Extraction and Representation of a Supervisor Using Guards in Extended Finite Automata

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

***Abstract*— In supervisory control theory, an issue that often arises in real industrial applications is the huge number of states for the supervisor, which requires a lot of memory. Another problem that is typically encountered for the users of supervisory synthesis tools is lack of information and unreadability of the supervisor. In this paper, we introduce a method to characterize a controllable and non-blocking supervisor directly on the modular automata (sub-plants and sub-specifications), by extracting some guard conditions from the synthesized supervisor and the synchronized automaton. Thus, the presented approach may potentially model a complex supervisor using a compact representation whilst not infringe the original modular structure. Furthermore, the guard conditions, which are generated from a set of states, may give the user of the synthesis procedure a better understanding of which states that were removed during the synthesis. In order to obtain more compact guard expressions, we include some unnecessary states (unreachable and extended forbidden states) in the set of states that will be used for guard generation. By exploiting this extra information, it is possible to reduce the logical expressions to more compact guard conditions.**

I. INTRODUCTION

In the last decades, there has been a lot of effort to design controllers for complex systems automatically. One approach suggested by Wonham and Ramadge, is the *Supervisory control theory* for discrete event systems [1]. It is a framework for automatically synthesizing a discrete event supervisor for a plant so that the closed-loop system fulfills given specifications. The plant and the specifications are most often modeled by finite state automata. Both the plant and the supervisor are typically modeled by a number of interacting sub-automata.

The standard way of synthesizing a supervisor is to enumerate all reachable states in the closed-loop system and then remove all states that does not fulfill the given specifications. This approach has three main problems:

1) Enumerating all reachable states in the closed-loop system is computational expensive due to the state-space explosion.
2) Typically the synthesized supervisor has a large number of states and representing them as a single automaton will require much more memory than the memory in the hardware used to realize the supervisor.
3) While the input models to a supervisory synthesis problem typically consists of multiple automata, the output from the synthesis procedure (the supervisor) is in most cases a monolithic automaton. The relation between the original modular input models and the monolithic output automaton is weak and it is troublesome for the users of such a system to really understand how the synthesis procedure restricts the input automata models. Thus, a third problem that is typically encountered for users of supervisory synthesis tools, e.g. [2], [3], is that they cannot manually explore the synthesis result. More specifically, the user retrieves the final supervisor for the system without any specific information regarding the events causing undesirable states.

The authors in [4] propose an algorithm for manufacturing cell controllers to extract the relations between the operations defining the work in the cell from the synthesized supervisor. The main advantage of these relations is to give an easy-to-read representation of the control function and make the method usable in an industrial setting. However, not much attention is paid on how to reduce the final relational expressions for more complex systems which is the case in many industrial applications. Moreover, some restrictive conditions have been assumed for the original models, which should be satisfied in order to benefit the method.

In [5], [6] an implementation of decentralized supervisory control was presented. This is performed "by embedding the control map in the plant's local Finite State Machines and employing private sets of Boolean variables to encode the control information for each component supervisor" [6]. Although this process will assist the simplicity and clearness of the supervisors, the main focus in these papers is to solve the problem of decentralized communicating controllers.

The authors in [7], [8], [9], [10], [11] have proposed another class of approaches for supervisory synthesis based on the linear algebraic representation of Petri nets model of the plants. In these methods, the specifications are added to the plants in the form of linear predicates which can be considered as constraint conditions. The resulting controller can also be formulated in a similar way as suggested in this paper. However, each approach has some restrictions. The non-blocking problem is not considered in [7]. In addition, in order to employ this approach, the system should satisfy a particular structural condition: the uncontrollable subnet extracted from the Petri net model must be loop free. In [8] the liveness problem is considered but only for controlled marked graphs. The approach proposed in [9] is applicable if the supervisory net has a convex reachability set. The focus is mainly on efficient automatic verification. In [10] the request for a maximally permissive supervisor is abandoned, in favor

of a more easily computed but also more restrictive control function.

In this paper we introduce a method for characterizing the controllable and non-blocking supervisor directly on the modular automata by using extended finite automata (EFA) [12]. The main idea is to generate a supervisor that could be represented using the original modular structure that was used to represent the sub-plants and sub-specifications. This is performed by introducing guard conditions on the modular automata so that the resulting reachable states become the same states as in the supervisor.

The synthesis procedure is divided into three steps. In step 1, the monolithic supervisor is synthesized in the traditional way by using binary decision diagrams (BDDs) [13] in order to do the computation symbolically. Using binary decision diagrams make the synthesis problem tractable for many industrial problems including extremely large number of states [14], [15]. In step 2 the guard conditions, formed as logic expressions, are extracted from the monolithic supervisor - represented by BDD. Finally, the guard conditions are added to the modular automata in step 3.

A crucial step is to reduce the guard conditions to compact expressions. If the guard conditions are minimized enough, the suggested approach can also save a large amount of memory for supervisors with numerous states. We suggest some alternative state-sets, including unnecessary states (unreachable and certain forbidden states), that more probably yield compact expressions.

Since the presented approach is suitable for implementations based on BDDs, it makes it tractable for larger problems. Moreover, by using this method, the clearness and simplicity of the supervisor is enhanced. The method could indeed be used for any standard supervisory control problem and is thus applicable to any applications where the supervisory control could be used. One possible application could be to automatically generate conditions for how concurrently executing operations in a manufacturing should be coordinated such that the product could be successfully produced, see e.g. [4].

This paper is organized as follows: Section II is devoted to some preliminaries for the theory. The process of adding guards to modular automata is discussed in Section III. Section IV describes how the guard extraction from a monolithic system is performed. In Section V a BDD representation for the state-sets is presented. Finally, Section VI provides some conclusions and suggestions for future work.

II. PRELIMINARIES

In this section, we present some basic concepts that are required in order to get a better understanding of the rest of this article.

A. Finite Automaton

A finite automaton (FA) is a 5-tuple $\langle Q, \Sigma, \delta, q_i, Q_m \rangle$ where Q is a set of finite states; Σ is a finite set of events (the alphabet); and $\delta \colon Q \times \Sigma \to Q$ is a partial transition function which describes the state transitions. When $\delta(q, \sigma)$ is defined, it means that there exists a transition for the state $q \in Q$ and the event $\sigma \in \Sigma$. The next state is denoted by q', i.e. $\delta(q, \sigma) = q'$. There are also some $marked\ states\ Q_m \subseteq Q$, which are the set of states that are desired to be reached after one or several transitions.

The composition of two automata $A = \langle Q^A, \Sigma^A, \delta^A, q_i^A, Q_m^A \rangle$ and $B = \langle Q^B, \Sigma^B, \delta^B, q_i^B, Q_m^B \rangle$ is defined by the *full synchronous composition (FSC)* operator $\parallel$ [16], which results in a total system $A \parallel B = \langle Q, \Sigma^A \cup \Sigma^B, \delta, q_i, Q_m \rangle$ where $Q \subseteq Q^A \times Q^B$, $q_i = \langle q_i^A, q_i^B \rangle$ and $Q_m = \{\langle q^A, q^B \rangle \mid q^A \in Q_m^A, q^B \in Q_m^B\}$. The transition function δ for $A \parallel B$ is defined as in [16].

B. Supervisory Control Theory

Supervisory Control Theory (SCT) [1], [17] is a method for automatically synthesizing supervisors that restrict the conduct of a plant (or a number of plants) in order to satisfy some given specifications. These specifications describe the required or allowed behaviors. In an attempt to restrict the execution of the plant to the specifications, a *supervisor (controller)* is used. In automata theory, the supervisor is the automaton which enables or disables the events in the plant.

Unlike model checking [18], [19], where the goal is to verify if the model contains any incorrectness, in SCT all incorrect situations, e.g. undesirable deadlocks, should be identified and avoided in order to guarantee that the system never violates given specifications.

In SCT, events are divided into two disjoint subsets: $controllable\ events\ \Sigma_c$, i.e. the events that can be influenced by the supervisor, and $uncontrollable\ events\ \Sigma_u$, i.e. the events that cannot be influenced by the supervisor.

For a plant model P where

$$P = P_1 \parallel P_2 \parallel \ldots \parallel P_\ell,$$

and a specification model Sp where

$$Sp = Sp_1 \parallel Sp_2 \parallel \ldots \parallel Sp_m,$$

$A = P \parallel Sp$ is the full synchronized automaton.

In the process of generating the final supervisor (after the synthesis), we do not distinguish between P and Sp and thus from now on we express A as:

$$A = A_1 \parallel A_2 \parallel \ldots \parallel A_n \tag{1}$$

Some states in A are explicitly defined to be avoided; which are called $forbidden\ states$. This includes uncontrollable states as well as user defined forbidden states. There could be some states that merely lead to the forbidden states and thus they should also be prohibited. We call such states the *extended forbidden states* and denote the corresponding set by Q_{ex} which follows the supervisory synthesis. Hence, the supervisor is generated by excluding Q_{ex} from the reachable states in A.

Following are notations for some state-sets which will be used later in the paper:

- Q: All states in $A_1 \times A_2 \times \ldots \times A_n$.
- Q^σ: $\{q \in Q \mid \exists\ q' \in Q, \sigma \in \Sigma.\ \delta(q, \sigma) = q'\}$. The states that enable σ.
- Q_{reach}: The reachable states. The states that can be reached from the initial state by a number of transitions.

Q_{sup} : All the states in the supervisor.
Q^{σ}_{sup} : $Q_{sup} \cap Q^{\sigma}$.
Q_{ex} : $Q_{reach} \backslash Q_{sup}$.

C. Extended Finite Automaton

An Extended Finite Automaton (EFA) presented in [20], [12], is an extension of the ordinary FA with guard (conditional) formulas and action functions including different variables. In this kind of automaton, a transition is enabled if the associated guard is $true$, and when the transition is taken, updating actions of a set of variables may follow. An EFA is a 6-tuple $\langle Q \times V, \Sigma, \mathcal{G}, A, \rightarrow, (q_0, v_0)\rangle$ where Q is a set of states; V is the domain of definition of variables; $Q \times V$ is the extended finite set of states; Σ is the alphabet; $\mathcal{G}$ is the set of guard predicates over V; A is a set of action functions, i.e. $\{a \mid a\colon V \rightarrow V\}$; $\rightarrow \subseteq Q \times \Sigma \times \mathcal{G} \times A \times Q$ is the state transition relation; and (q_0, v_0) is the initial state.

Fig. 1 shows a sample EFA where σ, $\mathcal{G}$, and A stand for $event$, $guard$, and $action$ respectively.

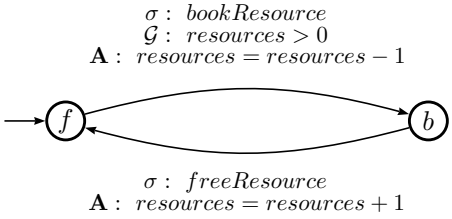


Fig. 1. A sample EFA.

III. ADDING GUARDS TO MODULAR SYSTEMS

As stated earlier, in order to synthesize the supervisor, the extended forbidden states (Q_{ex}) should be excluded from the reachable states (Q_{reach}) in synchronized automaton, A (1).

Another approach to generate the final supervisor is to add some restrictive guard conditions to the transitions of modular automata, i.e. sub-plants and sub-specifications, and avoid them to reach the extended forbidden states. Hence, by assuming that the systems are modeled by FAs, after adding the guards, they will form EFAs. This enables us to characterize the supervisor directly on the modular automata. The guards can be extracted from the *monolithic system* (the full synchronized composition of the modular automata) by using the information from the supervisor. Recall that we wish to determine the events in the modular automata that should be enabled or disabled. Thus, we will study the case for each event separately.

In order to generate the guard conditions, we will first determine the state-sets where an event σ can occur and extract the guard expressions from these state-sets. There are two different points of views one can consider for constructing the state-sets:

Case $\mathcal{A}$. States where σ is *allowed*, denoted by Q^{σ}_{a}.
Case $\mathcal{F}$. States where σ is *forbidden*, denoted by Q^{σ}_{f}.

Hence, we can either choose to restrict a transition by forcing it to be or not to be in a state-set while executing the event. As a result, there would be two types of guard conditions: *allowing* guard conditions ($\mathcal{G}^{\sigma}_{a}$) and *forbidding* guard conditions ($\mathcal{G}^{\sigma}_{f}$). Consequently, there are two approaches to construct a guard condition for an event σ:

1) $\mathcal{G}^{\sigma}_{a}$: The guard expression is *true* when Case $\mathcal{A}$ is satisfied.
2) $\mathcal{G}^{\sigma}_{f}$: The guard expression is *false* when Case $\mathcal{F}$ is satisfied.

From section II-B, recall that, A (1) is the full synchronous composition of n automata $A_1, A_2, \ldots, A_n$. Thus each state in the monolithic automaton has the following form:

$$q_j = \langle q^{A_1}_{j_1}, q^{A_2}_{j_2}, \ldots, q^{A_n}_{j_n}\rangle$$

For case $\mathcal{A}$, we just take into account the states that can be allowed: $\{\langle q^{A_1}_{k_1}, q^{A_2}_{k_2}, \ldots, q^{A_n}_{k_n}\rangle, \ldots, \langle q^{A_1}_{\ell_1}, q^{A_2}_{\ell_2}, \ldots, q^{A_n}_{\ell_n}\rangle\}$ and thus the expression will have the following form:

$$\mathcal{G}^{\sigma}_{a} = ((q^{A_1} = q^{A_1}_{k_1}) \wedge (q^{A_2} = q^{A_2}_{k_2}) \wedge \ldots \wedge (q^{A_n} = q^{A_n}_{k_n})) \vee \ldots \vee ((q^{A_1} = q^{A_1}_{\ell_1}) \wedge (q^{A_2} = q^{A_2}_{\ell_2}) \wedge \ldots \wedge (q^{A_n} = q^{A_n}_{\ell_n}))$$

On the other hand, for case $\mathcal{F}$, where we consider the states that must be forbidden $\{\langle q^{A_1}_{k'_1}, q^{A_2}_{k'_2}, \ldots, q^{A_n}_{k'_n}\rangle, \ldots, \langle q^{A_1}_{\ell'_1}, q^{A_2}_{\ell'_2}, \ldots, q^{A_n}_{\ell'_n}\rangle\}$, the guard expression that represents the state-set for forbidden states is:

$$\begin{aligned}\mathcal{G}^{\sigma}_{f} &= \neg((q^{A_1} = q^{A_1}_{k'_1}) \wedge (q^{A_2} = q^{A_2}_{k'_2}) \wedge \ldots \wedge (q^{A_n} = q^{A_n}_{k'_n})) \wedge \ldots \wedge \neg((q^{A_1} = q^{A_1}_{\ell'_1}) \wedge (q^{A_2} = q^{A_2}_{\ell'_2}) \wedge \ldots \wedge (q^{A_n} = q^{A_n}_{\ell'_n})) \\ &= ((q^{A_1} \neq q^{A_1}_{k'_1}) \vee (q^{A_2} \neq q^{A_2}_{k'_2}) \vee \ldots \vee (q^{A_n} \neq q^{A_n}_{k'_n})) \wedge \ldots \wedge ((q^{A_1} \neq q^{A_1}_{\ell'_1}) \vee (q^{A_2} \neq q^{A_2}_{\ell'_2}) \vee \ldots \vee (q^{A_n} \neq q^{A_n}_{\ell'_n})),\end{aligned}$$

The final guard expression that will be added to transition $\delta(q^{A_j}_{r_j}, \sigma)$ is computed by removing the terms that include $q^{A_j}_{r_j}$ from the expression. In order to get a more simplified expression, standard algorithms for minimization of logic expressions, e.g. [21], [22], will be performed on the final guard condition. The guards expressions can either be represented in disjunctive normal form (DNF) or conjunctive normal form (CNF). For each specific example, the form that has a simpler comprehension for the *user*, will be selected. We clarify the above process by the following example.

Example 1: Consider the classical resource booking problem where there are users that will use two resources but in opposite order. Thus it can be directly implied that there would be a deadlock in the system when the users use a common resource at the same time. Fig. 2 shows the resource automata models plus the monolithic automaton for this system. Note that state $\langle q^A_2, q^B_2, q^C_2, q^D_2\rangle$ in Fig. 2(b) is a deadlock state. Now consider the guard expression for event a_1. We study this case for each of the approaches mentioned earlier:

1) The states that must be allowed for event a_1 are $\{\langle q^A_1, q^B_1, q^C_1, q^D_1\rangle, \langle q^A_1, q^B_3, q^C_1, q^D_1\rangle\}$. Hence the guard expression will be:

$$\begin{aligned}\mathcal{G}^{a_1}_{a} = &((q^A = q^A_1) \wedge (q^B = q^B_1) \wedge (q^C = q^C_1) \wedge (q^D = q^D_1)) \\ &\vee ((q^A = q^A_1) \wedge (q^B = q^B_3) \wedge (q^C = q^C_1) \wedge (q^D = q^D_1))\end{aligned}$$

which can be simplified to

$$\begin{aligned}\mathcal{G}^{a_1}_{a} = &((q^A = q^A_1) \wedge (q^C = q^C_1) \wedge (q^D = q^D_1)) \wedge \\ &((q^B = q^B_1) \vee (q^B = q^B_3))\end{aligned}$$

Thus for transition $\delta(q_1^A, a_1)$, the guard will be

$$\mathcal{G}_a^{a_1} = ((q^C = q_1^C) \wedge (q^D = q_1^D)) \wedge ((q^B = q_1^B) \vee (q^B = q_3^B))$$

2) The state-set where a_1 should be forbidden is $\langle q_1^A, q_2^B, q_1^C, q_2^D \rangle$. Thus we will have

$$\mathcal{G}_f^{a_1} = (q^A \neq q_1^A) \vee (q^B \neq q_2^B) \vee (q^C \neq q_1^C) \vee (q^D \neq q_2^D)$$

which will be

$$\mathcal{G}_f^{a_1} = (q^B \neq q_2^B) \vee (q^C \neq q_1^C) \vee (q^D \neq q_2^D)$$

for transition $\delta(q_1^A, a_1)$.

Fig. 2(c) shows the automata A and B after adding $\mathcal{G}_f^{a_1}$ and $\mathcal{G}_f^{b_2}$ respectively. Note that all of the four modular automata operate in a synchronized manner to obtain the desired supervisor.

IV. EXTRACTING GUARDS FROM A MONOLITHIC SYSTEM

In the previous section we mentioned how we can characterize the supervisor by adding restricting guard conditions to the modular automata. Now the question is how we can extract the guard expressions from the synchronized model A and the supervisor S.

As stated earlier, there are two cases we could consider in an attempt to construct the guards. For each case we study two levels of *certainty* by introducing the following definitions.

Definition 1 (Upper bound of Q_a^σ : $U(Q_a^\sigma)$): The states where σ **can** be allowed. Hence, if the set $U(Q_a^\sigma)$ is extended to include a state in $C(U(Q_a^\sigma))$, then the guard expressions generated from the extended set of $U(Q_a^\sigma)$ will make it possible for the closed loop system to enter a state that was removed in the synthesis procedure, i.e. Q_{ex}.

Definition 2 (Lower bound of Q_a^σ : $L(Q_a^\sigma)$): The states where σ **must** be allowed. Hence, if the set $L(Q_a^\sigma)$ is restricted to not include a state in $L(Q_a^\sigma)$, then the guard expressions generated from the restricted set of $L(Q_a^\sigma)$ will not make it possible for the closed loop system to enter a state that was retained after the synthesis procedure, i.e. Q_{sup}^σ.

Definition 3 (Upper bound of Q_f^σ : $U(Q_f^\sigma)$): The states where σ **can** be forbidden. Hence, if the set $U(Q_f^\sigma)$ is extended to include a state in $U(Q_f^\sigma)$, then the guard expressions generated from the extended set of $U(Q_f^\sigma)$ will not make it possible for the closed loop system to enter a state that was retained after the synthesis procedure, i.e. Q_{sup}^σ.

Definition 4 (Lower bound of Q_f^σ : $L(Q_f^\sigma)$): The states where σ **must** be forbidden. Hence, if the set $L(Q_f^\sigma)$ is restricted to not include a state in $L(Q_f^\sigma)$, then the guard expressions generated from the restricted set of $L(Q_f^\sigma)$ will make it possible for the closed loop system to enter a state that was removed in the synthesis procedure, i.e. Q_{ex}.

It can directly be observed that there is a *duality* relation between the upper and lower bounds for each case. Hence,

$$U(Q_a^\sigma) = C(L(Q_f^\sigma)) \quad or \quad L(Q_f^\sigma) = C(U(Q_a^\sigma))$$
$$L(Q_a^\sigma) = C(U(Q_f^\sigma)) \quad or \quad U(Q_f^\sigma) = C(L(Q_a^\sigma))$$

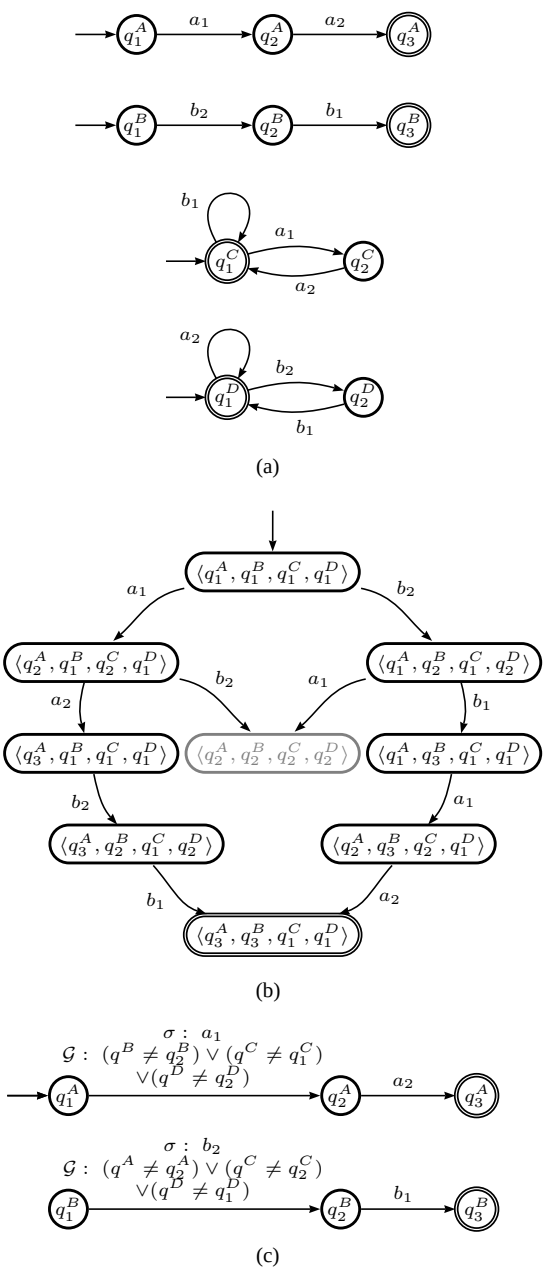


Fig. 2. Example 1. a) Product descriptions and resource models. b) Full synchronized composition of the automata ($A \parallel B \parallel C \parallel D$). c) Automata A and B after adding $\mathcal{G}_f^{a_1}$ and $\mathcal{G}_f^{b_2}$ respectively.

where $C(X)$ denotes the complement of set X by having Q as the universal set.

By definition of Q_{sup}^σ stated in section II-B, it is straightforward that

$$L(Q_a^\sigma) = Q_{sup}^\sigma;$$

and thus

$$U(Q_f^\sigma) = C(Q_{sup}^\sigma)$$

The lower bound of Q_f^σ will be shown in the following lemma and theorem.

Lemma 1: For every state that belongs to $L(Q_f^\sigma)$, there exists an event σ which leads to a state in Q_{ex}. More

formally, let q be an arbitrary state in $L(Q_f^\sigma)$, then it holds that $\delta(q,\sigma) \in Q_{ex}$.

Proof: The proof will be shown by contradiction. Let $q \in L(Q_f^\sigma)$. Assume that there is exists a state $q' = \delta(q,\sigma) \notin Q_{ex}$. Thus:

$$\begin{aligned} & q' \in C(Q_{ex}) \\ & \Rightarrow q' \in C(Q_{reach} \backslash Q_{sup}) \\ & \Rightarrow q' \in C(Q_{reach} \cap C(Q_{sup})) \\ & \Rightarrow q' \in C(Q_{reach}) \cup Q_{sup} \end{aligned}$$

This implies that q' belongs either to $C(Q_{reach})$ or Q_{sup}. If $q' \in C(Q_{reach})$, it means that an unreachable state that will never be reached is forbidden which violates the lower bound specifications. If $q' \in Q_{sup}$, it means that q should not be forbidden, but we had assumed that $q \in L(Q_f^\sigma)$ which leads to a contradiction. Hence, for both of the cases we will face contradictions and thus it implies that $\delta(q,\sigma) \in Q_{ex}$. ■

Theorem 1: The lower bound of Q_f^σ is

$$Q^\sigma \cap Q_{reach} \cap C(Q_{sup}^\sigma) \cap C(Q_{ex}).$$

Proof: The proof will be shown by contradiction. Assume there is a state-set $Q_\ell \subset L(Q_f^\sigma)$, where a state $q \in L(Q_f^\sigma) \backslash Q_\ell$. According to Lemma 1, $q' = \delta(q,\sigma) \in Q_{ex}$. Thus, if we generate the guard conditions from Q_ℓ, then we can reach a state $q' \in Q_{ex}$ after the supervisory synthesis which leads to a contradiction. ■

Based on the duality property, a direct deduction from this theorem is

$$U(Q_a^\sigma) = C(Q^\sigma) \cup C(Q_{reach}) \cup Q_{sup}^\sigma \cup Q_{ex}$$

This means that the states where σ can be allowed are the states that do not enable σ; or the unreachable states; or the states in the supervisor; or the extended forbidden states which will not be reached anyway.

A challenging issue is which approach between $\mathcal{A}$ and $\mathcal{F}$ is more convenient for extracting the guard conditions. To deal with this question, we first introduce two factors that can impact our decision:

- *Memory*: In most of the cases, the automata will be saved on a limited amount of memory, e.g. PLCs; therefore it is crucial to have guard expressions that are reduced as much as possible.
- *User*: From a user perspective, a *reduced* logic expression would be more readable and understandable. Nevertheless, sometimes if an expression is reduced too much, it can decrease the comprehension.

Definition 5 (Minimal Guard Expression (MGE)): Among a set of equivalent guard expressions (expressions with equal truth tables), MGE is the DNF (CNF) expression with the least number of conjunctive (disjunctive) clauses. This definition is based on this assumption that from a user perspective, a logic expression with fewer clauses is more comprehensible.

The goal is to find the MGE for a set of guard expressions. Depending on the system, one of the approaches can yield the MGE, and thus basically either of them can be desirable.

However, based on the following hypotheses, a proper choice can be the second case where the state-set is Q_f^σ and the guard $\mathcal{G}_f^\sigma$.

Hypothesis 1. It is of more importance for the user to realize what cannot occur in a system.

Hypothesis 2. Practically, there are very few situations where the synthesis restricts the events that can occur.

It is hard to say if there exists a state-set Q_{in} represented by set operations that always yield MGEs. Nonetheless, according to the lower and upper bounds of Q_f^σ, Q_{in} has the following restriction:

$$\begin{gathered} L(Q_f^\sigma) \subseteq Q_{in} \subseteq U(Q_f^\sigma) \\ Q^\sigma \cap Q_{reach} \cap C(Q_{sup}^\sigma) \cap C(Q_{ex}) \subseteq Q_{in} \subseteq C(Q_{sup}^\sigma) \end{gathered}$$

We can rewrite $L(Q_f^\sigma)$ as follows:

$$\begin{aligned} L(Q_f^\sigma) &= Q^\sigma \cap Q_{reach} \cap C(Q_{sup}^\sigma) \cap C(Q_{ex}) \\ &= (Q^\sigma \backslash Q_{sup}^\sigma) \cap Q_{reach} \cap C(Q_{reach} \cap C(Q_{sup})) \\ &= ((Q^\sigma \backslash Q_{sup}^\sigma) \cap Q_{reach} \cap C(Q_{reach})) \cup \\ &\quad ((Q^\sigma \backslash Q_{sup}^\sigma) \cap Q_{reach} \cap Q_{sup}) \\ &= (Q^\sigma \backslash Q_{sup}^\sigma) \cap Q_{reach} \cap Q_{sup} \end{aligned}$$

In a first glance, it seems that $L(Q_f^\sigma)$ produces MGE, however, this does not always hold. By including some unnecessary states (unreachable and extended forbidden states), it is possible to perform an additional reduction in the final minimization. Thus, there is a trade-off between retaining the expression as reduced as possible, and adding some unnecessary states for assisting the final minimization.

As a conclusion, four reasonable alternatives for Q_f^σ can be suggested:

a) $Q_{f_1}^\sigma = Q^\sigma \backslash Q_{sup}^\sigma$.
b) $Q_{f_2}^\sigma = Q^\sigma \backslash Q_{sup}^\sigma \backslash C(Q_{reach})$.
c) $Q_{f_3}^\sigma = Q^\sigma \backslash Q_{sup}^\sigma \backslash Q_{ex}$.
d) $Q_{f_4}^\sigma = Q^\sigma \backslash Q_{sup}^\sigma \backslash C(Q_{reach}) \backslash Q_{ex}$.

By computing the above state-sets for a number of examples, one can get a view of the alternative that likely yields the MGEs in most of the cases.

As a final remark, note that all the state-sets represented, i.e. Q^σ, Q_{sup}^σ, Q_{reach}, Q_{ex}, and their complements, can be effectively computed by BDDs and this is where we can take advantage of such data structures.

The theory extended in this section is illustrated by the following example.

Example 2: Consider the two sub-plant models P_1 and P_2 and two sub-specifications Sp_1 and Sp_2 shown in Fig. 3(a). Moreover, their full synchronous composition (S_0) is illustrated in Fig. 3(b). The states in the monolithic automaton have the following form:

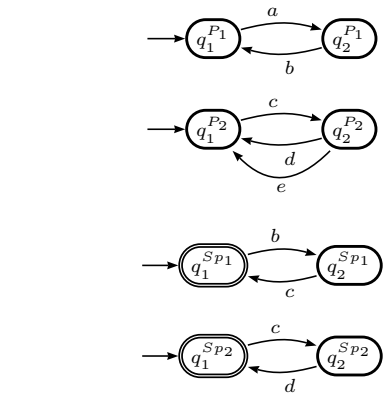
$$q_{rspt} = \langle q_r^{P_1}, q_s^{P_2}, q_p^{SP_1}, q_t^{SP_2} \rangle$$

We also use the following notations in the guard expressions:

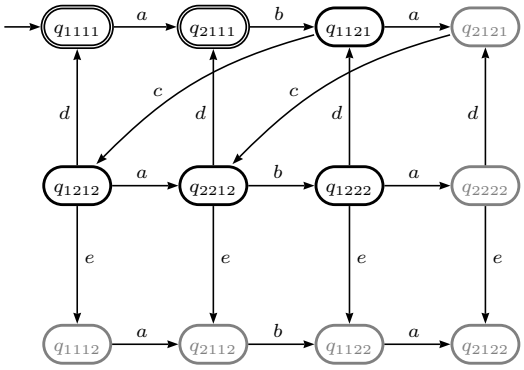
$$\backslash q_i^A \equiv (q^A \neq q_i^A)$$

where q_i^A means state i in automaton A.

Assume that the forbidden states are $\{q_{2121}, q_{2222}, q_{1112}, q_{2112}, q_{1122}, q_{2122}\}$. Moreover, the



(a)



(b)

Fig. 3. Example 2. a) Sub-plant models P_1 and P_2 and sub-specifications S_{P_1} and S_{P_2}. b) Full synchronized composition of the automata ($P_1 \parallel P_2 \parallel S_{P_1} \parallel S_{P_2}$).

unreachable states are $\{q_{1211}, q_{2211}, q_{2221}, q_{1221}\}$. We compute the alternative state-sets introduced earlier for events a and e plus their respective guard expressions:

a) $Q^a_{f_1} = \{q_{1121}, q_{1222}, q_{1112}, q_{1122}, q_{1211}, q_{2211}, q_{2221}, q_{1221}\} \Longrightarrow$

$$\begin{aligned}\mathcal{G}^a_{f_1} = &(\backslash q_1^{P_1} \vee \backslash q_1^{P_2} \vee \backslash q_2^{SP_1} \vee \backslash q_1^{SP_2}) \\ &\wedge(\backslash q_1^{P_1} \vee \backslash q_2^{P_2} \vee \backslash q_2^{SP_1} \vee \backslash q_2^{SP_2}) \\ &\wedge(\backslash q_1^{P_1} \vee \backslash q_1^{P_2} \vee \backslash q_1^{SP_1} \vee \backslash q_2^{SP_2}) \\ &\wedge(\backslash q_1^{P_1} \vee \backslash q_1^{P_2} \vee \backslash q_2^{SP_1} \vee \backslash q_2^{SP_2}) \\ &\wedge(\backslash q_1^{P_1} \vee \backslash q_2^{P_2} \vee \backslash q_1^{SP_1} \vee \backslash q_1^{SP_2}) \\ &\wedge(\backslash q_2^{P_1} \vee \backslash q_2^{P_2} \vee \backslash q_1^{SP_1} \vee \backslash q_1^{SP_2}) \\ &\wedge(\backslash q_2^{P_1} \vee \backslash q_2^{P_2} \vee \backslash q_2^{SP_1} \vee \backslash q_1^{SP_2}) \\ &\wedge(\backslash q_1^{P_1} \vee \backslash q_2^{P_2} \vee \backslash q_2^{SP_1} \vee \backslash q_1^{SP_2})\end{aligned}$$

By performing a minimization algorithm on this logic expression and applying it on state $q_1^{P_1}$, the only state that enables event a, it can be reduced to

$$\mathcal{G}^{\mathbf{a}}_{\mathbf{f_1}} = (\backslash \mathbf{q_2^{P_2}} \wedge \backslash \mathbf{q_2^{SP_1}} \wedge \backslash \mathbf{q_2^{SP_2}}) \vee (\backslash \mathbf{q_1^{P_2}} \wedge \backslash \mathbf{q_2^{SP_1}} \wedge \backslash \mathbf{q_1^{SP_2}})$$

For the rest of the expressions, we merely show the reduced representations for events a and e, on states $q_1^{P_1}$ and $q_2^{P_2}$ respectively.

$Q^e_{f_1} = \{q_{1212}, q_{2212}, q_{1222}, q_{2222}, q_{1211}, q_{2221}, q_{1221}\} \Longrightarrow$

$$\mathcal{G}^{\mathbf{e}}_{\mathbf{f_1}} = (\backslash \mathbf{q_2^{P_2}})$$

which becomes $false$ for state $q_2^{P_2}$.

b) $Q^a_{f_2} = \{q_{1121}, q_{1222}, q_{1112}, q_{1122}\} \Longrightarrow$

$$\mathcal{G}^{\mathbf{a}}_{\mathbf{f_2}} = (\backslash \mathbf{q_2^{SP_1}} \wedge \backslash \mathbf{q_2^{SP_2}}) \vee (\backslash \mathbf{q_1^{P_2}} \wedge \backslash \mathbf{q_2^{SP_2}}) \vee (\backslash \mathbf{q_1^{P_2}} \wedge \backslash \mathbf{q_2^{SP_1}})$$

$Q^e_{f_2} = \{q_{1212}, q_{2212}, q_{1222}, q_{2222}\} \Longrightarrow false$

c) $Q^a_{f_3} = \{q_{1121}, q_{1222}, q_{1211}, q_{2211}, q_{2221}, q_{1221}\} \Longrightarrow$

$$\mathcal{G}^{\mathbf{a}}_{\mathbf{f_3}} = (\backslash \mathbf{q_2^{SP_1}} \wedge \backslash \mathbf{q_1^{SP_2}}) \vee (\backslash \mathbf{q_2^{P_2}} \wedge \backslash \mathbf{q_1^{SP_2}}) \vee (\backslash \mathbf{q_2^{SP_1}} \wedge \backslash \mathbf{q_2^{P_2}})$$

$Q^e_{f_3} = \{q_{1212}, q_{2212}, q_{1222}, q_{1211}, q_{2221}, q_{1221}\} \Longrightarrow$

$$\mathcal{G}^{\mathbf{e}}_{\mathbf{f_3}} = (\backslash \mathbf{q_1^{P_1}} \wedge \backslash \mathbf{q_1^{SP_1}} \wedge \backslash \mathbf{q_1^{SP_2}})$$

d) $Q^a_{f_4} = \{q_{1121}, q_{1222}\} \Longrightarrow$

$$\mathcal{G}^{\mathbf{a}}_{\mathbf{f_4}} = (\backslash \mathbf{q_1^{P_2}} \wedge \backslash \mathbf{q_2^{SP_2}}) \vee (\backslash \mathbf{q_2^{P_2}} \wedge \backslash \mathbf{q_1^{SP_2}}) \vee (\backslash \mathbf{q_2^{SP_1}})$$

$Q^e_{f_4} = \{q_{1212}, q_{2212}, q_{1222}\} \Longrightarrow$

$$\mathcal{G}^{\mathbf{e}}_{\mathbf{f_4}} = (\backslash \mathbf{q_1^{P_1}} \wedge \backslash \mathbf{q_1^{SP_1}}) \vee (\backslash \mathbf{q_2^{SP_2}})$$

We observe that for this specific example, alternative (a), i.e. $Q^{\sigma}_{f_1}$, yields MGEs. The resulted guard expressions for $Q^{\sigma}_{f_1}$ is shown in Fig. 4. Note that since the events a and e appear on P_1 and P_2, the guard conditions will just be added on those automata. In general, after these eliminations, one could perform a further reduction on the final expression. Since the reduction is performed on a new expression, it is possible to obtain a more reduced one.

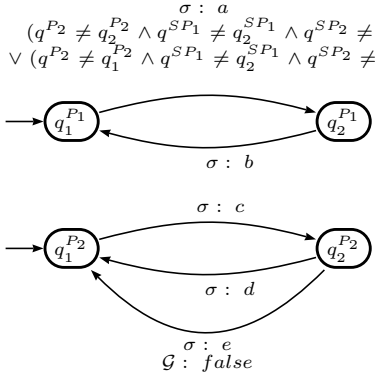


Fig. 4. The resulted modular automata in Example 2 with guard conditions.

V. BDD REPRESENTATION FOR STATE-SETS

As discussed, the extraction and addition of guards deal with various state-sets of the automata such as Q^{σ}, $C(Q_{reach})$, etc., and a number of set-operations are performed on these sets. Thus, in order to have an efficient implementation of the system, one should take advantage of a good data structure to represent the automata and the state-sets. A powerful symbolic representation for an automaton is Binary Decision Diagram (BDD) [13]. Given a set of Boolean variables V, a BDD is a Boolean function

$f\colon 2^V \rightarrow \{0,1\}$ represented as a directed acyclic graph (DAG) which consists of two types of nodes: *decision nodes* and *terminal nodes*. A terminal node can either be 0-terminal or 1-terminal. If the variables in the BDD follow a total order, it is called $Ordered\ BDD\ (OBDD)$. The main idea behind OBDD is that it can be reduced to a compact and canonical data representation of a Boolean function which is often called $Reduced\ OBDD$ [23]. In order to represent complex structures such as automata with BDDs, a construct called characteristic function is often used. Having a finite set S, for every subset A of S, the characteristic function is defined as follows:

$$\chi_A(\alpha) = \begin{cases} 1 & \alpha \in A \\ 0 & \alpha \notin A \end{cases}$$

Hence, the basic set-operations such as *union*, *complement* and *comparison* can be applied to characteristic functions using Boolean operators. For instance, if $A_1, A_2 \subseteq S$, then $A_1 \cup A_2$ can be expressed as $\chi_{A_1} \vee \chi_{A_2}$, since $A_1 \cup A_2 = \{a \in S \mid a \in \chi_{A_1} \vee a \in \chi_{A_2}\}$ Consequently, the state-sets mentioned above can easily be represented by BDDs. For instance, consider the reachable states for an automaton (Q_{reach}). By starting from the initial state and performing iterative $fixpoint$ computations, in each step of the computation, a new set of reachable states, i.e. the states that are one transition away from the states in Q_{reach}, will be added to the new state-set. This procedure will be repeated until no more new states are found; or in other words until the global $fixpoint$ is reached. Afterwards, one can easily compute the $C(Q_{reach})$. It is just sufficient to replace the 0-terminals with 1-terminals and vice versa for the BDD of Q_{reach}. Similarly, other state-sets can also be represented by BDDs.

To conclude, the representation of state-sets and set-operations are preferably computed by BDDs. Using binary decision diagrams make the synthesis problem tractable for many industrial problems [14], [15]. We can also benefit of these data structures in the minimization process of logic expressions.

VI. CONCLUSIONS AND FUTURE WORKS

In this paper, we introduced a method for characterizing a supervisor directly on the modular automata by extracting guard conditions from the monolithic system. The extraction process is performed by first determine some state-sets in the synchronized automaton where a given event should be prohibited in order to prevent the system to reach the forbidden states, and second to convert the state-sets to guard expressions. We presented some suggestions for state-sets including unnecessary states (unreachable and certain forbidden states) in order to reduce the logical expressions to more compact guard conditions. Furthermore, we showed how BDDs can be used to represent the state-sets used in the guard extraction and why they are counted as powerful data structures for large systems.

There are some directions in which we could extend and optimize our method. In this paper, we have assumed that the modular automata are always ordinary finite automata and then after adding the guard conditions they become EFAs. Thus we start the whole process from FAs. An extension to this could be to have EFAs as the modular automata from the beginning and perform the guard extraction and minimization based on these models. This would require another structure with some analogous parts to the method presented here.

As discussed, we cannot make a certain and general conclusion which state-set that gives the minimal guard expression among the four suggested alternatives. A possible future work is to investigate for which state-set it is more probable to retrieve a more reduced expression, especially for large systems based on BDD computations.

REFERENCES

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

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

[3] L. Feng and W. Wonham, "Tct: A computation tool for supervisory control synthesis," *Discrete Event Systems, 2006 8th International Workshop on*, pp. 388–389, 2006.

[4] K. Andersson, J. Richardsson, B. Lennartson, and M. Fabian, "Coordinated operations by relation extraction for manufacturing cell controllers," Signals and Systems, Chalmers, Göteborg, Sweden, Tech. Rep. R017/2006, 2006.

[5] Y. Yang and P. Gohari, "Embedded supervisory control of discrete-event systems," in *2005*, Edmonton, Canada, August 2005, pp. 410–415.

[6] A. Mannani, Y. Yang, and P. Gohari, "Distributed extended finite-state machines: communication and control," *Discrete Event Systems, 2006 8th International Workshop on*, pp. 161–167, 10-12 July 2006.

[7] Y. Li and W. M. Wonham, "Control of vector discrete-event systems II—Controller synthesis," *IEEE*, vol. 39, no. 3, pp. 512–531, 1994.

[8] L. Holloway and B. Krogh, "On closed-loop liveness of discrete event systems under maximally permissive control," *IEEE Transactions on Automatic Control*, vol. 37, no. 5, pp. 692–697, 1992.

[9] A. Giua and F. DiCesare, "Blocking and controllability of Petri nets in supervisory control," *IEEE*, vol. 39, no. 4, pp. 818–823, 1994.

[10] K. Yamalidou, J. O. Moody, M. D. Lemmon, and P. J. Antsaklis, "Feedback control of Petri nets based on place invariats," *Automatica*, vol. 32, no. 1, pp. 15–28, 1996.

[11] L. E. Holloway, B. H. Krogh, and A. Giua, "A survey of Petri net methods for controlled discrete event systems," *Discrete Event Dynamic Systems*, no. 7, pp. 151–190, 1997.

[12] M. Sköldstam, K. Åkesson, and M. Fabian, "Supervisory control applied to automata extended with variables," Signals and Systems, Chalmers, Göteborg, Sweden, Tech. Rep. R003/2007, 2007.

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

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

[15] A. Vahidi, "Efficient analysis of discrete event systems," Ph.D. dissertation, Signals and Systems, Chalmers, Göteborg, Sweden, 2004.

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

[17] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Kluwer, Sep. 1999.

[18] J.-P. Queille and J. Sifakis, "Specification and verification of concurrent systems in cesar," in *Proceedings of the 5th Colloquium on International Symposium on Programming*. London, UK: Springer-Verlag, 1982, pp. 337–351.

[19] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications." *ACM Transactions on Programming Languages and Systems*, vol. 8, no. 2, pp. 244–263, 1986.

[20] Y.-L. Chen and F. Lin, "Modeling of discrete event systems using finite state machines with parameters," in *CCA00*, Anchorage, Alaska, Sep. 2000.

[21] J. P. Tremblay and R. Manohar, *Discrete Mathematical Structures with Applications to Computer Science*. McGraw-Hill, 1987.

[22] B. Reusch, “Generation of prime implicants from subfunctions and a unifying approach to the covering problem,” *IEEE Trans. Comput.*, vol. 24, no. 9, pp. 924–930, 1975.
[23] R. E. Bryant, “Symbolic boolean manipulation with ordered binary-decision diagrams,” *ACM Comput. Surv.*, vol. 24, no. 3, pp. 293–318, 1992.