A BDD-Based Approach for Modeling Plant and Supervisor by Extended Finite Automata

Sajed Miremadi, Bengt Lennartson, *Member, IEEE*, and Knut Åkesson

***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. This will provide a framework for the users where they can both model their system and obtain the supervisor in form of EFAs. In order to 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*—Binary decision diagrams (BDDs), extended finite automata (EFA), supervisor representation, supervisory control theory (SCT), symbolic representation.**

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]–[4].

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 the following 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.

Many of the frameworks that allow compact representations of complex and large state-space systems are inspired by Statecharts [5]; an extension of ordinary automata with hierarchy, concurrency and communication using variables, guards and actions. However, Statecharts are not completely suitable for the SCT framework. "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" [6]. In addition, there is a causality between subsystems in Statecharts, which is not desired in the SCT framework. There exist a number of frameworks that are based on automata extended with variables such as [7]–[9]. In [7], it is assumed that a variable can be updated by at most one finite state machine, and in order to do synthesis the state-space needs to be extended by additional states. In [8], finite-state machines 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 (not transitions) of the model. In [9], to ensure a least restrictive supervisor it is assumed that all variables are local, i.e., not shared between automata.

In [6], an extended framework called *extended finite automata* (EFA) is presented that overcomes the above-mentioned restrictions making the framework suitable for SCT. An EFA is an augmentation of an ordinary automaton extended with variables, guard expressions and action functions. The guards

Manuscript received March 10, 2011; revised July 04, 2011; accepted July 26, 2011. Manuscript received in final form August 29, 2011. Recommended by Associate Editor A. Giua. This work was supported by the Swedish Governmental Agency for Innovation Systems (VINNOVA).

The authors are with the Department of Signals and Systems, Chalmers University of Technology, SE-412 96 Gothenburg, Sweden (e-mail: miremads@chalmers.se; bengt.lennartson@chalmers.se; knut@chalmers.se).

Digital Object Identifier 10.1109/TCST.2011.2167150

and action functions are attached to the transitions, which admits local design techniques of systems consisting of many different parts. It is also possible to update the variables, which are global, in different automata, and to use EFAs to model both plant and supervisor. In [10], 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) [11]; powerful data structures for representing Boolean functions. Several researchers have tackled the state-space explosion problem in the context of SCT using BDDs such as [12], [13], however, most of them are based on state transition systems without the introduction of variables.

Regarding the third problem, in [13] an approach is presented, which aims to obtain comprehensible control functions. In that article, the supervisor is represented as a set of control functions expressed by a set of simplified BDDs. They model the system by hierarchial models called state tree structures, which do not include variables and could yield complex models for some industrial applications. In addition, the synthesis result is represented as BDDs, which is not always easy to interpret by users.

In [14], an algorithm for reducing the state size of the supremal (minimally restrictive) supervisor is proposed. The algorithm determines how a control-equivalent monolithic supervisor with reduced number of states can be computed. Since the proposed algorithm manipulates the states explicitly, for huge systems with industrial complexity, e.g., 10^{20} states, the computations can be very costly, both in terms of time and memory. In addition, even though the supervisor can be decreased significantly in many cases, the control action of the resultant model could still be hard for the designers to render.

There also exists a number of papers dealing with synthesis on parameterized models on systems with infinite states, where the supervisor is represented as a set of logical expressions [15]–[17]. In these papers the focus is on the computational aspects of synthesis and no attention has been paid to the comprehensibility of the resultant supervisor. Besides, the nonblocking property has not been considered in [15] and [17]. Furthermore, since all the papers deal with infinite systems, there does not exist an exact and automatic computation of the supervisor.

This paper has the following three main contributions:

1) development of a framework, where both the plant/specification and the supervisor are modeled by EFAs;

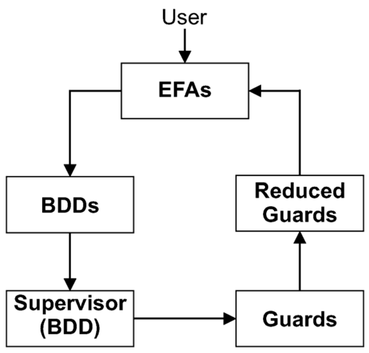

Fig. 1. Process overview of the approach.

2) representation of EFAs and their nontrivial full synchronous composition operator by BDDs including proof of correctness;
3) applying the framework to a set of industrially relevant benchmark problems and showing that the results can be obtained efficiently.

In particular, based on a plant and a specification modeled by EFAs, initially, the EFAs are transformed to BDDs and the corresponding BDD for the states of the monolithic supervisor is computed. Next, guard expressions are extracted from the generated BDD, which will be attached to the initial EFAs, forming a modular supervisor. Hence, the only difference between the initial and final EFAs is that the guards are extended in the latter model. Fig. 1 shows a process overview of the approach. The guard generation procedure, based on ordinary automata without variables, has been explained thoroughly in [18].

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 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 [19] and [20]) 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 [21]. Also, from an engineering perspective, EFAs are attractive models due to their similarity to UML and state diagrams.

This paper is organized as follows. Section II provides some preliminaries including extended finite automata and binary decision diagrams, which are the fundamental concepts in our work. Supervisory control theory and the synthesis algorithms are described in Section III. In Section IV, we show how the closed-loop model can be symbolically represented. Section V explains how the supervisor can be represented based on the initial EFA models. The whole process is applied to some case studies in Section VI. Finally, Section VII provides some conclusions and suggestions for future work.

II. Preliminaries

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

A. Extended Finite Automata

An EFA, introduced in [6], 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* $\chi_W : U \rightarrow \mathbb{B}$ 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 $\theta : U \rightarrow \mathbb{B}^m$ 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

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

v^i denotes the ith 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

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

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) $\mathcal{G} = \{\chi_W | W \in 2^V\}$ is the set of guard predicates over V;
(iv) $\mathcal{A} = \{a | a : V \rightarrow V\}$ is a collection of action functions;
(v) $\rightarrow \subseteq L^E \times \Sigma^E \times \mathcal{G} \times \mathcal{A} \times L^E$ is the transition relation;
(vi) $(\ell_0^E, v_0) \in L^E \times V$ is the initial state.

The finite set $V = V^1 \times \ldots \times V^n$ 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

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

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

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

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 $g(v)$

$$\mathsf{SAT}\mathcal{G}(g) \triangleq \{v \in V | v \models g\} \tag{4}$$

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

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

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 $\ell \xrightarrow{\sigma}_{g/a} \acute{\ell}$ by $\mapsto_{\ell \xrightarrow{\sigma}_{g/a} \acute{\ell}}$.

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

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

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 EFSC of E_1 and E_2 is

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

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

1) $(\ell^{E_1}, \ell^{E_2}) \xrightarrow{\sigma}_{g/a} (\acute{\ell}^{E_1}, \acute{\ell}^{E_2}), \sigma \in \Sigma_1 \cap \Sigma_2$ if $\exists \ell^{E_1} \xrightarrow{\sigma}_{g_1/a_1} \acute{\ell}^{E_1} \in \rightarrow_{E_1}$ and $\exists \ell^{E_2} \xrightarrow{\sigma}_{g_2/a_2} \acute{\ell}^{E_2} \in \rightarrow_{E_2}$ such that:
   a) $g = g_1 \wedge g_2$,
   b) 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) $(\ell^{E_1}, \ell^{E_2}) \xrightarrow{\sigma}_{g/a} (\acute{\ell}^{E_1}, \acute{\ell}^{E_2}), \sigma \in \Sigma_1 \setminus \Sigma_2$ if $(\ell^{E_1}, \sigma, g, a, \acute{\ell}^{E_1}) \in \rightarrow_{E_1}$ and $\ell^{E_2} = \acute{\ell}^{E_2}$;
3) $(\ell^{E_1}, \ell^{E_2}) \xrightarrow{\sigma}_{g/a} (\acute{\ell}^{E_1}, \acute{\ell}^{E_2}), \sigma \in \Sigma_2 \setminus \Sigma_1$ 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 [6] including the procedure of converting an EFA model to an FA model.

B. Binary Decision Diagrams

BDDs [11] are powerful data structures for representing Boolean functions. For large systems where the number of states grows exponentially, BDDs can improve the efficiency of set and Boolean operations performed on the state sets dramatically [12], [22]–[24].

Given a set of Boolean variables B, a BDD is a Boolean function $h : 2^B \rightarrow \{0, 1\}$, which can be expressed using Shannon's decomposition [25]

$$h = \left(\neg b_j \wedge h|_{b_j=0}\right) \vee \left(b_j \wedge h|_{b_j=1}\right) \quad b_j \in B$$

where $h|_{b_j=0}$ and $h|_{b_j=1}$ refer to assigning 0 and 1 to all occurrences of the Boolean variable b_j, respectively. A BDD is represented as a directed acyclic graph, which consists of two types of nodes: *decision nodes* and *terminal nodes*. A terminal node can either be *0-terminal* or *1-terminal*. Each decision node is labeled by a Boolean variable and has two edges to its *low-child* and *high-child*. The low- and high-child corresponds to the cases in the above equation where b_j is 0 (graphically represented by a *dotted* line) and 1 (graphically represented by a *solid* line), respectively. The *size* of a BDD refers to the number of decision nodes.

The power of BDDs lies in their simplicity and efficiency to perform binary operations. A binary operator $\mathtt{op}$ between two BDDs h and g can be computed as

$$h \ \mathtt{op}\ g = \left[\neg b_j \wedge \left(h|_{b_j=0} \ \mathtt{op}\ g|_{b_j=0}\right)\right] \vee \left[b_j \wedge \left(h|_{b_j=1} \ \mathtt{op}\ g|_{b_j=1}\right)\right].$$

If the operator is implemented based on dynamic programming, the time complexity of the algorithm will be $O(|h| \cdot |g|)$, where $|h|$ and $|g|$ are the sizes of the BDDs. A BDD operation that has been used extensively in our implementation is the *existential quantification* over a set of Boolean variables

$$\exists b.h = h|_{b_j=0} \vee h|_{b_j=1}.$$

The time complexity for quantification is exponential in the worst case. The implementation of the BDD operators has been discussed in more detail in [26].

The corresponding BDD for a finite set $W \subseteq U$ (U is the universal set), can be represented using the characteristic function χ in (1).

In a BDD graph, a variable b_1 has a lower (higher) *order* than variable b_2 if b_1 is closer (further) to the root and is denoted by $b_1 \prec b_2$ ($b_2 \prec b_1$). The variable ordering will impact the size of the BDD, however, finding an optimal variable ordering of a BDD is an NP-complete problem [27].

In our implementation, a BDD follows a fixed variable ordering based on the method presented in [28]. In this method, the variable ordering is influenced by the ordering of interacting automata based on weighted search in the process communication graph (PCG). A PCG for a set of automata is a weighted undirected graph, where the weight between two automata A_1 and A_2 is defined as $|\Sigma^{A_1} \cap \Sigma^{A_2}|$. In some cases, the ordering can be improved [12]. This is however beyond the scope of this paper.

For a more elaborate and verbose exposition of BDDs and the implementation of different operators, refer to [26] and [29].

III. Supervisory Control Theory

SCT [1], [30] 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*, Q_m^E. 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], [30].

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 $Q_{\sup}$.

Blocking states of an EFA E, are states where no marked states are reachable

$$Q_{bl} \triangleq \Big\{(\ell, v) \in Q^E_{\text{reach}} | \ \nexists s \in \Sigma^* \Rightarrow (\ell, v) \overset{s}{\mapsto}_E (\acute{\ell}, \acute{v}) \text{ and } (\acute{\ell}, \acute{v}) \in Q^E_m \Big\}.$$

For a supervisor candidate $\hat{S}$ that is a sub-automaton of S_0, the set of uncontrollable states are defined as

$$Q_{uc} \triangleq \Big\{(\ell^P, \ell^{\hat{S}}, v) \in Q^{P\|\hat{S}}_{\text{reach}} | \exists \sigma_{uc} \in \Sigma_{uc} \cap \Sigma^{\hat{S}} \Rightarrow (\ell^P, v) \overset{\sigma_{uc}}{\mapsto}_P (\acute{\ell}^P, \acute{v}) \text{ and } \sigma_{uc} \notin \Gamma^{\hat{S}}(\ell^{\hat{S}}, v) \Big\}$$

that is, 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 synthesized by fixed point computations [12]. There are two operators that are used frequently in the fixed point computations: $\texttt{IMAGE}$ and $\texttt{PRE} - \texttt{IMAGE}$. Given a set of states $W \subseteq Q$; $\texttt{IMAGE}(W, \mapsto)$ computes the set of states that can be reached in one transition

$$\texttt{IMAGE}(W, \mapsto) \triangleq \Big\{(\acute{\ell}, \acute{v}) \in Q | \exists (\ell, v) \in W, \sigma \in \Sigma \Rightarrow (\ell, v) \overset{\sigma}{\mapsto} (\acute{\ell}, \acute{v}) \Big\} \quad (6)$$

and $\texttt{PRE} - \texttt{IMAGE}(W, \mapsto)$ computes the set of states that in one transition can reach a state in W

$$\texttt{PRE} - \texttt{IMAGE}(W, \mapsto) \triangleq \Big\{(\ell, v) \in Q | \exists \sigma \in \Sigma \Rightarrow (\ell, v) \overset{\sigma}{\mapsto} (\acute{\ell}, \acute{v}) \text{ and } (\acute{\ell}, \acute{v}) \in W \Big\}. \quad (7)$$

The BDD-based implementations of these operators are described in Section IV. For a more formal and detailed explanation of supervisory synthesis, see [1], [20], and [30].

IV. Symbolic Computation of S_0

In the sequel, we will use the characteristic function χ, presented in (1), to represent a BDD. This section describes how $S_0 = P \| Sp$ can be symbolically represented. In particular, we explain how to compute $\chi_{\mapsto_{S_0}}$ and $\chi_{\{q_0^{S_0}\}}$, used as inputs to $\texttt{IMAGE}(\{q_0^{S_0}\}, \mapsto_{S_0})$ (and $\texttt{PRE} - \texttt{IMAGE}$) in the synthesis computations mentioned in Section III.

There are basically the following 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 [6]. $\chi_{\mapsto_{S_0}}$ 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. Regarding this approach, the following two main issues should be resolved:

1) how to represent the transition functions of EFAs by BDDs?
2) how to represent the full synchronous composition on EFAs by BDDs?

In the sequel, we elaborate these issues.

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

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

Proof: The proof follows immediately from (2), (3), and Definition II.3. ■

For brevity, we write $\chi_{\ell \xrightarrow{\sigma}_{g/a} \acute{\ell}}$ representing the characteristic function of $\mapsto_{\ell \xrightarrow{\sigma}_{g/a} \acute{\ell}}$. We denote the characteristic function where the Boolean variables $\acute{b}^{V^i}$ have been removed, i.e., $\chi_{\mapsto_{\ell \xrightarrow{\sigma}_{g/a} \acute{\ell}}}(b^{V^1}, \ldots, b^{V^n}, b^L, \acute{b}^L, b^\Sigma)$, by $\chi'_{\ell \xrightarrow{\sigma}_{g/a} \acute{\ell}}$. Hence, we have

$$\chi_{\ell \xrightarrow{\sigma}_{g/a} \acute{\ell}} \wedge \chi'_{\ell \xrightarrow{\sigma}_{g/a} \acute{\ell}} = \chi_{\ell \xrightarrow{\sigma}_{g/a} \acute{\ell}}.$$

We represent integers in the two's complement system as an array of BDDs [31]. 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

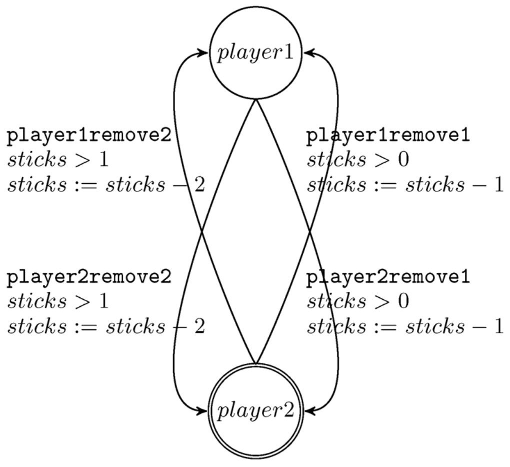

Fig. 2. EFA model for Example IV.1.

the variables. Consequently, the characteristic function of the transition relation of an EFA E will be

$$\chi_{\mapsto_E} = \bigvee_{\ell \xrightarrow{\sigma}_{g/a} \acute{\ell} \in \to_E} \chi_{\ell \xrightarrow{\sigma}_{g/a} \acute{\ell}} \wedge \bigwedge_{i=1}^{n} \chi_{V^i}\left(b^{V^i}\right) \wedge \bigwedge_{i=1}^{n} \chi_{V^i}\left(\acute{b}^{V^i}\right). \quad (8)$$

Based on $\chi_{\mapsto_E}$, different set of states can be extracted. For instance, let Q^{σ} denote the set of states where an event σ can be executed from. Then, χ_{Q^σ} can be computed as follows:

$$\begin{aligned} f &= \chi_{\mapsto_E} \wedge \chi_{\{\sigma\}} \\ h &= \exists \left(\acute{b}^L \cup \acute{b}^{V^i}\right).f \\ \chi_{Q^\sigma} &= \exists\, b^{\Sigma}.h. \end{aligned} \quad (9)$$

f represents all transitions that include event σ. By excluding the BDD-variables used for representing the target-states from f, h will be obtained. Finally, the BDD-variables used for representing the events are excluded, which will yield the states in E that enable σ.

The following example shows how the transition function of an EFA can be represented by a BDD.

Example IV.1: Consider a nim game with 5 sticks on a table, and two players that take turn by removing one or two sticks. The winner is the player that takes the last stick(s). Fig. 2 depicts the EFA model for this game.

Fig. 3 shows the corresponding transition function for the EFA shown in Fig. 2. Note that the BDD does not contain the cases where $sticks < 0$ and $sticks > 5$. The BDD variables in the figure are labeled with numbers as follows:

$$\begin{aligned} b^{\Sigma} &= \left(b_1^{\Sigma}, b_0^{\Sigma}\right) = (\text{'1'}, \text{'0'}) \\ b^L &= \left(b_0^L\right) = (\text{'2'}) \\ \acute{b}^L &= \left(\acute{b}_0^L\right) = (\text{'3'}) \\ b^{\text{sticks}} &= \left(b_3^{\text{sticks}}, b_2^{\text{sticks}}, b_1^{\text{sticks}}, b_0^{\text{sticks}}\right) \\ &= (\text{'7'}, \text{'6'}, \text{'5'}, \text{'4'}) \\ \acute{b}^{\text{sticks}} &= \left(\acute{b}_3^{\text{sticks}}, \acute{b}_2^{\text{sticks}}, \acute{b}_1^{sticks}, \acute{b}_0^{\text{sticks}}\right) \\ &= (\text{'1'}, \text{'10'}, \text{'9'}, \text{'8'}) \end{aligned}$$

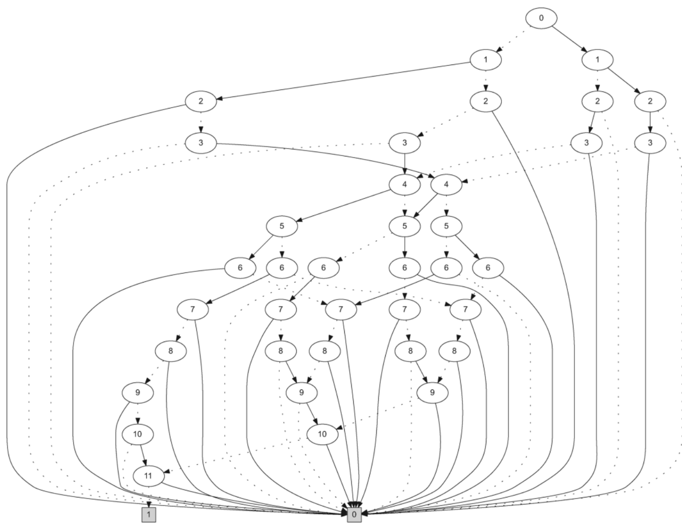

Fig. 3. Corresponding BDD for the transition function of the EFA in Fig. 2.

TABLE I
EVENT AND LOCATION ENCODING FOR THE EFA IN FIG. 2

Event	$b_1^{\Sigma} b_0^{\Sigma}$	Location	b_0^L
player1remove1	0 0	$player1$	0
player1remove2	0 1	$player2$	1
player2remove1	1 0		
player2remove2	1 1		

where b_0 is the least significant bit. Note that since the integers are represented in two's complement, four Boolean variables are used to represent $sticks$ because of the sign-bit. The location and event encoding is shown in Table I.

For instance, let's track the transition

$$(\texttt{player2},\ \texttt{player2remove2},\ \text{sticks} > 1,\ \text{sticks} := \text{sticks} - 2,\ \texttt{player1})$$

on the BDD in Fig. 3. Event player2remove2 is identified by starting from node "0", following the high-child to node "1" and following the high-child to node "2", i.e., $b_1^{\Sigma} \wedge b_0^{\Sigma}$. The location player2 is identified by following the high-child from node "2", i.e., b_0^L, and location player1 is identified by following the low-child from node "1", i.e., $\neg\acute{b}_0^L$. The guard and action are identified by all the paths from node "3" to node "11".

As it can be observed, the BDD in this example is larger than its corresponding EFA, however, for larger models the BDDs typically become much more compact.

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 the following 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 (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

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

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

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

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.

- I_1: $a_1^i(v) = a_2^i(v)$; both actions update the variable to the same value.
- T_1: $a^i(v) = a_1^i(v)$.
- I_2: $a_2^i(v) = \xi$; the first action updates the variable but not the second action.
- T_2: $a^i(v) = a_1^i(v)$.
- I_3: $a_1^i(v) = \xi$; the second action updates the variable but not the first action.
- T_3: $a^i(v) = a_2^i(v)$.
- I_4: $otherwise$; either none of the actions update the variable, or the actions update the variable but to different values.
- T_4: $a^i(v) = v^i$.

Definition IV. 3 (Interaction Transition Relation, $\overset{C_j}{\looparrowright}_{v^i,E_1\|E_2}$): For two EFAs E_1 and E_2 and a variable v^i, the interaction transition relation, denoted by $\overset{C_j}{\looparrowright}_{v^i,E_1\|E_2}$, represents a set of partial transitions that satisfies C_j in Definition II.5

$$\overset{C_1}{\looparrowright}_{v^i,E_1\|E_2} \triangleq \Big\{\left((\ell^{E_1}, \ell^{E_2}), v, \sigma, (\acute{\ell}^{E_1}, \acute{\ell}^{E_2}), \acute{v}\right) \mid \\ (\ell^{E_1}, v, \sigma, \acute{\ell}^{E_1}, \acute{v}) \in \looparrowright_{v^i,E_1} \\ \wedge (\ell^{E_2}, v, \sigma, \acute{\ell}^{E_2}, \acute{v}) \in \looparrowright_{v^i,E_2}\Big\} \tag{10}$$

$$\overset{C_2}{\looparrowright}_{v^i,E_1\|E_2} \triangleq \Big\{\left((\ell^{E_1}, \ell^{E_2}), v, \sigma, (\acute{\ell}^{E_1}, \acute{\ell}^{E_2}), \acute{v}\right) \mid \\ (\ell^{E_1}, v, \sigma, \acute{\ell}^{E_1}, \acute{v}) \in \looparrowright_{v^i,E_1} \\ \wedge (\ell^{E_2}, v, \sigma, \acute{\ell}^{E_2}, \acute{\mu}) \in \looparrowright_{E_2} \setminus \looparrowright_{v^i,E_2}\Big\} \tag{11}$$

$$\overset{C_3}{\looparrowright}_{v^i,E_1\|E_2} \triangleq \Big\{\left((\ell^{E_1}, \ell^{E_2}), v, \sigma, (\acute{\ell}^{E_1}, \acute{\ell}^{E_2}), \acute{v}\right) \mid \\ (\ell^{E_1}, v, \sigma, \acute{\ell}^{E_1}, \acute{\mu}) \in \looparrowright_{E_1} \setminus \looparrowright_{v^i,E_1} \\ \wedge (\ell^{E_2}, v, \sigma, \acute{\ell}^{E_2}, \acute{v}) \in \looparrowright_{v^i,E_2}\Big\} \tag{12}$$

$$\overset{C_4}{\looparrowright}_{v^i,E_1\|E_2} \triangleq \left\{ \left((\ell^{E_1}, \ell^{E_2}), v, \sigma, (\acute{\ell}^{E_1}, \acute{\ell}^{E_2}), \acute{v}\right) \mid \\ \left((\ell^{E_1}, \ell^{E_2}), v, \sigma, (\acute{\ell}^{E_1}, \acute{\ell}^{E_2}), \acute{v}\right) \notin \\ \bigcup_{j=1}^{3} \overset{C_j}{\looparrowright}_{v^i,E_1\|E_2} \right\}. \tag{13}$$

Hence, by definition we have

$$\chi_{\looparrowright_{v^i,E_1\|E_2}} = \bigvee_{j=1}^{4}\left(\chi_{\overset{C_j}{\looparrowright}_{v^i,E_1\|E_2}}\right). \tag{14}$$

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

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

where

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

and

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

Then, the following statement holds:

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

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

Proof: The lemma can be proved by induction.

(i) For the basic step, we check if the lemma holds for $k = 2$, where

$$\begin{aligned}
\chi_{\looparrowright_{v^i,2}} &= \bigvee_{j=1}^{4}\left(\widehat{\chi}^{I_j}_{v^i,2} \wedge \widehat{\chi}^{T_j}_{v^i,2}\right)\\
\widehat{\chi}^{I_1}_{v^i,2} &= \widehat{\chi}^{T_1}_{v^i,2} = \chi_{\looparrowright_{v^i,E_1}} \wedge \chi_{\looparrowright_{v^i,E_2}}\\
\widehat{\chi}^{I_2}_{v^i,2} &= \chi'_{\looparrowright_{v^i,E_1}} \wedge \neg\chi'_{\looparrowright_{v^i,E_2}}\\
\widehat{\chi}^{T_2}_{v^i,2} &= \chi_{\looparrowright_{v^i,E_1}}\\
\widehat{\chi}^{I_3}_{v^i,2} &= \neg\chi'_{\looparrowright_{v^i,E_1}} \wedge \chi'_{\looparrowright_{v^i,E_2}}\\
\widehat{\chi}^{T_3}_{v^i,2} &= \chi_{\looparrowright_{v^i,E_2}}\\
\widehat{\chi}^{I_4}_{v^i,2} &= \neg\left(\widehat{\chi}^{I_1}_{v^i,2} \vee \widehat{\chi}^{I_2}_{v^i,2} \vee \widehat{\chi}^{I_3}_{v^i,2}\right)\\
\widehat{\chi}^{T_4}_{v^i,2} &= \chi_{\mathsf{SAT}\mathcal{A}(\acute{v}^i=v^i)}.
\end{aligned}$$

We have to prove that

$$\bigwedge_{i=1}^{n} \chi_{\looparrowright_{v^i,2}} = \chi_{\looparrowright_{E_1\|E_2}} \vee \psi$$

where $\psi \wedge \chi'_{\looparrowright_{E_1\|E_2}} \models$ `false`. First, we compute $\widehat{\chi}^{I_j}_{v^i,2} \wedge \widehat{\chi}^{T_j}_{v^i,2}$ for $j = 1,\dots,4$. Based on Proposition IV.1, we have (15), shown at the bottom of the page. Now, lets compute $\widehat{\chi}^{I_2}_{v^i,2} \wedge \widehat{\chi}^{T_2}_{v^i,2}$. For $\widehat{\chi}^{I_2}_{v^i,2}$, we begin to compute each term separately, as shown in (16) and (17), shown at the bottom of the next page, where ω is a propositional formula including invalid partial transitions (some assignments do not have corresponding partial transitions). It can be observed that $\chi'_{\looparrowright_{v^i,E_1}} \wedge \neg\chi'_{\looparrowright_{v^i,E_2}}$ represents all the partial transitions, where v^i has been updated in EFA E_1 but not in E_2, together with many other valid and invalid partial transitions, i.e., ω. Furthermore, based on Definition II.5, $\widehat{\chi}^{T_2}_{v^i,2}$ should represent the partial transitions in EFA E_1, where variable v^i is

$$\begin{aligned}
\widehat{\chi}^{I_1}_{v^i,2} \wedge \widehat{\chi}^{T_1}_{v^i,2} &= \chi_{\looparrowright_{v^i,E_1}} \wedge \chi_{\looparrowright_{v^i,E_2}}\\
&= \left(\bigvee_{\left(\ell^{E_1},v,\sigma,\acute{\ell}^{E_1},\acute{v}\right)\in\looparrowright_{v^i,E_1}} \left(b^{V^i} \leftrightarrow \theta\left(v^i\right) \wedge \acute{b}^{V^i} \leftrightarrow \theta\left(\acute{v}^i\right) \wedge b^{L^{E_1}} \leftrightarrow \theta\left(\ell^{E_1}\right) \wedge \acute{b}^{L^{E_1}} \leftrightarrow \theta\left(\acute{\ell}^{E_1}\right) \wedge b^{\Sigma^{E_1\|E_2}} \leftrightarrow \theta\left(\sigma\right)\right)\right)\\
&\quad\wedge \left(\bigvee_{\left(\ell^{E_2},v,\sigma,\acute{\ell}^{E_2},\acute{v}\right)\in\looparrowright_{v^i,E_2}} \left(b^{V^i} \leftrightarrow \theta\left(v^i\right) \wedge \acute{b}^{V^i} \leftrightarrow \theta\left(\acute{v}^i\right) \wedge b^{L^{E_2}} \leftrightarrow \theta\left(\ell^{E_2}\right) \wedge \acute{b}^{L^{E_2}} \leftrightarrow \theta\left(\acute{\ell}^{E_2}\right) \wedge b^{\Sigma^{E_1\|E_2}} \leftrightarrow \theta\left(\sigma\right)\right)\right)\\
&= \bigvee_{\overset{C_1}{\looparrowright}_{v^i,E_1\|E_2}} \Big(b^{V^i} \leftrightarrow \theta\left(v^i\right) \wedge \acute{b}^{V^i} \leftrightarrow \theta\left(\acute{v}^i\right) \wedge b^{L^{E_1}} \leftrightarrow \theta\left(\ell^{E_1}\right) \wedge \acute{b}^{L^{E_1}} \leftrightarrow \theta\left(\acute{\ell}^{E_1}\right)\\
&\qquad \wedge b^{L^{E_2}} \leftrightarrow \theta(\ell^{E_2}) \wedge \acute{b}^{L^{E_2}} \leftrightarrow \theta(\acute{\ell}^{E_2}) \wedge b^{\Sigma^{E_1\|E_2}} \leftrightarrow \theta(\sigma)\Big)\\
&= \chi_{\overset{C_1}{\looparrowright}_{v^i,E_1\|E_2}}
\end{aligned} \tag{15}$$

updated, i.e., $\chi_{\hookrightarrow_{v^i,E_1}}$. Hence, we have (18), shown at the bottom of the page. By contradiction, we prove that

$$\chi'_{\overset{C_2}{\hookrightarrow}_{v^i,E_1\|E_2}} \wedge \alpha \models \texttt{false}. \tag{19}$$

Let's assume that

$$\chi'_{\overset{C_2\Delta}{\hookrightarrow}_{v^i,E_1\|E_2}} \wedge \alpha \not\models \texttt{false}.$$

Then, by assuming that $\alpha \not\models false$, there should exist at least one term in α that evaluates to `true` when being conjuncted with

$$\chi'_{\overset{C_2\Delta}{\hookrightarrow}_{v^i,E_1\|E_2}}$$

which is contradictory from (16) and (17). Note that if $\widehat{\chi}^{I_2}_{v^i,2} = \chi_{\hookrightarrow_{v^i,E_1}} \wedge \neg\chi'_{\hookrightarrow_{v^i,E_2}}$, the result for $\widehat{\chi}^{I_4}_{v^i,2}$ would not be correct.
Similarly, it can be shown that

$$\widehat{\chi}^{I_3}_{v^i,2} \wedge \widehat{\chi}^{T_3}_{v^i,2} = \chi_{\overset{C_3}{\hookrightarrow}_{v^i,E_1\|E_2}} \vee \beta \tag{20}$$

$$\widehat{\chi}^{I_4}_{v^i,2} \wedge \widehat{\chi}^{T_4}_{v^i,2} = \chi_{\overset{C_4}{\hookrightarrow}_{v^i,E_1\|E_2}} \vee \gamma \tag{21}$$

where

$$\chi'_{\overset{C_3}{\hookrightarrow}_{v^i,E_1\|E_2}} \wedge \beta \models \texttt{false} \tag{22}$$

$$\chi'_{\overset{C_4}{\hookrightarrow}_{v^i,E_1\|E_2}} \wedge \gamma \models \texttt{false}. \tag{23}$$

$$\chi'_{\hookrightarrow_{v^i,E_1}} = \bigvee_{(\ell^{E_1},v,\sigma,\acute{\ell}^{E_1},\acute{v})\in\hookrightarrow_{v^i,E_1}} \Big(b^{V^i}\leftrightarrow\theta(v^i)\wedge b^{L^{E_1}}\leftrightarrow\theta(\ell^{E_1})\wedge \acute{b}^{L^{E_1}}\leftrightarrow\theta(\acute{\ell}^{E_1})\wedge b^{\Sigma^{E_1\|E_2}}\leftrightarrow\theta(\sigma)\Big)$$

$$\begin{aligned}\neg\chi'_{\hookrightarrow_{v^i,E_2}} &= \neg\left(\bigvee_{(\ell^{E_2},v,\sigma,\acute{\ell}^{E_2},\acute{v})\in\hookrightarrow_{v^i,E_2}} \Big(b^{V^i}\leftrightarrow\theta(v^i)\wedge b^{L^{E_2}}\leftrightarrow\theta(\ell^{E_2})\wedge \acute{b}^{L^{E_2}}\leftrightarrow\theta(\acute{\ell}^{E_2})\wedge b^{\Sigma^{E_1\|E_2}}\leftrightarrow\theta(\sigma)\Big)\right)\\ &= \bigwedge_{(\ell^{E_2},v,\sigma,\acute{\ell}^{E_2},\acute{v})\in\hookrightarrow_{v^i,E_2}} \Big(\neg b^{V^i}\leftrightarrow\theta(v^i)\vee \neg b^{L^{E_2}}\leftrightarrow\theta(\ell^{E_2})\vee \neg\acute{b}^{L^{E_2}}\leftrightarrow\theta(\acute{\ell}^{E_2})\vee \neg b^{\Sigma^{E_1\|E_2}}\leftrightarrow\theta(\sigma)\Big) && (16)\\ &= \omega \vee \bigvee_{(\ell^{E_2},v,\sigma,\acute{\ell}^{E_2},\acute{v})\in\hookrightarrow_{E_2}\setminus\hookrightarrow_{v^i,E_2}} \Big(b^{V^i}\leftrightarrow\theta(v^i)\wedge b^{\Sigma^{E_1\|E_2}}\leftrightarrow\theta(\sigma)\wedge b^{L^{E_2}}\leftrightarrow\theta(\ell^{E_2})\wedge \acute{b}^{L^{E_2}}\leftrightarrow\theta(\acute{\ell}^{E_2})\Big) && (17)\end{aligned}$$

$$\begin{aligned}\widehat{\chi}^{I_2}_{v^i,2} \wedge \widehat{\chi}^{T_2}_{v^i,2} &= \chi'_{\hookrightarrow_{v^i,E_1}} \wedge \neg\chi'_{\hookrightarrow_{v^i,E_2}} \wedge \chi_{\hookrightarrow_{v^i,E_1}}\\ &= \chi_{\hookrightarrow_{v^i,E_1}} \wedge \neg\chi'_{\hookrightarrow_{v^i,E_2}}\\ &= \left(\bigvee_{(\ell^{E_1},v,\sigma,\acute{\ell}^{E_1},\acute{v})\in\hookrightarrow_{v^i,E_1}} \Big(b^{V^i}\leftrightarrow\theta(v^i)\wedge \acute{b}^{V^i}\leftrightarrow\theta(\acute{v}^i)\wedge b^{L^{E_1}}\leftrightarrow\theta(\ell^{E_1})\wedge \acute{b}^{L^{E_1}}\leftrightarrow\theta(\acute{\ell}^{E_1})\wedge b^{\Sigma^{E_1\|E_2}}\leftrightarrow\theta(\sigma)\Big)\right)\\ &\quad\wedge\left(\omega \vee \bigvee_{(\ell^{E_2},v,\sigma,\acute{\ell}^{E_2},\acute{v})\in\hookrightarrow_{E_2}\setminus\hookrightarrow_{v^i,E_2}} \Big(b^{V^i}\leftrightarrow\theta(v^i)\wedge b^{\Sigma^{E_1\|E_2}}\leftrightarrow\theta(\sigma)\wedge b^{L^{E_2}}\leftrightarrow\theta(\ell^{E_2})\wedge \acute{b}^{L^{E_2}}\leftrightarrow\theta(\acute{\ell}^{E_2})\Big)\right)\\ &= \bigvee_{\overset{C_2}{\hookrightarrow}_{v^i,E_1\|E_2}} \Big(b^{V^i}\leftrightarrow\theta(v^i)\wedge \acute{b}^{V^i}\leftrightarrow\theta(\acute{v}^i)\wedge b^{L^{E_1}}\leftrightarrow\theta(\ell^{E_1})\wedge \acute{b}^{L^{E_1}}\leftrightarrow\theta(\acute{\ell}^{E_1})\\ &\qquad\qquad \wedge b^{L^{E_2}}\leftrightarrow\theta(\ell^{E_2})\wedge \acute{b}^{L^{E_2}}\leftrightarrow\theta(\acute{\ell}^{E_2})\wedge b^{\Sigma^{E_1\|E_2}}\leftrightarrow\theta(\sigma)\Big)\vee\alpha\\ &= \chi_{\overset{C_2}{\hookrightarrow}_{v^i,E_1\|E_2}} \vee \alpha. && (18)\end{aligned}$$

If we logically disjunct equations (15), (18), (20), and (21), we get

$$
\begin{aligned}
(15) \vee (18) \vee (20) \vee (21) &= \bigvee_{j=1}^{4} \widehat{\chi}_{v^i,2}^{I_j} \wedge \widehat{\chi}_{v^i,2}^{T_j} \\
&= \bigvee_{j=1}^{4} \chi_{C_{j_{\looparrowright_{v^i,E_1\|E_2}}}} \vee \alpha \vee \beta \vee \gamma \\
&= \bigvee_{j=1}^{4} \chi_{C_{j_{\looparrowright_{v^i,E_1\|E_2}}}} \vee \psi_i .
\end{aligned}
$$

Hence

$$
\begin{aligned}
\chi_{\looparrowright_{v^i,2}} &= \bigvee_{j=1}^{4} \chi_{C_{j_{\looparrowright_{v^i,E_1\|E_2}}}} \vee \psi_i \\
&= \chi_{\looparrowright_{v^i,E_1\|E_2}} \vee \psi_i
\end{aligned}
$$

where the last equation is deduced from (14). Based on (19), (22), and (23), it is straightforward that $\psi_i \wedge \chi'_{\looparrowright_{E_1\|E_2}} \models \texttt{false}$. Since the above equation holds for all variables in v, we will have n such equations. If we logically conjunct these n equations, we will get

$$
\begin{aligned}
\bigwedge_{i=1}^{n} \chi_{\looparrowright_{v^i,2}} &= \bigwedge_{i=1}^{n} \left(\chi_{\looparrowright_{v^i,E_1\|E_2}} \vee \psi_i \right) \\
&= \chi_{\looparrowright_{E_1\|E_2}} \vee \psi .
\end{aligned}
$$

Since each variable v^i is represented by unique Boolean variables, i.e., b^{V^i} and $\acute{b}^{V^i}$, it can be deduced that $\psi \wedge \chi'_{\looparrowright_{E_1\|E_2}} \models \texttt{false}$. This proves the correctness of the basic step.

(ii) For the inductive step we assume that

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

We have to show that

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

Let $\widetilde{E} = E_1\|\ldots\|E_k$. Hence, we need to prove

$$
\bigwedge_{i=1}^{n} \chi_{\looparrowright_{v^i,2}} = \chi_{\looparrowright_{\widetilde{E}\|E_{k+1}}} \vee \acute{\psi}
$$

which becomes the basic step that is already proven to be correct. Notice that we utilized the associativity and commutativity properties of the logical binary operators and the EFSC operator.

Consequently, the lemma is proved by induction. ■

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:

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

■

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 the following three steps:

1) compute a BDD representing the safe states, i.e., the corresponding BDD for $\chi_{Q_{\text{sup}}}$;
2) transform the computed BDD to guard expressions;
3) attach the guards to the original EFAs.

$\chi_{Q_{\text{sup}}}$ is computed by fixed point computations based on the synthesis algorithm described in [12]. 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 $\chi_{\mapsto_{S_0}^{uc}}$. Q_x is the union of the explicitly forbidden states and the initially uncontrollable states, described in Section III. In the last argument, $\mapsto_{S_0}^{uc}$ denotes the transitions in S_0 that include uncontrollable events. $\chi_{\mapsto_{S_0}^{uc}}$ can be computed as follows:

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

In stage 2, based on Q_{sup}, we create the following two sets of states [18].

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

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

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

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.

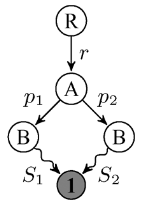

Fig. 4. Recursive representation of an IDD.

A. Guard Generation

The guards are computed in the following consequent steps:

- compute the corresponding BDDs for $Q^\sigma_{\mathbf{a}}$ and $Q^\sigma_{\mathbf{f}}$;
- convert the BDDs to integer decision diagrams;
- convert the integer decision diagrams to guards.

First, the corresponding BDDs for the state sets are computed similar to χ_{Q^σ} in (9). Next, the BDDs are converted to their corresponding *integer decision diagrams* (IDDs) [32], which will be used to generate the guards in the last step. An IDD is an extension to 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 $\langle q^{A_1}, q^{A_2}, \ldots, q^{A_n}\rangle$ in the closed-loop automaton $A_1\|\ldots\|A_n$, each IDD-variable is associated to an automaton A_i that has Q^{A_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 A_i represents a state in A_i. Hence the maximum number of edges from a node A_i is $|Q^A_i|$. 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 automata and states, respectively. On the other side different manipulations can be carried out more efficiently on BDDs compared to IDDs. The procedure of converting a BDD to an IDD is presented in [18].

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. 4 is

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

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 is presented in [18].

B. Guard Attachment

Since $q^{E_i} \in L^{E_i} \times V$, the generated guard will be a combination of $\ell^{E_i} = \ell^{E_i}_\imath$ (or $\ell^{E_i} \neq \ell^{E_i}_\imath$) and $v^i = v^i_\jmath$ (or $v^i \neq v^i_\jmath$) 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:

$$\rightarrow^+_{E_i} = \Big\{\ell^{E_i} \xrightarrow{\sigma}_{g/a^+} \acute{\ell}^{E_i} | \forall \ell^{E_i} \xrightarrow{\sigma}_{g/a} \acute{\ell}^{E_i} \in \rightarrow_{E_i},$$
$$a^+(v^+) = \Big(a^1(v), \ldots, a^n(v), \ell^{E_1}, \ldots, \acute{\ell}^{E_i}, \ldots, \ell^{E_N}\Big)\Big\}.$$

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^{sup}_i, where

$$\rightarrow_{E^{\text{sup}}_i} = \Big\{\ell \xrightarrow{\sigma}_{g^+/a^+} \acute{\ell} | \forall \ell \xrightarrow{\sigma}_{g/a^+} \acute{\ell} \in \rightarrow^+_{E_i}, g^+ = g \wedge \mathcal{G}^\sigma\Big\}.$$

Consequently, the supervisor can be represented in a modular manner, deducing that $E^{\text{sup}}_1\|\ldots\|E^{\text{sup}}_N$ satisfies the specification without any forbidden states.

Example IV.1: Recall the nim game in Example IV.1. The controllable events are `player1remove1` and `player1remove2`. After converting the model and performing the fixed point computations, we get 10 reachable states. The nonblocking and controllable supervisor consists of 5 states. After the guard generation procedure, the following guards are obtained:

$$\mathcal{G}^{\texttt{player1remove1}} : \text{sticks} = 1,$$
$$\mathcal{G}^{\texttt{player1remove2}} : \texttt{true}.$$

This means that event `player1remove2` is allowed to be executed everywhere in the closed-loop model. On the other hand, 5 forbidden states are prevented to be reached by adding $sticks = 1$ to the guard in the transition of the EFA that has `player1remove1` as its event.

VI. Case Studies

In this section we will show the results of applying our framework to a set of academic and industrial benchmark examples.

A. Model Classification

In order to be able to compare different models, we can classify them from two perspectives: *computational* and *tractability*. In the sequel, we assume that a model is given as a set of N EFAs $E_1, \ldots, E_N$ and n variables $v^1, \ldots, v^n$. Based on this model a supervisor is computed and represented as guards added to the original EFAs.

1) Computational Measures: By introducing a number of measures we try to get a clue on how hard the reachability analysis is. One way is to analyze the interaction between the automata by looking at their common events. Originally, the idea comes from [28] by introducing a simple communication map for interacting ordinary automata, i.e., not EFAs, called process communication graph (PCG). Each node in a PCG corresponds to an automaton. Each edge between two nodes is weighted, where the weight is equal to the number of common events between the two automata. The *dependency set* of an automaton A_i, denoted by $D(A_i)$, is A_i itself plus the set of automata directly connected to A_i. Hence, for each automaton it is possible to calculate a measure that shows how dependent it is to the rest of the automata. For an automaton A_i, we call this measure $LD(A_i)$ and compute it is as follows:

$$LD(A_i) = |D(A_i)| / \texttt{SIZE}$$

where $|D(A_i)|$ is the cardinality of the set $D(A_i)$ and SIZE is the number of automata in model. Since this measure is defined on ordinary automata, we have to convert our EFAs to DFAs based on [6].

The measures that will be used for computational purposes related to synthesis are as follows.

- SIZE: Number of EFAs and variables in the model.[1]
- $|Q_T|$: Number of theoretically reachable states in the model, which is equal to $\prod_{k=1}^{N} |L^{E_k}| \cdot \prod_{i=1}^{n} |V^i|$.
- LD: The minimum, maximum, and average LD among the automata.
- $|Q_{\text{reach}}|$: The number of reachable states in the closed-loop model, i.e., S_0.
- $|Q_{\text{sup}}|$: The number of states in the supervisor.

The number of EFAs and the number of reachable states are the most common measures used in SCT. However, it has been shown that there is a stronger connection between LD and the hardness of a problem compared to SIZE and Q_T [33]. In particular, the higher LD is, the harder it will be to perform reachability analysis.

2) Tractability Measures: In this part we use some measures to decide the compactness of a guard and how much advantageous it will we for the users to deal with guards on modular supervisors rather than a monolithic supervisor. Hence, for these measures we assume that the supervisor has been computed. The considered measures are as follows.

- $|Q_{ex}|$: The number of forbidden states, which is equal to $|Q_{\text{reach}}| - |Q_{\text{sup}}|$.
- $|\Sigma_c|$: The number of controllable events in the model, which is equal to the number of generated guards.
- $|\mathcal{G}|$: The minimum, maximum, and average size of the guards.

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. For instance, the size of the guard expression

$$(a = 2 \wedge b = 3) \vee c \neq 0$$

is 3. Often smaller guards are more tractable and understandable for the users.

[1]This is equal to the number of automata in the flattened model.

B. Benchmark Examples

We will first present the examples that are going to be analyzed.

1) Resource Allocation System (RAS): In [34], a RAS representing a flexibly automated robotic cell has been presented. The system includes seven resource types $\{R_1, \ldots, R_7\}$ with capacities $C_1 = C_2 = C_3 = 1$ and $C_4 = C_5 = C_6 = C_7 = 2$. It also has four product types, $\{\Pi_1, \Pi_2, \Pi_3, \Pi_4\}$ showing in which order the resources will be used

$$\begin{aligned}
\Pi_1 &: R_1 \rightarrow R_6 \rightarrow R_2 \rightarrow R_7 \rightarrow R_3 \\
\Pi_2 &: R_1 \rightarrow R_4 \rightarrow R_2 \rightarrow R_5 \rightarrow R_3 \\
\Pi_3 &: R_2 \rightarrow R_5 \rightarrow R_2 \\
\Pi_4 &: R_3 \rightarrow R_7 \rightarrow R_2 \rightarrow R_6 \rightarrow R_1.
\end{aligned}$$

It is possible to have multiple instances of a product. The control problem is to get the maximum number of instances without having a blocking system.

2) Fabrication Production System (FBS): This benchmark concerns a fabrication production system that can build different products [35]. There exist two product families, each one consisting of two product types. There also exist two fixture types that can be moved to be used in different parts of the system. The goal is to combine the product types from different families to get a new product and it is possible to have several instances of a product type. This system can be modeled by EFAs, and the restrictions are that some combinations are not allowed. Furthermore, the fixtures should be available at certain stations when needed. The maximum number of instances of a product type, in the system, may be given with a variable. The control problem is to have a blocking free system.

3) Collision Avoidance System (CAS): A robot cell with five robots, a fixture, a conveyor and an automated guided vehicle (AGV) is analyzed, see Fig. 5. There are two stations in the cell. At the first station there are two robots close to the conveyor moving two different parts. The robots will pick up the parts and weld them together. In the second station there are two other robots. The fifth robot will move the parts between different stations and put the final product on the AGV that will move it out from the cell. We model this system by EFAs and based on [36] we extend the guards to ensure a collision-free system. The final EFA model will be the input to the framework presented in this paper to generate a nonblocking supervisor.

4) Control Logic Development (CLD): A working cell that contains a fixture for holding a workpiece and an automatic carriage is considered. The workpiece is processed in the fixture. After processing the carriage is brought to the fixture as the clamp is opened. When the carriage is in place the workpiece is pushed out and falls into the carriage. The carriage transports the workpiece to a buffer. The cell is shown in Fig. 6. In [37], an EFA model for its execution model in the IEC 61499 standard has been generated and used to develop the control logic.

5) Automated Guided Vehicles (AGVs): A model of a flexible manufacturing cell was introduced in [38]. The cell consists of three workstations, two input stations and one output station and five AGVs, each one responsible to route some parts through the cell by following certain paths. The control problem is that the

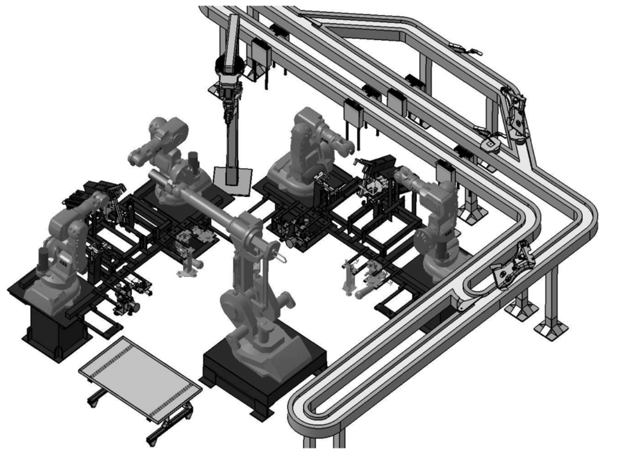

Fig. 5. CAS.

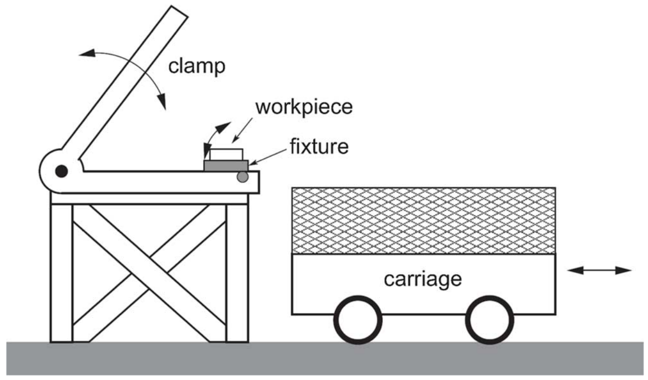

Fig. 6. CLD.

TABLE II
COMPUTATIONAL MEASURES

Model	SIZE	$\|Q_T\|$	LD		
			min	max	avg
RAS	26	7.3×10^8	0.17	0.74	0.29
AGV	16	5.2×10^{10}	0.13	0.38	0.22
FBS	25	9.8×10^{10}	0.11	0.89	0.46
CLD	20	2.1×10^{12}	0.2	0.5	0.32
CAS	142	5.6×10^{67}	0.02	0.25	0.07

routes intersect or are very close to each other and thus there are zones in which no two AGVs are allowed to be at the same time.

C. Results

The framework described in this paper has been implemented and integrated in the supervisory control tool Supremica [39], [40], which uses *JavaBDD* [41] as the BDD package. The examples were conducted on a standard PC (Intel Core 2 Quad CPU @ 2.4 GHz and 3 GB RAM) running Windows 7.

Table II shows the computational measures. The LD values are computed for each automaton, then the minimum, maximum and average values are calculated for the whole model.

Table III shows the results of the reachability analysis. The table includes the number of reachable states in the closed-loop model, the number of states in the supervisor and the time for computing the supervisor. From Table III it can be observed that it takes more time for the models CLD and CAS to compute the supervisor. This can be deduced from Table II. The LD values for the CLD model are higher than the other models (except the FBS model), which means that more iterations might be needed to reach a fixed point. The CAS model has the largest state-space which can make the BDD computations more time consuming.

TABLE III
REACHABILITY ANALYSIS

Model	$\|Q_{reach}\|$	$\|Q_{sup}\|$	Time (s)
RAS	26750	21581	4
AGV	2.6×10^7	1.7×10^7	5
FBS	72846	33966	1
CLD	121	110	27
CAS	5.4×10^8	2×10^8	9

TABLE IV
TRACTABILITY ANALYSIS

Model	$\|Q_{ex}\|$	$\|\Sigma_c\|$	$\|\mathcal{G}\|$			Time (s)
			min	max	avg	
RAS	5169	20	1	178	31.5	1
AGV	9×10^6	10	4	44	17.6	1
FBS	38880	41	1	122	8.5	1
CLD	11	3	1	4	2.3	1
CAS	3.4×10^8	142	1	28	1.4	1

Table IV shows the results of the guard generation process. For instance, in the AGV model, 9 million states are prevented to be reached by only 10 guards with an average size of 17.6. In the FBS model around 53% of the reachable states are prevented to be reached by 41 guards with an average size of 8.5 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 track the synthesis results. It can be observed that with 1 second computation time, the algorithm works quite efficiently for these examples. We believe that by complementing our BDD-based algorithms by partitioning techniques [12], it is possible to efficiently compute the supervisor and the guards for much larger and more complicated examples.

Table IV shows the guard compactness measures.

VII. CONCLUSION

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.

VIII. Acknowledgement

This work was carried out at the Wingquist Laboratory VINN Excellence Centre within the Area of Advance-Production at Chalmers.

References

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

[2] S. Balemi, G. J. Hoffmann, P. Gyugyi, H. Wong-Toi, and G. F. Franklin, "Supervisory control of a rapid thermal multiprocessor," *IEEE Trans. Autom. Control*, vol. 38, no. 7, pp. 1040–1059, Jul. 1993.

[3] 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.

[4] K. Andersson, J. Richardsson, B. Lennartson, and M. Fabian, "Coordination of operations by relation extraction for manufacturing cell controllers," *IEEE Trans. Control Syst. Technol.*, vol. 18, no. 2, pp. 414–429, Mar. 2010.

[5] D. Harel, "Statecharts: A visual formalism for complex systems," *Sci. Comput. Program.*, vol. 8, pp. 231–274, 1987.

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

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

[8] Y. Yang and R. Gohari, "Embedded supervisory control of discrete-event systems," in *Proc. IEEE Int. Conf. Autom. Sci. Eng.*, 2005, pp. 410–415.

[9] B. Gaudin and P. H. Deussen, "Supervisory control on concurrent discrete event systems with variables," in *Proc. Amer. Control Conf. (ACC)*, 2007, pp. 4274–4279.

[10] L. Ouedraogo, R. Kumar, R. Malik, and K. Åkesson, "Symbolic approach to nonblocking and safe control of extended finite automata," in *Proc. IEEE Int. Conf. Autom. Sci. Eng.*, 2010, pp. 471–476.

[11] S. B. Akers, "Binary decision diagrams," *IEEE Trans. Comput.*, vol. 27, pp. 509–516, Jun. 1978.

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

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

[14] R. Su and W. M. Wonham, "Supervisor reduction for discrete event systems," *Discrete Event Dyn. Syst.*, vol. 14, no. 1, pp. 31–53.

[15] R. Kumar and V. K. Garg, "On computation of state avoidance control for infinite state systems in assignment program framework," *IEEE Trans. Autom. Sci. Eng.*, vol. 2, no. 1, pp. 87–91, 2005.

[16] T. Le Gall, B. Jeannet, and H. Marchand, "Supervisory control of infinite symbolic systems using abstract interpretation," in *Proc. 44th IEEE Conf. Decision Control, Euro. Control Conf. (CDC-ECC)*, 2005, pp. 30–35.

[17] C. de Oliveira, J. E. R. Cury, and C. A. A. Kaestner, "Synthesis of supervisors for parameterized and infinity non-regular discrete event systems," in *Proc. 1st IFAC Workshop Depend. Control Discrete Syst. (DCDS)*, 2007, pp. 77–82.

[18] S. Miremadi, B. Lennartson, and K. Åkesson, "Symbolic computation of reduced guards in supervisory control," *IEEE Trans. Autom. Sci. Eng.*, vol. 8, no. 4, pp. 754–765, Oct. 2011.

[19] A. Hellgren, B. Lennartson, and M. Fabian, "Modelling and PLC-based implementation of modular supervisory control," in *Proc. 6th Int. Workshop Discrete Event Syst.*, 2002, pp. 371–376.

[20] K. Åkesson, "Methods and tools in supervisory control theory: Operator aspects, computation efficiency and applications," Ph.D. dissertation, Dept. Signals Syst., Chalmers Univ. Technol., Göteborg, Sweden, 2002.

[21] A. Voronov and K. Åkesson, "Verification of supervisory control properties of finite automata extended with variables," Dept. Signals Syst., Chalmers Univ. Technol., Göteborg, Sweden, 2009.

[22] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang, "Symbolic model checking: 10^{20} states and beyond," in *Proc. 5th Annu. IEEE Symp. e Logic Comput. Sci. (LICS)*, 1990, pp. 428–439.

[23] S. Miremadi, K. Åkesson, M. Fabian, A. Vahidi, and B. Lennartson, "Solving two supervisory control benchmark problems using supremica," in *Proc. 9th Int. Workshop Discrete Event Syst. (WODES)*, 2008, pp. 131–136.

[24] C. Ma and W. M. Wonham, "STSLib and its application to two benchmarks," in *Proc. 9th Int. Workshop Discrete Event Syst. (WODES)*, 2008, pp. 119–124.

[25] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, pp. 379–423, 1948.

[26] H. R. Andersen, "An introduction to binary decision diagrams," Dept. Inf. Technol., Tech. Univ. Denmark, 1997.

[27] B. Bollig and I. Wegener, "Improving the variable ordering of OBDDs Is NP-complete," *IEEE Trans. Comput.*, vol. 45, no. 9, pp. 993–1002, Sep. 1996.

[28] A. Aziz, S. Tasiran, and R. K. Brayton, "BDD variable ordering for interacting finite state machines," in *Proc. 31st Annu. Design Autom. Conf. (DAC)*, 1994, pp. 283–288.

[29] R. E. Bryant, "Symbolic Boolean manipulation with ordered binary-decision diagrams," *ACM Comput. Surv.*, vol. 24, no. 3, pp. 293–318, 1992.

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

[31] E. M. Clarke, K. L. Mcmillan, X. Zhao, M. Fujita, and J. Yang, "Spectral transforms for large Boolean functions with applications to technology mapping," *Form. Methods Syst. Des.*, vol. 10, no. 2–3, pp. 137–148, 1997.

[32] J. Gunnarsson, "Symbolic methods and tools for discrete event dynamic systems," Ph.D. dissertation, Dept. Elect. Eng., Linköping Univ., Linköping, Sweden, 1997.

[33] A. Vahidi, "Efficient analysis of discrete event systems," Ph.D. dissertation, Dept. Signals Syst., Chalmers Univ. Technol., Göteborg, Sweden, 2004.

[34] A. Nazeem and S. Reveliotis, "A practical approach to the design of maximally permissive liveness-enforcing supervisors for complex resource allocation systems," in *Proc. 6th IEEE Conf. Autom. Sci. Eng. (CASE)*, Toronto, ON, Canada, 2010, pp. 451–458 [Online]. Available: http://www.isye.gatech.edu/~spyros/publications/CASE-2010.pdf

[35] P. Magnusson, N. Sundström, K. Bengtsson, B. Lennartson, P. Falkman, and M. Fabian, "Planning transport sequences for exible manufacturing systems," presented at the 18th IFAC World Congr. Milano, Italy, 2011.

[36] M. R. Shoaei, B. Lennartson, and S. Miremadi, "Automatic generation of controllers for collision-free flexible manufacturing systems," in *Proc. 6th IEEE Int. Conf. Autom. Sci. Eng.*, 2010, pp. 368–373.

[37] G. Čengić, O. Ljungkrantz, and K. Åkesson, "Formal modeling of function block applications running in IEC 61499 execution runtime," presented at the 11th IEEE Int. Conf. Emerg. Technol. Factory Autom., Prague, Czech Republic, 2006.

[38] L. E. Holloway and B. H. Krogh, "Synthesis of feedback control logic for a class of controlled petri nets," *IEEE Trans. Autom. Control*, vol. 35, no. 5, pp. 514–523, May 1990.

[39] K. Åkesson, M. Fabian, H. Flordal, and R. Malik, "Supremica—An integrated environment for verification, synthesis and simulation of discrete event systems," in *Proc. 8th Int. Workshop Discrete Event Syst. (WODES)*, 2006, pp. 384–385.

[40] K. Åkesson, M. Fabian, H. Flordal, and A. Vahidi, "Supremica—A tool for verification and synthesis of discrete event supervisors," presented at the 11th Med. Conf. Control Autom., Rhodos, Greece, 2003.

[41] JavaBDD, 2007. [Online]. Available: http://javabdd.sourceforge.net

Sajed Miremadi was born in Linköping, Sweden, in 1983. He received the B.Sc. degree in computer engineering from Sharif University of Technology, Tehran, Iran, in 2006 and the M.Sc. degree in computer science from Linköping University, Linköping, Sweden, in 2008. He is currently pursuing the Ph.D. degree in automation from Chalmers University of Technology, Gothenburg, Sweden.

His current research interests include supervisory control and optimization of (timed) discrete event systems, using formal methods.

Bengt Lennartson (M'10) was born in Gnosjö, Sweden, in 1956. He received the Ph.D. degree in automatic control from Chalmers University of Technology, Gothenburg, Sweden, in 1986.

Since 1999, he has been a Professor of the Chair of Automation, Department of Signals and Systems, Chalmers University of Technology, and was Dean of Education from 2004 to 2007. Since 2005, he is a Guest Professor with University West, Trollhättan, Sweden. , He is (co)author of two books and 180 peer reviewed international papers with more than 2200 citations. His main areas of interest include discrete event and hybrid systems, especially for manufacturing applications, as well as robust feedback control.

Dr. Lennartson was the Chairman of the Ninth International Workshop on Discrete Event Systems, WODES'08, and currently is a member of the Advisory Board for IEEE TASE. He was an Associate Editor for *Automatica*.

Knut Åkesson received the M.S. degree in computer science and engineering from Lund Institute of Technology, Lund, Sweden, in 1997 and the Ph.D. degree in control engineering from Chalmers University of Technology, Gothenburg, Sweden, in 2002.

Currently, he is an Associate Professor with the Department of Signals and Systems, Chalmers University of Technology, where his main research interest is to develop and applying formal methods for verification and synthesis of control logic.