CHALMERS

Supervisory Control Applied to Automata Extended with Variables - Revised

M. SKÖLDSTAM, K. ÅKESSON, M. FABIAN

Automation
Department of Signals and Systems
Chalmers University of Technology
Göteborg, Sweden, 2008 R001/2008

Supervisory Control Applied to Automata Extended with Variables - Revised

Markus Sköldstam, Knut Åkesson, Martin Fabian

Abstract— **To get industrial acceptance of supervisory control theory, there is a need to bridge the gap between the signal-based industrial reality and the event-based supervisory control framework. This report tries to shorten this gap by introducing a modeling formalism with automata extended with variables, guard expressions and action functions. The formalism is suitable for modeling plants and specifications in the supervisory control framework. No restrictions are made on the sharing of variables between concurrent automata and don't care updating of shared variables is allowed. This leads to frame problems since unreachable states of subsystems can become reachable in the entire system. To define supervisory control problems in this general setting we introduce the concept of controllable languages with respect to the entire system which is a generalization of the classical definition of controllability. An algorithm that transforms supervisory control problems modeled by automata with shared variables into equivalent ordinary automata supervisory control problems, is presented. This allows the user to model complex behaviors with a compact representation, and at the same time use existing algorithms for synthesis and verification. The proposed approach has been implemented in the supervisory control tool, Supremica.**

Index Terms— **Automata, Modeling, Supervisory control, System analysis and design, State space methods, Software verification and validation**

I. Introduction

Discrete event systems (DES) are models of systems that at each time instant occupy a discrete state, and perform state-changes on the occurrence of events. Examples of such systems are manufacturing systems, communication networks and embedded systems. The behavior of a DES is described by the sequences of events that may occur, and the sequences of states that may be visited.

Supervisory control theory [1] is a general approach to synthesize control systems for DES. A supervisor may be generated using models of the plant and the specification, such that it is minimally restrictive with respect to the plant behavior, while still guaranteeing that the specification is upheld. Traditionally, regular languages and finite automata [2] have been used both for modeling and analysis of discrete event systems in the supervisory control community.

Though a large amount of promising research results have been achieved in academia, industrial acceptance of the supervisory control theory is scarce. Only a few examples have been reported [3]. A number of issues that hinder industrial use have been identified by various researchers [3], [4], [5]. Two main issues are the discrepancy between the signal-based reality and the event-based automata framework, and the lack of a compact representation of large models.

M. Sköldstam, K. Åkesson and M. Fabian are with the Department of Signals and Systems, Chalmers University of Technology, SE-412 96 Gothenburg, Sweden

In many industrial applications, parts of a system, such as sensors, actuators and buffers, are conveniently modeled using variables. Guard expressions are used to restrict the behavior of the system and action functions are used to update variables. Physical signals that are stored in memories or sent between controllers are naturally modeled as global variables in DES models. Using variables, guards and actions help us to compactly represent large and complicated DES.

A number of frameworks have been introduced that allow compact representations of discrete event systems with complex behavior and large state-spaces. Many of these are inspired by Statecharts [6], which extends automata with hierarchy, concurrency, and communication using variables, guards and actions. While most of the concepts introduced in Statecharts are useful for modeling supervisory control problems, Statecharts is not in its original formulation suitable for supervisory control. In the supervisory control framework it is essential to model what *may* occur instead of what *should* occur, this has large consequences for how the interaction between subsystems are modeled. In Statecharts there is a causality between subsystems, this is not desired in the supervisory control framework.

Modeling frameworks based on automata extended with variables, suitable for supervisory control, are presented in [7], [8], [9], [10]. In [7] it is assumed that a variable can be updated by at the most one extended automaton and in order to do synthesis the state-space needs to be extended by additional states. In [8] automata with variables are used to implement a supervisor. The authors encode the states of a given supervisor using boolean variables. The variables are used in guards and actions attached to the events of the model. In [9] supervisory control is applied to a number of automata with variables. To ensure a least restrictive supervisor it is assumed that all variables are local i.e. not shared between automata. This is a quite strong restriction because variables cannot be used to model any interaction between subsystems. In [10] a state transition structure with a data collection used for parameterized and non-regular discrete event systems is introduced.

Though extended frameworks allow compact representations of huge state-spaces, and hence simplify the modeling of systems of industrially interesting sizes, the states do not disappear and hence potentially pose a problem when it comes to analysis. The main problem is the state-space explosion that typically occurs when the behavior of interacting sub-systems is studied. For ordinary automata, especially in the context of supervisory control, there exists a large body of work for fighting this state-space explosion. For extended frameworks, less has been done. An attractive approach is to develop algorithms that benefit from the structure given by extended modeling frameworks, see [11] and its references.

Without doubts, developing effective synthesis algorithms for automata that share variables needs to be explored further. However, it is equally important to be able to use existing algorithms that have been tested and have been proven to handle systems with large state-spaces. In this report we present no new algorithms for analysis of extended automata models. Our approach is to transform extended models into ordinary automata models with the same behavior. This way we can use extended automata for modeling and regular automata for analysis. In order to fully understand the relation between regular finite automata and extended finite automata, we feel that a new and very detailed definition of extended automata is needed. Since we only model physical systems with finite state-spaces and the current variable values are part of the system state, we only consider variables that have *finite* domains of definition.

We present a modeling formalism with automata extended with variables, guard expressions and action functions. We attach guards and actions functions to transitions since this admits local design techniques of systems consisting of many different parts. In comparison to previous work we do not put any restrictions on how variables are shared between extended automata, thus all extended automata are allowed to update all variables as long as the composition is well defined. The presented framework has been implemented in the supervisory control tool Supremica [12], [13]. In [14] it is argued that the extended finite automata (EFA) introduced in this report may be used as a basis for efficiently representing control problems that consists of mixed logic and supervisory control problems.

We start with a motivating example of an EFA model of a system with a complex specification (section II). Section III provides the formal notations and definitions used for regular languages and finite automata (FA). Section IV introduces extended finite automata (EFA), deterministic EFA, full synchronous composition between EFA and action consistent EFA. In section V we present and prove a basic algorithm that transforms a single EFA to an isomorphic FA. It is explained how the algorithm is extended to handle an arbitrary number of interacting EFA and how it is to be used to effectively perform analysis. Controllability of arbitrary EFA specifications that share variables with a plant EFA is defined in Section VI. We also define the controllable language of an EFA model and show how the standard definition of controllability can be deduced if all EFA in the model have distinct variables. The approach that is suggested in this report is to use existing algorithms developed for ordinary automata for synthesis and verification of EFA models. Section VII presents a transformation algorithm that makes this approach feasible. The algorithm transforms supervisory control problems modeled by EFA into equivalent FA control problems. The procedure is illustrated with an example that contains a number of difficulties that needs to be solved. Step by step the complete transformation algorithm is developed. It is proven that the generated FA model can be used both for verification and synthesis of the original EFA model.

II. Modeling a Dosing Tank

This section illustrates some of the advantages of using EFA as a modeling tool compared with FA. We have chosen to model a unit in a chemical batch plant. The system consists of a tank and a user. The tank has an inlet valve, an outlet valve and two sensors to check the filling of the tank, S1 at the bottom and S2 at the top of the tank. Filling the tank or emptying the tank, can be requested to start or stop by the user. To meet the user requests, a supervisor/controller is designed that closes and opens the valves appropriately.

The plant and the supervisor are modeled in figures 1 and 2, respectively. State changes are modeled by the regular events **s1_on, s1_off, s2_on, s2_off, req_stop, req_start, close_in, open_in, close_out** and **open_out**. Sensor signals are modeled by the variables v_{s1} and v_{s2}, request signals from the user by the variable v_{req} and control signals to the valves are modeled by the variables v_{in} and v_{out}. All variables have domain $\{0, 1\}$ and zero as initial value.

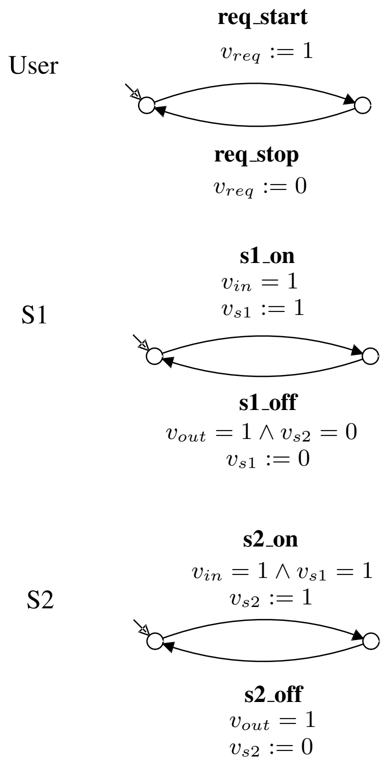

Fig. 1. A plant model of the dosing tank.

The plant model consists of the extended automata User, S1 and S2. A sensor signal can go high if the inlet valve is open ($v_{in} = 1$) and sensor signal can go low if the outlet valve is open ($v_{out} = 1$). The supervisor's task is to enable or disable the opening/closing of the valves such that the following conditions are satisfied:

a. the inlet and outlet valves are never open at the same time;

b. discharging or filling can only start when a request is present, $v_{req} = 1$;

c. discharging can only start when the tank is completely filled, $v_{s2} = 1$;

d. filling can only start when the tank is empty, $v_{s1} = 0$;

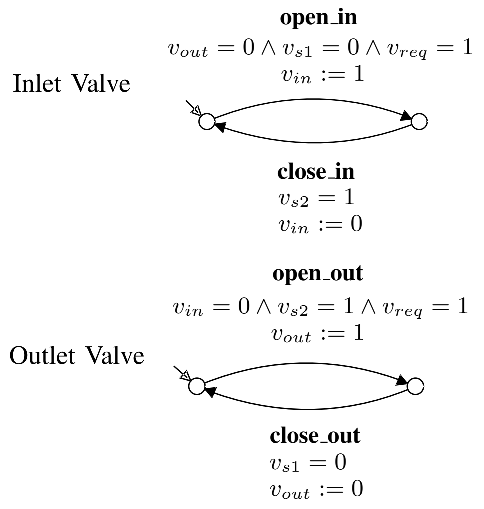

Fig. 2. A supervisor for the dosing tank.

e. for the inlet valve to close the tank must be full, $v_{s2} = 1$;

f. for the outlet valve to close the tank must be empty, $v_{s1} = 0$.

In figure 2 all these requirements are expressed using guard formulas over the variables. The presented EFA model of the dosing tank may be compared to the FA model in [15] where the same process is modeled without variables. It is clear that the use of EFA facilitates the modeling of many systems and it is reasonable to expect that the benefits of using EFA for modeling increase when the complexity of the modeled system increases. The model above does not illustrate all features of EFA or fully explain how EFA can be used to model supervisory control problems. Since we have not distinguished between controllable and uncontrollable events it is quite easy to generate an equivalent ordinary automata model. The equivalent ordinary automata model of the EFA model above, obtained from a basic transformation algorithm, is presented in figure 3.

III. Preliminaries

A. Events and Languages

The event set of a DES is called an *alphabet* and is denoted by Σ. For the purpose of supervisory control, the alphabet is divided into the disjoint set of the *controllable* events and the set Σ_u of *uncontrollable* events. To denote that an event is uncontrollable it is prefixed by an exclamation mark (!). Σ^* denotes the set of all finite *strings* of the form $\sigma_1\sigma_2\ldots\sigma_k$ of events from Σ, including the *empty string* ϵ. A subset of $\mathcal{L} \subseteq \Sigma^*$ is called a *language*. Languages are used to describe the behavior of DES. Besides the usual set operations, the key operation involved when building languages is *concatenation*. The concatenation of two strings $s, t \in \mathcal{L}$ is written st. Languages and alphabets can also be concatenated, $\mathcal{L}\Sigma = \{s\sigma \mid s \in \mathcal{L}, \sigma \in \Sigma\}$. The *prefix closure* of a language $\mathcal{L} \subseteq \Sigma^*$, denoted $\overline{\mathcal{L}}$, is $\overline{\mathcal{L}} = \{s \in \Sigma^* \mid \exists t \in \Sigma^*, st \in \mathcal{L}\}$.

B. Automata

Typically, DES are modeled using deterministic finite-state automata. In what follows the word ”ordinary” will be used synonymously with ”deterministic finite-state”.

Definition 1 (Automaton):
A deterministic finite-state automaton A is a 4-tuple

$$A = \langle Q, \Sigma, \rightarrow, q_0 \rangle,$$

where Q is a finite set of states; Σ (the alphabet) is a nonempty finite set of events; $\rightarrow \subseteq Q \times \Sigma \times Q$ is the state transition function mapping elements of $Q \times \Sigma$ into singletons of Q and $q_0 \in Q$ is the initial state.
The transition function is written in infix notation $p \xrightarrow{\sigma} q$. In particular, $p \xrightarrow{\sigma}$ denotes that there exists a state q such that $p \xrightarrow{\sigma} q$, and $p \overset{\sigma}{\nrightarrow}$ denotes that it does not exist such a state. This notation is extended to strings in Σ^* in the natural way by letting

$$\begin{aligned} &p \xrightarrow{\epsilon} p \text{ for all } p \in Q;\\ &p \xrightarrow{s\sigma} q \text{ if } p \xrightarrow{s} r \text{ and } r \xrightarrow{\sigma} q \text{ for some } r \in Q. \end{aligned}$$

For convenience, the notation $A \xrightarrow{s} q$ is introduced as a short hand for $q_0 \xrightarrow{s} q$, where q_0 is the initial state of A.

The behavior of an ordinary automaton is described by its language. Let $A = \langle Q, \Sigma, \rightarrow, q_0 \rangle$ be an ordinary automaton. The language of A denoted $L(A)$ is defined as $L(A) = \{s \in \Sigma^* \mid q_0 \xrightarrow{s}\}$.

To deal with interacting automata we use full synchronous composition (FSC) [16]. This composition operator models that an event can occur in the synchronized system if and only if it can occur in all automata that share the event.

Definition 2 (FSC, Automata):
Let $A_j = \langle Q_j, \Sigma_j, \rightarrow_j, q_0^j \rangle$, $j = 1, 2$ be two automata. The full synchronous composition (FSC) of A_1 and A_2 is

$$A_1 || A_2 = \langle Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \rightarrow, (q_0^1, q_0^2) \rangle,$$

where
$(p_1, p_2) \xrightarrow{\sigma} (q_1, q_2)$, $\sigma \in \Sigma_1 \cap \Sigma_2$ if
$p_i \xrightarrow{\sigma}_i q_i$, $i = 1, 2$;
$(p_1, p_2) \xrightarrow{\sigma} (q_1, q_2)$, $\sigma \in \Sigma_1 \setminus \Sigma_2$ if
$p_1 \xrightarrow{\sigma}_1 q_1$ and $p_2 = q_2$;
$(p_1, p_2) \xrightarrow{\sigma} (q_1, q_2)$, $\sigma \in \Sigma_2 \setminus \Sigma_1$ if
$p_2 \xrightarrow{\sigma}_2 q_2$ and $p_1 = q_1$.

The composition operator is easily extended to simultaneous composition of multiple automata.

IV. Extended Automata

An Extended Finite Automaton (EFA) is an augmentation of the ordinary automaton with guard formulas and action functions. We associate the guards and actions to the transitions in the automaton. The transitions in the EFA are enabled if and only if the guard formula is true and when a transition is taken, updating actions of a set of variables may follow. To define guard predicates we use the characteristic function χ_W of a set W. χ_W is defined by

$$\chi_W(v) = \begin{cases} 1 & \text{if } v \in W \\ 0 & \text{if } v \notin W \end{cases},$$

and is sometimes called the indicator function of W. If $\chi_W(v) = 1$ the predicate "$v \in W$" is true, and if $\chi_W(v) = 0$ the predicate is false.

Definition 3 (Extended Automaton):
An extended finite-state automaton E is a 6-tuple

$$E = \langle Q \times V, \Sigma, \mathcal{G}, \mathcal{A}, \rightarrow, (q_0, v_0)\rangle,$$

where:

(i) $Q \times V$ is the extended finite set of states, where Q is a finite set of *locations* and V is the finite domain of definition of the variables;
(ii) Σ is a nonempty finite set of events (the alphabet);
(iii) $\mathcal{G} = \{\chi_W \mid W \in 2^V\}$ is the set of guard predicates over V.
(iv) $\mathcal{A} = \{a \mid a$ is a function from V to $V\}$ is a collection of action functions.
(v) $\rightarrow \subseteq Q \times \Sigma \times \mathcal{G} \times \mathcal{A} \times Q$ is the state transition relation.
(vi) $(q_0, v_0) \in Q \times V$ is the initial state.

We have extended the states of the ordinary automaton to $Q \times V$, where $V = V^1 \times ... \times V^n$. The finite set V is the domain of definition of an n-tuple of variables $v = (v^1, \dots, v^n)$ with initial values $v_0 = (v_0^1, \dots, v_0^n) \in V$.

Formally speaking, the inclusion of $\mathcal{G}$ and $\mathcal{A}$ in the definition of E are superfluous since they only depend on V. In what follows, we therefore omit explicitly writing $\mathcal{G}$ and $\mathcal{A}$ when specifying an EFA. The *guards* are predicates over the variables that relate each element of V, to either 1 (true) or 0 (false). We are interested in deterministic EFA, and therefore the *actions* are functions. Action functions $a \in \mathcal{A}$ maps the variable values of the present state to the variable values of the next state. Guards and actions are written as

$$\begin{aligned} w &= g(v), \text{ where } w \in \{0,1\}; \\ w &:= a(v) = (a^1(v), \dots, a^n(v)), \\ &\quad \text{where } w \in V. \end{aligned}$$

For convenience we use the symbol Ξ to denote implicit actions that update variables to their current value. Unlike explicit actions Ξ can be overridden when EFA are synchronized, see [17]. If $a^i = \Xi$, we say that a^i is a *don't care updating of the variable* v^i.

The transition relation is written as $p \xrightarrow{\sigma}_{g/a} q$, where $p, q \in Q$, $\sigma \in \Sigma$, $g \in \mathcal{G}$ and $a \in \mathcal{A}$. If g is absent, it is assumed that g always evaluates to true and the transition takes place when σ occurs. If a is absent, it is assumed that $a = (\Xi, \Xi, \dots, \Xi)$ and no variable is updated during the transition [1]. Note that, the state transition relation is well defined when the guard always evaluates to false and no transition can take place. One reason for using actions and guards is that they can be used to hide variable values in system transitions. It is sometimes convenient to write out the states (locations and variable values) explicitly in system transitions.

Definition 4 (Explicit State Transition Relation): Let $E = \langle Q \times V, \Sigma, \rightarrow, (q_0, v_0)\rangle$ be an EFA. The explicit state transition relation of E is defined as

$$\begin{aligned} \mapsto \;:=\; & \{(p, v, \sigma, q, w) \in Q \times V \times \Sigma \times Q \times V \mid \\ & \exists p \xrightarrow{\sigma}_{g/a} q \text{ such that} \\ & g(v) = 1 \text{ and } a(v) =: w \text{ or} \\ & g(v) = 1, v = w \text{ and } a = \Xi\}. \end{aligned}$$

The explicit state transition relation is written $(p, v) \overset{\sigma}{\mapsto} (q, v')$ and it is extended to strings in Σ^* in the usual recursive way

$$\begin{aligned} & (p, v) \overset{\epsilon}{\mapsto} (p, v) \text{ for all } (p, v) \in Q \times V, \\ & (p, v) \overset{s\sigma}{\mapsto} (q, v') \text{ if } (p, v) \overset{s}{\mapsto} (r, y) \text{ and} \\ & (r, y) \overset{\sigma}{\mapsto} (q, v') \text{ for some } (r, y) \in Q \times V. \end{aligned}$$

The language $L(E)$ of an EFA E is $L(E) = \{s \in \Sigma^* \mid (q_0, v_0) \overset{s}{\mapsto}\}$. In section VII, we will need the notion of deterministic EFA.

Definition 5 (Deterministic EFA):
An EFA $E = \langle Q \times V, \Sigma, \rightarrow, (q_0, v_0)\rangle$ is deterministic if $(p, v) \overset{\sigma}{\mapsto} (q, v')$ and $(p, v) \overset{\sigma}{\mapsto} (q', v'')$ always implies $(q, v') = (q', v'')$.
Note that for an EFA to be deterministic all explicit transitions (not just the reachable) must have this property. This seemingly strong condition is needed to ensure that the synchronized product of two deterministic EFA also is deterministic. Observe that if the guards are distinct, we support the possibility of having multiple transitions from the same location triggered by the same event. Verifying if a given EFA fulfills Definition 5 might be non-trivial for certain EFA with large state-spaces. However, sufficient conditions that guarantee deterministic EFA are straightforward to formulate.

A. Full Synchronous Composition, EFA

To simplify the notation when defining the full synchronous product of EFA, we assume that the EFA share all variables. This is no restriction since it is always possible to add don't care variables that are never updated. For the synchronous product to exist, a necessary and sufficient condition is that shared variables must have the same initial values. As for ordinary automata, the composition operator models that an event can occur in the synchronized system if and only if it can occur in all EFA that share the event.

Definition 6 (FSC, EFA):
Let $E_k = \langle Q_k \times V, \Sigma_k, \rightarrow_k, (q_0^k, v_0)\rangle$, $k = 1, 2$, be two EFA using the shared variables $v = (v^1, \dots, v^n)$. The Full Synchronous Composition (FSC) of E_1 and E_2 is

$$\begin{aligned} E_1 \| E_2 \;=\; & \langle Q_1 \times Q_2 \times V, \Sigma_1 \cup \Sigma_2, \\ & \rightarrow, (q_0^1, q_0^2, v_0)\rangle, \end{aligned}$$

where the state transition relation $\rightarrow$ is defined as

* $(p_1, p_2) \xrightarrow{\sigma}_{g/a} (q_1, q_2)$, $\sigma \in \Sigma_1 \cap \Sigma_2$ if $\exists(p_1, \sigma, g_1, a_1, q_1) \in \rightarrow_1, \exists(p_2, \sigma, g_2, a_2, q_2) \in \rightarrow_2$ such that:
  (i) $g = g_1 \wedge g_2$,

[1]We consider event driven transitions. Dynamic transitions that take place when the guard becomes true are not considered here.

(ii) For $i = 1, \dots, n$ and $\forall v \in V$:

$$a^i(v) = \begin{cases} a_1^i(v) & \text{if } a_1^i(v) = a_2^i(v) \\ a_1^i(v) & \text{if } a_2^i(v) = \Xi \\ a_2^i(v) & \text{if } a_1^i(v) = \Xi \\ v^i & \text{otherwise} \end{cases} ;$$

- $*$ $(p_1, p_2) \xrightarrow{\sigma}_{g/a} (q_1, q_2)$, $\sigma \in \Sigma_1 \setminus \Sigma_2$ if $(p_1, \sigma, g, a, q_1) \in \rightarrow_1$ and $p_2 = q_2$;
- $*$ $(p_1, p_2) \xrightarrow{\sigma}_{g/a} (q_1, q_2)$, $\sigma \in \Sigma_2 \setminus \Sigma_1$ if $(p_2, \sigma, g, a, q_2) \in \rightarrow_2$ and $p_1 = q_1$.

Note that if the action functions of E_1 and E_2 explicitly tries to update a shared variable to different values, the variable is, by default, not updated. This implies that the synchronized EFA may not have the intended behavior. A sufficient condition that can be used to avoid this possibility is:

Definition 7 (Action Consistent EFA): Let $E_k = \langle Q_k \times V, \Sigma_k, \rightarrow_k, (q_0^k, v_0)\rangle$, $k = 1, 2$, be two extended automata with shared variables $v = (v^1, \dots, v^n)$. E_1 and E_2 are *action consistent* if $\forall(p_1, \sigma, g_1, a_1, q_1) \in \rightarrow_1$ and $\forall(p_2, \sigma, g_2, a_2, q_2) \in \rightarrow_2$ it is true that:

- $*$ $\forall v \in V$ such that $g_1(v) \wedge g_2(v)$ is true then $a_1^i(v) = a_2^i(v)$ or one of $a_1^i(v)$ and $a_2^i(v)$ is a don't care updating of v^i, $i = 1 \dots n$.

Similar to Definition 5, the action consistency condition is a global requirement and it may be computational expensive to check.

V. Transforming EFA to FA

Much research has been put into developing efficient algorithms and data structures for solving supervisory control problems formulated with ordinary automata. It has therefore been an important goal of our research to show how a set of EFA may be transformed into another set of FA having the same properties.

In this section a basic algorithm for transforming EFA into an isomorphic FA is presented. The transformation is inspired by a translation from UML Statecharts to Finite-State Machines discussed in [18]. The algorithm relies on variables with *finite* domain of definition. It collects the information stored in the guards and actions and builds two kinds of automata, *variable automata* and *location automata*, both with relabeled event sets. The variable automata model the updating of the variables, and the location automata has the same structure as the original extended automata. From a practical point of view it is important to point out that the transformation does not destroy modular structure, this is important because the modular structure may be exploited by efficient verification and synthesis algorithms.

Definition 8: (Isomorphic FA) Let $E = \langle Q \times V, \Sigma_E, \rightarrow_E, (q_0, v_0)\rangle$ be an EFA and $A = \langle R, \Sigma_A, \rightarrow_A, r_0\rangle$ be a FA. Let $\mapsto_E$ be the explicit state transition relation of E. E and A are isomorphic if the following conditions hold.

(i) Σ_E=Σ_A

(ii) There exists a bijective function F from $Q \times V$ to R, such that $F(q_0, v_0) = r_0$ and $(q, v) \overset{s}{\mapsto}_E (q', v') \Leftrightarrow F(q, v) \xrightarrow{s}_A F(q', v')$.

We place no restriction on the naming of the states in automata and therefore we call all ordinary automata that are isomorphic with the extended automaton E, *the isomorphic FA of E*. Note that, the isomorphic FA of a given EFA is obtained by replacing the state transitions relation $\rightarrow$ with the explicit state transition relation $\mapsto$.

A benefit of using EFA as a modeling tool is that the values of the variables in state transitions can be hidden. Usually, the explicit state transition relation $\mapsto$ is not known. Instead the notation $p \xrightarrow{\sigma}_{g/a} q$ is used to describe system transitions in models. Here, we present an algorithm that flattens out EFA models into FA models by extracting the information in the guard and action functions. This is done by building location automata and variable automata, introducing relabeled events, composing the system using the full synchronous composition and in the last step, changing back to the original event names.

Algorithm 1 presents in detail how a single EFA is transformed to its isomorphic FA. It is assumed that the guards have been parsed and written in disjunctive normal form

$$g = g^1 \vee \dots \vee g^j,$$

where each and-clause

$$g^i(v) = g^{i,1}(v^1) \wedge \dots \wedge g^{i,n}(v^n),\ i = 1 \dots j,$$

compares the variables with a constant in $V = V^1 \times \cdots \times V^n$. We also assume that all actions are written as $a(v) = (a^1(v^1), \dots, a^n(v^n))$ where the new value of v^i only depends on its previous value. Any transition $p \xrightarrow{\sigma}_{g/a} q$ can be decomposed into multiple transitions $p \xrightarrow{\sigma}_{g^k/a^k} q$, $k = 1 \dots m$ of this form. It can be achieved by stepping through the domain of definition of all variables and creating multiple assignment functions a^k and new updated guards g^k.

Example: The transition $p \xrightarrow{\sigma}_{x:=y} q$ where $y \in \{0, 1\}$ can be decomposed into: $p \xrightarrow{\sigma}_{y=0/x:=0} q$ and $p \xrightarrow{\sigma}_{y=1/x:=1} q$.

In Algorithm 1 $|g|$ denotes the number of and-clauses of a guard written in disjunctive normal form and if $a^k(v^k) = \Xi$, it is understood that $v^k \xrightarrow{\sigma_i} a^k(v^k)$ means a self loop at v^k.

Algorithm 1 (Basic Transformation Algorithm): Let $E = \langle Q \times V, \Sigma, \rightarrow, (q_0, v_0)\rangle$ be an extended finite automaton where $V = (V^1, \dots, V^n)$ is the domain of definition for the variables $v = (v^1, \dots, v^n)$. The following steps build the isomorphic finite ordinary automaton A and define a renaming function $\Psi(\cdot)$:

1 For each transition $p \xrightarrow{\sigma}_{g/a} q$ in E, introduce $|g|$ new events, all with *unique* names. Create a one to one mapping between each renamed event and an and-clause g^i in the guard g.

2 Collect all relabeled events in the alphabet Σ'.

3 Build a location automaton $A_{loc} = \langle Q, \Sigma', \rightarrow, q_0\rangle$ representing the location changes of E. Each transition $p \xrightarrow{\sigma}_{g/a} q$ is divided into regular transitions in A_{loc} using the relabeled events in Σ'. The number of and-clauses $|g|$ of the guard determine the number of transitions obtained from $p \xrightarrow{\sigma}_{g/a} q$.

4 Build variable automata $A_v^k = \langle V^k, \Sigma', \rightarrow, v_0^k \rangle$ $k = 1 \dots n$, representing the updating of the variables. For each event σ_i in Σ', create transitions $v^k \overset{\sigma_i}{\rightarrow} a^k(v^k)$ in A_v^k if $g^{i,k}(v^k)$ is true.

5 Implement the guard by synchronizing all ordinary automata

$$A_{loc} \| A_v^1 \| \cdots \| A_v^n.$$

6 Let $\Psi(\cdot)$ be the mapping that maps each relabeled event in Σ' to its original event in Σ. The finite automaton A is obtained by applying $\Psi(\cdot)$ to all events in $A_{loc} \| A_v^1 \| \cdots \| A_v^n$.

Proposition 1: Algorithm 1 transforms an extended automaton $E = \langle Q \times V, \Sigma, \rightarrow, (q_0, v_0) \rangle$ into its isomorphic ordinary automaton A.

Proof: It follows immediately that A has the same alphabet as E. The states of A are $Q \times V^1 \times \cdots \times V^n$ so we have a trivial bijective mapping between the states of A and the states of E. It remains to prove that the transition relation of A equals the explicit transition relation $\mapsto$ of E. According to Definition 4, we need to show that the transitions in A obtained from $p \overset{\sigma}{\rightarrow}_{g/a} q$ are all transitions $(p, v) \overset{\sigma}{\mapsto} (q, a(v))$, such that $g(v)$ is true. Let $V_i = V_i^1 \times \ldots \times V_i^n \subseteq V$ be the set where the and-clause g^i evaluates to true. The set of all v such that $g(v)$ is true can then be written as $V_1 \cup \ldots \cup V_j$, where $j = |g|$. For each transition $p \overset{\sigma}{\rightarrow}_{g/a} q$ in E, Algorithm 1 can be described as follows.

For $i = 1, \dots, |g|$:

(i) Create $\alpha \in \Sigma'$ and $p \overset{\alpha}{\rightarrow} q$ in A_{loc}, where α is unique.

(ii) For $k = 1, \dots n$, create $v^k \overset{\alpha}{\rightarrow} a^k(v^k)$ for all $v^k \in V_i^k$ in A_v^k.

(iii) Synchronizing all transitions in the ordinary automata triggered by α gives:
$(p, v) \overset{\alpha}{\rightarrow} (q, a(v))$, $\forall v \in V_i$ in $A_{loc} \| A_v^1 \| \cdots \| A_v^n$.

(iv) Replacing all transitions in the synchronized system triggered by α with transitions triggered by σ gives:
$(p, v) \overset{\sigma}{\rightarrow} (q, a(v))$ $\forall v \in V_i$ in A.

Hence, $p \overset{\sigma}{\rightarrow}_{g/a} q$ in E is mapped to $(p, v) \overset{\sigma}{\mapsto} (q, a(v))$ in A, if $g(v)$ is true. ■

Algorithm 1 transforms an EFA into a single monolithic ordinary automaton. We are interested in synthesis and verification of DES and we want to avoid the state-space explosion problem when all components are synchronized. *Therefore we only implement the first four steps of the algorithm.* Before the monolithic automaton is computed in step five, we have a model consisting of a number of ordinary automata $A_{loc}, A_v^1, \dots, A_v^n$, whose alphabet Σ', consists of relabeled events from the original alphabet Σ. We can relate $L(E)$ to $L(A_{loc} \| A_v^1 \| \cdots \| A_v^n)$ by extending the mapping $\Psi(\cdot)$ to strings. This is done in the usual way by letting $\Psi(\epsilon) = \epsilon$, and $\Psi(s\sigma) = t$ if $\Psi(s) = r$ and $r\Psi(\sigma) = t$ for some $r \in \Sigma^*$. Since $\Psi(\cdot)$ projects relabeled events to their original events, it follows that $\Psi(L(A_{loc} \| A_v^1 \| \cdots \| A_v^n)) = L(E)$.

Typically, models are built using a number of interacting EFA. EFA that share variables and interact via the FSC can through their guard-action pairs exchange information during the synchronization process. In order to build ordinary automata that represent the synchronized behavior of multiple EFA, access to all guards and updating actions is needed for each transition in the synchronized system. If we transform interacting EFA separately to FA information is lost. The transformation must consider all components simultaneously. This fact implies that Algorithm 1 must be applied to all combinations of transitions in the synchronized system.

Let $E_k = \langle Q_k \times V, \Sigma_k, \rightarrow_k, (q_0^k, v_0) \rangle$, $k = 1, \dots, m$ be interacting EFA whose shared variables $v = (v^1, \dots, v^n)$ have domain $V = (V^1, \dots, V^n)$. The overall behavior of the system is described by the composition of all components using the FSC

$$E = E_1 \| \cdots \| E_m.$$

To obtain an isomorphic ordinary automaton A of the EFA E, we apply Algorithm 1 to all transitions $p \overset{\sigma}{\rightarrow}_{g/a} q$ of E, where $p = (p_1, \dots, p_m)$ and $q = (q_1, \dots, q_m)$ are locations in $E_1 \| \cdots \| E_m$. The difference is that the number of location automata in step 3 increase to m, i.e. instead of only one location automaton we have $A_{loc}^k = \langle Q^k, \Sigma', \rightarrow, q_0^k \rangle$, $k = 1, \dots, m$. By building the location automata and variable automata transition by transition, the modular ordinary automata model (in step 4 of Algorithm 1) can be implemented using an algorithm that only consumes a polynomial amount of space.

For each transition $p \overset{\sigma}{\rightarrow}_{g/a} q$ in E:

1 Introduce $|g|$ new events, all with *unique* names. Create a one to one mapping between each renamed event σ_i and an and-clause g^i in the guard g.

2 Add the relabeled events to Σ'.

3 For $i = 1 \dots |g|$:

a for $k = 1, \dots, m$, create $p_k \overset{\sigma_i}{\rightarrow} q_k$ in A_{loc}^k;

b for $k = 1, \dots, n$, create $v^k \overset{\sigma_i}{\rightarrow} a^k(v^k)$ in A_v^k if $g^{i,k}(v^k)$ is true.

In the dosing tank example in section II, each event is associated to a single transition whose guard expression consists of a single and-clause. This implies that $\Psi(\cdot)$ will be a bijective mapping between Σ' and Σ, and that a renaming of events is unnecessary. A FA model of the dosing tank, without relabeled events, is given figure 3.

VI. Controllability

The goal of this section is to define controllability for the EFA defined in section IV and to relate the new definition to the classical one. Since we allow don't care updating of shared variables, properties of subsystems are not necessarily true for the entire system. For instance, if E_1 and E_2 are EFA, it is possible that a state (p_1, p_2, v) in $E_1 \| E_2$ is reachable even though neither of the states (p_1, v) in E_1, or (p_2, v) in E_2, are reachable. In particular, it makes no sense to speak of the language of individual extended automata. The language of the components of a model can both be larger than or smaller than the language of the synchronized system. An example of a system consisting of two EFA where the languages of the components are smaller than the

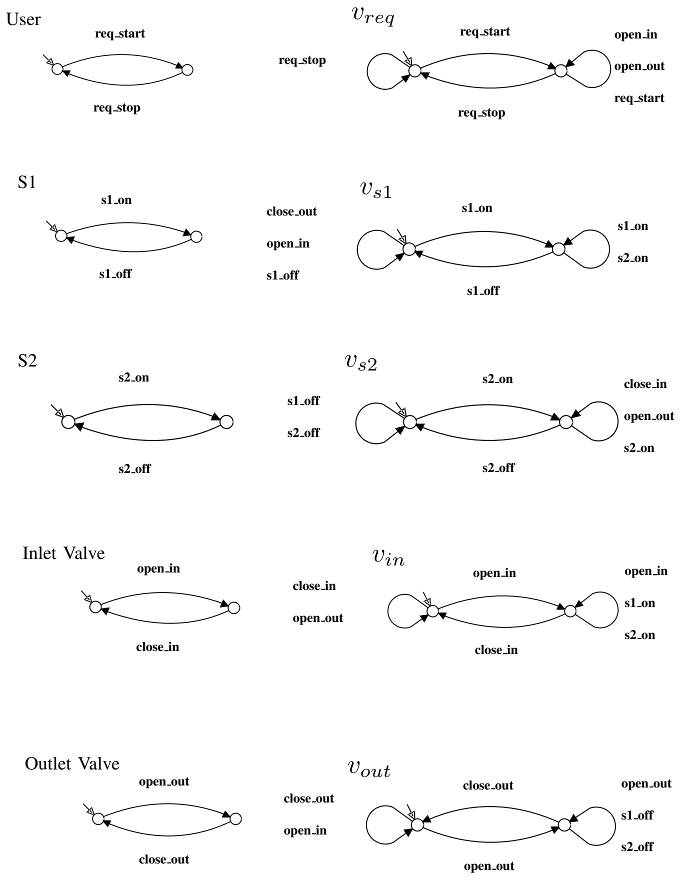

Fig. 3. A model of the dosing tank example consisting of ordinary automata. It has been generated using the first four steps of Algorithm 1.

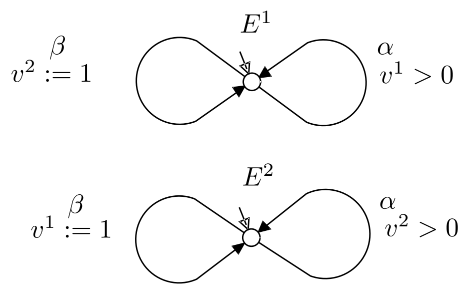

Fig. 4. Two EFA E^1 and E^2 sharing variables (v^1, v^2) with initial value $(0, 0)$. The event α can occur in $E^1 \| E^1$ but neither in E^1 nor in E^2.

language of the synchronized system, is given in figure 4. In general, the only meaningful language is the generated language of the synchronous product of all EFA in the model. However, the language of the synchronized system is not a suitable reference language for supervisory control purposes. Because of this, we will not speak of languages being controllable with respect to other languages, instead we shall talk about controllable languages with respect to entire systems.

The traditional definition of controllability from [1], formulated using ordinary automata is as follows.

Definition 9 (Controllability Automata):
Let G and K be deterministic finite state automata over the same alphabet and Σ_u be the set of uncontrollable events. K is controllable with respect to G if

$$L(K)\Sigma_u \cap L(G) \subseteq L(K).$$

If K and G where EFA with shared variables the languages $L(K)$ and $L(G)$ would, in general, be meaningless. A definition that neither depends on the language of subsystems nor the language of the entire system is required. For this, we make use the active event function and the concept of controllable/uncontrollable states.

Definition 10 (Active Event Function):
Let $G = \langle Q \times V, \Sigma, \rightarrow, (q_i, v_0)\rangle$ be an EFA with explicit state transition relation $\mapsto$. The active event function $\Gamma : Q \times V \rightarrow 2^{\Sigma}$ of G is defined as

$$\Gamma(p, v) \quad = \quad \{\sigma \in \Sigma \mid (p, v) \overset{\sigma}{\mapsto}\}.$$

Definition 11 (Controllable State):
Let G and K be two EFA using shared variables with domain V. Let Σ_u be the set of uncontrollable events and Σ_K be the alphabet of K. A state $(p_G, p_K, v) \in Q_G \times Q_K \times V$ in the synchronized automaton $G \| K$, is controllable if the following statement holds:

$$\Sigma_K \cap \Sigma_u \cap \Gamma(p_G, v) \subseteq \Gamma(p_K, v).$$

Uncontrollable states, are states of $G \| K$ where G allows an uncontrollable event but K disables the same event, via the FSC.

Definition 12 (Controllability EFA):
Let G be a plant, K be a specification and Σ_u be the set of uncontrollable events. K is controllable with respect to G and Σ_u, if and only if all *reachable* states of $G \| K$, are controllable.

The above definition is a straightforward generalization of Definition 9 that is feasible for non-deterministic automata. For deterministic finite state automata over the same alphabet they coincide. We are interested in synthesizing languages and therefore we want a language definition of controllability.

Definition 13 (Controllable Language EFA):
Let G and K be two EFA using shared variables with domain V. Let Σ_u be the set of uncontrollable events and $\mapsto$ be the explicit state transition relation of $G \| K$. A language $\mathcal{L}$ is controllable with respect to the system G, K, Σ_u if for all $s \in \overline{\mathcal{L}}$ and states (p_G, p_K, v) in $G \| K$ such that $G \| K \overset{s}{\mapsto} (p_G, p_K, v)$, it holds that:

$$s\big(\Sigma_u \cap \Gamma(p_G, v)\big) \subseteq \mathcal{L}.$$

Hence, a controllable language must always be able to follow uncontrollable events generated by the plant. Note that controllability with respect to systems is, as in the usual definition of controllability, a property of the prefix-closure of a language, i.e. $\mathcal{L}$ is controllable if and only if $\overline{\mathcal{L}}$ is controllable. A couple of important observations regarding Definition 13 are worth formulating into separate propositions.

Proposition 2: If K is controllable with respect to G and Σ_u then the language of the closed loop $L(G \| K)$ is a controllable language with respect to the system itself.

Proof: Assume that K is controllable with respect to G and Σ_u. For any $s \in L(G \| K)$ there exists (p_G, p_K, v)

such that $G\|K \overset{s}{\mapsto} (p_G, p_K, v)$. Since, K is controllable with respect to G and Σ_u, the reachable state (p_G, p_K, v) is controllable. This implies that for all $\sigma \in \Sigma_u \cap \Gamma(p_G, v)$, it holds that $s\sigma \in L(G\|K)$. ■

Proposition 3: If $\mathcal{L} \subseteq L(G\|K)$ and $\mathcal{L}$ is controllable with respect to the system G, K, Σ_u then $\mathcal{L}$ can never reach any uncontrollable states of $G\|K$.

Proof: Assume that $\mathcal{L} \subseteq L(G\|K)$ is controllable with respect to the system G, K, Σ_u and that $\mathcal{L}$ can reach uncontrollable states of $G\|K$. This means that $\exists s \in \mathcal{L}$ and an uncontrollable state (p_G, p_K, v) in $G\|K$ such that $G\|K \overset{s}{\mapsto} (p_G, p_K, v)$. Since the state (p_G, p_K, v) is uncontrollable there exists an uncontrollable event σ in $\Sigma_K \cap \Sigma_u \cap \Gamma(p_G, v)$ that does not belong to $\Gamma(p_K, v)$. This implies that $s\sigma \notin L(G\|K)$. By assumption $\mathcal{L}$ was controllable so $s\sigma \in \mathcal{L} \subseteq L(G\|K)$. Hence, the assumption was wrong and the statement follows. ■

If $L(G\|K)$ is not controllable with respect to the system, we want to find the "largest" sub-language of $L(G\|K)$ that is controllable. We let $L(G\|K)^{\uparrow C}$ denote the *supremal controllable sub-language* of $L(G\|K)$. A model where the EFA share variables does not have any natural reference language. As explained previously, none of the languages $L(G)$ or $L(K)$ makes any sense, and $L(G\|K)$ is just a test language that we hope is controllable with respect to the system. In the special case when G and K do not share variables and have the same alphabet, $L(G\|K) = L(G) \cap L(K)$ and Definition 13 breaks down to the classical controllability definition.

Proposition 4: Let G and K be two EFA using the same alphabet and having distinct variables with domains V_G and V_K, respectively. Let Σ_u be the set of uncontrollable events. A language $\mathcal{L}$ is controllable with respect to the system G, K, Σ_u if and only if

$$\overline{\mathcal{L}}\Sigma_u \cap L(G) \subseteq \overline{\mathcal{L}}.$$

Proof: $\Rightarrow$) Assume that $\mathcal{L}$ is controllable with respect to the system G, K, Σ_u. Let $s \in \overline{\mathcal{L}}$ such that $G\|K \overset{s}{\mapsto}$, be given. Let $\Gamma_{G(s)}$ denote the union of all active events in states (p_G, v_G) in G such that $G\|K \overset{s}{\mapsto} (p_G, p_K, v_G, v_K)$ for some state (p_K, v_K) in K. Since $\mathcal{L}$ is controllable with respect to the system G, K, Σ_u we have:

$$s\big(\Sigma_u \cap \Gamma_{G(s)}\big) \subseteq \overline{\mathcal{L}}.$$

G and K have distinct variables and the same alphabet so $s\Gamma_{G(s)} \subseteq L(G)$. In particular, the language $s\Gamma_{G(s)}$ contains precisely all concatenations of s with events σ such that $s\sigma \in L(G)$. This implies that

$$s\Sigma_u \cap L(G) \subseteq s\big(\Sigma_u \cap \Gamma_{G(s)}\big) \subseteq \overline{\mathcal{L}}.$$

$\Leftarrow$) Assume that $\overline{\mathcal{L}}\Sigma_u \cap L(G) \subseteq \overline{\mathcal{L}}$. Given any $s \in \overline{\mathcal{L}}$ and a state (p_G, p_K, v_G, v_K) in $G\|K$ such that $G\|K \overset{s}{\mapsto} (p_G, p_K, v_G, v_K)$, we have:

$$s\Sigma_u \cap L(G) \subseteq \overline{\mathcal{L}}.$$

Again $s \in L(G)$ and since $s\big(\Sigma_u \cap \Gamma(p_G, v_G)\big) \subseteq s\Sigma_u$ and $s\big(\Sigma_u \cap \Gamma(p_G, v_G)\big) \subseteq L(G)$, it follows that

$$s\big(\Sigma_u \cap \Gamma(p_G, v_G)\big) \subseteq s\Sigma_u \cap L(G) \subseteq \overline{\mathcal{L}}.$$

■

In the standard definition of controllability it is possible to compare languages with a reference language generated by the automata model. We have introduced the notion of a controllable language with respect to an entire system modeled by EFA. As we have seen, it is a generalization of the classical definition of controllability for ordinary automata. In what follows, when we speak about a controllable language, it will always be with respect to the entire system, i.e. Definition 13.

VII. Transforming Supervisory Control Problems

The approach that is suggested in this report is to use existing algorithms developed for ordinary automata for synthesis and verification of EFA models. This section presents a transformation algorithm that makes this possible. The algorithm transforms supervisory control problems modeled by EFA into equivalent FA control problems. The transformation algorithm is the key result of this report. The modularity and structure of the EFA model is preserved making the FA model feasible for analysis. The procedure is illustrated with an example that contains a number of difficulties and the presented algorithm is proven to be sound. We start out by explaining supervisory control and synthesis.

A. Supervisory Control and Synthesis

Supervisors are used to restrict the behaviors of plants represented by automata in accordance to some given control objective. A supervisor observes the sequence of events occurring in the plant and enables or disables events. A subset of the plant events are uncontrollable and not subject to disablement. Classical supervisory control theory [1] concerns a single plant automaton G and a single specification automaton K. The plant G models the system to be controlled the specification K models the control objective and the composition $G\|K$ models the desired behavior or the closed loop. Uncontrollable states, with respect to G, of the synchronous product $G\|K$ are states where G allows an uncontrollable event but K disables the same event, via the FSC. States that are not uncontrollable are called controllable. If all *reachable* states of $G\|K$ are controllable with respect to G, then the desired closed loop can be achieved, and K is a supervisor that implements the desired behavior $L(G\|K)$. When the desired behavior $L(G\|K)$ can not be obtained, it is known that for any finite regular plant and specification, a unique controllable *least restrictive* (or *maximally permissive*) supervisor exists. The language that a maximally permissive supervisor admits is called the *supremal controllable sub-language of* $L(G\|K)$, denoted $L(G\|K)^{\uparrow C}$.

The *synthesis* task is to find $L(G\|K)^{\uparrow C}$ when the desired closed loop can not be obtained. The standard monolithic synthesis algorithm takes a specification K and a plant G, computes $G\|K$ and searches the state-space. A known problem with this approach is the state space explosion problem when the product of all components is built. In [19] different approaches to fight the state-space explosion is described and divided into six main categorizes: modular approaches, hierarchical approaches, symbolic representation, partial order techniques, compositional approaches and exploiting symmetry. All of these strategies are directly applicable to ordinary automata models.

B. Transformation Setup and Requirements

The setup for the transformation algorithm is a modular supervisory control problem consisting of a number of plant EFA $G = G^1\| \cdots \|G^k$ and specification EFA $K = K^1\| \cdots \|K^m$. It is assumed that all EFA are deterministic. The algorithm then transforms the supervisory control problem into a modular supervisory control problem consisting of only ordinary plant automata

$$\tilde{G} = \tilde{G}^1\| \cdots \|\tilde{G}^i$$

and ordinary specification automata

$$\tilde{K} = \tilde{K}^1\| \cdots \|\tilde{K}^j.$$

To be able to verify if $L(G\|K)$ is controllable or to find $L(G\|K)^{\uparrow C}$ from analysis of the FA model the transformation must have a number of properties. Obviously, a relation between the alphabets and languages of the two models is needed. We also require that reachable states of EFA model are mapped to reachable states of the FA model and that controllable/uncontrollable states of the EFA model are mapped to controllable/uncontrollable states of the FA model. The algorithm begins with the first four steps of Algorithm 1 giving us a modular supervisory control problem with a plant

$$G^1_{loc}\| \cdots \|G^k_{loc}\|G^1_v\| \cdots \|G^n_v$$

and a specification

$$K^1_{loc}\| \cdots \|K^m_{loc}.$$

$G^1_{loc}, \ldots, G^k_{loc}$ and $K^1_{loc}, \ldots, K^m_{loc}$ are location automata, and $G^1_v, \ldots, G^n_v$ are variable automata. There are two main problems with the above model. It may contain fictional *uncontrollable states caused by the renaming of events*, and since all variable automata have been chosen to be plants, it is possible that *uncontrollable states have been removed*. To illustrate the required modifications of ordinary automata models generated from the first four steps of Algorithm 1, we use the example depicted in figure 5 and figure 6.

Figure 5 shows a plant G and specification K sharing the variable v with domain $V = \{0, 1, 2\}$ and initial value $v_0 = 0$. If the controllable event c occurs then the specification forbids the uncontrollable event a. Hence, the specification is not controllable and a supervisor needs to be synthesized.

Figure 6 shows the corresponding ordinary automata model generated from the first four steps of Algorithm 1. The relation between the languages of the two models is $\Psi(L(G_{loc}\|K_{loc}\|G_v)) = L(G\|K)$, where $\Psi(\cdot)$ is the renaming function defined in section V. It is quite clear that reachable states are mapped to reachable states in the

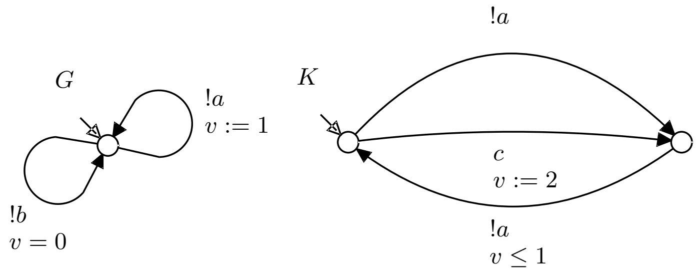

Fig. 5. G is a plant and K is a specification sharing the variable v with domain $V = \{0, 1, 2\}$ and initial value $v_0 = 0$. If the controllable event c occurs then the specification forbids the uncontrollable event a.

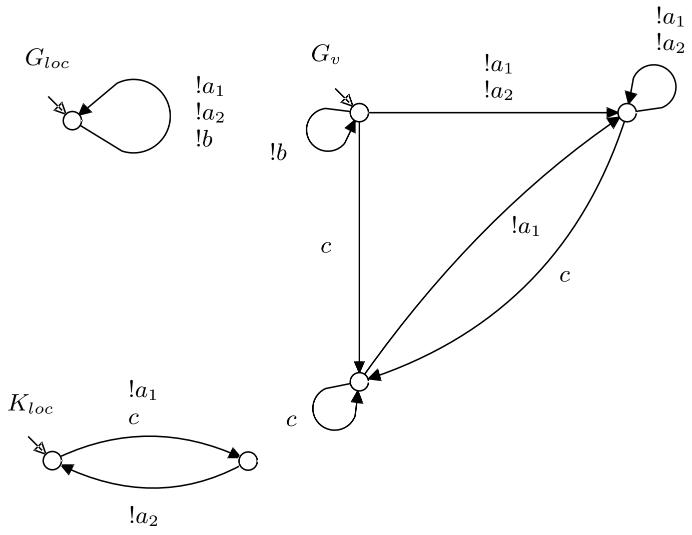

Fig. 6. G_{loc} and K_{loc} are location automata and G_v is a variable automaton. The automata are obtained by transforming the EFA model in figure 5 using the first four steps of Algorithm 1. The renaming function $\Psi(\cdot)$ is defined as $\Psi(b) = b$, $\Psi(c) = c$ and $\Psi(a_i) = a$, $i = 1, 2$.

two models. However, the last requirement that controllable/uncontrollable states of the EFA model are mapped to controllable/uncontrollable states of the FA model is not fulfilled.

C. Uncontrollable States Caused by Renaming

Consider figures 5 and 6. The renaming of the uncontrollable event a into a_1 and a_2 causes fictional uncontrollable states in the model in figure 6. The uncontrollable states of the synchronized system caused by renaming in figure 6 are of two kinds:

1) states where K_{loc} disables a_1/a_2 and enables a_2/a_1, and the plant enables both a_1 and a_2,
2) the state where K_{loc} disables a_1 and enables a_2, and the plant enables a_1.

The renaming of events was designed to implement the guard expressions in the EFA model. To avoid fictional uncontrollable states caused by the renaming in Algorithm 1 we shall add "plantified" specifications to the plant.

There are many ways of transforming specifications into the plants. In [20], the complete plant automaton was introduced. It is the plant automaton replacing a specification, where uncontrollable states have been transformed into blocking states. Here, we want to transform fictional uncontrollable states, caused by the renaming of events, into controllable states. To do this we define the function $P_\Psi(\cdot)$ that transforms specification automata into plant automata in a special way.

Definition 14: Let $K = \langle Q, \Sigma, \rightarrow, q_0\rangle$ be a specification, Σ_u a set of uncontrollable events and Ψ be a function with domain of definition Σ. The function $P_\Psi(K)$ is defined as

$$P_\Psi(K) = \langle Q, \Sigma, \rightarrow_\Psi, q_0\rangle,$$

where

$$\begin{aligned}\rightarrow_\Psi \quad = \quad & \rightarrow \cup \{(q,\sigma,q) \mid q \in Q, \sigma \in \Sigma_u \cap \Sigma \\ & \text{and } \forall \sigma' \in \Psi^{-1}(\Psi(\sigma)), \quad q \overset{\sigma'}{\not\rightarrow}\}.\end{aligned}$$

The plant $P_\Psi(K)$ is obtained from the specification K by adding self loops in special way: if no relabeled uncontrollable event in $\Psi^{-1}(\Psi(\sigma))$ is enabled in the state q then all uncontrollable events in $\Psi^{-1}(\Psi(\sigma))$ are self looped at q. Note that $P_\Psi(K)\|K$ and K are isomorphic, and since we pay no attention to the naming of the states in automata we write $P_\Psi(K)\|K = K$. In order to avoid fictional uncontrollable states caused by renaming we extend the plant model to

$$G^1_{loc}\| \cdots \|G^k_{loc}\|G^1_v\| \cdots \|G^n_v\|P_\Psi(K_{loc})$$

where

$$P_\Psi(K_{loc}) = P_\Psi(K^1_{loc})\| \cdots \|P_\Psi(K^m_{loc}).$$

In states where the specification K_{loc} enables uncontrollable relabeled events the plant automata $P_\Psi(K_{loc})$ enables the same uncontrollable relabeled events. A consequence is that no fictional uncontrollable states caused by the renaming of events exits in the extended plant model. This is exemplified in figure 7, which shows the extended plant model of our running example problem.

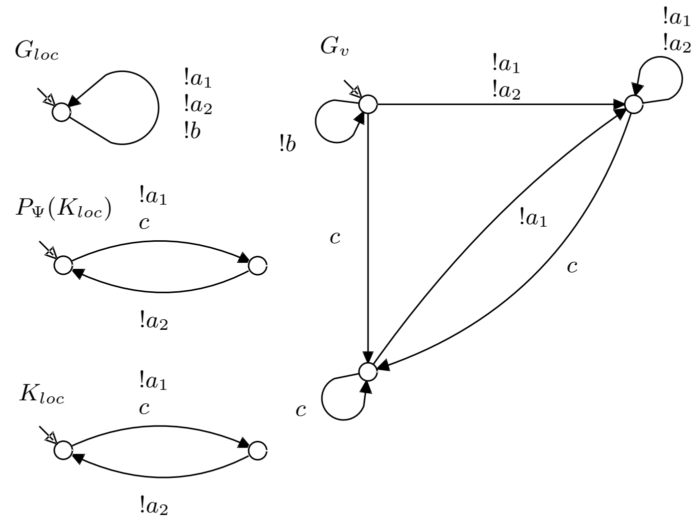

Fig. 7. In order to avoid that fictional uncontrollable events are introduced in the ordinary automata model the plant has been extended with $P_\Psi(K_{loc})$. However, since the Algorithm 1 does not distinguish between guards from plants and guards from specifications the uncontrollable state reached after the event c is transformed into a controllable state.

D. Uncontrollable States Caused by Guards

Algorithm 1 removes transitions triggered by uncontrollable events in the plant that do not fulfil the guards without checking if they were removed by the plant guard or the specification guard. To avoid erroneous uncontrollable states in the FA model when a plant guard blocks uncontrollable transitions, we want the variable automata to be plants. On the other hand, to introduce uncontrollable states correctly when a specification guard blocks uncontrollable transitions, we want the variable automata to be specifications. The choice can become ambiguous when guards on the same variable occur both in specification transitions and in plant transitions. Consider the extended ordinary automata model in figure 7. In the variable automaton G_v, a plant guard ($v = 0$) have disabled the uncontrollable event b at the states corresponding to $v = 1$ and $v = 2$ and a specification guard ($v \leq 1$) have disabled the relabeled uncontrollable event a_2 at the state corresponding to $v = 2$. If G_v is a plant then all states are controllable and if G_v is a specification then we get an additional uncontrollable state at the initial state. Clearly, we can not solve this ambiguity by choosing the appropriate status, plant or specification, of the variable automaton G_v.

We solve this issue in three steps. First only the plant guards are considered in the algorithm that builds the variable automata and all variable automata are given a plant status. Then the uncontrollable states caused by specification guards are identified. These illegal states are made uncontrollable in the ordinary automata model in the last step.

Collecting Uncontrollable States: We have chosen to give all variable automata a plant status in the ordinary automata model. We have also extended the plant model by adding "plantified" specifications. This implies that all uncontrollable states caused by specification guards that blocks uncontrollable transitions have been transformed into controllable states. To find these "illegal" states, we distinguish between *plant guards* g_G and *specification guards* g_K, and build a new guard expression. Let $p = (p_G, p_K)$ and $q = (q_G, q_K)$ be locations in $G\|K$, σ be an uncontrollable event and $p \xrightarrow{\sigma}_{g/a} q$ be a transition in $G\|K$, where $g = g_G \wedge g_K$. Consider the guard expression

$$\hat{g} = g_G \wedge \neg g_K.$$

We assume that $\hat{g}$ have been parsed and written in disjunctive normal form

$$\hat{g} = \hat{g}^1 \vee \cdots \vee \hat{g}^l.$$

A state $p = (p_G, p_K, v)$ in $G\|K$ where $\hat{g}(v)$ is true, is uncontrollable in the EFA model since at that state a specification guard blocks a transition triggered by an uncontrollable event. However, the corresponding state in the FA model $G_{loc}\|G_v\|P_\Psi(K)\|K_{loc}$ is controllable. Each and-clause $\hat{g}^i$ defines a set of uncontrollable states in $G\|K$

$$F^i = \{(p_G, p_K, v) \in Q_G \times Q_K \times V \mid \hat{g}^i(v) = 1\}.$$

Each set of states in the EFA model corresponds to a set of states in the FA model, where locations corresponds to states in the location automata and variables values corresponds to states in the variable automata. Therefore, we also let F^i denote the corresponding set of illegal states in the FA model. Since we have extended the plant model with $P_\Psi(K_{loc})$ it suffices to consider the plant states of the FA model.

Proposition 5: Let G, $P_\Psi(K)$ be plant automata, K and $\hat{K}$ be a specification automata. Let (p,q,r) be a state in $G\|K\|\hat{K}$ and (p,q',r) be the corresponding state in $G\|P_\Psi(K)\|\hat{K}$. Then the following implication holds:

$$(p,q',r) \text{ is uncontrollable in } G\|P_\Psi(K)\|\hat{K}$$
$$\Rightarrow$$
$$(p,q,r) \text{ is uncontrollable in } G\|K\|\hat{K}.$$

Proof: Since $G\|K\|\hat{K} = G\|P_\Psi(K)\|K\|\hat{K}$, and $K\|\hat{K}$ disables more events than $\hat{K}$ alone the statement follows. ■

According to Proposition 5, we can create uncontrollable states in $G_{loc}\|G_v\|P_\Psi(K)$ and thereby obtain uncontrollable states in $G_{loc}\|G_v\|P_\Psi(K)\|K_{loc}$. Benefitting from the structure the FA model, we represent all illegal states obtained from specification guards as

$$F = \bigcup_{i=1}^{L} F^i = \bigcup_{i=1}^{L} F_1^i \times \cdots \times F_{k+n+m}^i,$$

where $F_1^i \times \cdots \times F_k^i$ is a subset of the states in G_{loc}, $F_{k+1}^i \times \cdots \times F_{k+n}^i$ is a subset of the states in G_v and $F_{k+n+1}^i \times \cdots \times F_{k+n+m}^i$ is a subset of the states in $P_\Psi(K_{loc})$.

Making States Uncontrollable: In this section we present a general way of rewriting static specifications that identifies illegal states in ordinary automata models, into dynamic automata specifications. This is done such that illegal states become uncontrollable. The method can be fully automated and preserves the modularity of the model.

We start with a modular plant model consisting of a number of ordinary automata $G = G_1\|\ldots\|G_m$ and a set that explicitly specifies the illegal states of the plant. We assume that the bad states are given as Cartesian product sets

$$F = \bigcup_{j=1}^{n} F^j = \bigcup_{j=1}^{n} F_1^j \times \cdots \times F_m^j,$$

where each F_k^j is a subset of the states of the plant automaton G_k. It is assumed that all F^j's are proper subsets of the plant states. The following algorithm replaces the static specification F with automata specifications and modify the plant model in such a way that the illegal plant states are transformed into uncontrollable states.

Algorithm 2: Let $G_k = \langle Q_k, \Sigma_k, \rightarrow_k, q_0^k \rangle$ $k = 1,\ldots,m$, be plant automata and $F = \bigcup_{j=1}^{n} F^j$ be illegal states of $G = G_1\|\cdots\|G_m$. The following algorithm transforms the illegal states of G into uncontrollable states of $G_\circlearrowleft\|\hat{K}$, where $G_\circlearrowleft$ is a modified plant automaton and $\hat{K} = \hat{K}_1\|\cdots\|\hat{K}_n$ is a specification automaton.

1) $\forall F^j \in F$:
   a) create a new unique *uncontrollable* event u_j,
   b) create a specification automaton $\hat{K}_j = \langle \{q\}, \{u_j\}, \emptyset, q, \rangle$, that forbids u_j,
   c) $\forall F_k^j \in F^j$ such that $F_k^j \neq Q_k$:
      i) add u_j to Σ_k,
      ii) $\forall q \in F_k^j$ add self loops $q \overset{u_j}{\rightarrow}_k q$ to $\rightarrow_k$.
2) Update all plant automata G_k that has modified alphabets Σ_k and transition relations $\rightarrow_k$ and denote the modified plant with $G_\circlearrowleft$.

Proof: The constructed specifications $\hat{K}_j$ are single state automata that forbids the corresponding uncontrollable events u_j. Hence, all states where u_j are active, are uncontrollable. Let $q = (q_1,\ldots,q_m)$ be a state in the plant $G_\circlearrowleft$ that belongs to the set F of illegal states. Then q must belong to at least one of the product sets of F, say $F^j = F_1^j \times \cdots \times F_m^j$. For the local-state $q_k \in Q_k$, there are two possibilities, either $u_j \notin \Sigma_k$ or there is a self loop at q_k triggered by the uncontrollable event u_j. Since we assumed that not all F_k^j equals Q_k, at least one local-state of $q = (q_1,\ldots,q_m)$ must have self loops triggered by u_j. Thus, it follows from the FSC that u_j can indeed occur in q and therefore the state is uncontrollable. ■

Algorithm 2 adds uncontrollable self loops in all illegal plant states and forbids the corresponding events. The subscript $\circlearrowleft$ is used to denote the modified plants i.e. the transformed supervisory control problem is given by the plant automaton $G_\circlearrowleft$ and the specification automaton $\hat{K}$.

E. A Supervisory Transformation Algorithm

We believe that EFA will help to bridge the gap between industry logic control programs and the classical supervisory theory. It is therefore of great importance to develop effective synthesis algorithms for EFA. However, since action functions may change when EFA are composed with other EFA, properties that are true for a subsystem are not necessarily true for the entire synchronized system. Unreachable states of single EFA can become reachable after synchronization. It is therefore unfeasible to reason in a modular way about EFA and it is difficult to develop effective synthesis algorithms for EFA. Our approach is to transform EFA models into isomorphic FA models. This way we can rely on existing algorithms developed for FA that have been tested and have been proven to handle systems with large state-spaces. Here, we present a transformation algorithm that transforms a modular supervisory control problem modeled by EFA into an equivalent FA supervisory control problem. The transformation preserves the structure of the EFA model. The equivalent FA model can be used both for synthesis and verification of the EFA model.

Let $G = G^1\|\cdots\|G^k$ be plant EFA and $K = K^1\|\cdots\|K^m$ be specification EFA and assume that all EFA are action consistent. The product $G\|K$ is a model of the desired closed loop. In order to verify if $L(G\|K)$ is controllable or to calculate $L(G\|K)^{\uparrow C}$, we transform the system into a modular supervisory control problem consisting of ordinary automata. The event set of the ordinary supervisory control problem is obtained by renaming events in the original alphabets of G and K. $\Psi(\cdot)$ is the function defined in Algorithm 1 that maps relabeled events to their original events.

Algorithm 3 (Transforming Control Problems): Let $G = G^1\|\cdots\|G^k$ be plant EFA and $K = K^1\|\cdots\|K^m$ be specification EFA using the shared variables $v = (v^1,\ldots,v^n)$. Assume that the EFA are deterministic. The modular supervisory control problem consisting of ordinary automata given by the plant

$$\tilde{G} = \big(G_{loc}\|G_v\|P_\Psi(K_{loc})\big)_{\circlearrowleft},$$

and the specification

$$\tilde{K} = K_{loc}\|\hat{K},$$

is generated from the EFA model in three steps:

(i) For each transition $p \xrightarrow{\sigma}_{g_G\wedge g_K/a} q$ in $G\|K$:
  a) If σ is *controllable*, apply the first 4 steps of Algorithm 1 to $p \xrightarrow{\sigma}_{g_G\wedge g_K/a} q$.
  b) If σ is *uncontrollable*, apply the first 4 steps of Algorithm 1 to $p \xrightarrow{\sigma}_{g_G/a} q$, and collect all states (p,v) such that $g_G \wedge \neg g_K$ is true into the set F.

(ii) Use the function $P_\Psi(\cdot)$ in Definition 14 to extend the plant with $P_\Psi(K_{loc})$.

(iii) Transform the collected states F into uncontrollable states using Algorithm 2, where F is interpreted as states in $G_{loc}\|G_v\|P_\Psi(K_{loc})$.

$\tilde{G}\|\tilde{K}$ is the isomorphic FA of the EFA $G\|K$ (up to renaming of events) so we do not need to distinguish between the states of $\tilde{G}\|\tilde{K}$ and the states of $G\|K$. By construction the algorithm maps controllable/uncontrollable states of $G\|K$ to controllable/uncontrollable states of $\tilde{G}\|\tilde{K}$. Because of this can use the ordinary automata model to verify the controllability of the EFA model.

Proposition 6 (Verification): Let $G = G^1\|\cdots\|G^k$ be plant EFA and $K = K^1\|\cdots\|K^m$ be specification EFA that are deterministic. Let $\tilde{G}$ and $\tilde{K}$ be the plant and specification obtained by applying Algorithm 3 to G and K. Then the following two statements are equivalent.

(i) $L(G\|K)$ is controllable with respect to G and K.
(ii) $L(\tilde{G}\|\tilde{K})$ is controllable with respect to $\tilde{G}$ and $\tilde{K}$.

Proof: From Proposition 1 it follows that if we rename the events in $\tilde{G}\|\tilde{K}$ with $\Psi(\cdot)$ then $\tilde{G}\|\tilde{K}$ is the isomorphic FA of $G\|K$. Hence, reachable states of $G\|K$ are mapped to reachable states of $\tilde{G}\|\tilde{K}$. Since $P_\Psi(K_{loc})$ is included in $\tilde{G}$ there exits no fictional uncontrollable states caused by the renaming of events. Uncontrollable states of $G\|K$ caused by specification guards are collected in the set F. From the proof of Algorithm 2, it follows that these states are transformed correctly into uncontrollable states of $\tilde{G}\|\tilde{K}$. Hence, controllable/uncontrollable states of $G\|K$ are transformed into controllable/uncontrollable states of $\tilde{G}\|\tilde{K}$ and the statement follows. ■

Let $\sim$ be the equivalence relation on strings s, t in $L(\tilde{G}\|\tilde{K})$, where $s \sim t$ if $\Psi(s) = \Psi(t)$. Let $\mathcal{L} \subseteq L(G\|K)$ and $\tilde{\mathcal{L}} \subseteq L(\tilde{G}\|\tilde{K})$ be such that $\Psi(\tilde{\mathcal{L}}) = \mathcal{L}$. The fact that $G\|K$ is deterministic implies that strings in $s,t \in L(\tilde{G}\|\tilde{K})$, where $s \sim t$ visit exactly the same states in $\tilde{G}\|\tilde{K}$. An important consequence is that the sub-languages $\mathcal{L}$ and $\tilde{\mathcal{L}}$ also must visit the same states. The following lemma formalizes the fact that, the set of uncontrollable active events of a plant state in the FA model, are surjectively mapped to the set of uncontrollable active events of the corresponding plant state of the EFA model.

Lemma 1: Let G be a plant EFA and K be a specification EFA using the shared variables $v = (v^1,\ldots,v^n)$. Assume that the EFA are deterministic. Let $\tilde{G}$ and $\tilde{K}$ be the plant and specification obtained by applying Algorithm 3 to G and K. Let (p_G, v) be a state in G and $p_{\tilde{G}}$ be the corresponding state in $\tilde{G}$. Then $\Psi(\Gamma(p_{\tilde{G}}) \cap \Psi^{-1}(\Sigma_u)) = \Gamma(p_G,v) \cap \Sigma_u$.

Proof: Let $\sigma \in \Gamma(p_G,v) \cap \Sigma_u$ be given. Then there must exist a transition $p \xrightarrow{\sigma}_{g_G\wedge g_K/a} q$ of $G\|K$ where the plant guard $g_G(v) = 1$. If $g_K(v) = 1$ then relabeled events in $\Psi^{-1}(\sigma)$ are added to $\Gamma(p_{\tilde{G}}) \cap \Psi^{-1}(\Sigma_u)$ in step 3 and 4 of Algorithm 1. If $g_K(v) = 0$ then $p_{\tilde{G}}$ is collected to the set of uncontrollable states and relabeled events in $\Psi^{-1}(\sigma)$ are added to $\Gamma(p_{\tilde{G}}) \cap \Psi^{-1}(\Sigma_u)$ as uncontrollable self loops. ■

Now its possible to prove that the ordinary automata control problem can be used for synthesis.

Proposition 7 (Synthesis): Let $G = G^1\|\cdots\|G^k$ be plant EFA and $K = K^1\|\cdots\|K^m$ be specification EFA using the shared variables $v = (v^1,\ldots,v^n)$. Assume that all EFA are deterministic. Let $\tilde{G}$ and $\tilde{K}$ be the plant and specification obtained by applying Algorithm 3 to G and K. Let $\mathcal{L} \subseteq L(G\|K)$ and $\tilde{\mathcal{L}} = sup\{\mathcal{N} \subseteq L(\tilde{G}\|\tilde{K}) \mid \Psi(\mathcal{N}) = \mathcal{L}\}$. Then the following two statements are equivalent.

(i) $\mathcal{L}$ is controllable with respect to G and K.
(ii) $\tilde{\mathcal{L}}$ is controllable with respect to $\tilde{G}$ and $\tilde{K}$.

Proof: $(i) \Rightarrow (ii)$. Assume that (i) is true and (ii) is false. Let $\tilde{s} \in \overline{\tilde{\mathcal{L}}}$ and $(p_{\tilde{G}}, p_{\tilde{K}}) \in \tilde{G}\|\tilde{K}$ be such that $\tilde{G}\|\tilde{K} \xrightarrow{\tilde{s}} (p_{\tilde{G}}, p_{\tilde{K}})$, and $\tilde{\sigma} \in \Psi^{-1}(\Sigma_u) \cap \Gamma(p_{\tilde{G}})$, where $\tilde{s}\tilde{\sigma} \notin \tilde{\mathcal{L}}$. Let (p_G, v) be the corresponding state in $G\|K$ i.e. $G\|K \overset{\Psi(\tilde{s})}{\mapsto} (p_G, p_K, v)$. By assumption $\mathcal{L}$ is controllable so by Definition 13, we have $\Psi(\tilde{s})\big(\Sigma_u \cap \Gamma(p_G,v)\big) \subseteq \mathcal{L}$. From Lemma 1 it follows that:

$$\Psi\big(\tilde{s}\big(\Psi^{-1}(\Sigma_u) \cap \Gamma(p_{\tilde{G}})\big)\big) \subseteq \mathcal{L}.$$

This implies that $\Psi(\tilde{\mathcal{L}} \cup \{\tilde{s}\sigma\}) = \mathcal{L}$ which is a contradiction since $\tilde{\mathcal{L}} \cup \{\tilde{s}\sigma\}$ is larger than $\tilde{\mathcal{L}}$.

$(ii) \Rightarrow (i)$. Assume that (ii) holds. Let $s \in \overline{\mathcal{L}}$ and (p_G, p_K, v) in $G\|K$ be such that $G\|K \overset{s}{\mapsto} (p_G, p_K, v)$. Let $(p_{\tilde{G}}, p_{\tilde{K}})$ be the corresponding state in $\tilde{G}\|\tilde{K}$. Since $\tilde{\mathcal{L}}$ is controllable with respect to $\tilde{G}$ and $\tilde{K}$ there exists $\tilde{s} \in \tilde{\mathcal{L}}$ such that $\Psi(\tilde{s}) = s$ and

$$\tilde{s}\big(\Psi^{-1}(\Sigma_u) \cap \Gamma(p_{\tilde{G}})\big) \subseteq \tilde{\mathcal{L}}.$$

By Lemma 1 applying $\Psi(\cdot)$ to both sides of the above equation gives us $s\big(\Sigma_u \cap \Gamma(p_G,v)\big) \subseteq \mathcal{L}$ and the statement follows. ■

A consequence of Proposition 7 is that

$$\Psi(L(\tilde{G}\|\tilde{K})^{\uparrow C}) = \Psi(L(\tilde{G}\|\tilde{K}))^{\uparrow C} = L(G\|K)^{\uparrow C},$$

and therefore we can use the ordinary automata model to do both synthesis and verification. An FA model that can be used to synthesize a supervisor for the EFA model presented in figure 5 is depict in figure 8. It has been generated by Algorithm 3.

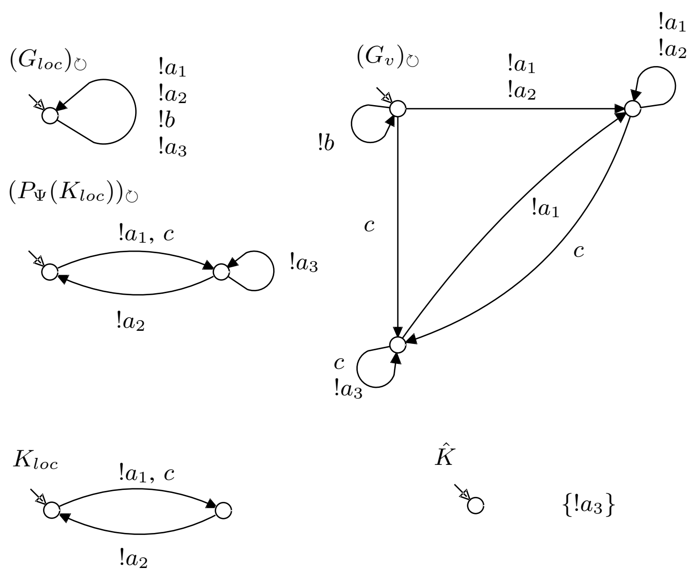
$(G_{loc})_{\circlearrowright}$
$!a_1$ $!a_2$ $!b$ $!a_3$
$(G_v)_{\circlearrowright}$
$!a_1$ $!a_2$
$!a_1$ $!a_2$
$!b$
c
$!a_1$
c
c $!a_3$
$(P_\Psi(K_{loc}))_{\circlearrowright}$
$!a_1,\ c$
$!a_3$
$!a_2$
K_{loc}
$!a_1,\ c$
$!a_2$
$\hat{K}$
$\{!a_3\}$

Fig. 8. The above model is an equivalent ordinary automata control problem of the EFA control problem given in figure 5. It has been generated by Algorithm 3. The renaming function $\Psi(\cdot)$ is defined as $\Psi(b) = b$, $\Psi(c) = c$ and $\Psi(a_i) = a$, $i = 1, 2, 3$.

VIII. CONCLUSIONS

The proposed modeling framework of extended finite automata (EFA) can be used to design supervisors for complex systems where ordinary finite automata (FA) modeling requires complicated and possibly non-intuitive solutions. Due to the use of variables, guard expressions and action functions, the EFA formalism can hide information and represent systems more compact than FA.

We have considered a general setting where no restrictions are made on the sharing of variables between concurrent EFA and don't care updating of shared variables is allowed. The don't care assumption states that everything that is not mentioned explicitly in an action does not change. Since action functions may change when EFA are composed with other EFA, properties that are true for a subsystem are not necessarily true for the entire synchronized system. Unreachable states of single EFA can become reachable after synchronization. It is therefore unfeasible to reason in a modular way about EFA.

In order to formulate supervisory control problems modeled by EFA, we have generalized the standard definition of controllability. However, developing effective synthesis algorithms for EFA is not an easy task. To overcome this difficulty we provide an algorithm that transforms supervisory control problems modeled by automata with shared variables into equivalent ordinary automata control problems. By examining the information exchange between all components of the model, we avoid building the product of the extended model and instead, we obtain an equivalent modular FA model. The algorithm is feasible for any finite modular system whose EFA share variables with finite domain.

The transformation algorithm is applied to all combinations of transitions in the synchronized system. Since the algorithm builds the ordinary automata transition by transition, it only consumes a polynomial amount of space. However, for each transition in the synchronized EFA model, transitions with relabeled events are created in the FA model. This can lead to an explosion of events and transitions in the ordinary automata model. One possible way of dealing with this problem is to minimize the relabelling of events and to remove unnecessary transitions.

The presented algorithms and the EFA framework have been implemented in the supervisory control tool Supremica. Since Supremica also implements state-of-the-art algorithms and data structures for dealing with large scale problems, we hope that this work will facilitate the adaptation of the supervisory control ideas into industrial applications. Since a synthesized supervisor can be implemented by adding additional guards to the original EFA model, see [8], we believe that, unlike FA, the EFA framework can be used to close the loop of the logic development i.e. provide feedback to the model designers.

The authors would like to compare the performance of algorithms on automata generated from EFA models with their performance on equivalent handwritten automata models. It is also of interest to develop effective synthesis and verification algorithms directly on EFA models.

ACKNOWLEDGEMENT

The authors would like to thank Robi Malik at University of Waikato, New Zealand, for helpful discussions about the EFA concept and for guidance with the implementation.

REFERENCES

[1] P. J. Ramadge and W. M. Wonham, "The control of discrete event systems," *IEEE*, vol. 77, no. 1, pp. 81–98, Jan. 1989.
[2] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, 2nd ed., ser. Series in Computer Science. Addison-Wesley, 2003.
[3] S. Balemi, G. J. Hoffmann, P. Gyugyi, H. Wong-Toi, and G. F. Franklin, "Supervisory control of a rapid thermal multiprocessor," *IEEE*, vol. 38, no. 7, pp. 1040–1059, 1993.
[4] M. Fabian and A. Hellgren, "PLC-based implementation of supervisory control for discrete event systems," in *37th Decision and Control*, Tampa, FL, USA, 1998.
[5] X.-R. Cao, G. Cohen, A. Giua, W. M. Wonham, and J. H. van Schuppen, "Unity in diversity, diversity in unity: Retrospective and prospective views on control of discrete event systems," *Discrete Event Dynamic Systems*, vol. 12, pp. 253–264, 2002.
[6] D. Harel, "Statecharts: A visual formalism for complex systems," *Science of Computer Programming*, vol. 8, pp. 231–274, 1987.
[7] Y.-L. Chen and F. Lin, "Modeling of discrete event systems using finite state machines with parameters," in *CCA00*, Anchorage, Alaska, Sept. 2000.
[8] Y. Yang and P. Gohari, "Embedded supervisory control of discrete-event systems," in *2005*, Edmonton, Canada, August 2005, pp. 410–415.
[9] B. Gaudin and P. H. Deussen, "Supervisory control on concurrent discrete event systems with variables (extended version)," Technical University, Berlin, Tech. Rep., 2006. [Online]. Available: http://www.benoit.gaudin1.free.fr
[10] C. de Oliveira, J. Cury, and C. Kaestner, "Synthesis of supervisors for parameterized and infinity non-regular discrete event systems," in *Proceedings of the 1st IFAC Workshop on Dependable Control of Discrete Systems (DCDS'07)*, June 2007, pp. 77–82.
[11] C. Ma and W. Wonham, "Nonblocking supervisory control of state tree structures," *IEEE*, vol. 51, no. 5, pp. 782–793, May 2006.
[12] Supremica, "`www.supremica.org`. The official website for the Supremica project," 2007.

[13] K. Åkesson, M. Fabian, H. Flordal, and R. Malik, "Supremica—an integrated environment for verification, synthesis and simulation of discrete event systems," in *8th Discrete Event Systems, WODES '06*, Ann Arbor, MI, USA, July 2006, pp. 384–385.

[14] K. Åkesson and M. Sköldstam, "Towards a framework for integrated supervisory and logic control," in *1st Dependable Control of Discrete Event Systems'07*, Paris, France, 2007, pp. 83–88.

[15] P. Malik and R. Malik, "Modular control-loop detection," in *8th Discrete Event Systems, WODES '06*, Ann Arbor, MI, USA, July 2006, pp. 119–124.

[16] C. A. R. Hoare, *Communicating sequential processes*, ser. Series in Computer Science. Prentice-Hall, 1985.

[17] J. M. Spivey, *The Z Notation: A Reference Manual*, 2nd ed. Prentice-Hall, 1992.

[18] R. Malik and R. Mühlfeld, "A case study in verification of uml statecharts: the profisafe protocol," *Universal Computer Science*, vol. 9, no. 2, pp. 138–151, Feb. 2003.

[19] H. Flordal, "Compositional approaches in supervisory control—with application to automatic generation of robot interlocking policies," Ph.D. dissertation, Signals and Systems, Chalmers, Göteborg, Sweden, Oct. 2006.

[20] H. Flordal and R. Malik, "Supervision equivalence," in *8th Discrete Event Systems, WODES '06*, Ann Arbor, MI, USA, July 2006, pp. 155–160.