BDD-based supervisory control on extended finite automata

2011 IEEE International Conference on
Automation Science and Engineering
Trieste, Italy - August 24-27, 2011

BDD-based Supervisory Control on Extended Finite Automata

S. Miremadi, B. Lennartson and K. Åkesson
Department of Signals and Systems, Chalmers University of Technology
SE-412 96 Gothenburg, Sweden
{miremads, bengt.lennartson, knut}@chalmers.se

Abstract— **In this paper, we settle some problems that are encountered when modeling and synthesizing complex industrial systems by the supervisory control theory. First, modeling such huge systems with explicit state-transition models typically results in an intractable model. An alternative modeling approach is to use extended finite automata (EFAs), which is an augmentation of ordinary automata with variables. The main advantage of utilizing EFAs for modeling is that more compact models are obtained. The second problem concerns the ease to understand and implement the supervisor. To handle this problem, we represent the supervisor in a modular manner by extending the original EFAs by compact conditional expressions generated from the monolithic supervisor. In order to, potentially, be able to handle complex systems efficiently, the models are symbolically represented by binary decision diagrams (BDDs). All computations that are performed in this framework are based on BDD operations. The framework has been implemented in a supervisory control tool and applied to industrially relevant benchmark problems.**

Index Terms— **Supervisory control theory, extended finite automata, supervisor representation, symbolic representation, binary decision diagrams.**

I. Introduction

When designing control functions for discrete event systems, a model-based approach may be used to conveniently understand the system's behavior. It is also possible to easily apply different modifications to models and decrease the testing and debugging time. A well known example of such a model-based approach is supervisory control theory (SCT) [1]. Having a plant (the system to be controlled) and a specification, SCT automatically synthesizes a control function, called *supervisor*, that restricts the conduct of the plant to ensure that the system never violates the given specification. SCT has various applications in different areas such as automated manufacturing and embedded systems, e.g. [2], [3].

Generally, a supervisor is a function that, given a set of events, restricts the plant to execute some events so that the specification is satisfied. A typical issue is how to compute such a control function efficiently and represent it lucidly for the users. A standard approach is to model the system by finite automata, synthesize the supervisor, and then explicitly represent all the states that are allowed to be reached in the closed-loop system.

However, regarding systems of industrially interesting sizes, the standard approach has some drawbacks:

- Modeling complex systems with ordinary automata can make the model large and intractable.
- Exploring all reachable states in the closed-loop system explicitly is computationally expensive, in terms of both time and memory, due to the state-space explosion problem.
- The monolithic supervisor for such systems, typically, consists of a huge number of states, which makes it difficult for the user to understand it thoroughly. In addition, representing the supervisor as a single automaton will require more memory than available on the hardware.

Various researchers have settled these issues, yet no work has considered all three topics together.

One way to obtain compact models is to use variables. The variables can then appear in *guards* and *actions*. Guard expressions at the transitions restrict the behavior of the system, while actions update the variables. Naturally, physical signals that are stored in memories or sent between controllers can be modeled as global variables, e.g., sensors, actuators and buffers.

In [4], a framework called *extended finite automata (EFAs)* was presented, which is an augmentation of an ordinary automaton extended with variables, guard expressions and action functions. The guards and action functions are attached to the transitions, which admits local design techniques of systems consisting of many different parts. The main feature of EFAs is that they are suitable for the SCT framework.

In [5], the authors present an approach to compute the optimal nonblocking supervisor based on a number of EFAs. The principle is that based on an EFA plant and a set of forbidden locations, iteratively strengthen the guards of the plant so that forbidden or blocking states become unreachable in the controlled plant. For problems with a huge number of EFAs, the approach can suffer from an early state-space explosion while generating the plant with the forbidden locations. In addition, the focus is not to obtain comprehensible guards for the users.

Although extended frameworks allow compact representations of huge state-spaces, when it comes to analysis the number of states will not be affected and could potentially cause state-space explosion problem that typically occurs when the behavior of interacting sub-systems is studied. A well-known approach to handle this problem is to symbolically represent the state-space and transitions using binary decision diagrams (BDDs) [6]; powerful data structures for representing Boolean functions. Several researchers have tackled the state-space explosion problem in the context of

This work was carried out at the Wingquist Laboratory VINN Excellence Centre within the Area of Advance - Production at Chalmers, supported by the Swedish Governmental Agency for Innovation Systems (VINNOVA). The support is gratefully acknowledged.

978-1-4577-1732-1/11/$26.00 ©2011 IEEE

SCT using BDDs such as [7], [8], however, most of them are based on state transition systems without the introduction of variables.

The contribution of this paper is the development of a framework, where both the plant/specification and the supervisor are modeled by EFAs. In addition, we show how EFAs and their nontrivial full synchronous composition operator by BDDs including proof of correctness. The framework has been applied to a set of industrially relevant benchmark problems and showing that the results can be obtained efficiently.

Our approach has some advantages from different perspectives. By modeling a system based on EFAs, a compact representation of complex systems with huge state-space can potentially be obtained. Another advantage is that the system is symbolically represented using BDDs, and all the computations are based on BDD operations, making it possible to handle large systems and overcome the state-space explosion problem in many cases. Representing the supervisor by EFAs in a modular manner, in contrast to a monolithic manner, also makes it more comprehensible and tractable for the users. In addition, typically, a modular supervisor consumes less memory in a controller. The reason is that the synchronization will be performed online in the controller (see [9], [10]) which can alleviate the problem of exponential growth of the number of states in the synchronization. Furthermore, since EFAs include guards and actions, they are often easier to interpret than purely event based ordinary modular automata. They can also easily be converted to controller programming languages e.g. SFC or ladder diagrams. EFAs can also easily be converted to well-known verification tools such as NuSMV [11]. Also, from an engineering perspective, EFAs are attractive models due to their similarity to UML and state diagrams.

II. Preliminaries

This section provides some preliminaries that are used throughout this paper.

A. Extended Finite Automata

An EFA, introduced in [4], is an augmentation of the ordinary finite automaton (FA) with guard predicates and action functions. The guard predicates and actions are associated to the transitions of the automaton. A transition in an EFA is enabled if and only if its corresponding guard predicate is evaluated to `true`, and when a transition is taken, updating actions of a set of variables may follow. Guard predicates can be realized by their characteristic functions.

Definition II.1 (Characteristic Function)**.** Let W be a finite set so that $W \subseteq U$, where U is the finite universal set. A *characteristic function* $\chi_W : U \to \mathbb{B}$ is defined by:

$$\chi_W(a) = \begin{cases} 1 & \text{iff } a \in W \\ 0 & \text{iff } a \notin W \end{cases}. \tag{1}$$

Since the set U is finite, say with size n, in practice its elements are represented with numbers in $\mathbb{Z}_n$ or binary m-tuples in $\mathbb{B}^m$ ($m = \lceil \log_2^n \rceil$). For binary characteristic functions, an injective function $\theta : U \to \mathbb{B}^m$ is used to map the elements in U to elements in $\mathbb{B}^m$. In general, $\chi_W(a)$ is constructed as

$$\chi_W(a) = \bigvee_{w \in W} a \leftrightarrow \theta(w), \tag{2}$$

where $\leftrightarrow$ on two m-tuples v_1 and v_2 is defined as

$$v_1 \leftrightarrow v_2 \triangleq \bigwedge_{0 \le i < m} (v_1^i \leftrightarrow v_2^i). \tag{3}$$

v^i denotes the i:th element in the binary m-tuple v.

As we will see later, characteristic functions can also be used to represent BDDs.

Definition II.2 (Extended Finite Automaton)**.**
An extended finite-state automaton E is a 6-tuple

$$E = \langle L^E \times V, \Sigma^E, \mathcal{G}, \mathcal{A}, \rightarrow, (\ell_0^E, v_0) \rangle,$$

where:

- (i) $L^E \times V$ is the extended finite set of states, denoted by Q, where L^E is a set of *locations* and V is the domain of definition of the variables;
- (ii) Σ^E is a nonempty finite set of events;
- (iii) $\mathcal{G} = \{\chi_W \mid W \in 2^V\}$ is the set of guard predicates over V;
- (iv) $\mathcal{A} = \{a \mid a : V \to V\}$ is a collection of action functions;
- (v) $\rightarrow \subseteq L^E \times \Sigma \times \mathcal{G} \times \mathcal{A} \times L^E$ is the transition relation;
- (vi) $(\ell_0^E, v_0) \in L^E \times V$ is the initial state.

The finite set $V = V^1 \times ... \times V^n$ is the domain of definition of an n-tuple of variables $v = (v^1, \ldots, v^n)$ with initial values $v_0 = (v_0^1, \ldots, v_0^n) \in V$. A *guard* $g(v)$ is a predicate over the variables that relate each element of V to either 1 (`true`) or 0 (`false`). Actions are written as

$$\acute{v} \quad := \quad a(v) = (a^1(v), \ldots, a^n(v)), \text{where } \acute{v} \in V.$$

The symbol ξ is used to denote implicit actions that do not update the value of variables. For instance, if $a^i(v) = \xi$, it means that action a^i does not update variable v^i, i.e. $\acute{v}^i = v^i$.

A partial transition relation is written as $\ell \xrightarrow{\sigma}_{g/a} \acute{\ell}$, where $\ell, \acute{\ell} \in L$, $\sigma \in \Sigma$, $g \in \mathcal{G}$ and $a \in \mathcal{A}$. If g is absent, denoted by $\ell \xrightarrow{\sigma}_{a} \acute{\ell}$, it is assumed that g always evaluates to `true`. If a is absent, denoted by $\ell \xrightarrow{\sigma}_{g} \acute{\ell}$, it is assumed that $a(v) = \Xi$, where Ξ is the vector notation for $(\xi, \xi, \ldots, \xi)$, indicating that no variable is updated during the transition.

For convenience, the states (locations and variable values) can explicitly be written out in system transitions according to the following definition.

Definition II.3 (Explicit State Transition Relation)**.**
Let $E = \langle L^E \times V, \Sigma^E, \mapsto, (\ell_0^E, v_0) \rangle$ be an EFA. The explicit state transition relation of E is defined as

$$\begin{aligned} \mapsto \quad \triangleq \quad & \{(\ell^E, v, \sigma, \acute{\ell}^E, \acute{v}) \in L^E \times V \times \Sigma \times L^E \times V \mid \\ & \exists \ell^E \xrightarrow{\sigma}_{g/a} \acute{\ell}^E : v \in \mathsf{SAT}\mathcal{G}(g) \wedge (v, \acute{v}) \in \mathsf{SAT}\mathcal{A}(a)\}, \end{aligned}$$

where v and $\acute{v}$ are the values of the variables before and after executing the transition, respectively; $\mathsf{SAT}\mathcal{G}$ denotes the set of variable assignments that satisfies the guard $g(v)$,

$$\mathsf{SAT}\mathcal{G}(g) \triangleq \{v \in V \mid v \vDash g\}; \tag{4}$$

and $\mathsf{SAT}\mathcal{A}$ denotes the following set:

$$\mathsf{SAT}\mathcal{A}(a) \triangleq \{(v, \acute{v}) \in V \times V |\ \acute{v} = a(v)\}. \tag{5}$$

Note that a special case of $\acute{v} = a(v)$ is when $\acute{v} = v$, that is $a(v) = \Xi$. The explicit state transition relation is written $(\ell, v) \overset{\sigma}{\mapsto} (\acute{\ell}, \acute{v})$ and can recursively be extended to strings in Σ^*.

We denote the explicit representation of a partial transition $\ell \overset{\sigma}{\rightarrow}_{g/a} \acute{\ell}$ by $\mapsto_{\ell \overset{\sigma}{\rightarrow}_{g/a} \acute{\ell}}$.

For an EFA E, we write $\Gamma^E(\ell^E, v)$ to denote all the events that are defined from a state $(\ell^E, v) \in L^E \times V$. Formally,

$$\Gamma^E(\ell^E, v) = \{\sigma \in \Sigma^E \mid \exists (\acute{\ell}^E, \acute{v}) \Rightarrow (\ell^E, v) \overset{\sigma}{\mapsto} (\acute{\ell}^E, \acute{v})\}.$$

Definition II.4 (Deterministic EFA)**.**
An EFA $E = \langle L^E \times V, \Sigma, \mapsto, (\ell_0^E, v_0)\rangle$ is deterministic if $(\ell^E, v) \overset{\sigma}{\mapsto} (\acute{\ell}^E, \acute{v})$ and $(\ell^E, v) \overset{\sigma}{\mapsto} (\grave{\ell}^E, \grave{v})$ always implies $(\acute{\ell}^E, \acute{v}) = (\grave{\ell}^E, \grave{v})$.

Since we are interested in deterministic systems, we merely focus on deterministic EFAs. In the sequel, for the sake of brevity, we simply write EFAs for deterministic EFAs.

The composition of two EFAs is defined by the *extended full synchronous composition (EFSC)*.

Definition II.5 (Extended Full Synchronous Composition)**.**
Let $E_k = \langle L^{E_k} \times V, \Sigma^{E_k}, \rightarrow_{E_k}, (\ell_0^{E_k}, v_0)\rangle$, $k = 1, 2$, be two EFAs using the shared variables $v = (v^1, \ldots, v^n)$. The Extended Full Synchronous Composition (EFSC) of E_1 and E_2 is

$$E_1 \| E_2 = \langle L^{E_1} \times L^{E_2} \times V, \Sigma^{E_1} \cup \Sigma^{E_2}, \rightarrow, (\ell_0^{E_1}, \ell_0^{E_2}, v_0)\rangle$$

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

1) $(\ell^{E_1}, \ell^{E_2}) \overset{\sigma}{\rightarrow}_{g/a} (\acute{\ell}^{E_1}, \acute{\ell}^{E_2})$, $\sigma \in \Sigma_1 \cap \Sigma_2$ if
   $\exists\ \ell^{E_1} \overset{\sigma}{\rightarrow}_{g_1/a_1} \acute{\ell}^{E_1} \in \rightarrow_{E_1}$ and
   $\exists\ \ell^{E_2} \overset{\sigma}{\rightarrow}_{g_2/a_2} \acute{\ell}^{E_2} \in \rightarrow_{E_2}$ such that:
   (i) $g = g_1 \wedge g_2$,
   (ii) For $i = 1, \ldots, n$ and $\forall v \in V$:

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

2) $(\ell^{E_1}, \ell^{E_2}) \overset{\sigma}{\rightarrow}_{g/a} (\acute{\ell}^{E_1}, \acute{\ell}^{E_2}), \sigma \in \Sigma_1 \backslash \Sigma_2$ if
   $(\ell^{E_1}, \sigma, g, a, \acute{\ell}^{E_1}) \in \rightarrow_{E_1}$ and $\ell^{E_2} = \acute{\ell}^{E_2}$;

3) $(\ell^{E_1}, \ell^{E_2}) \overset{\sigma}{\rightarrow}_{g/a} (\acute{\ell}^{E_1}, \acute{\ell}^{E_2}), \sigma \in \Sigma_2 \backslash \Sigma_1$ if
   $(\ell^{E_2}, \sigma, g, a, \acute{\ell}^{E_2}) \in \rightarrow_{E_2}$ and $\ell^{E_1} = \acute{\ell}^{E_1}$.

The EFSC operator is both commutative and associative. Note that, in the case where the action functions of E_1 and E_2 explicitly try to update a shared variable to different values, we assume that the variable is not updated. It can indeed be discussed whether the transition should be executed. In that case, the definition of EFSC need to be more modified compared to FSC, which is not desired. In addition, a situation where two values are conflicting, is usually a consequence of bad modeling, and thus it is more reasonable to inform the user by a message rather than disabling the transition. For more details about EFAs, refer to [4] including the procedure of converting an EFA model to an FA model.

III. Supervisory Control Theory

Supervisory Control Theory (SCT) [1], [12] is a general theory to automatically synthesize supervisors based on a given plant and specification. A specification describes the allowed and inhibited behaviors. A *supervisor* restricts the conduct of the plant to guarantee that the system never violates the given specification. In SCT, some states of an automaton E, which is typically a specification, are considered as *marked states*. These are the states that are desired to be reached from the initial state. The set of marked states of a composed automaton $E_1 \| E_2$ is the cartesian product of the corresponding sets of marked states. In addition, some states can be specified as *explicitly forbidden*, Q_{ex}^E,which are states that should not be reached from the initial state. The set of forbidden states of a composed automaton $E_1 \| E_2$ is $Q_{ex}^{E_1} \times Q^{E_2} \cup Q^{E_1} \times Q_{ex}^{E_2}$. In SCT, the events are divided into two disjoint subsets: *controllable events*, denoted by Σ_c, that can be prevented from executing by the supervisor; and *uncontrollable events*, denoted by Σ_{uc}, which cannot be influenced by the supervisor [1], [12]. A plant P can be described by the synchronization of a number of sub-plants $P = P_1 \| P_2 \| \ldots \| P_l$, and similarly for a specification $Sp = Sp_1 \| Sp_2 \| \ldots \| Sp_m$. In our computations, we assume that a supervisor S always refines the plant, i.e. $S = S||P$. A first candidate of the supervisor is the composed automaton $P \| Sp$, which we refer to as S_0 in the sequel. After the synthesis procedure, some states are identified as *blocking* or *uncontrollable*, referred to as *forbidden states*, which should be excluded from S_0 in order to obtain the supervisor. The states that belong to the supervisor are called *safe states*, denoted by Q_{sup}. For and EFA E, blocking states are states where no marked states are reachable. For a supervisor candidate $\hat{S}$ that is a sub-automaton of S_0, the set of uncontrollable states are the reachable states in $P \| \hat{S}$ for which an uncontrollable event is defined for the plant P but not for the supervisor $\hat{S}$. The safe states can be computed by fixed point iterations [7]. For a more formal and detailed explanation of supervisory synthesis, see [1], [10], [12]. For large systems the number of states can grow exponentially. To this end, we use BDDs to represent the EFAs and perform different operations. BDDs can improve the efficiency of set and Boolean operations performed on the state sets dramatically [7], [13], [14]. The corresponding BDD for a finite set $W \subseteq U$ (U is the universal set), can be represented using the characteristic function χ presented in Equation (1). In the sequel, we will use characteristic functions to represent BDDs.

IV. Symbolic Computation of S_0

This section describes how $S_0 = P \| Sp$ can be symbolically represented.

There are basically two approaches for computing $\chi_{\mapsto_{S_0}}$:

1) Transforming the EFAs to FAs and then applying the synthesis procedure.

2) Applying the synthesis procedure directly on the EFAs without transforming them to FAs.

In the former case, the EFAs are initially transformed to FAs based on an algorithm explained in [4]. $\chi_{\mapsto_{S_0}}$ can then be computed based on the FAs. A drawback of this approach is that the number of transitions often grows very rapidly when transforming EFAs to FAs, incurring an inefficient performance.

To overcome the above-mentioned obstacle we settle on the second approach, that is showing how $\chi_{\mapsto_{S_0}}$ can be computed without transforming EFAs to FAs.

A. BDD representation of an EFA

The characteristic function of the transition function of an EFA can be computed based on Definition II.3. Two different sets of boolean (BDD) variables are used to represent the current values of different locations and variables, denoted by b^L and b^{V^i}, respectively. Since we have to differ between the boolean variables used to represent current and updated values, $\acute{b}^L$ and $\acute{b}^{V^i}$ are used to represent the updated values. b^Σ denotes the boolean variables used to represent the alphabet.

Proposition IV.1. *The characteristic function of an explicit partial transition $\mapsto_{\ell \xrightarrow{\sigma}_{g/a} \acute{\ell}}$ is:*

$$\chi_{\mapsto_{\ell \xrightarrow{\sigma}_{g/a} \ell}} = \Big(\bigvee_{(v,\acute{v}) \in \mathsf{SAT}\mathcal{A}(a) | v \in \mathsf{SAT}\mathcal{G}(g)} \bigwedge_{i=1}^{n} b^{V^i} \leftrightarrow \theta(v^i) \ \wedge \ \acute{b}^{V^i} \leftrightarrow \theta(\acute{v}^i) \Big) \wedge b^L \leftrightarrow \theta(\ell) \ \wedge \ \acute{b}^L \leftrightarrow \theta(\acute{\ell}) \ \wedge \ b^\Sigma \leftrightarrow \theta(\sigma).$$

For brevity, we write $\chi_{\ell \xrightarrow{\sigma}_{g/a} \acute{\ell}}$ rather than $\chi_{\mapsto_{\ell \xrightarrow{\sigma}_{g/a} \acute{\ell}}}$. We represent integers in the two's complement system as an array of BDDs [15]. In our framework, we assume that overflows on variables are not allowed and thus we omit the cases where an overflow occurs. This is performed by removing all the variable assignments that result in values outside the domain of the variables. Consequently, the characteristic function of the transition relation of an EFA E will be

$$\chi_{\mapsto_E} = \bigvee_{\ell \xrightarrow{\sigma}_{g/a} \acute{\ell} \in \rightarrow_E} \chi_{\ell \xrightarrow{\sigma}_{g/a} \acute{\ell}} \ \wedge \ \bigwedge_{i=1}^{n} \chi_{V^i}(b^{V^i}) \wedge \bigwedge_{i=1}^{n} \chi_{V^i}(\acute{b}^{V^i}). \quad (6)$$

B. BDD representation of EFSC on EFAs

Based on Definition II.5 for the extended full synchronous composition, we compute $\chi_{\mapsto_{S_0}}$ in three steps:

1) Compute a characteristic function, representing $\mapsto_{S_0}$ without including the actions, denoted by $\chi'_{\mapsto_{S_0}}$.
2) Compute a characteristic function, representing the update of the EFA variables, denoted by $\chi_{\mapsto^v_{S_0}}$.
3) Based on $\chi'_{\mapsto_{S_0}}$ and $\chi_{\mapsto^v_{S_0}}$, compute $\chi_{\mapsto_{S_0}}$.

Since S_0 is the synchronization of a number of sub-plants and sub-specifications in form of EFAs, in all of the following computations we focus on $N \geq 2$ EFAs $E_1, \ldots, E_N$.

Note that the result will be incorrect if steps 1 and 2 are carried out in a single step. For deterministic FAs without variables, this is not the case. For N FAs $A_1, \ldots, A_N$, we have $\chi_{\mapsto_{A_1 \| \ldots \| A_N}} = \bigwedge_{k=1}^{N} \chi_{\mapsto_{A_k}}$. This comes from the fact that the full synchronous operator corresponds to 'intersection' on languages, and 'intersection' corresponds to the AND operator on characteristic functions. For $N \geq 2$ EFAs $E_1, \ldots, E_N$,

$$\chi_{\mapsto_{E_1 \| \ldots \| E_N}} \neq \bigwedge_{k=1}^{N} \chi_{\mapsto_{E_k}}.$$

Because then it would not be possible to keep track of the variables that are not updated (don't-care updates). Furthermore, the action conflicts will disable the corresponding events. However, based on Definition II.5, the result should be a transition where the variables will be remained unchanged.

When computing the synchronous composition based on the characteristic functions, we have to assume that the EFAs have the same alphabet. To make this possible we extend the transition relations of each EFA by adding self-loops with events that are not in the alphabet of the EFA.

Definition IV.1 (Extended explicit transition relation, $\looparrowright_{E_k}$)**.** For $N \geq 2$ EFAs $E_1, \ldots, E_N$, the extended explicit transition relation of E_k, denoted by $\looparrowright_{E_k}$, represents the explicit transition relation of E_k together with self-loops on all states with events that are not in the alphabet of E_k

$$\looparrowright_{E_k} \triangleq \mapsto_{E_k} \cup \{(\ell, v, \sigma, \acute{\ell}, \acute{v}) \mid \forall \ell \in L^{E_k}, \forall v, \acute{v} \in V : \sigma \in (\Sigma^{E_1 \| \ldots \| E_N} \setminus \Sigma^{E_k}) \ \wedge \ \ell = \acute{\ell}\}.$$

By this extension, all EFAs in the model will have the same alphabet and thus the definition of extended full synchronous composition (Definition II.5) will be simplified to case 1 that only considers common events.

Proposition IV.2. *Let $E_1, \ldots, E_N$ be $N \geq 2$ EFAs. Then,*

$$\chi'_{\looparrowright_{E_1 \| \ldots \| E_N}} = \bigwedge_{k=1}^{N} \chi'_{\looparrowright_{E_k}}.$$

At this stage, we are done with step 1 in the procedure of computing $\mapsto_{E_1 \| \ldots \| E_N}$. The next step is to compute a characteristic function that represents the updating of EFA variables. First, we have to compute a characteristic function that represents all partial transitions that include the resulting action function of synchronizing N EFAs based on Definition II.5. In the following computations, we start to focus on a single variable v^i and then extend it to all variables in the model, i.e., v. Hence, for each EFA E_k and each variable v^i in the model, it is necessary to compute the transitions in E_k on which the variable v^i is updated.

Definition IV.2 (Updated transition relation, $\looparrowright_{v^i,E}$)**.** For an EFA E and a variable v^i, the updated transition relation for variable v^i, denoted by $\looparrowright_{v^i,E}$, represents the

set of partial transitions in E on which the variable v^i is updated:

$$\looparrowright_{v^i,E} \triangleq \{(\ell, v, \sigma, \acute{\ell}, \acute{v}) \mid \forall(\ell, v, \sigma, \acute{\ell}, \acute{v}) \in \looparrowright_E \wedge \acute{v}^i \neq v^i\}.$$

Remark. In a deterministic EFA, the combination of source-location, event, guard and target-location will uniquely define a transition.

Recall that, from Definition II.5, the result of $a^i(v)$ can be divided into four if-then constructs, which we denote by C_j. Each C_j consists of an `if` part, denoted by I_j, and a `then` part, denoted by T_j.

Lemma IV.3. *Let v^i be an arbitrary variable of an n-tuple v, and for $k \geq 2$ EFAs $E_1, \ldots, E_k$, let $\looparrowright_{v^i,k}$ be defined as follows:*

$$\chi_{\looparrowright_{v^i,k}} := \bigvee_{j=1}^{4} (\widehat{\chi}^{I_j}_{v^i,k} \wedge \widehat{\chi}^{T_j}_{v^i,k}),$$

where

$$\widehat{\chi}^{I_1}_{v^i,k} := \widehat{\chi}^{T_1}_{v^i,k} = \chi_{\looparrowright_{v^i,k-1}} \wedge \chi_{\looparrowright_{v^i,E_k}};$$
$$\widehat{\chi}^{I_2}_{v^i,k} := \chi'_{\looparrowright_{v^i,k-1}} \wedge \neg\chi'_{\looparrowright_{v^i,E_k}} \quad , \quad \widehat{\chi}^{T_2}_{v^i,k} := \chi_{\looparrowright_{v^i,k-1}};$$
$$\widehat{\chi}^{I_3}_{v^i,k} := \neg\chi'_{\looparrowright_{v^i,k-1}} \wedge \chi'_{\looparrowright_{v^i,E_k}} \quad , \quad \widehat{\chi}^{T_3}_{v^i,k} := \chi_{\looparrowright_{v^i,E_k}};$$
$$\widehat{\chi}^{I_4}_{v^i,k} := \neg(\widehat{\chi}^{I_1}_{v^i,k} \vee \widehat{\chi}^{I_2}_{v^i,k} \vee \widehat{\chi}^{I_3}_{v^i,k}) \quad , \quad \widehat{\chi}^{T_4}_{v^i,k} := \chi_{\mathsf{SAT}\mathcal{A}(\acute{v}^i = v^i)};$$

and

$$\chi'_{\looparrowright_{v^i,k}} := \chi'_{\looparrowright_{v^i,k-1}} \vee \chi'_{\looparrowright_{v^i,E_k}};$$
$$\chi_{\looparrowright_{v^i,1}} := \chi_{\looparrowright_{v^i,E_1}};$$
$$\chi'_{\looparrowright_{v^i,1}} := \chi'_{\looparrowright_{v^i,E_1}}.$$

Then, the following statement holds:

$$\bigwedge_{i=1}^{n} \chi_{\looparrowright_{v^i,k}} = \chi_{\looparrowright_{E_1 \| \ldots \| E_k}} \vee \psi,$$

where $\psi \wedge \chi'_{\looparrowright_{E_1 \| \ldots \| E_k}} \models$ `false`.

For the proof, see [16].

Theorem IV.4. *For $N \geq 2$ EFAs $E_1, \ldots, E_N$, and an n-tuple of variables $v = (v^1, \ldots, v^n)$, the following statement holds:*

$$\chi_{\looparrowright_{E_1 \| \ldots \| E_N}} = \bigwedge_{i=1}^{n} \chi_{\looparrowright_{v^i,N}} \wedge \chi'_{\looparrowright_{E_1 \| \ldots \| E_N}}.$$

Proof.

$$\begin{aligned}
&\bigwedge_{i=1}^{n} \chi_{\looparrowright_{v^i,N}} \wedge \chi'_{\looparrowright_{E_1 \| \ldots \| E_N}} = \\
&\quad (\chi_{\looparrowright_{E_1 \| \ldots \| E_k}} \vee \psi) \wedge \chi'_{\looparrowright_{E_1 \| \ldots \| E_N}} \\
&\quad (\chi_{\looparrowright_{E_1 \| \ldots \| E_k}} \wedge \chi'_{\looparrowright_{E_1 \| \ldots \| E_N}}) \vee (\psi \wedge \chi'_{\looparrowright_{E_1 \| \ldots \| E_N}}) \\
&\quad (\chi_{\looparrowright_{E_1 \| \ldots \| E_k}} \wedge \chi'_{\looparrowright_{E_1 \| \ldots \| E_N}}) \vee \texttt{false} \\
&\quad = \chi_{\looparrowright_{E_1 \| \ldots \| E_N}}.
\end{aligned}$$

□

Consequently, for a plant P and a specification Sp, $\chi_{\looparrowright_{P \| Sp}}$, i.e., $\chi_{\looparrowright_{S_0}}$, can be computed based on Lemma IV.4. Furthermore, since $\chi_{\looparrowright_{S_0}}$ and $\chi_{\mapsto_{S_0}}$ have the same alphabet, $\chi_{\looparrowright_{S_0}}$ and $\chi_{\mapsto_{S_0}}$ are equal.

V. Representation of the Supervisor as EFAs

The last step is to compute the supervisor represented as EFAs. This computation is performed in three steps:

1) Compute a BDD representing the safe states, i.e., the corresponding BDD for $\chi_{Q_{sup}}$.
2) Transform the computed BDD to guard expressions.
3) Attach the guards to the original EFAs.

$\chi_{Q_{sup}}$ is computed by fixed point computations based on the synthesis algorithm described in [7]. Note that for a set of EFAs, the reachability algorithms performed on $\chi_{\mapsto_{S_0}}$ do not differ from the algorithms used for FAs. The algorithm requires four arguments: $\chi_{\{q_0^{S_0}\}}$, $\chi_{\mapsto_{S_0}}$, χ_{Q_x} and $\chi_{\mapsto^{uc}_{S_0}}$. Q_x is the union of the explicitly forbidden states and the initially uncontrollable states, described in in Section III. In the last argument, $\mapsto^{uc}_{S_0}$ denotes the transitions in S_0 that include uncontrollable events. $\chi_{\mapsto^{uc}_{S_0}}$ can be computed as follows:

$$\chi_{\mapsto^{uc}_{S_0}} = \chi_{\mapsto_{S_0}} \wedge \chi_{\Sigma_{uc}}.$$

In stage 2, based on Q_{sup}, we create two sets of states [17]:

- $Q^{\sigma}_{\mathbf{a}}$: The set of states in the supervisor where the execution of σ is defined for the supervisor.
- $Q^{\sigma}_{\mathbf{f}}$: The set of states in the supervisor where the execution of σ is defined for S_0, but not for the supervisor.

By utilizing $Q^{\sigma}_{\mathbf{a}}$ and $Q^{\sigma}_{\mathbf{f}}$ a guard expression $\mathcal{G}^{\sigma}(\langle q^{E_1}, q^{E_2}, \ldots, q^{E_n}\rangle)$ is generated for each controllable event $\sigma \in \Sigma_c^{S_0}$:

$$\mathcal{G}^{\sigma}(\langle q^{E_1}, q^{E_2}, \ldots, q^{E_n}\rangle) = \begin{cases} \text{true} & (\langle q^{E_1}, q^{E_2}, \ldots, q^{E_n}\rangle) \in Q^{\sigma}_{\mathbf{a}} \\ \text{false} & (\langle q^{E_1}, q^{E_2}, \ldots, q^{E_n}\rangle) \in Q^{\sigma}_{\mathbf{f}} \\ \text{don't} - \text{care} & \text{otherwise} \end{cases}$$

where q^{E_i} represents the current state of EFA E_i. $\mathcal{G}^{\sigma}(\langle q^{E_1}, q^{E_2}, \ldots, q^{E_n}\rangle)$ evaluates to true if σ is allowed to be executed from the state $\langle q^{E_1}, q^{E_2}, \ldots, q^{E_n}\rangle$. The size of a guard $\mathcal{G}$, denoted by $|\mathcal{G}|$, is defined by the number of atomic equality and nonequality terms in the guard expression.

A. Guard Generation

The guards are computed in three consequent steps. First, the corresponding BDDs for the state sets are computed. Next, the BDDs are converted to their corresponding *integer decision diagrams* (IDDs) [18], which will be used to generate the guards in the last step. An IDD is an extension of a BDD where the number of terminals is arbitrary and the domain of the variables in the graph is an arbitrary set of integers. For our purpose, we use an IDD with two terminals, 0-terminal and 1-terminal.

To represent a state $\langle q^{E_1}, q^{E_2}, \ldots, q^{E_n}\rangle$ in the closed-loop automaton $E_1 \parallel \ldots \parallel E_n$, each IDD-variable is associated to an EFA E_i that has Q^{E_i} as its domain. This domain can be mapped to an integer that is represented as an IDD. In other

words, each outgoing edge from node E_i represents a state in E_i. Hence the maximum number of edges from a node E_i is $|Q_i^E|$. As for BDDs the number of edges and nodes for an IDD can also be reduced. For simplicity, we use the names of the states on the IDD-edges rather than integers in the sequel.

Using IDDs to generate guards has some advantages in comparison to BDDs: 1) they make it easier to handle and manipulate propositional formulae; 2) they exploit some of the common subexpressions in a guard yielding a more factorized and smaller formula; 3) they depict a more understandable model of the state set, since the nodes and edges represent names of the EFAs and states, respectively.The procedure of converting a BDD to an IDD is presented in [17].

The result is correct under the assumption that the BDD has a fixed variable ordering. A pseudo-algorithm of this process has been presented in [17].

The last step of obtaining the guard is to convert the IDDs to propositional formulae. For a given IDD, a top-down depth first search is used to traverse the graph and generate its corresponding propositional formula. The algorithm starts from the root and visits the nodes whilst generating the expression and ends at the 1-terminal.

For each node in the IDD, the corresponding expressions of the edges belonging to the same level (the children of that node) are logically disjuncted and if the edges belong to different levels they are logically conjuncted. Hence, the propositional formula for the IDD in Fig. 1 is

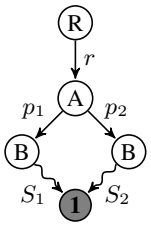

Fig. 1: Recursive representation of an IDD.

$$r \wedge ((p_1 \wedge S_1) \vee (p_2 \wedge S_2)),$$

where p_i is the corresponding expression of the edge that lead to one of A's children and S_i is the corresponding expression from the node to the 1-terminal, that is recursively computed. A pseudo-algorithm of this process has been presented in [17].

B. Guard Attachment

Since $q^{E_i} \in L^{E_i} \times V$, the generated guard will be a combination of $\ell^{E_i} = \ell_i^{E_i}$ (or $\ell^{E_i} \neq \ell_i^{E_i}$) and $v^i = v_j^i$ (or $v^i \neq v_j^i$) expressions. Each variable ℓ^{E_i} holds the current location of EFA E_i. However, since they are not defined in the model, they should be declared and added to the set of variables in the model. Thus, the variable v is extended to $v^+ = (v^1, \ldots, v^n, \ell^{E_1}, \ldots, \ell^{E_N})$. Hence, the transition function of each automaton E_i is extended as follows:

$$\begin{aligned}\rightarrow_{E_i}^+ = \{&\ell^{E_i} \xrightarrow{\sigma}_{g/a^+} \acute{\ell}^{E_i} \mid \forall \ell^{E_i} \xrightarrow{\sigma}_{g/a} \acute{\ell}^{E_i} \in \rightarrow_{E_i},\\ &a^+(v^+) = (a^1(v), \ldots, a^n(v), \ell^{E_1}, \ldots, \acute{\ell}^{E_i}, \ldots, \ell^{E_N}).\end{aligned}$$

Nevertheless, this extension can be performed implicitly so that it becomes transparent to the user. Finally, for each EFA E_i in the model, each generated guard $\mathcal{G}^\sigma$ is conjuncted with the guards in $\rightarrow_{E_i}^+$ that include event σ; forming a new EFA E_i^{sup} where

$$\begin{aligned}\rightarrow_{E_i^{sup}} = \{&\ell \xrightarrow{\sigma}_{g^+/a^+} \acute{\ell} \mid\\ &\forall \ell \xrightarrow{\sigma}_{g/a^+} \acute{\ell} \in \rightarrow_{E_i}^+, g^+ = g \wedge \mathcal{G}^\sigma\}.\end{aligned}$$

Consequently, the supervisor can be represented in a modular manner, deducing that $E_1^{sup} \parallel \ldots \parallel E_N^{sup}$ satisfies the specification without any forbidden states.

VI. Case Study

We have applied the presented framework to a set of industrial benchmark examples. The framework has been implemented and integrated in the supervisory control tool Supremica [19], [20], which uses *JavaBDD* [21] as the BDD package. The examples were conducted on a standard PC (Intel Core 2 Quad CPU @ 2.4 GHz and 3GB RAM) running Windows 7. In our implementation, the BDDs follow a fixed variable ordering based on the approach presented in [22].

The benchmark examples are: Resource Allocation System (RAS) [23], Collision Avoidance System (CAS) [24], Control Logic Development (CLD) [25], Automated Guided Vehicles (AGV) [26].

Table I shows the results of the reachability analysis. `SIZE` represents the number of EFAs and variables in the model. $|Q_T|$ is the number of theoretically reachable states in the model, which is equal to $\prod_{k=1}^{N} |L^{E_k}| \cdot \prod_{i=1}^{n} |V^i|$. $|Q_{reach}|$ represents the number of reachable states in the closed-loop model. The table also includes the time for computing the supervisor.

Table II shows the results of the guard generation process. $|Q_\otimes|$ is the number of forbidden states, equal to $|Q_{reach}| - |Q_{sup}|$. The number of controllable events, $|\Sigma_c|$, is equal to the number of generated guards. The table also includes the minimum, maximum and average sizes of the guards and the time for generating the guards. The table tries to give an overview of how much easier it will be for the user to tract the synthesis results. For instance, in the AGV model, 9 million states are prevented to be reached by introducing only 10 new guards with an average size of 17.6 terms. Furthermore, in the CAS model around 63% of the reachable states are prevented to be reached by 142 guards with an average size of 1.4 terms. Hence, it would be easier for the users to tract the synthesis results. It can be observed that with 1 second computation time, the algorithm works quite efficiently for these examples. We believe that it is possible to efficiently generate guards for much larger and more complicated examples, however, due to state-space explosion in the synthesis procedure we were not able to compute the supervisor for larger examples.

TABLE I: Reachability analysis.

Model	`SIZE`	$\lvert Q_T \rvert$	$\lvert Q_{reach} \rvert$	$\lvert Q_{sup} \rvert$	Time (s)
RAS	26	7.3×10^8	26750	21581	4
AGV	16	5.2×10^{10}	2.6×10^7	1.7×10^7	5
CLD	20	2.1×10^{12}	121	110	27
CAS	142	5.6×10^{67}	5.4×10^8	2×10^8	9

TABLE II: Tractability analysis.

Model	$\lvert Q_{\otimes}\rvert$	$\lvert\Sigma_c\rvert$	$\lvert\mathcal{G}\rvert$			Time (s)
			min	max	avg	
RAS	5169	20	1	178	31.5	1
AGV	9×10^6	10	4	44	17.6	1
CLD	11	3	1	4	2.3	1
CAS	3.4×10^8	142	1	28	1.4	1

VII. Conclusions

In this paper we presented an approach that, given a system modeled by EFAs, symbolically computes the supervisor. In particular, this approach provides a seamless framework for generating and modifying control functions that are modeled by EFAs. Specifically, after modeling a system with EFAs, the users can obtain the control function in form of the original EFAs extended with some additional guards. Hence, during the design phase, the users remain in the same model domain, i.e., EFAs. The main advantage of this approach is that the users can iteratively update both the models and the intermediate control functions. All the computations are performed by BDDs, which are transparent to the users, and the only interface the users deal with is the EFA framework.

The entire procedure was applied to a set of academic and industrial benchmark examples.

There are some possible directions for future work that are worth pursuing. As mentioned in Section VI, the BDD-based algorithms need to be complemented by partitioning techniques that are normally used for ordinary automata. Then, it would be possible to handle much larger and more complicated systems. In addition, there is a potential to improve the variable ordering of the BDDs. We also believe that the guards can be more reduced in some cases, which is a work in progress.

References

[1] P. Ramadge and W. Wonham, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

[2] L. Feng, W. M. Wonham, and P. S. Thiagarajan, "Designing communicating transaction processes by supervisory control theory," *Form. Methods Syst. Des.*, vol. 30, no. 2, pp. 117–141, 2007.

[3] K. Andersson, J. Richardsson, B. Lennartson, and M. Fabian, "Coordination of Operations by Relation Extraction for Manufacturing Cell Controllers," *IEEE Trans. on Control Systems Technology*, vol. 18, no. 2, pp. 414–429, 2010.

[4] M. Sköldstam, K. Åkesson, and M. Fabian, "Modeling of discrete event systems using finite automata with variables," *Decision and Control, 2007 46th IEEE Conference on*, pp. 3387–3392, 2007.

[5] L. Ouedraogo, R. Kumar, R. Malik, and K. Åkesson, "Symbolic approach to nonblocking and safe control of Extended Finite Automata," in *2010 IEEE International Conference on Automation Science and Engineering*. IEEE, Aug. 2010, pp. 471–476.

[6] S. B. Akers, "Binary Decision Diagrams," *IEEE Transactions on Computers*, vol. 27, pp. 509–516, Jun. 1978.

[7] A. Vahidi, M. Fabian, and B. Lennartson, "Efficient supervisory synthesis of large systems," *Control Engineering Practice*, vol. 14, no. 10, pp. 1157–1167, Oct. 2006.

[8] C. Ma and W. M. Wonham, "Nonblocking supervisory control of state tree structures," *IEEE Transactions on Automatic Control*, vol. 51, no. 5, pp. 782–793, May 2006.

[9] A. Hellgren, B. Lennartson, and M. Fabian, "Modelling and PLC-based implementation of modular supervisory control," in *Discrete Event Systems, 2002. Proceedings. Sixth International Workshop on*, 2002, pp. 371–376.

[10] K. Åkesson, "Methods and tools in supervisory control theory: operator aspects, computation efficiency and applications," Ph.D. dissertation, Signals and Systems,Chalmers University of Technology, Gothenburg, Sweden, 2002.

[11] A. Voronov and K. Åkesson, "Verification of Supervisory Control Properties of Finite Automata Extended with Variables," Department of Signals and Systems, Chalmers University of Technology, Tech. Rep., 2009.

[12] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. Springer, 2008.

[13] S. Miremadi, K. Åkesson, M. Fabian, A. Vahidi, and B. Lennartson, "Solving two supervisory control benchmark problems using Supremica," in *9th International Workshop on Discrete Event Systems, 2008. WODES 2008.*, May 2008, pp. 131–136.

[14] C. Ma and W. M. Wonham, "STSLib and its application to two benchmarks," in *9th International Workshop on Discrete Event Systems, 2008. WODES 2008.*, May 2008, pp. 119–124.

[15] E. M. Clarke, K. L. Mcmillan, X. Zhao, M. Fujita, and J. Yang, "Spectral Transforms for Large Boolean Functions withApplications to Technology Mapping," *Form. Methods Syst. Des.*, vol. 10, no. 2-3, pp. 137–148, 1997.

[16] S. Miremadi, B. Lennartson, and K. Åkesson, "A BDD-based Approach for Modeling Plant and Supervisor by Extended Finite Automata," 2011.

[17] S. Miremadi, K. Åkesson, and B. Lennartson, "Symbolic Computation of Reduced Guards in Supervisory Control," *Accepted for IEEE Transactions on Automation Science and Engineering*, 2011.

[18] J. Gunnarsson, "Symbolic Methods and Tools for Discrete Event Dynamic Systems," Ph.D. dissertation, Electrical Engineering, Linköping University, Linköping, Sweden, 1997.

[19] K. Å kesson, M. Fabian, H. Flordal, and R. Malik, "Supremica—an integrated environment for verification, synthesis and simulation of discrete event systems," in *Proceedings of the 8th international Workshop on Discrete Event Systems, WODES'08*, Ann Arbor, MI, USA, 2006, pp. 384–385.

[20] K. Å kesson, M. Fabian, H. Flordal, and A. Vahidi, "Supremica—a Tool for Verification and Synthesis of Discrete Event Supervisors," in *11th Mediterranean Conference on Control and Automation*, Rhodos, Greece, 2003.

[21] "JavaBDD." [Online]. Available: http://javabdd.sourceforge.net

[22] A. Aziz, S. Tasiran, and R. K. Brayton, "BDD variable ordering for interacting finite state machines," in *Proceedings of the 31st annual Design Automation Conference, DAC '94*. New York, NY, USA: ACM, 1994, pp. 283–288.

[23] A. Nazeem and S. Reveliotis, "A practical approach to the design of maximally permissive liveness-enforcing supervisors for complex resource allocation systems," in *6th IEEE Conference on Automation Science and Engineering (CASE)*, Toronto, Ontario, Canada, 2010, pp. 451–458. [Online]. Available: http://www.isye.gatech.edu/∼spyros/publications/CASE-2010.pdf

[24] M. R. Shoaei, B. Lennartson, and S. Miremadi, "Automatic generation of controllers for collision-free flexible manufacturing systems," in *6th IEEE International Conference on Automation Science and Engineering*, Aug. 2010, pp. 368–373.

[25] G. Čengić, O. Ljungkrantz, and K. Åkesson, "Formal Modeling of Function Block Applications Running in IEC 61499 Execution Runtime," in *Proceedings of the 11th IEEE International Conference on Emerging Technologies and Factory Automation*, Prague, Czech Republic, 2006.

[26] L. E. Holloway and B. H. Krogh, "Synthesis of Feedback Control Logic for a Class of Controlled Petri Nets," *IEEE Transactions on Automatic Control*, vol. 35, no. 5, pp. 514–523, 1990.