p becomes true for these next-state values in combination with the current values of the current-state variables in p. Any such valuation of the variables in $\mathrm{vars}'(p)$, extended with the current-state values of the unchanged variables gives a possible new valuation after execution of the transition.

When an EFSM is part of a larger system, the transition relation is conceptually extended for events not in the event set of the EFSM, i.e., for $\sigma \notin \Sigma$, by defining $x \xrightarrow{\sigma:\mathrm{true}} x$ for all locations $x \in Q$. That is, events not in the event set can occur at any time without any location or variable changes.

Example 1 Consider the EFSM F in Fig. 1, which has only one variable z with domain $\mathrm{dom}(z) = \{0, \ldots, 5\}$ and initial value $z^\circ = 0$. The update $z' = z + 1$ of the β-transition changes the variable z by adding z to its current value, if it currently is less than 5. Otherwise (if $z = 5$) the transition is disabled. The update $z = 2$ disables its transition unless $z = 2$ currently, and the value of z after the transition is unchanged if the transition is taken. Differently, the update $z' = 3$ always enables its transition, and the value of z after the transition is forced to be 3.

Given an EFSM $F = \langle \Sigma, Q, Q^\circ, \to \rangle$ and event $\sigma \in \Sigma$, the *referenced variable set* is $\mathrm{vars}(F, \sigma) = \bigcup\{ \mathrm{vars}(p) \mid x \xrightarrow{\sigma:p} y \}$. Furthermore, $\mathrm{vars}(F, \Sigma') = \bigcup_{\sigma \in \Sigma'} \mathrm{vars}(F, \sigma)$ for a set of events $\Sigma' \subseteq \Sigma$, and then $\mathrm{vars}(F) = \mathrm{vars}(F, \Sigma)$ is the set of all variables in F. 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 [12] Let $F = \langle \Sigma, Q, Q^\circ, \to \rangle$ be an EFSM.

(i) F is *next-state variable normalised* (vars'*-normalised* for short) 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 $\mathrm{vars}'(p_1) = \mathrm{vars}'(p_2)$.

(ii) F is *pure* if $\mathrm{vars}'(F) = \emptyset$.

(iii) F is *location-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 next-state variable normalised EFSM, if an event is associated with different updates on different transitions, the set of variables changed by these updates is always the same. This assumption helps to identify the implicitly unchanged variables after synchronous composition. Next-state variable normalisation is a weaker property than normalisation [16], which requires transitions with the same event to have identical updates. Still, a process similar to normalisation [16] can be used to transform every EFSM into a vars'-normalised EFSM that is equivalent up to the renaming of some events.

A pure EFSM does not include any next-state variables in its updates. It cannot assign any variables and only restricts events. This is a stronger condition than normalisation: every pure EFSM is also vars'-normalised.

There are different notions of determinism that can be applied to EFSMs, e.g., in [25] it is required that all locations and variable values after a transition are uniquely determined for a given event and valuation of variables. Location-determinism is a weaker condition that only ensures that the target locations are uniquely determined from the source location, event, and valuation. This condition is sufficient for supervisors to track the location of the plant by the observation of events and variable values.

Example 2 Consider again the EFSM F in Fig. 1. This EFSM is *not* vars'-normalised, because it has α-transitions with updates $z = 1$ and $z' = 3$, and $\mathrm{vars}'(z = 1) = \emptyset \neq \{z\} = \mathrm{vars}'(z' = 3)$. That is, some α-transitions explicitly change z while others leave z implicitly unchanged. The EFSM F is also not pure, for example $\mathrm{vars}'(z' = 3) \neq \emptyset$.

On the other hand, F is location-deterministic: although there are two α-transitions with different targets originating from the initial location, the dependence of the guards on the value of z ensures that these transitions cannot be enabled at the same time. The conjunction $z = 1 \wedge z = 2$ is unsatisfiable as z can never have both the values 1 and 2.

In this paper, plants are modelled by vars'-normalised location-deterministic EFSMs, while specifications are pure location-deterministic EFSMs. The synthesised supervisor is also vars'-normalised and location-deterministic but, unlike the specification, not necessarily pure so that it can restrict variables.

An EFSM $F = \langle \Sigma, Q, Q^\circ, \to \rangle$ can be *unfolded* [16,28] 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 $\hat{v} \in \mathrm{dom}(\mathrm{vars}(F))$. More specifically, the unfolded transition relation is defined as follows.

Definition 3 Let $F = \langle \Sigma, Q, Q^\circ, \to \rangle$ be an EFSM, and let $V \supseteq \mathrm{vars}(F)$. The *unfolded transition relation* $\to \subseteq (Q \times \mathrm{dom}(V)) \times \Sigma \times (Q \times \mathrm{dom}(V))$ is defined such that $(x, \hat{v}) \xrightarrow{\sigma} (y, \hat{w})$ if and only if there exists a transition $x \xrightarrow{\sigma:p} y$ in F such that $(\hat{v} \lhd\!\!\rhd \hat{w}')(p)$ is true and $\hat{v}(z) = \hat{w}(z)$ for all variables $z \in V \setminus \mathrm{vars}'(p)$.

Thus, an unfolded transition between two states $(x, \hat{v}) \xrightarrow{\sigma} (y, \hat{w})$ exists if F contains a transition $x \xrightarrow{\sigma:p} y$ 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 p are unchanged between $\hat{v}$ and $\hat{w}$. This transition relation is extended to events not in the EFSM's event set Σ, which are always enabled without changing the EFSM's location or any variables.

The $\to$ notation is extended to traces, state sets, and EFSMs in the same way as for FSMs. For example, a transition sequence

$$(x^0, \hat{v}^0) \xrightarrow{\sigma_1} (x^1, \hat{v}^1) \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_n} (x^n, \hat{v}^n) \tag{4}$$

is written $(x^0, \hat{v}^0) \xrightarrow{s} (x^n, \hat{v}^n)$ for $s = \sigma_1 \cdots \sigma_n$. This transition sequence is a *path* in F if it starts in an initial location of F and with initial variable values, i.e., if $x^0 \in Q^\circ$ and $\hat{v}^0 = V^\circ$, which is also written as $F \xrightarrow{s} (x^n, \hat{v}^n)$. Based on this, the set of *accessible states* of an EFSM F is

$$Q^{\mathrm{acc}}(F) = \{\, (x, \hat{v}) \in Q \times \mathrm{dom}(\mathrm{vars}(F)) \mid F \xrightarrow{s} (x, \hat{v}) \text{ for some } s \in \Sigma^* \,\} \,. \tag{5}$$

For the control of an EFSM, it is of interest to *restrict* its behaviour to a subset $X \subseteq Q \times \mathrm{dom}(V)$ of its unfolded states.

Definition 4 Let $F = \langle \Sigma, Q, Q^\circ, \to \rangle$ be an EFSM with $V = \mathrm{vars}(F)$, and let $X \subseteq Q \times \mathrm{dom}(V)$. The valuations to restrict a transition $x \xrightarrow{\sigma:p} y$ to X are:

$$\begin{aligned} \mathcal{R}_X[x \xrightarrow{\sigma:p} y] = \{\, (\hat{v}, \hat{w}) \in \mathrm{dom}(V) \times \mathrm{dom}(V) \mid (\hat{v} \lhd\!\!\rhd \hat{w}')(p) = \text{true and} \\ \hat{v}{\restriction}_{V \setminus \mathrm{vars}'(p)} = \hat{w}{\restriction}_{V \setminus \mathrm{vars}'(p)} \text{ and } (y, \hat{w}) \in X \,\} \,. \end{aligned} \tag{6}$$

The restriction relation contains pairs of valuations $\hat{v}$ before the transition and $\hat{w}$ after the transition, which are consistent with the unfolded transition relation, $(x, \hat{v}) \xrightarrow{\sigma} (y, \hat{w})$, and where the system satisfies the constraints imposed by X after the transition. By converting this relation to an update formula, a restricted EFSM can be constructed symbolically.

Definition 5 Let $F = \langle \Sigma, Q, Q^\circ, \to \rangle$ be an EFSM with $V = \mathrm{vars}(F)$, and let $X \subseteq Q \times \mathrm{dom}(V)$. The *symbolic restriction* of F to X is an EFSM $F \restriction X = \langle \Sigma, Q_{|X}, Q^\circ_{|X}, \to_{|X} \rangle$, where

$$Q_{|X} = \{\, x \in Q \mid (x, \hat{v}) \in X \text{ for some } \hat{v} \in \mathrm{dom}(V) \,\} \,; \tag{7}$$

$$Q^\circ_{|X} = \{\, x^\circ \in Q^\circ \mid (x^\circ, V^\circ) \in X \,\} \,; \tag{8}$$

and $x \xrightarrow{\sigma:R}_{|X} y$, if $x, y \in Q_{|X}$ and $x \xrightarrow{\sigma:p} y$, and $R \in \Pi$ is an update with $\mathrm{vars}(R) \subseteq V$ and $\mathrm{vars}'(R) \subseteq \mathrm{vars}'(p)$ such that $(\hat{v} \mathbin{\triangleleft} \hat{w}')(R) = \mathrm{true}$ if and only if $(\hat{v}, \hat{w}) \in \mathcal{R}_X[x \xrightarrow{\sigma:p} y]$ for all valuations $\hat{v}, \hat{w} \in \mathrm{dom}(V)$.

Example 3 Consider an EFSM with variables y and z, both with domain $\{0, 1, 2\}$, and a transition

$$a \xrightarrow{\sigma:z'=z-1} b \ . \tag{9}$$

Assume the EFSM is to be restricted to ensure $y = z = 0$ whenever the system is in location b. This restriction is defined by the set $X \subseteq Q \times \mathrm{dom}(y) \times \mathrm{dom}(z)$ with $X = \{(b, 0, 0)\} \cup ((Q \setminus \{b\}) \times \mathrm{dom}(y) \times \mathrm{dom}(z))$. As $y = z = 0$ must hold after the transition (9), which leaves y unchanged, it is clear that $y = 0$ and $z = 1$ before the transition. Based on (6), there is only one pair of valuations allowed for the restricted transition,

$$\mathcal{R}_X[a \xrightarrow{\sigma:z'=z-1} b] = \{((0, 1), (0, 0))\} \ . \tag{10}$$

If an EFSM representing the restriction to X is to be constructed, the set of valuations (10) is written as an update formula using only primed variables that appear in (9). As an example, this transition can be replaced by

$$a \xrightarrow{\sigma:y=0 \wedge z=1 \wedge z'=0}_{|X} b \ . \tag{11}$$

The transition (11) can only be taken when $y = 0$ and $z = 1$, thus ensuring $y = z = 0$ when the system enters location b.

Although not unique, it is always possible to construct a formula R that matches $\mathcal{R}_X[x \xrightarrow{\sigma:p} y]$, e.g., *conjunctive normal form* [8]. In the following it is assumed that the symbolic restriction is obtained deterministically by an appropriate algorithm.

3.3 Synchronous Composition

EFSMs are composed using lock-step synchronisation on shared events, like ordinary FSMs, but in addition the updates are combined by conjunction.

Definition 6 The *synchronous composition* of two EFSMs $F_1 = \langle \Sigma_1, Q_1, Q_1^\circ, \to_1 \rangle$ and $F_2 = \langle \Sigma_2, Q_2, Q_2^\circ, \to_2 \rangle$ is

$$F_1 \parallel F_2 = \langle \Sigma_1 \cup \Sigma_2, Q_1 \times Q_2, Q_1^\circ \times Q_2^\circ, \to \rangle \ , \tag{12}$$

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. For example, if $x_1 \xrightarrow{\sigma:p}_1 y_1$ in F_1 and F_2 does not synchronise on this event, $\sigma \notin \Sigma_2$, then the extended transition relation of F_2 includes $x_2 \xrightarrow{\sigma:\mathrm{true}}_2 x_2$ for every location $x_2 \in Q_2$. This results in synchronised transitions $(x_1, x_2) \xrightarrow{\sigma:p \wedge \mathrm{true}} (y_1, x_2)$, or equivalently $(x_1, x_2) \xrightarrow{\sigma:p} (y_1, x_2)$, in $F_1 \parallel F_2$.

As a result of the conjunctive combination of updates, they may cancel each other out. For example, if $x_1 \xrightarrow{\sigma:z'=0}_1 y_1$ in F_1 and $x_2 \xrightarrow{\sigma:z'=1}_2 y_2$ in F_2, then the conjunction $z' = 0 \wedge z' = 1$ is unsatisfiable, or equivalently there is no such transition in the final composed system. Synchronous composition can override the assumption of implicitly unchanged variables in an EFSM. If $x_1 \xrightarrow{\sigma:z=0}_1 y_1$ and $x_2 \xrightarrow{\sigma:z'=z+1}_2 y_2$, e.g., then $(x_1, x_2) \xrightarrow{\sigma:z=0\wedge z'=z+1} (y_1, y_2)$. So the value of z changes from 0 to 1 in $F_1 \parallel F_2$ although implicitly unchanged in F_1.

EFSM synchronous composition is associative and commutative, apart from the renaming of locations and rewriting of updates into equivalent formulas. Synchronous composition is not idempotent as $F \parallel F = F$ does not generally hold for non-deterministic state machines. If F is a location-deterministic EFSM, then $F \parallel F = F$ holds up to isomorphism, after deletion of transitions with unsatisfiable updates and inaccessible locations.

This paper considers systems modelled as the synchronous composition of several EFSMs. Then the model consists of a *set* of EFSMs, $\mathcal{F} = \{F_1, \ldots, F_n\}$, and the notation $\Big\|(\mathcal{F}) = F_1 \parallel \cdots \parallel F_n$ denotes their synchronous composition. As a special case, $\Big\|(\emptyset) = \langle \emptyset, \{x^\circ\}, \{x^\circ\}, \emptyset \rangle$ is the neutral element of synchronous composition. This is a one-location EFSM without events, which by definition accepts all events without changing its location or assigning any variables.

The following lemma describes a criterion to determine the presence of a transition in the unfolded transition relation of a synchronous composition. Such a transition exists if each of the composed EFSMs has a transition whose update evaluates to true, and the variables that do not appear as next-state variables in any of these updates are unchanged.

Lemma 2 Let $F_1, \ldots, F_n$ be EFSMs, $V \supseteq \mathrm{vars}(F_1) \cup \cdots \cup \mathrm{vars}(F_n)$, and $\hat{v}, \hat{w} \in \mathrm{dom}(V)$. Then

$$(x_1, \ldots, x_n, \hat{v}) \xrightarrow{\sigma} (y_1, \ldots, y_n, \hat{w}) \qquad \text{in } F_1 \parallel \cdots \parallel F_n \tag{13}$$

if and only if $x_i \xrightarrow{\sigma:p_i} y_i$ in each F_i, with $(\hat{v} \lhd \hat{w}')(p_i) = \mathrm{true}$ and $\hat{v}\restriction_U = \hat{w}\restriction_U$ where $U = V \setminus (\mathrm{vars}'(p_1) \cup \cdots \cup \mathrm{vars}'(p_n))$.

The proof of this lemma is immediate from Defs. 3 and 6 as the EFSM $F_1 \parallel \cdots \parallel F_n$ must contain a transition $(x_1, \ldots, x_n, \hat{v}) \xrightarrow{\sigma:p_1\wedge\cdots\wedge p_n} (y_1, \ldots, y_n, \hat{w})$, with the variables not appearing primed in the updates remaining unchanged. Under the assumption of vars'-normalisation, the set U of unchanged variables can also be written as $U = V \setminus (\mathrm{vars}'(F_1, \sigma) \cup \cdots \cup \mathrm{vars}'(F_n, \sigma))$. Note that if the event σ is not in the event set of some F_i, then by definition $x_i \xrightarrow{\sigma:\mathrm{true}} x_i$.

3.4 Behavioural Inclusion

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

$$(x_0, \hat{v}_0) \xrightarrow{\sigma_1} (x_1, \hat{v}_1) \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_n} (x_n, \hat{v}_n) \qquad \text{in } F_1 \tag{14}$$

with $\hat{v}_i \in \mathrm{dom}(\mathrm{vars}(F_1) \cup \mathrm{vars}(F_2))$, there exists a path

$$(y_0, \hat{v}_0) \xrightarrow{\sigma_1} (y_1, \hat{v}_1) \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_n} (y_n, \hat{v}_n) \qquad \text{in } F_2\ . \tag{15}$$

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 valuations of variables. Variables not present in F_1 remain unchanged by F_1 according to Def. 3, and as a consequence must also be unchanged in F_2. This semantics of implicitly unchanged variables makes behavioural inclusion of EFSMs different from language inclusion of FSMs, and several intuitively expected properties do not hold. The properties of behavioural inclusion as defined above are investigated in [13]. Most importantly, the relation is reflexive and transitive.

Lemma 3 [13] Let A, B, and C be EFSMs. Then the following properties hold.

(i) $A \subseteq_{\mathrm{v}} A$.
(ii) If $A \subseteq_{\mathrm{v}} B$ and $B \subseteq_{\mathrm{v}} C$ then $A \subseteq_{\mathrm{v}} C$.

3.5 Controllability and Synthesis

For the supervisory control of EFSM systems, this paper assumes that the variables are part of the plant [12]. The plant is modelled by a set of vars′-normalised EFSMs that represent the possible system behaviour including all possible variable changes. The specification is 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 8 [12] 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 $\mu \in \Sigma_{\mathrm{u}}$, 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 $\mu \in \Sigma_{\mathrm{u}}$ is eligible in the plant, then it is also eligible in the specification. In addition, any valuation of next-state variables allowed by the plant must remain possible, either by a pure specification following the plant, or by a supervisor making only consistent changes. 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 valuation of next-state variables.

In the case that an uncontrollable event is not mentioned in the plant, $\mu \notin \Sigma_G$, based on the extended definition of the transition relation, the transition

is always possible in the plant and does not change variables. In order to be controllable, the specification must always enable μ without changing any variables on its occurrence.

Remark 1 Given a plant G and pure specification E, it can be assumed without loss of generality that $\mathrm{vars}(E) \subseteq \mathrm{vars}(G)$. This is because any variable z that appears only in E and not in G, cannot appear as next-state variable in G or E as $\mathrm{vars}'(E) = \emptyset$. This variable z remains unchanged on all transitions of the synchronous composition $G \parallel E$, so all its occurrences can be replaced by a constant representing its initial value z°, resulting in an EFSM system with equivalent behaviour.

If a specification is not controllable, *synthesis* is used to find a *supervisor*. 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 restrict the possible values of variables after a controllable transition.

Definition 9 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) $G \parallel S \subseteq_\mathrm{v} G \parallel E$;
(ii) S is Σ_u-controllable w.r.t. G;
(iii) For any EFSM S' that satisfies (i) and (ii), it holds that $G \parallel S' \subseteq_\mathrm{v} G \parallel S$.

Def. 9 characterises the possible synthesis results for a plant G and specification E. A correct supervisor must satisfy the specification through behavioural inclusion after composition with the plant (i), and it must be controllable (ii). It also must be *least restrictive* or *supremal*, i.e., any other controllable supervisor for the specification has less possible behaviour, again in composition with the plant (iii).

A supervisor satisfying these three conditions can be computed by means of a standard fixpoint iteration on the unfolded state set of $G \parallel E$, using the following operator.

Definition 10 [11] 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, let $V = \mathrm{vars}(G) \cup \mathrm{vars}(E)$, and let Σ_u be a set of events. The *extended synthesis step* operator $\Theta_{G,E,\Sigma_\mathrm{u}} \colon 2^{Q_G \times Q_E \times \mathrm{dom}(V)} \to 2^{Q_G \times Q_E \times \mathrm{dom}(V)}$ w.r.t. G, E, and Σ_u is defined as

$$\Theta_{G,E,\Sigma_\mathrm{u}}(X) = \{\, (x_G, x_E, \hat{v}) \in Q_G \times Q_E \times \mathrm{dom}(V) \mid \text{if } (x_G, \hat{v}) \xrightarrow{\mu} (y_G, \hat{w}) \text{ for some } \mu \in \Sigma_\mathrm{u} \text{ and } \hat{w} \in \mathrm{dom}(V), \text{ then there exists } y_E \in Q_E \text{ such that } (x_G, x_E, \hat{v}) \xrightarrow{\mu} (y_G, y_E, \hat{w}) \in X \,\} \,. \quad (16)$$

For a set X of combinations of locations and valuations of variables, the operator $\Theta_{G,E,\Sigma_\mathrm{u}}$ removes from X any uncontrollable states, i.e., states where the plant enables some uncontrollable transition not enabled by the specification, and any states from where the system could uncontrollably reach some combination of location and valuation not contained in X. The operator $\Theta_{G,E,\Sigma_\mathrm{u}}$ is monotonic

and has a greatest fixpoint $\hat{\Theta}_{G,E,\Sigma_\mathrm{u}}$ according to the Knaster-Tarski theorem [26]. In the finite-state case, this fixpoint is calculated as the limit of the iteration

$$X^0 = Q_Q \times Q_E \times \mathrm{dom}(V) \; ; \tag{17}$$

$$X^{j+1} = \Theta_{G,E,\Sigma_\mathrm{u}}(X^j) \; . \tag{18}$$

The result of EFSM synthesis is then obtained by restricting the system to this fixpoint.

Definition 11 [11] Let G and E be two EFSMs, and let Σ_u be a set of events. The *supremal Σ_u-controllable sub-EFSM* of G and E is

$$\sup\mathcal{C}(G, E, \Sigma_\mathrm{u}) = (G \parallel E) \restriction \hat{\Theta}_{G,E,\Sigma_\mathrm{u}} \; , \tag{19}$$

where $\hat{\Theta}_{G,E,\Sigma_\mathrm{u}}$ is the greatest fixpoint of the operator $\Theta_{G,E,\Sigma_\mathrm{u}}$ from Def. 10.

It is shown in [11,13] that this operation indeed gives a correct synthesis result according to Def. 9. Care must be taken, as the definition compares the supervisors after composition with the plant, and this requires location-determinism and vars′-normalisation to ensure synchrony of the states. The correctness result is stated as follows.

Proposition 4 [11, 13] Let G and E be location-deterministic EFSMs such that G is vars′-normalised, and let Σ_u be a set of events. Then $\sup\mathcal{C}(G, E, \Sigma_\mathrm{u})$ is a supremal supervisor for E w.r.t. G and Σ_u.

Given this result, the iteration defined by (17) and (18) can be used to compute the largest set of states $\hat{\Theta}_{G,E,\Sigma_\mathrm{u}}$ that can be allowed by any controllable supervisor, and from there the supervisor can be constructed. Although the definitions are based on explicit state sets, the locations and valuations can be encoded symbolically, e.g. using BDDs [3], to avoid the overhead of explicit state enumeration.

3.6 Chaos Abstraction

This paper is concerned with methods to modify or rewrite EFSMs or systems of composed EFSMs to simplify them and make synthesis procedures more efficient. Ordinary FSMs have useful modularity properties, according to which synchronous composition of a state machine with another only ever restricts the behaviour [2]. This makes it possible, for example, to remove components from a synchronous composition while preserving safety properties such as controllability, i.e., the controllability w.r.t. a part of the plant implies controllability w.r.t. the entire plant.

EFSMs do not have this property. When they are combined in synchronous composition, new next-state variables can be added to transitions, possibly changing variables that were implicitly unchanged. To obtain modularity properties similar to those known for FSMs, one solution [12] 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 12 [12] Given an event σ and a variable z, the *chaos EFSM* for σ and z is

$$\mathrm{chaos}(\sigma, z) = \langle \{\sigma\}, \{c\}, \{c\}, \{(c, \sigma, z' = *, c)\} \rangle \; . \tag{20}$$

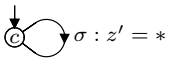

Fig. 2 The EFSM chaos(σ, z).

The EFSM chaos(σ, z) is shown in Fig. 2. The update $z' = *$ means that the variable z 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 z' so that z is no longer implicitly unchanged.

In the synchronous composition $F_1 \| F_2$ of two EFSMs, some variables in F_1 may be changed by transitions in F_2. A variable z can be changed after composition of a transition in F_1 that does not mention z' with a transition in F_2 that mentions z'; 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 5 shows how to identify the specific chaos EFSMs to capture possible variable changes in another EFSM.

Lemma 5 [12] Let $F_1 = \langle \Sigma_1, Q_1, Q_1^\circ, \to_1 \rangle$ and $F_2 = \langle \Sigma_2, Q_2, Q_2^\circ, \to_2 \rangle$ be two EFSMs, and let

$$C = \Big\| (\{\, \mathrm{chaos}(\sigma, z) \mid z \in \mathrm{vars}(F_1) \cap \mathrm{vars}'(F_2, \sigma) \,\}) \ . \tag{21}$$

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

In a synchronous composition $F_1 \parallel F_2$, the chaos abstraction of F_2 as defined by (21) is the composition of chaos EFSMs for variables in F_1 and events with transitions assigning to these variables in F_2. The condition $z \in \mathrm{vars}(F_1) \cap \mathrm{vars}'(F_2, \sigma)$ in (21) can only hold for variables shared between F_1 and F_2 and for events of F_2, so that the construction can be restricted to $\sigma \in \Sigma_2$. The composition C of these chaos EFSMs is a one-state EFSM with selfloop transitions $\sigma : z' = *$ for all events $\sigma \in \Sigma_2$ and variables $z \in \mathrm{vars}(F_1) \cap \mathrm{vars}'(F_2, \sigma)$. Lemma 5 allows F_2 to be replaced by this chaos EFSM C, such that all transitions in the composition $F_1 \parallel F_2$ are also possible in the abstraction $F_1 \parallel C$.

3.7 Existential Abstraction

An important feature of the results in this paper is the ability to simplify EFSMs through *variable abstraction* [28] using the existential quantifier. If $p \in \Pi$ is an update and z is a variable, then $\exists z\, p$ is an update that is true if and only if p can be made true by choosing some value for the variable z from its domain. That is, the range of quantification is always limited to the defined domain of the quantified variable.

In this paper, quantification is generalised to sets of variables as follows. For $W = \{z_0, z_1, \ldots, z_n\}$, it is defined that

$$\exists W\, p \ \equiv\ \exists z_0 \exists z_0' \exists z_1 \exists z_1' \cdots \exists z_n \exists z_n'\, p \ ; \tag{22}$$

$$\exists W'\, p \ \equiv\ \exists z_0' \exists z_1' \cdots \exists z_n'\, p \ . \tag{23}$$

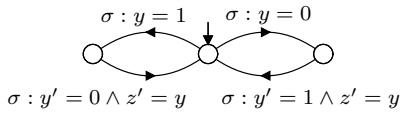

Fig. 3 Example of always enabled and unconstrained events, where $\mathrm{dom}(y) = \mathrm{dom}(z) = \{0, 1\}$ and $y^\circ = z^\circ = 0$.

That is, an update is quantified over variable set W by quantifying over both the current-state and next-state variables of W. Differently, quantification over W' means to quantify over the next-state variables only. The same notation is introduced for the universal quantifier, so $\forall W\, p$ is true if and only if p is true for all possible values of the current and next-state variables of W.

An EFSM is abstracted by existentially quantifying the updates on all the transitions.

Definition 13 [28] Let $F = \langle \Sigma, Q, Q^\circ, \to \rangle$ be an EFSM, and let W be a set of variables. The *existential abstraction* of F with respect to W is the EFSM $\exists W\, F = \langle \Sigma, Q, Q^\circ, \to_\exists \rangle$ where $x \xrightarrow{\sigma : \exists W\, p}_\exists y$ in $\exists W\, F$ if and only if $x \xrightarrow{\sigma : p} y$ in F.

The existential abstraction of a set $\mathcal{F} = \{F_1, \ldots, F_n\}$ of EFSMs is the set $\exists W\, \mathcal{F} = \{\exists W\, F_1, \ldots, \exists W\, F_n\}$ of the abstractions of the individual EFSMs.

Existential abstraction results in an EFSM that is independent of the quantified variables, i.e., $\mathrm{vars}(\exists W\, F) \cap W = \emptyset$. A transition in the abstraction is possible if there exist values for the quantified variables to make the update in the original EFSM F true. It is clear that existential abstraction preserves the EFSM properties of vars′-normalisation and purity. Location-determinism is not preserved, however, so it has to be required explicitly that an abstraction is location-deterministic in order for it to be used in synthesis.

Another concept related to existential quantification is that of always enabled events. An event in an ordinary FSM is always enabled if it has a transition from every state. Algorithm 1 uses this idea to avoid plants that never disable the uncontrollable events in question. With EFSMs, it may additionally be of interest that the event is enabled for all values of variables. The following definition requires an event that is enabled for all current-state values and some next-state values of the variables, while the next-state values for other variables only need to exist.

Definition 14 [12] Let $F = \langle \Sigma, Q, Q^\circ, \to \rangle$ be an EFSM, and let $W \subseteq \mathrm{vars}(F)$. An event $\sigma \in \Sigma$ is *always enabled* in F w.r.t. W if for all locations $x \in Q$ with σ-transitions

$$x \xrightarrow{\sigma : p_1} y_1 \quad \cdots \quad x \xrightarrow{\sigma : p_n} y_n \tag{24}$$

the formula

$$\exists W'\, (p_1 \vee \cdots \vee p_n) \tag{25}$$

is valid. An event set $\Sigma' \subseteq \Sigma$ is always enabled in F w.r.t. W if every event $\sigma \in \Sigma'$ has this property.

Example 4 Consider the EFSM F in Fig. 3 with $\mathrm{dom}(y) = \mathrm{dom}(z) = \{0, 1\}$. Event σ is always enabled in F w.r.t. $W_1 = \{y, z\}$, because independently of the current value of the variables y and z, it is always possible to choose values for y

and z such that some other location can be reached. But σ is *not* always enabled w.r.t. $W_2 = \{y\}$, because the update $y' = 0 \wedge z' = y$ is not possible when the next value of z is constrained to be different from the current value of y through synchronisation with some other component. Formally, $\exists y' \, (y' = 0 \wedge z' = y)$ is not valid because, if $y = 1$ and $z' = 0$ then it can never be that $z' = y$.

The following definition introduces the related but slightly different condition of unconstrained events, whose enablement does not depend on certain variables.

Definition 15 Let $F = \langle \Sigma, Q, Q^\circ, \rightarrow \rangle$ be an EFSM, and let $W \subseteq \mathrm{vars}(F)$. An event $\sigma \in \Sigma$ is *unconstrained* in F w.r.t. W if, for all transitions $x \xrightarrow{\sigma:p} y$ in F the formula

$$\exists W p \Rightarrow \forall W \exists W' p \tag{26}$$

is valid. An event set $\Sigma' \subseteq \Sigma$ is unconstrained in F w.r.t. W if every event $\sigma \in \Sigma'$ has this property.

The symbol $\Rightarrow$ in (26) denotes logical implication [8]. By convention (22) the quantification $\exists W$ and $\forall W$ is over both current-state and next-state variables, but in $\forall W \exists W'$ the universal quantification of the next-state variables is immediately overridden by $\exists W'$.

An update is unconstrained by a variable z if, in all cases where the update formula is true for some value of z, then it is also true for all other values of z in the current state, but possibly with different values of z' in the next state. While an always enabled event is enabled in every location, an unconstrained event, if it is enabled, is enabled independently of given variables.

Example 5 Consider again the EFSM F in Fig. 3. The event σ is *not* unconstrained w.r.t. $W_1 = \{y\}$, because the update $y = 1$ is possible when $y = 1$, so $\exists y \exists y' \, y = 1$ or equivalently $\exists y \; y = 1$ is true, but not for all other values of y, namely $\forall y \exists y' \, y = 1$ or equivalently $\forall y \; y = 1$ is not true. But σ is unconstrained w.r.t. $W_2 = \{z\}$, because all the transitions are enabled independently of the current value of z. That is, if a transition is enabled for some current-state value of z, then the transition can also be taken for all other values of z, possibly with different values for z' in the next state.

4 Abstraction of EFSMs in Modular Synthesis

This section proposes modular synthesis algorithms for composed EFSM systems. In general, the plant and specification consist of several EFSMs each, and the goal is to compute a least restrictive supervisor for the composed specification $E = E_1 \parallel \cdots \parallel E_k$ w.r.t. the composed plant $G = G_1 \parallel \cdots \parallel G_m$. As indicated in Section 3.2, all the EFSMs are assumed to be location-deterministic and vars$'$-normalised, and the specifications are in addition assumed to be pure.

To simplify the synthesis task, it is of interest to simplify both the plant and specification by reducing the number of components and by simplifying individual components through abstraction. It is known from modularity results for FSMs [2] and EFSMs [12] that the synthesis problem can be simplified by considering only one specification at a time, and the opportunities for simplification and abstraction

can be investigated separately for the plant and specification. Consequently, the following Section 4.1 starts by considering abstraction of the plant under the assumption that the specification consists of only one component. Later, Section 4.2 shows how a single specification can be simplified by abstraction, and finally Section 4.3 combines and generalises the results for synthesis with multiple specifications.

4.1 Abstracting the Plant

This section considers the possibilities of abstraction of the plant. It is assumed that the plant is defined as the synchronous composition of a set $\mathcal{G} = \{G_1, \ldots, G_m\}$ of vars'-normalised location-deterministic EFSMs, and the specification is given by a single pure location-deterministic EFSM E.

The idea of modular synthesis for FSMs as proposed in Algorithm 1 is to simplify the plant $\mathcal{G} = \{G_1, \ldots, G_m\}$ by selecting some of its components, and discarding the others, in such a way that the result is equivalent to that of synthesis w.r.t. the complete plant. In the EFSM setting, this is now generalised to the selection of plant components *and* variables. Algorithm 2 is further developed from Algorithm 1 for modular FSM synthesis by considering EFSMs, and also from the modular EFSM synthesis algorithm [12], in that it considers existential abstraction of variables in addition to component selection and chaos abstraction.

Similarly to Algorithm 1, the idea is to gradually increase the sets of plants $\mathcal{G}^i$, variables V^i, and uncontrollable events Σ_{u}^i, until it is guaranteed that the optimal result has been found. At each iteration, the algorithm considers sets of plants $\mathcal{G}^i \subseteq \mathcal{G}$ and variables $V^i \subseteq \text{vars}(\mathcal{G}) \cup \text{vars}(E)$, and also uncontrollable events $\Sigma_{\text{u}}^i \subseteq \Sigma_{\text{u}}$. Throughout the algorithm, $\bar{\mathcal{G}}^i$ and $\bar{V}^i$ are always the complements of $\mathcal{G}^i$ and V^i.

Initially, Algorithm 2 performs synthesis using only the specification E and its variables, as $\mathcal{G}^0 = \emptyset$ (line 3) and $V^0 = \text{vars}(E)$ (line 4). Following Lemma 5, the plants $\bar{\mathcal{G}}^0 = \mathcal{G}$ are replaced by chaos EFSMs $\mathcal{C}^0$ for the included variables (line 5). The set of uncontrollable events Σ_{u}^0 is initially empty (line 2), i.e., synthesis is first performed under the pretence that all events are controllable. In this case, the synthesis result S^0 is equal to the specification E (line 6).

Therefore, on entering the loop for the first time, the loop entry condition on line 8 checks whether the specification $S^0 = E$ is controllable with respect to only the chaos EFSM $\mathcal{C}^0$ (recall that $\mathcal{G}^0 = \emptyset$), based on the full set Σ_{u} of uncontrollable events. This may succeed if, for example, the specification has only controllable events, in which case $S^0 = E$ is returned as the least restrictive solution. Otherwise the loop is entered and synthesis is performed w.r.t. increased subsets of plants, variables, and uncontrollable events, which are computed as follows.

First, line 9 calculates a new set Σ_{u}^{i+1} of uncontrollable events. As the current supervisor S^i is not controllable by the loop entry condition, there must be some uncontrollable event that is possible in the plant but not in the specification. These events are called the *causes of uncontrollability*, as per the following definition.

Algorithm 2: Modular abstracting EFSM synthesis for a single specification

Input: vars′-normalised location-deterministic plants $\mathcal{G} = \{G_1, \ldots, G_m\}$;
pure location-deterministic specification E;
uncontrollable events Σ_u.

Output: supremal supervisor S^k for $\big\|(\mathcal{G})$ w.r.t. E and Σ_u.

1 $V \leftarrow \mathrm{vars}(G) \cup \mathrm{vars}(E)$;
2 $\Sigma_\mathrm{u}^0 \leftarrow \emptyset$;
3 $\mathcal{G}^0 \leftarrow \emptyset$; $\bar{\mathcal{G}}^0 \leftarrow \mathcal{G}$;
4 $V^0 \leftarrow \mathrm{vars}(E)$; $\bar{V}^0 \leftarrow V \setminus V^0$;
5 $\mathcal{C}^0 \leftarrow \{\, \mathrm{chaos}(\sigma, v) \mid v \in V^0 \cap \mathrm{vars}'(\bar{\mathcal{G}}^0, \sigma) \,\}$;
6 $S^0 \leftarrow E$;
7 $i \leftarrow 0$;
8 **while** S^i is not Σ_u-controllable w.r.t. $\big\|(\exists \bar{V}^i\, \mathcal{G}^i) \parallel \big\|(\mathcal{C}^i)$ **do**
9 $\quad \Sigma_\mathrm{u}^{i+1} \leftarrow \Sigma_\mathrm{u}^i \cup \mathrm{uncont}(\big\|(\exists \bar{V}^i\, \mathcal{G}^i) \parallel \big\|(\mathcal{C}^i), S^i, \Sigma_\mathrm{u})$;
10 $\quad$ Choose $\mathcal{G}^{i+1} \subseteq \mathcal{G}$, $V^{i+1} \subseteq V$, $\bar{\mathcal{G}}^{i+1} = \mathcal{G} \setminus \mathcal{G}^{i+1}$, and $\bar{V}^{i+1} = V \setminus V^{i+1}$
such that $\mathrm{vars}(E) \subseteq V^{i+1} \subseteq \mathrm{vars}(\mathcal{G}^{i+1}) \cup \mathrm{vars}(E)$
and Σ_u^{i+1} is always enabled in $\bar{\mathcal{G}}^{i+1}$ w.r.t. $\bar{V}^{i+1}$
and Σ_u^{i+1} is unconstrained in $\mathcal{G}^{i+1}$ w.r.t. $\bar{V}^{i+1}$
and $\mathrm{vars}'(\mathcal{G}^{i+1}, \mu) \cap \mathrm{vars}'(\bar{\mathcal{G}}^{i+1}, \mu) \cap \bar{V}^{i+1} = \emptyset$ for each $\mu \in \Sigma_\mathrm{u}^{i+1}$
and $\exists \bar{V}^{i+1}\, \mathcal{G}^{i+1}$ is location-deterministic;
11 $\quad \mathcal{C}^{i+1} \leftarrow \{\, \mathrm{chaos}(\sigma, z) \mid z \in V^{i+1} \cap \mathrm{vars}'(\bar{\mathcal{G}}^{i+1}, \sigma) \,\}$;
12 $\quad S^{i+1} \leftarrow \sup\mathcal{C}\big(\, \big\|(\exists \bar{V}^{i+1}\, \mathcal{G}^{i+1}) \parallel \big\|(\mathcal{C}^{i+1}), E, \Sigma_\mathrm{u}^{i+1}\big)$;
13 $\quad i \leftarrow i + 1$;
14 **end**
15 **return** S^i;

Definition 16 Let G and E be two EFSMs, and let Σ_u be a set of events. The set of *causes of Σ_u-uncontrollability* of E w.r.t. G is the set of events

$$\begin{aligned}\mathrm{uncont}(G, E, \Sigma_\mathrm{u}) = \{\, \mu \in \Sigma_\mathrm{u} \mid \text{there exist } (x_G, x_E, \hat{v}) \in Q^{\mathrm{acc}}(G \parallel E) \text{ and } \\ (x_G, \hat{v}) \xrightarrow{\mu} (x_G, \hat{w}) \text{ in } G, \text{ and there is no location} \\ y_E \text{ in } E \text{ such that } (x_G, x_E, \hat{v}) \xrightarrow{\mu} (y_G, y_E, \hat{w}) \,\}\ .\end{aligned} \tag{27}$$

It follows from Def. 8 that the set of causes of uncontrollability is empty, $\mathrm{uncont}(G, E, \Sigma_\mathrm{u}) = \emptyset$, if and only if G is Σ_u-controllable w.r.t. E. As the loop entry condition has found the supervisor S^i to be not Σ_u^i-controllable w.r.t. the plant abstraction $\big\|(\exists \bar{V}^i\, \mathcal{G}^i) \parallel \big\|(\mathcal{C}^i)$, there exist some causes of uncontrollability, which are included in the next set Σ_u^{i+1} of uncontrollable events on line 9. This ensures that they are treated as uncontrollable for the next synthesis attempt. For other uncontrollable events, the algorithm will continue to pretend that they are controllable.

Next, line 10 chooses new plants $\mathcal{G}^{i+1}$ and variables V^{i+1} to form an improved approximation. First, all variables used in the specification are retained,

$$\mathrm{vars}(E) \subseteq V^{i+1}\ . \tag{28}$$

In this section, variables that appear in the specification are not abstracted. Later, in Section 4.2, it is shown how the amount of variables in the specification can be reduced by specification abstraction before Algorithm 2 is invoked.

The main part of the logic in Algorithm 2 is the selection of plant components and variables from them. To ensure a least restrictive synthesis result, all plant components that can disable some uncontrollable event are included [1,12]. Therefore it is required that the uncontrollable events must be always enabled by the plants $\bar{\mathcal{G}}^{i+1}$ and variables $\bar{V}^{i+1}$ not included in synthesis according to Def. 14,

$$\Sigma_{\mathrm{u}}^{i+1} \text{ is always enabled in } \bar{\mathcal{G}}^{i+1} \text{ w.r.t. } \bar{V}^{i+1}. \tag{29}$$

This condition ensures that any plant components that could ever cause disablement of some uncontrollable event from $\Sigma_{\mathrm{u}}^{i+1}$ are included in the abstraction. Furthermore, all variables that can constrain these uncontrollable events in the selected plant components $\mathcal{G}^{i+1}$ must also be included. This can be ensured by including all variables that appear in the selected plants on transitions with the selected uncontrollable events, or as weaker condition it is enough that the uncontrollable events are unconstrained by the other variables according to Def. 15,

$$\Sigma_{\mathrm{u}}^{i+1} \text{ is unconstrained in } \mathcal{G}^{i+1} \text{ w.r.t. } \bar{V}^{i+1}. \tag{30}$$

In order to consider the conditions (29) and (30) separately for $\mathcal{G}^{i+1}$ and $\bar{\mathcal{G}}^{i+1}$, it is furthermore necessary that there are no conflicting values of the same abstracted variable. Therefore it is required that $\mathcal{G}^{i+1}$ and $\bar{\mathcal{G}}^{i+1}$ do not share these variables in their primed form,

$$\mathrm{vars}'(\mathcal{G}^{i+1}, \mu) \cap \mathrm{vars}'(\bar{\mathcal{G}}^{i+1}, \mu) \cap \bar{V}^{i+1} = \emptyset \quad \text{for each } \mu \in \Sigma_{\mathrm{u}}^{i+1} \ . \tag{31}$$

This condition can be checked separately for each event μ considered as uncontrollable in the current iteration.

To summarise, in addition to including all variables from the specification (28), the selected uncontrollable events must be always enabled (29) in the plants $\bar{\mathcal{G}}^{i+1}$ *not* included in the approximation and unconstrained (30) in the plants $\mathcal{G}^{i+1}$ that are included, and the two parts $\mathcal{G}^{i+1}$ and $\bar{\mathcal{G}}^{i+1}$ of the plant must not use any of the abstracted variables in their primed form with the same uncontrollable event (31).

The variables not included in V^{i+1}, i.e., those in $\bar{V}^{i+1}$, are removed by existential abstraction. In order for synthesis to be well-defined, the abstraction must remain location-deterministic. This is ensured by the last condition,

$$\exists \bar{V}^{i+1}\, \mathcal{G}^{i+1} \text{ is location-deterministic} \ . \tag{32}$$

After the plants and variables for the next step have been chosen, line 11 introduces chaos EFSMs $\mathcal{C}^{i+1}$ to replace the plants $\bar{\mathcal{G}}^{i+1}$ that are not included, such that any possible changes to the included variables V^{i+1} are reflected in the abstraction [12].

Then line 12 performs synthesis for the plant abstraction $\big\|(\exists \bar{V}^{i+1}\, \mathcal{G}^{i+1}) \parallel \big\|(\mathcal{C}^{i+1})$ and the chosen set $\Sigma_{\mathrm{u}}^{i+1}$ of uncontrollable events. If the resulting supervisor is controllable w.r.t. the full set Σ_{u} of uncontrollable events (line 8), then it is returned as the result. Otherwise more uncontrollable events need to be included, resulting in a new plant abstraction. The loop continues until a Σ_{u}-controllable solution is found.

Line 10 of Algorithm 2 may be difficult to implement as it is not specified how the plant components $\mathcal{G}^{i+1}$ and variables V^{i+1} can be chosen such that conditions (28)–(32) are satisfied at the same time. A simple approach is to start with the variables of the specification (28) and the plants that disable an uncontrollable event from Σ_{u}^i in some location (29), and then gradually add more plants and variables until all conditions are satisfied. The search may be simpler for well-designed EFSM models in practice, such as the flexible manufacturing system presented in Section 5 where not all conditions need to be considered.

A formal correctness proof of Algorithm 2 is given in [13]. It is clear that termination is guaranteed because the set Σ_{u}^i of uncontrollable events increases with every iteration, and it is bounded by the finite set Σ_{u} of all uncontrollable events. In order to see that the algorithm returns a correct result, i.e., that the supervisor S^i returned from line 15 is indeed a supremal supervisor for E w.r.t. $\mathcal{G}$ and Σ_{u}, it is first observed from line 12 that

$$S^i = \sup\mathcal{C}\big(\big\|(\exists \bar{V}^i\, \mathcal{G}^i) \parallel \big\|(\mathcal{C}^i), E, \Sigma_{\mathrm{u}}^i\big) \tag{33}$$

is a supremal supervisor for E w.r.t. $\big\|(\exists \bar{V}^i\, \mathcal{G}^i) \parallel \big\|(\mathcal{C}^i)$ and Σ_{u}^i. Then it can be shown from the properties of the abstraction that, for all i,

$$S^i \text{ is a supremal supervisor for } E \text{ w.r.t. } \mathcal{G} \text{ and } \Sigma_{\mathrm{u}}^i\ . \tag{34}$$

That is, the computed supervisor is supremal not only for the plant abstraction but also for the original, unabstracted plant $\mathcal{G}$. The controllability of S^i and its inclusion in the specification follow from the fact that this supervisor is computed w.r.t. an over-approximation of the plant, i.e., because the original plant $\mathcal{G}$ is behaviourally included in its abstraction. The least restrictiveness of S^i is more difficult to ensure, and it depends on the precise conditions (28)–(31) for the choice of the abstraction.

Once (34) is established, the only difference between S^i and the desired result is the uncontrollable event set: S^i is synthesised w.r.t. $\Sigma_{\mathrm{u}}^i \subseteq \Sigma_{\mathrm{u}}$. As S^i uses fewer uncontrollable events, it can be shown that it over-approximates the synthesis result w.r.t. the full uncontrollable event set Σ_{u},

$$\sup\mathcal{C}(\big\|(\mathcal{G}), E, \Sigma_{\mathrm{u}}) \subseteq_{\mathrm{v}} S^i\ . \tag{35}$$

On termination of the loop, S^i is not only Σ_{u}^i-controllable but also Σ_{u}-controllable. At this point, it follows from (34) and (35) that S^i is a supremal supervisor for E w.r.t. $\mathcal{G}$ and Σ_{u}.

These observations lead to the following correctness result of Algorithm 2.

Theorem 6 [13] Algorithm 2 terminates, and upon termination the result S^i is a supremal supervisor for E w.r.t. $\big\|(\mathcal{G})$ and Σ_{u}.

To estimate the time complexity of Algorithm 2, it is first observed that each iteration of the loop on line 8 must add at least one variable to V^{i+1} or one plant component to $\mathcal{G}^{i+1}$, so the size of the subsystem abstraction increases by at least one component. Thus, the number of iterations is bounded by the number $|\mathcal{G}|$ of plant components plus the number $|V|$ of variables. In the loop, line 10 requires several operations to identify and compute existential abstractions, but these can be performed locally by analysing individual EFSM components rather than their

synchronous composition. Therefore, the time complexity of the loop body is dominated by the synthesis step on line 12, which is polynomial in the size of the state space but exponential in the number of components included.

Then the worst-case time complexity of Algorithm 2 is obtained by multiplying the exponential complexity of synthesis on line 12 with the number of plant components plus variables, $|\mathcal{G}| + |V|$. In the worst case, Algorithm 2 is slower than ordinary synthesis based on Prop. 4 by a linear factor. However, this worst case only arises if all the variables and plant components are needed to synthesise the supervisor. It is more likely that the loop terminates early, in which case the synthesis is performed with a smaller number of components, resulting in an exponential reduction of the number of states and the runtime, while multiplying it by only a linear factor in the number of components and variables. This behaviour has been observed experimentally for FSM verification [2], and it also occurs in the example in Section 5 below.

4.2 Abstracting the Specification

This subsection considers the case of a single specification EFSM E, assumed to be location-deterministic and pure, and a single vars'-normalised and location-deterministic plant EFSM G. The developed results can be combined with Section 4.1 above to perform synthesis for a plant that consists of several EFSM components. The concern here is whether any variables can be existentially abstracted from the specification E.

As noted in Remark 1 on page 12, variables that appear only in the pure specification and not in the plant, can be removed by replacing them with a constant representing their initial value. This trivial case is not considered further. The question then is whether any variables shared between the plant and specification can be existentially abstracted from the specification.

By closely inspecting the synthesis process, it can be observed that only the updates of the uncontrollable events are relevant for the removal of states. This suggests the existential quantification of variables that only appear on transitions with controllable events in the specification. That is, if a set of variables $\bar{V} \subseteq \text{vars}(E)$ does not contain any variables used with uncontrollable events in E,

$$\bar{V} \cap \text{vars}(E, \Sigma_{\text{u}}) = \emptyset \ , \tag{36}$$

then it is enough to synthesise for the abstracted specification $\exists \bar{V} E$ instead of E. Such synthesis only makes sense for a location-deterministic abstraction, so a second assumption is made that

$$\exists \bar{V} E \text{ is location-deterministic} \ . \tag{37}$$

Then synthesis will ensure that all constraints associated with uncontrollable events in E are enforced by a controllable supervisor. However, the constraints associated with controllable events are not properly included in the abstraction $\exists \bar{V} E$ and may not be carried forward in the synthesis result. Fortunately, controllable constraints can easily be enforced in a supervisor without the need for synthesis—it is enough to use the updates on the controllable transitions in E on the corresponding transitions in the synthesis result. Under the assumption of location-

determinism, this can be achieved by composing the synthesis result for the abstracted specification with the original specification.

Therefore, to compute a supervisor for a specification E, it is possible to first find an abstraction $\exists \bar{V} E$ subject to (36) and (37), and then compute a supervisor $S_\exists = \sup\mathcal{C}(G, \exists \bar{V} E, \Sigma_\mathrm{u})$ for the abstraction, e.g., using Algorithm 2. Then a supervisor for the original specification is obtained by composing the result $S_\exists$ obtained with the abstraction with the original specification E, i.e.,

$$S_\exists \parallel E \text{ is a supremal supervisor for } E \text{ w.r.t. } G \text{ and } \Sigma_\mathrm{u} \ . \tag{38}$$

The following main result for specification abstraction states that (38) holds under the assumptions (36) and (37) for every supremal supervisor $S_\exists$ for $\exists \bar{V} E$ w.r.t. G and Σ_u.

Theorem 7 [13] Let G and E be EFSMs such that E is pure, let Σ_u be a set of events, and let $\bar{V} \subseteq \mathrm{vars}(G) \cap \mathrm{vars}(E)$ such that $\bar{V} \cap \mathrm{vars}(E, \Sigma_\mathrm{u}) = \emptyset$ and $\exists \bar{V} E$ is location-deterministic. If $S_\exists$ is a supremal supervisor for $\exists \bar{V} E$ w.r.t. G and Σ_u, then $S_\exists \parallel E$ is a supremal supervisor for E w.r.t. G and Σ_u.

The proof of this result is given in [13]. Inclusion in the specification is clear because the proposed supervisor $S_\exists \parallel E$ includes the specification, and least restrictiveness follows because synthesis is done w.r.t. the more liberal specification $\exists \bar{V} E$. The crucial issue here is controllability, which depends on the condition (36) to ensure that the updates associated with uncontrollable events are completely retained in the abstraction.

To use Theorem 7 for synthesis, one has to find a set $\bar{V}$ of variables satisfying (36) and (37) and compute the abstraction $\exists \bar{V} E$. While the variables satisfying (36) are easily found by removing the variables used on transitions with uncontrollable events in the specification, the requirement (37) of location-determinism is more difficult to ensure algorithmically. A simple solution is to start with all variables used only on transitions with controllable events in the specification, $\bar{V} = \mathrm{vars}(E) \backslash \mathrm{vars}(E, \Sigma_\mathrm{u})$, and gradually remove variables that cause failure of $\exists \bar{V} E$ being location-deterministic, until (37) is satisfied. Once an appropriate set $\bar{V}$ of variables for abstraction is found, Algorithm 2 can be used to compute a synthesis result for $\sup\mathcal{C}(G, \exists \bar{V} E, \Sigma_\mathrm{u})$, which then can be combined with the specification E to obtain a correct supervisor for the original synthesis problem.

4.3 Synthesis with Multiple Specifications

The results presented in the previous Sections 4.1 and 4.2 show how abstractions can be used to compute a supervisor for a single specification. In general, the specification is given in modular form, as a synchronous composition $E_1 \parallel \cdots \parallel E_k$ of several EFSMs. In this case, it is known [1,2] for ordinary FSMs that synthesis can be performed separately for each specification, and the resulting supervisors can be combined to form a least restrictive controllable supervisor for the combined specification. Under the assumption of pure specifications, these results can be generalised directly for EFSMs [11, 12].

Algorithm 3 uses this idea to synthesise a modular supervisor for an EFSM system composed of several plants $G_1 \parallel \cdots \parallel G_m$ and specifications $E_1 \parallel \cdots \parallel E_k$,

Algorithm 3: Modular abstracting EFSM synthesis for multiple specifications

Input: vars′-normalised location-deterministic plants $\mathcal{G} = \{G_1, \ldots, G_m\}$;
pure specifications $\mathcal{E} = \{E_1, \ldots, E_k\}$;
uncontrollable events Σ_u.

Output: collection $\mathcal{S}$ of supervisors such that $\|(\mathcal{S})$ is a supremal supervisor for $\|(\mathcal{E})$ w.r.t. $\|(\mathcal{G})$ and Σ_u.

1 $\mathcal{S} \leftarrow \emptyset$;
2 **foreach** $E_j \in \mathcal{E}$ **do**
3 | Choose $\bar{V}_j \subseteq \mathrm{vars}(E_j) \setminus \mathrm{vars}(E_j, \Sigma_\mathrm{u})$ such that $\exists \bar{V}_j\, E_j$ is location-deterministic;
4 | Calculate S_j using Algorithm 2 with $E = \exists \bar{V}_j\, E_j$;
5 | $\mathcal{S} \leftarrow \mathcal{S} \cup \{S_j, E_j\}$;
6 **end**
7 **return** $\mathcal{S}$;

while incorporating the results from Sections 4.1 and 4.2. The loop on line 2 processes each specification E_j, by first abstracting it according to Section 4.2 and then computing a supervisor using plant abstractions according to Section 4.1. On line 3, the variables $\bar{V}_j$ for abstraction of the specification E_j are chosen to satisfy assumptions (36) and (37), and then line 4 invokes Algorithm 2 to compute a supremal supervisor S_j for the specification abstraction $\exists \bar{V}_j\, E_j$. In this case, Theorem 7 states that the composition $S_j \parallel E_j$ of the supervisor computed using the specification abstraction and the original specification is a supremal supervisor, and therefore both EFSMs S_j and E_j are added to the modular supervisor $\mathcal{S}$ on line 5.

The main argument for the correctness of Algorithm 3 is the observation that synthesis for a modular specification can be performed separately for each specification. This known result for ordinary FSMs [1, 2] is easily lifted to EFSMs.

Proposition 8 [13] Let G be a vars′-normalised EFSM, let $E_1, \ldots, E_k$ be pure EFSMs for some $k \geq 0$, let Σ_u be a set of events, and let S_j be a supremal supervisor for E_j w.r.t. G and Σ_u for $j = 1, \ldots, k$. Then $S_1 \parallel \cdots \parallel S_k$ is a supremal supervisor for $E_1 \parallel \cdots \parallel E_k$ w.r.t. G and Σ_u.

A proof of this result first appears in [11], and a revised version based on the modified definition of behavioural inclusion in this paper is given in [13].

The result from Prop. 8 about separate synthesis leads to the correctness of Algorithm 3. Each iteration in the loop gives a supremal supervisor by Theorems 6 and 7, and then their combination also is a supremal supervisor by Prop. 8. It is also clear that the algorithm terminates because the loop performs exactly one iteration for each specification $E_1, \ldots, E_k$.

Theorem 9 [13] Algorithm 3 terminates, and upon termination the composition $\|(\mathcal{S})$ of the results is a supremal supervisor for the composed specification $\|(\mathcal{E})$ w.r.t. the composed plant $\|(\mathcal{G})$ and uncontrollable event set Σ_u.

To estimate the time complexity of Algorithm 3, it is noted that the loop on line 2 performs one iteration per specification component, i.e., $|\mathcal{E}|$ iterations.

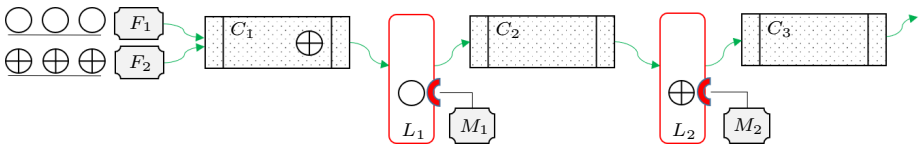

Fig. 4 Flexible Manufacturing System Layout.

Within the loop, the operations to choose the variable set $\bar{V}_j$ and compute the existential quantification on line 3 only depend on a single component E_j, so they are dominated by the complexity of executing Algorithm 2 on line 4, which is exponential in the number of plant components plus variables, $|\mathcal{G}|+|V|$. Multiplying this by the number $|\mathcal{E}|$ of loop iterations, gives the result that the runtime of Algorithm 3 is exponential in $|\mathcal{G}| + |V|$ and linear in $|\mathcal{E}|$. This is an exponential improvement over ordinary synthesis based on Prop. 4, which is exponential in the number of components plus variables, i.e., exponential in $|\mathcal{G}| + |\mathcal{E}| + |V|$.

This completes the exposition of the modular abstraction-based synthesis algorithm for EFSMs and its correctness. Algorithm 3 can be used to compute a least restrictive supervisor for any combination of vars′-normalised location-deterministic plant and pure location-deterministic specification EFSMs.

5 Flexible Manufacturing System Example

This section applies the proposed synthesis procedure from Algorithm 3 to compute a modular least restrictive controllable supervisor for an EFSM model of a flexible manufacturing system. This model is a further developed version of an earlier example [24]. Fig. 4 shows the layout of the system.

Workpieces enter the system through one of two feeders (F_1 or F_2) and are placed on the first conveyor (C_1), which delivers them to the first production line (L_1). In L_1, the workpieces may be processed by the first machine (M_1) and then put on the second conveyor (C_2), or they may be put on C_2 immediately without processing. After passing conveyor C_2 the workpieces may be processed by a similar production line (L_2) and machine (M_2), and after that they are put on the last conveyor (C_3) before exiting the system.

The two feeders deliver two different types of workpieces: F_1 delivers type 1 workpieces, and F_2 delivers type 2 workpieces. The decision which feeder is used is outside of the scope of the model. The objective is to control the system in such a way that type 1 workpieces are only processed by machine M_1, and likewise type 2 workpieces are only processed by M_2.

The modelling of the different workpiece types is facilitated by the use of EFSM variables, as demonstrated in the plant model in Fig. 5. The variables c_1, c_2, c_3, l_1, and l_2 represent the contents of the conveyors and production lines. Their domain is $\{0, 1, 2, 1^\dagger, 2^\dagger\}$, where the initial value is 0 and means that the corresponding conveyor or production line is empty. A value of 1 or 2 indicates the presence of a raw workpiece of type 1 or 2, and a value of $1^\dagger$ or $2^\dagger$ indicates the presence of a workpiece of type 1 or 2 that has been processed by its corresponding machine M_1 or M_2.

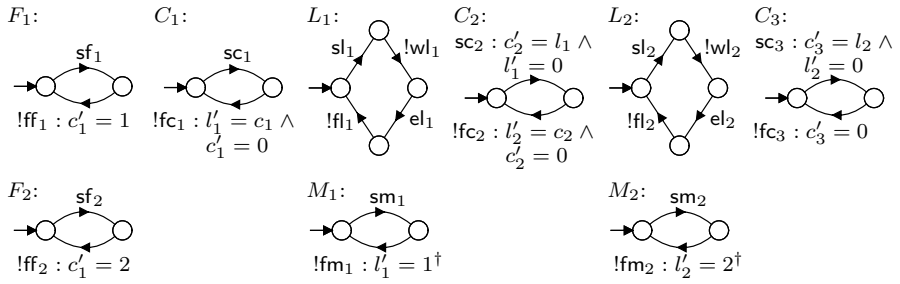

Fig. 5 Flexible Manufacturing System Plants.

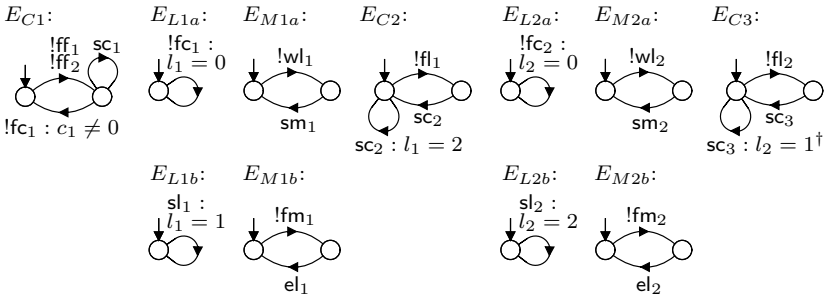

Fig. 6 Flexible Manufacturing System Specifications.

In the model, uncontrollable events are prefixed with an exclamation mark (!) to distinguish them from the controllable events. The plant EFSM F_1 shows that feeder F_1 can be started controllably with event sf_1, then finishes uncontrollably with event $!\mathsf{ff}_1$ and upon finishing puts a type 1 workpiece on conveyor C_1 as indicated by the update $c_1' = 1$. Plant F_2 describes the analogous behaviour of feeder F_2. Similarly, conveyor C_1 is started with sc_1, and upon finishing with $!\mathsf{fc}_1$ its workpiece is put into the first production line, $l_1' = c_1$, and removed from the conveyor, $c_1' = 0$. Conveyors C_2 and C_3 are similar, but in addition remove a workpiece from the production line in front of them when starting. Production line L_1 is requested to pick up a workpiece with controllable event sl_1, and the completion of the pick-up is indicated by uncontrollable event $!\mathsf{wl}_1$ after which the workpiece is available for machine M_1. Then the production line can be requested to eject the workpiece (el_1) and upon completion ($!\mathsf{fl}_1$) the workpiece again becomes available for conveyor C_2. Machine M_1 can be requested to start processing (sm_1), and when it finishes ($!\mathsf{fm}_1$) it changes the workpiece in the production line, $l_1' = 1^\dagger$, to indicate a processed workpiece of type 1. Production line L_2 and machine M_2 work in the same way.

Fig. 6 shows specification EFSMs that capture several control requirements for the flexible manufacturing system. Specification E_{C1} controls the synchronisation between the feeders and conveyor C_1. Conveyor C_1 can only start (sc_1) after having been loaded with a workpiece, i.e., after one of the feeders has completed ($!\mathsf{ff}_1$ or $!\mathsf{ff}_2$), and the feeders can only start again after the conveyor has finished ($!\mathsf{fc}_1$). Through the guard $c_1 \neq 0$ it is also required that there must be a workpiece on the conveyor when it finishes. Specification E_{L1a} rules out overflow of production line L_1, because conveyor C_1 is only allowed to deliver a workpiece ($!\mathsf{fc}_1$) when

the line is empty, $l_1 = 0$. Specification E_{L1b} requires that only unprocessed type 1 workpiece may enter production line L_1. Specifications E_{M1a} and E_{M1b} require that machine M_1 only starts (sm_1) when there is a workpiece for it to process ($!\mathsf{wl}_1$), and the workpiece is only ejected (el_1) after being processed by the machine ($!\mathsf{fm}_1$). Specification E_{C2} constrains the starting (sc_2) of conveyor C_2: conveyor C_2 may start when production line L_1 contains a type 2 workpiece, $l_1 = 2$, as these workpieces should bypass L_1, or after production line L_1 has returned a processed workpiece ($!\mathsf{fl}_1$). Specifications E_{L2a}, E_{L2b}, E_{M2a}, E_{M2b}, and E_{C3} constrain the behaviour of production line L_2 and conveyor C_3 in a similar way.

It is clear that the EFSM model satisfies the structural requirements outlined for Algorithms 2 and 3. All the plants are vars′-normalised and all the specifications are pure. Also, all the EFSMs are location-deterministic, and so are all possible abstractions as no location has more than one outgoing transition for any given event. Moreover, it can be seen in Fig. 5 that for any given event, no next-state variable appears in more than one EFSM on transitions with that event, so that condition (31) will be trivially satisfied for any abstraction considered. It is quite typical for well-designed EFSM models to have such properties, particularly in the manufacturing context.

To synthesise a least restrictive supervisor, Algorithm 3 processes each of the specifications in Fig. 6. The order in which the specifications are processed is not important, so the following explanation starts with the easiest cases. The supervisors computed by Algorithm 2 are shown in Fig. 7.

- Specification E_{L1b} has only one controllable event, sl_1, so its variable l_1 is only used controllably and can be abstracted. Algorithm 3 forms the abstraction $\exists l_1 E_{L1b}$, and as $\exists l_1\ l_1 = 1$ is true, this simplifies to a one-state FSM with a controllable selfloop, which is trivially controllable and returned unchanged by Algorithm 2.
  Then the abstraction $\exists l_1 E_{L1b}$ becomes the first supervisor collected by Algorithm 3. It appears in Fig. 7 as S_{L1b}. It performs no control as a supervisor. Therefore Algorithm 3 also includes the original specification E_{L1b} as a supervisor, which ensures through the update $l_1 = 1$ that production line L_1 only starts (sl_1) when a workpiece of type 1 is available.
- Specification E_{M1a} has no variables, so Algorithm 3 passes it to Algorithm 2 unchanged. As E_{M1a} disables the uncontrollable event $!\mathsf{wl}_1$, it is found not controllable at the beginning of Algorithm 2, so the algorithm assigns $\Sigma_{\mathrm{u}}^1 = \{!\mathsf{wl}_1\}$ and searches for plants that disable this cause of uncontrollability. This yields L_1, which also has no variables. Therefore Algorithm 2 chooses $\mathcal{G}^1 = \{L_1\}$, $V^1 = \emptyset$, and $\mathcal{C}^1 = \emptyset$. Synthesis results in the supervisor $S^1 = \sup\mathcal{C}(L_1, E_{M1a}, \{!\mathsf{wl}_1\})$, which is also $!\mathsf{fl}_1$-controllable.
  This supervisor is shown as S_{M1a} in Fig. 7. It is returned from Algorithm 2 and collected by Algorithm 3. The supervisor S_{M1a} ensures that machine M_1 is only started (sm_1) when a workpiece is available, i.e., after $!\mathsf{wl}_1$ has occurred, and that the production line can only be started again after starting M_1.
- Specification E_{M1b} has no variables, so Algorithm 3 passes it to Algorithm 2 unchanged. As E_{M1b} disables the uncontrollable event $!\mathsf{fm}_1$, it is found not controllable at the beginning of Algorithm 2, so the algorithm assigns $\Sigma_{\mathrm{u}}^1 = \{!\mathsf{fm}_1\}$ and searches for plants that disable this cause of uncontrollability. This yields M_1, which includes the variable l_1. Yet on closer inspection $!\mathsf{fm}_1$ is uncon-

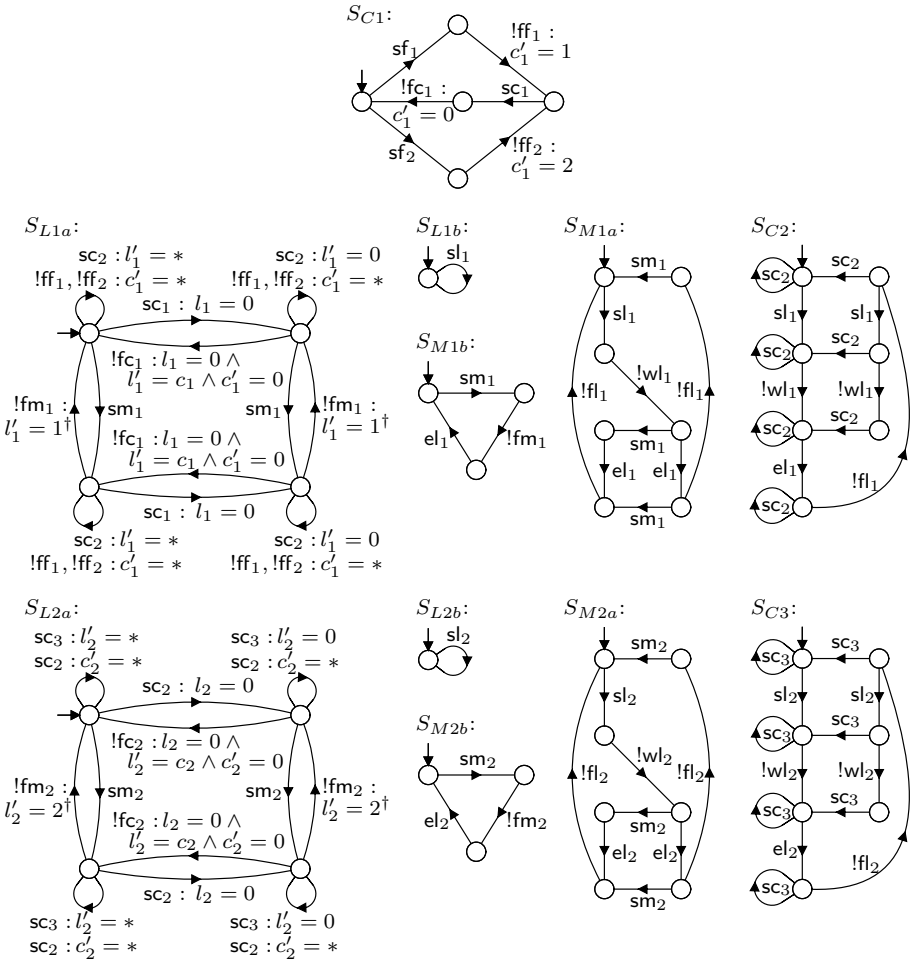

Fig. 7 Synthesised supervisors for flexible manufacturing system.

strained in M_1 w.r.t. l_1 (note that $\forall l_1 \exists l_1'\ l_1' = 1$ is true, which is enough to establish the validity of (26) in Def. 15). Therefore Algorithm 2 chooses the plant abstraction $\mathcal{G}^1 = \{\exists l_1 M_1\}$, no variables, $V^1 = \emptyset$, and no chaos EFSMs, $\mathcal{C}^1 = \emptyset$. Then synthesis results in the supervisor $S^1 = \sup\mathcal{C}(\exists l_1 M_1, E_{M1b}, \{!\mathsf{fm}_1\})$, which is controllable as no other uncontrollable events are involved.

This supervisor, shown as S_{M1b} in Fig. 7, is returned from Algorithm 2 and collected by Algorithm 3. It ensures that production line L_1 only starts the ejection (el_1) of a workpiece after it has been processed ($!\mathsf{fm}_1$) by machine M_1.

– Specification E_{C1} includes the variable c_1, which is used uncontrollably with event $!\mathsf{fc}_1$ and cannot be abstracted by Algorithm 3, so E_{C1} is passed to Algorithm 2 unchanged. This specification is not controllable by itself as it disables uncontrollable events $!\mathsf{ff}_1$, $!\mathsf{ff}_2$, and $!\mathsf{fc}_1$. In its first iteration, Algorithm 2 selects $\Sigma_{\mathrm{u}}^1 = \{!\mathsf{ff}_1, !\mathsf{ff}_2, !\mathsf{fc}_1\}$, which leads it to identify the plants F_1, F_2, and C_1 and the variable c_1 that appears in E_{C1}. Plant C_1 also includes the variable l_1, but

Σ_u^1 is unconstrained in C_1 w.r.t. l_1, because for the $!\mathsf{fc}_1$-transition in C_1 the condition (26) in Def. 15 becomes

$$\exists l_1 \exists l_1' (l_1' = c_1 \wedge c_1' = 0) \Rightarrow \forall l_1 \exists l_1' (l_1' = c_1 \wedge c_1' = 0) \ , \tag{39}$$

which can be simplified by removing the unused variable l_1 from quantification to give

$$\exists l_1' (l_1' = c_1 \wedge c_1' = 0) \Rightarrow \exists l_1' (l_1' = c_1 \wedge c_1' = 0) \ , \tag{40}$$

and the latter is clearly valid. This shows that the transition is possible independently of the current value of l_1. Therefore Algorithm 2 replaces C_1 by the abstraction $\exists l_1 C_1$, which amounts to changing the update of the $!\mathsf{fc}_1$-transition to $c_1' = 0$. No chaos EFSMs are needed as c_1 does not appear in any other plant, so that $\mathcal{G}^1 = \{F_1, F_2, \exists l_1 C_1\}$, $V^1 = \{c_1\}$, and $\mathcal{C}^1 = \emptyset$. Synthesis results in $S^1 = \sup\mathcal{C}(F_1 \parallel F_2 \parallel \exists l_1 C_1, E_{C1}, \{!\mathsf{ff}_1, !\mathsf{ff}_2, !\mathsf{fc}_1\})$, which is controllable as no other uncontrollable events are involved.
This supervisor, shown as S_{C1} in Fig. 7, is returned and collected by Algorithm 3. Since the value of c_1 is initially 0, and c_1 only changes its value on events sf_1, sf_2, and sc_1, the supervisor S_{C1} ensures that the feeders only start when conveyor C_1 is empty, and this conveyor is only started after delivery of a workpiece from a feeder.

– Specification E_{C2} includes the variable l_1, but it is only used with the controllable event sc_2. Then Algorithm 3 passes $\exists l_1 E_{C2}$ to Algorithm 2, which amounts to deletion of the update from E_{C2}. In Algorithm 2, the specification is not controllable by itself as it disables the uncontrollable event $!\mathsf{fl}_1$. So the algorithm sets $\Sigma_u^1 = \{!\mathsf{fl}_1\}$ and finds that $!\mathsf{fl}_1$ is only disabled by plant L_1, which has no variables. Then $\mathcal{G}^1 = \{L_1\}$, $V^1 = \emptyset$, $\mathcal{C}^1 = \emptyset$, and synthesis gives $S^1 = \sup\mathcal{C}(L_1, \exists l_1 E_{C2}, \{!\mathsf{fl}_1\})$, which is also found to be $!\mathsf{wl}_1$-controllable
This supervisor, shown as S_{C2} in Fig. 7, is returned and collected by Algorithm 3. Together with the original specification E_{C2}, which also is collected by Algorithm 3, it ensures that conveyor C_2 only removes type 2 workpieces that should not be processed by production line L_1 or type 1 workpieces processed by L_1.
– Specification E_{L1a} includes variable l_1, which is used with the uncontrollable event $!\mathsf{fc}_1$ and therefore cannot be abstracted. Thus Algorithm 3 passes E_{L1a} unchanged to Algorithm 2. At the beginning of Algorithm 2, chaos EFSMs are constructed for all events that can change the variable l_1 in some plant, resulting in $\mathcal{C}^0 = \{\text{chaos}(!\mathsf{fc}_1, l_1), \text{chaos}(!\mathsf{fm}_1, l_1), \text{chaos}(\mathsf{sc}_2, l_1)\}$. Then E_{L1} is not controllable w.r.t. $\|(\mathcal{C}^0)$ because l_1 can uncontrollably change from its initial value 0 by $\text{chaos}(!\mathsf{fc}_1, l_1)$, and afterwards $!\mathsf{fc}_1$ is possible with the specification's guard $l_1 = 0$ being false.
The only uncontrollable event in the specification and cause of uncontrollability is $!\mathsf{fc}_1$, so $\Sigma_u^1 = \{!\mathsf{fc}_1\}$. This uncontrollable event is disabled by plant C_1, which therefore is included in the plant abstraction. C_1 also includes the variable c_1, which is *not* unconstrained, so that $\mathcal{G}^1 = \{C_1\}$ and $V^1 = \{c_1, l_1\}$. Apart from the selected plant C_1, the variable l_1 is still assigned on event $!\mathsf{fm}_1$ in M_1 and on event sc_2 in C_2, and the variable c_1 is assigned on event $!\mathsf{ff}_1$ in F_1 and on event $!\mathsf{ff}_2$ in F_2. Therefore the chaos EFSMs $\mathcal{C}^1 = \{\text{chaos}(!\mathsf{ff}_1, c_1), \text{chaos}(!\mathsf{ff}_2, c_1),$

$\mathrm{chaos}(!\mathsf{fm}_1, l_1), \mathrm{chaos}(\mathsf{sc}_2, l_1)\}$ are included. Synthesis gives a supervisor

$$S^1 = \sup\mathcal{C}(C_1 \parallel \mathrm{chaos}(!\mathsf{ff}_1, c_1) \parallel \mathrm{chaos}(!\mathsf{ff}_2, c_1) \parallel \mathrm{chaos}(!\mathsf{fm}_1, l_1) \parallel \mathrm{chaos}(\mathsf{sc}_2, l_1), E_{L1a}, \{!\mathsf{fc}_1\}) \ , \tag{41}$$

but it is not $!\mathsf{fm}_1$-controllable. This is because S^1 is synthesised under the pretence that $!\mathsf{fm}_1$ is controllable, and then S^1 can constrain $!\mathsf{fm}_1$ to prevent the system from entering states with $l_1 \neq 0$ where $!\mathsf{fc}_1$ is enabled. For example, S^1 allows sc_1 when initially $l_1 = 0$ and then disables $!\mathsf{fm}_1$ to prevent it from changing l_1 to a non-zero value before $!\mathsf{fc}_1$ occurs. But this is not acceptable since $!\mathsf{fm}_1$ really is uncontrollable.
Therefore Algorithm 2 enters another iteration with $\Sigma_{\mathrm{u}}^2 = \{!\mathsf{fc}_1, !\mathsf{fm}_1\}$. Now plant M_1 must also be included as it disables $!\mathsf{fm}_1$. There are no additional variables in M_1 so that $\mathcal{G}^2 = \{C_1, M_1\}$ and $V^2 = \{c_1, l_1\}$. Outside of the selected plants C_1 and M_1, the variable l_1 is only assigned on sc_2 in C_2, so that $\mathcal{C}^2 = \{\mathrm{chaos}(!\mathsf{ff}_1, c_1), \mathrm{chaos}(!\mathsf{ff}_2, c_1), \mathrm{chaos}(\mathsf{sc}_2, l_1)\}$. Then another supervisor is synthesised,

$$S^2 = \sup\mathcal{C}(C_1 \parallel M_1 \parallel \mathrm{chaos}(!\mathsf{ff}_1, c_1) \parallel \mathrm{chaos}(!\mathsf{ff}_2, c_1) \parallel \mathrm{chaos}(\mathsf{sc}_2, l_1), E_{L1a}, \{!\mathsf{fc}_1, !\mathsf{fm}_1\}) \ , \tag{42}$$

and this supervisor is found to be controllable w.r.t. the remaining uncontrollable events $!\mathsf{ff}_1$ and $!\mathsf{ff}_2$.
The supervisor, shown as S_{L1a} in Fig. 7, is returned and collected by Algorithm 3. It avoids overflow of production line L_1, because it only allows the conveyor C_1 to start delivery of a new workpiece (sc_1) when the production line is empty, $l_1 = 0$.

– The remaining specifications E_{L2b}, E_{M2a}, E_{M2b}, E_{C3}, and E_{L2a} are processed in a similar way as those above, resulting in further supervisors S_{L2b}, S_{M2a}, S_{M2b}, S_{C3}, and S_{L2a}.

On completion, Algorithm 3 returns the supervisors shown in Fig. 7 plus the original specifications in Fig. 6, which together control the flexible manufacturing system in the least restrictive controllable way. The largest supervisor EFSMs have seven locations, and the largest state spaces encountered are 100 unfolded states during synthesis of S_{L1a} and S_{L2a}. In comparison, a full monolithic synthesis for all the plants and specifications together, explores a state space of 14580 unfolded states and results in a single supervisor EFSM with 464 locations and 1551 unfolded states.

6 Conclusions

This paper presents an algorithm that combines modular and abstraction-based synthesis for extended finite-state machines (EFSM). The approach allows to calculate supervisors that control a system in the least restrictive controllable way. Through a combination of component selection and symbolic manipulation by means of existential quantification, the method avoids the exploration of the full state space as normally required in synthesis. The resulting supervisors are modular and can be presented as the composition of several small EFSMs, which

also facilitates human readability. These results improve the authors' previous work [12, 28] in that they allow abstraction by both component and variable selection.

In future work, the authors would like to implement the proposed synthesis algorithm and provide tooling support for the synthesis of modular supervisors with EFSMs. The best option is likely to be a symbolic implementation, e.g., using BDDs [3], which reduces the overheads of finding and computing the existential abstraction. Symbolic guard extraction techniques [14] can help to present the computed supervisors as EFSMs.

Another important aspect of future research is the synthesis of supervisors that are *nonblocking* in addition to being controllable and least restrictive. The problem with the nonblocking property is that it does not admit the modularity results this paper is based upon. If a system is blocking (or nonblocking) then it may well be nonblocking (or blocking) after composition with some other component. Therefore, the ideas of component selection and existential quantification cannot be applied directly when the nonblocking property is considered.

Although the results of this work will help to compute supervisors that are both controllable and nonblocking, additional concepts will be needed to handle the nonblocking property. Compositional nonblocking *verification* of EFSMs can be approached with conflict equivalence instead of behavioural inclusion [16]. Using this for synthesis requires different abstractions again, which so far has only been done for ordinary FSMs without variables [15]. Another idea could be to generalise concepts such as observer projection [29] and local control consistency [21] to EFSMs.

References

1. Åkesson, K., Flordal, H., Fabian, M.: Exploiting modularity for synthesis and verification of supervisors. In: 15th IFAC World Congress on Automatic Control (2002). DOI 10.3182/20020721-6-ES-1901.00517
2. Brandin, B.A., Malik, R., Malik, P.: Incremental verification and synthesis of discrete-event systems guided by counter-examples. IEEE Trans. Control Syst. Technol. **12**(3), 387–401 (2004). DOI 10.1109/TCST.2004.824795
3. Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. IEEE Trans. Comput. **35**(8), 677–691 (1986). DOI 10.1109/TC.1986.1676819
4. Cassandras, C.G., Lafortune, S.: Introduction to Discrete Event Systems, 2 edn. Springer Science & Business Media, New York, NY, USA (2008)
5. Chen, Y., Lin, F.: Modeling of discrete event systems using finite state machines with parameters. In: 2000 IEEE Int. Conf. Control Applications (CCA), pp. 941–946 (2000). DOI 10.1109/CCA.2000.897591
6. Cury, J.E.R., de Queiroz, M.H., Bouzon, G., Teixeira, M.: Supervisory control of discrete event systems with distinguishers. Automatica **56**, 93–104 (2015). DOI 10.1016/j.automatica.2015.03.025
7. Hoare, C.A.R.: Communicating Sequential Processes. Prentice-Hall (1985)
8. Huth, M., Ryan, M.: Logic in Computer Science. Cambridge University Press, Cambridge, UK (2004)
9. Komenda, J., van Schuppen, J.H.: Control of discrete-event systems with modular or distributed structure. Theoretical Comput. Sci. **388**, 199–226 (2007). DOI 10.1016/j.tcs.2007.07.049
10. Le Gall, T., Jeannet, B., Marchand, H.: Supervisory control of infinite symbolic systems using abstract interpretation. In: 46th IEEE Conf. Decision and Control, CDC '05, pp. 30–35 (2005). DOI 10.1109/CDC.2005.1582126

11. Malik, R., Teixeira, M.: 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). URL `http://hdl.handle.net/10289/9841`
12. Malik, R., Teixeira, M.: Modular supervisor synthesis for extended finite-state machines subject to controllability. In: 13th Int. Workshop on Discrete Event Systems, WODES '16, pp. 117–122. IEEE (2016). DOI 10.1109/WODES.2016.7497831
13. Malik, R., Teixeira, M.: Framework and proofs for synthesis of least restrictive controllable supervisors for extended finite-state machines witb variable abstraction. Working Paper 03/2018, Dept. of Computer Science, University of Waikato, Hamilton, New Zealand (2018). URL `http://hdl.handle.net/10289/12140`
14. Miremadi, S., Åkesson, K., Lennartson, B.: Symbolic computation of reduced guards in supervisory control. IEEE Trans. Autom. Sci. Eng. **8**(4), 754–764 (2011). DOI 10.1109/TASE.2011.2146249
15. Mohajerani, S., Malik, R., Fabian, M.: A framework for compositional synthesis of modular nonblocking supervisors. IEEE Trans. Autom. Control **59**(1), 150–162 (2014). DOI 10.1109/TAC.2013.2283109
16. Mohajerani, S., Malik, R., Fabian, M.: A framework for compositional nonblocking verification of extended finite-state machines. Discrete Event Dyn. Syst. **26**(1), 33–84 (2016). DOI 10.1007/s10626-015-0217-y
17. Ouedraogo, L., Kumar, R., Malik, R., Åkesson, K.: Nonblocking and safe control of discrete-event systems modeled as extended finite automata. IEEE Trans. Autom. Sci. Eng. **8**(3), 560–569 (2011). DOI 10.1109/TASE.2011.2124457
18. de Queiroz, M.H., Cury, J.E.R.: Modular supervisory control of large scale discrete event systems. In: R. Boel, G. Stremersch (eds.) Discrete Event Systems: Analysis and Control, SECS 569, pp. 103–118. Kluwer (2000). DOI 10.1007/978-1-4615-4493-7_10
19. Ramadge, P.J.G., Wonham, W.M.: The control of discrete event systems. Proc. IEEE **77**(1), 81–98 (1989). DOI 10.1109/5.21072
20. Rosa, M., Teixeira, M., Malik, R.: Exploiting approximations in supervisory control with distinguishers. IFAC PapersOnLine **51**(7), 13–18 (2018). DOI 10.1016/j.ifacol.2018.06.272
21. Schmidt, K., Breindl, C.: Maximally permissive hierarchical control of decentralized discrete event systems. IEEE Trans. Autom. Control **56**(4), 723–737 (2011). DOI 10.1109/TAC.2010.2067250
22. Shoaei, M.R., Feng, L., Lennartson, B.: Abstractions for nonblocking supervisory control of extended finite automata. In: 8th Int. Conf. Automation Science and Engineering, CASE 2012, pp. 364–370 (2012). DOI 10.1109/CoASE.2012.6386446
23. Shoaei, M.R., Feng, L., Lennartson, B.: On the computation of natural observers for extended finite automata. IFAC Proceedings **47**(3), 2448–2455 (2014). DOI 10.3182/20140824-6-ZA-1003.02178
24. Silva, A.L., Ribeiro, R., Teixeira, M.: Modeling and control of flexible context-dependent manufacturing systems. Inform. Sciences **421**, 1–14 (2017). DOI 10.1016/j.ins.2017.08.084
25. Sköldstam, M., Åkesson, K., Fabian, M.: Modeling of discrete event systems using finite automata with variables. In: 46th IEEE Conf. Decision and Control, CDC '07, pp. 3387–3392 (2007). DOI 10.1109/CDC.2007.4434894
26. Tarski, A.: A lattice-theoretical fixpoint theorem and its applications. Pacific J. Math. **5**(2), 285–309 (1955)
27. Teixeira, M., Cury, J.E.R., de Queiroz, M.H.: Exploiting distinguishers in local modular control of discrete-event systems. IEEE Trans. Autom. Sci. Eng. **15**(3), 1431–1437 (2018). DOI 10.1109/TASE.2018.2793963
28. Teixeira, M., Malik, R., Cury, J.E.R., de Queiroz, M.H.: Supervisory control of DES with extended finite-state machines and variable abstraction. IEEE Trans. Autom. Control **60**(1), 118–129 (2015). DOI 10.1109/TAC.2014.2337411
29. Wong, K.C., Wonham, W.M.: Modular control and coordination of discrete-event systems. Discrete Event Dyn. Syst. **8**(3), 247–297 (1998). DOI 10.1023/A:1008210519960