Automation Research Group
Department of Signals and Systems
Chalmers University of Technology

Efficient Supervisory Synthesis to Large-Scale Discrete Event Systems Modeled as Extended Finite Automata

Zhennan Fei and Sajed Miremadi

Date:	March 15, 2012
Subject:	Fundation of Automation
Mentors:	Knut Åkesson and Bengt Lennartson
Project term:	PhD Research
E-mail:	zhennan@chalmers.se
ISSN:	1403-266X
Report No:	R005/2012

Abstract

The state-space explosion problem, resulting from the reachability computation of the synthesis task, is one of main obstacles preventing the supervisory control theory (SCT) from having a major industrial breakthrough. To alleviate this problem, a well-known strategy is to utilize binary decision diagrams (BDDs) to represent system transition relations and compute supervisors symbolically. Based on this principle, we present in this paper an efficient reachability approach to large-scale discrete event systems modeled as finite automata with variables. By making use of the disjunctive partitioning technique, the proposed approach partitions the transition relation of a considered system into a set of partial transition relations according to included events. Then those partial transition relations are selected systematically to perform the reachability computation. Consequently, more iterations might be required to compute the fixed point, but the intermediate BDDs are smaller. As a supplement to prior framework, the approach has been implemented in the supervisory control tool Supremica and the efficiency is demonstrated on a set of industrially relevant benchmark problems.

1 Introduction

The analysis of discrete event systems (DESs) has been paid extensive attention by researchers and scientists in the computer science community. One typical analysis approach is to utilize formal verification techniques, such as model checking, to verify whether considered systems fulfill given specifications or not. However, from the control engineering point of view, instead of verifying the correctness of a DES model, a controller, which automatically conducts the system behavior without violating specifications, is a necessity. The supervisory control theory (SCT) [1, 2] provides a control-theoretic framework for control engineers to

design such a safety device, referred to as the *supervisor* of a system. Given a DES model to be controlled, the *plant*, and the intended behavior, the *specification*, the supervisor can be automatically synthesized, guaranteeing that the closed-loop system always achieves the given specification. SCT has been applied to a variety of areas such as automated manufacturing systems, communication networks and embedded systems [3, 4, 5].

When modeling and synthesizing large-scaled DESs by the supervisory control theory, a typical issue is to how to compute such a control function efficiently and represent it appropriately. The standard approach provided by SCT is to model the considered system by finite automata, synthesize the supervisor and then represent all of reachable states in the closed-loop system explicitly. However, regarding systems of industrially interesting sizes, the standard approach reveals some drawbacks: (a) Modeling complex systems with ordinary automata might make the model large and intractable; (b) The resultant supervisor, as a single automaton, typically consists of a huge number of states, which makes it difficult for users to understand thoroughly.

In prior work [6], a framework was presented where users can both model a system and obtain the supervisor in the form of *extended finite automata* (EFAs) [7], which is an augmentation of an ordinary automata extended with variables. By taking the advantage of EFAs, more compact and comprehensible system models can be obtained. In addition, instead of representing the supervisor as a single automaton, the guard generation procedure provided in the framework can extract a set of logic formulas. Those extracted formulas, referred to as guards, are attached to the corresponding transitions of original models, which results in a modular representation of the supervisor.

Whereas the aforementioned framework allows compact representation of large state-spaces, when it comes to analysis, the number of states are not affected and could potentially cause the *state-space explosion* problem that typically occurs when the behavior of interacting sub-systems is studied. Particularly, the state-space explosion problem arises from the synthesis task which involves a series of reachability computations. As DESs becoming more complicated, the traditional explicit state-space traversal algorithm may be intractable. The state-space explosion problem actually is one of the main obstacles which prevent SCT from having a major industrial breakthrough. Over decades, numerous researches have been dedicated to it and thus a variety of fruitful approaches have been proposed. Among these approaches to the state-space explosion problem, a well-known one is to symbolically represent system models and compute supervisors by using *binary decision diagrams* (BDDs) [8, 9]. In [6], a symbolic synthesis approach was indeed presented. After constructing the symbolic representation of the considered closed-loop system, the supervisor can be computed iteratively. However, the main problem of such monolithic approach is that during the reachability computations, the number of nodes in the intermediate BDDs might be significantly large, which quickly becomes a serious impediment to efficient computation, even though the final number of BDD nodes is manageable.

"To reach significant BDD reduction, it is crucial to explore the search space in an intelligent way. The key is to impose structure on the state-space exploration" [10]. Moreover, to realize such an intelligent state-space exploration, an important ingredient is the use of *partitioning techniques*, which was rigorously defined in [11] and used in a number of contexts [12, 13]. There also exists a number of papers dealing with the adaption of these techniques to the verification or synthesis task of SCT. In [10] and [14], a straightforward but non-trivial symbolic reachability approach was presented. The approach, based on the disjunctive partitioning technique, represents the monolithic transition relation of a fully synchronized DES by

a collection of partial transition relations. In [15], an improved version of the algorithm from [14] was proposed to deal with the restricted reachability state-space search where the set of forbidden states is expanded and excluded iteratively. However, these approaches are based on finite automata without the introduction of variables. At the time of writing this paper, to the knowledge of the authors, yet no work has been presented to adapt these partitioning techniques to DESs with variables.

In the context of the aforementioned research developments, motivated by the above remarks, in this paper, we present an alternative symbolic reachability approach. By making use of the disjunctive partitioning technique, the proposed approach partitions the transition relation of a considered system into a set of partial transition relations according to included events. Then those partial transition relations are selected systematically to perform the reachability computation. Generally, the approach can be easily adapted or extended to other alternative modeling formalisms such as [16, 17], even though in this paper, it is considered as an important supplement to our modeling and synthesis framework.

The paper has three main contributions:

- Suggesting a new symbolic way to partition DESs modeled as EFAs by using the disjunctive partitioning technique. The proposed approach constructs partial transition relations based on the alphabet of a DES. In addition, the correctness is formally proved.
- Proposing an alternative straightforward algorithm including the proof of correctness to realize the structural state-space exploration.
- Integrating this approach with our modeling framework and demonstrating the efficiency on a set of industrially relevant benchmark examples.

Generally, the approach proposed in this paper can be easily adapted or extended to other alternative modeling formalisms such as [16, 17], even though it is being as an important supplement to our modeling and synthesis framework in this paper. As stated in [6], the framework offers a number of advantages from different perspectives. By modeling a system based on EFAs, a relatively compact representation of complex systems with huge state-space can be potentially 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-scaled 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 [18, 19]). 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 ordinary modular automata. They can also easily be converted to controller programming languages e.g., SFC or ladder diagrams. EFA can also be easily be converted to well-known verification tools such as [20]. Also, from an engineering perspective, EFAs are attractive models due to their similarities to UML and state diagrams.

This paper is organized as follows: Section 2 provides some preliminaries including extended finite automata, binary decision diagrams and the supervisory control theory which are the fundamental concepts in our work. Section 3 illustrates the modeling of a simple resource allocation system configuration as EFAs and how to use BDDs to symbolically represent an

EFA. Section 4 and 5 present the main contributions by detailing the approach pursued by this paper. In particular, Section 4 describes how to construct the partial transition relations, while Section 5 shows an alternative symbolic algorithm, which is used to explore the state-space in a structural way. Section 6 shows the results of applying the approach to a set of benchmark examples. Finally, Section 7 concludes the paper by summarizing the main contributions and outlining some suggestions for the future work.

2 Preliminaries

In this section, some preliminaries used throughout the rest of the paper are provided and briefly explained.

2.1 Extended Finite Automata

An EFA, introduced in [7], is an augmentation of the ordinary finite automaton (FA) with guards predicates and actions 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 2.1 (Characteristic Function)**.** Let W be a finite set so that $W \subseteq U$, where U is the finite universal set. A *characteristic function* $\chi_W : U \to \mathbb{B}$ is defined by

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

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

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

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

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

where v^i denotes the i:th element in the binary m-tuple v. As we will see later, characteristic functions can also be used to represent BDDs.

Definition 2.2 (Extended Finite Automata)**.** An extended finite automaton E is a 6-tuple

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

where:

- $L^E \times V$ 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*;

- Σ^E is a non-empty finite set of events;
- $\mathcal{G} = \{\chi_W \mid W \in 2^V\}$ is the set of guard predicates over V;
- $\mathcal{A} = \{a \mid a\colon V \to V\}$ is a collection of action functions;
- $\to \subseteq L^E \times \Sigma^E \times \mathcal{G} \times \mathcal{A} \times L^E$ is the transition relation;
- $(\ell_0^E, v_0) \in L^E \times V$ 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 the 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

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

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

The transition relation can be 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 be explicitly written out in system transitions according to the following definition.

Definition 2.3 (Explicit State Transition Relation)**.** Let $E = \langle L^E \times V, \Sigma^E, \mapsto, (\ell_0^E, v_0)\rangle$ be an EFA. The explicit state transition relation of E is defined as

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

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

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

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

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

For brevity, we denote the explicit representation of a transition $\ell \xrightarrow{\sigma}_{g/a} \acute{\ell}$ by $\mapsto_{\ell \xrightarrow{\sigma}_{g/a} \acute{\ell}}$.

Definition 2.4 (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 2.5 (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 with the shared variables $v = (v^1, \ldots, v^n)$. The Extended Full Synchronous Composition (EFSC) of E_1 and E_2 is

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

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

1. $(\ell^{E_1}, \ell^{E_2}) \xrightarrow{\sigma}_{g/a} (\acute{\ell}^{E_1}, \acute{\ell}^{E_2}), \sigma \in \Sigma_1 \cap \Sigma_2$ 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:
   - $g = g_1 \wedge g_2$,
   - 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. $(\ell^{E_1}, \ell^{E_2}) \xrightarrow{\sigma}_{g/a} (\acute{\ell}^{E_1}, \acute{\ell}^{E_2}), \sigma \in \Sigma_1 \backslash \Sigma_2$ if
   $(\ell^{E_1}, \sigma, g, a, \ell_{E_1}) \in \rightarrow_{E_1}$ and $\ell^{E_2} = \acute{\ell}^{E_2}$;
3. $(\ell^{E_1}, \ell^{E_2}) \xrightarrow{\sigma}_{g/a} (\acute{\ell}^{E_1}, \acute{\ell}^{E_2}), \sigma \in \Sigma_2 \backslash \Sigma_1$ if
   $(\ell^{E_2}, \sigma, g, a, \ell_{E_2}) \in \rightarrow_{E_2}$ 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 [7] including the procedure of converting an EFA model to an FA model.

2.2 Binary Decision Diagrams

Binary Decision Diagrams (BDDs) [8, 9] 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 [14, 21, 22, 23].

Given a set of Boolean variables B, a BDD is a Boolean function $h\colon 2^B \rightarrow \{0, 1\}$, which can be expressed using Shannon's decomposition [24]:

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

where $h\,|_{b_j=0}$ and $h\,|_{b_j=1}$ refer to assignment 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 `op` between two BDDs h and g can be computed as

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

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 referring to the number of nodes excluding the terminal nodes. A BDD operation that is used extensively in our implementation is the *existential quantification* over 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 [25].

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 $b_1 \prec b_2 (b_2 \succ b_1)$. The variable ordering will impact the size of the BDD, however, finding an optimal variable ordering of a BDD is an NP-complete problem [26].

In the framework, a static BDD variable ordering is computed based on the method presented in [27]. 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 $|\Sigma^{A_1} \cap \Sigma^{A_2}|$. With respect to the DESs modeled as finite automata with variables, such as EFAs, the interaction of automata is affected not only by the alphabets but also the global variables (all variables in EFAs are global). To handle this issue, we add additional weights to the PCG by inspecting the guard predicates and actions functions of each transition. 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 [25, 28].

2.3 Supervisory Control Theory

As described in Section 1, SCT is a general theory to, given a plant P and specification Sp, automatically synthesize a minimally restrictive supervisor S restricting the plant towards the specification. Notice that if the plant is given as a number of sub-plants $P_1, \ldots, P_n$, the plant $P = P_1 \parallel \ldots \parallel P_n$. Similarly, $Sp = Sp_1 \parallel \ldots \parallel Sp_m$. For each sub-specification Sp_i, $\Sigma^{Sp_i} \subseteq \Sigma^P$, meaning the specification can not specify more than what the plant can achieve. Within the theory, some states of an automaton E, typically a specification, are identified as *marked states* Q_m^E. The marked states are these states that can be reached from the initial state. The set of marked states of a composed automaton $E_1 \parallel E_2$ is the cartesian product of the corresponding sets of marked states. In addition, events in the alphabet, Σ, can either

be controllable or uncontrollable. Thus the alphabet can be divided into two disjoint subsets, the controllable event set Σ_c, and the uncontrollable event set Σ_u.

In SCT, the supervisor of a DES to be synthesized is assumed be minimally restrictive, meaning that the plant is give the greatest amount of freedom to generate events. Moreover, there are two properties that the supervisor ought to have:

- *Controllability*: The supervisor S is never allowed to disable any uncontrollable event that might be generated by the plant P. This safety property can be formally expressed in terms of languages as:

$$L(Sp \parallel P)\Sigma_u \cap L(P) \subseteq L(Sp \parallel P).$$

- *Non-blocking*: The supervisor S guarantees that at least one marked state can be reached from every state in the closed-loop system $S \parallel P$. In terms of languages, the property can be described as:

$$L(Sp \parallel P) = \overline{L_m(Sp \parallel P)}.$$

The supervisory synthesis starts by generating the system $S_0 = P \parallel Sp$ and detecting a set of initiallly uncontrollable states. Through a series of reachability computations, forbidden states are iteratively excluded from Q^{S_0} until the remaining states are both controllable and nonblocking. The resulting system is the supervisor S and all of the included states are hereby called *safe states*, denoted by Q^S. For a more formal and detailed explanation of this approach, refer to [14].

3 A Motivation Example

As an example illustrating EFA modeling and BDD encoding, consider a following simple resource allocation system configuration in Fig. 1, which is borrowed from [29].

Example (A Simple Resource Allocation System)**.** The considered resource allocation system is constituted by two process types P_1 and P_2, each of which consists of three processing stages performing the linear structure. The system resource set is $\mathcal{R} = \{R_1, R_2, R_3\}$, with the capacity $C_i = 1, i = 1, 2, 3$. Each processing stage $\Xi_{ij} (i = 1, 2; j = 1, 2, 3)$ requests one single unit of one resource type.

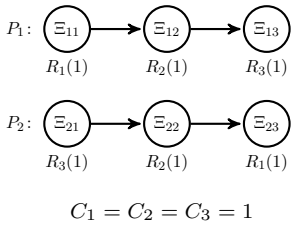


Figure 1: A flexibly automated robotic cell example in Section 2

In [30], an approach to modeling sequential allocation systems using EFA is presented. As a result, Figure. 2 shows the EFA model of this simple sequential resource allocation system,

where each EFA models one process type. To support the multiple instance execution, each EFA is defined to only have one location (both initial and marked) and all transitions labeled by events, for instance $\texttt{P}_1\texttt{\_book\_R}_1$, are added as self-loops. For each resource $R_i, i = 1, 2, 3$, the corresponding *resource variable* vR_i is declared to denote the number of available units of R_i. The domain of vR_i is denoted to be $\{0, \ldots, C_i\}$, where both of the initial and marked values of vR_i are equal to C_i. Moreover, for each processing stage except the last one of each process type. One *instance variable* is declared to denote the number of instances executing at the corresponding stage. Finally the resource and instance variables are used to construct the guards and actions. Guards are local formulae determining whether a process instance can advance to the next processing stage while actions are used to update the available resource units and instances for various processing stages. For a more detailed discussion, refer to [30]. In the sequel, we elaborate how to symbolically represent the transition relation of an EFA by using BDDs. For the sake of brevity, only the first EFA shown in Fig. 2 is considered.

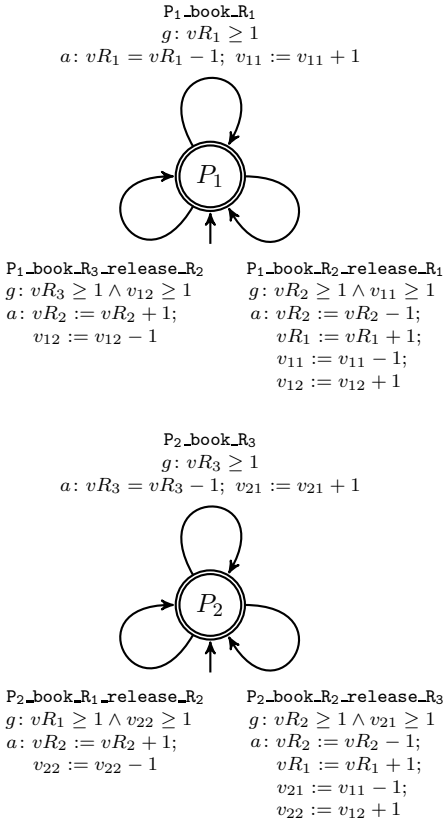


Figure 2: The EFA model of Example 3

As mentioned before, the characteristic functions can be used to represent the transition relation of an EFA. Based on Definition 2.1 and 2.3, the characteristic function of the explicit

state transition relation $\mapsto_{\ell \xrightarrow{\sigma}_{g/a} \acute{\ell}}$ can be rewritten as:

$$\begin{aligned}\chi_{\mapsto_{\ell \xrightarrow{\sigma}_{g/a} \acute{\ell}}}(b^{V^1},\ldots,b^{V^n},\acute{b}^{V^1},\ldots,\acute{b}^{V^n},b^L,\acute{b}^L,b^\Sigma) = \\ \Bigg(\bigvee_{(v,\acute{v})\in \mathsf{SAT}\mathcal{A}(a)|v\in\mathsf{SAT}\mathcal{G}(g)} \\ \bigwedge_{i=1}^{n}\left(b^{V^i}\leftrightarrow\theta(v^i)\ \wedge\ \acute{b}^{V^i}\leftrightarrow\theta(\acute{v}^i)\right)\Bigg)\ \wedge \\ b^L\leftrightarrow\theta(\ell)\ \wedge\ \acute{b}^L\leftrightarrow\theta(\acute{\ell})\ \wedge\ b^\Sigma\leftrightarrow\theta(\sigma),\end{aligned} \tag{6}$$

where b^Σ denotes the Boolean variables representing the alphabet while b^L and $\acute{b}^L$ are two different sets of Boolean variables representing the current and updated locations. For an EFA where n variables are defined, b^{V^i} and $\acute{b}^{V^i}$ denote the current and updated integer values of variables. In our framework, integers are represented in the two's complement system as array of BDDs [31]. In practice, it is very often that values of variables are defined as or updated to non-negative integers. In that case, for computational purposes, the BDD variable representing the sign-bit of a non-negative integer is omitted and thus only the magnitude is encoded. Besides, we assume that overflows are not allowed and thus we omit the cases where an overflow occurs. This is performed by removing all the variable assignments that result in values outside the domain of the variables. Consequently, the characteristic function of the transition relation of an EFA E will be

$$\begin{aligned}\chi_{\mapsto_E} = \bigvee_{\ell \xrightarrow{\sigma}_{g/a} \acute{\ell}\in\rightarrow_E} \chi_{\mapsto_{\ell \xrightarrow{\sigma}_{g/a} \acute{\ell}}}\ \wedge \\ \bigwedge_{i=1}^{n}\chi_{V^i}(b^{V^i})\ \wedge\ \bigwedge_{i=1}^{n}\chi_{V^i}(\acute{b}^{V^i}).\end{aligned} \tag{7}$$

Regarding the EFA model of process type P_1 of Example 3, Fig. 3 shows the corresponding transition relation and Table 1 shows the location and event encoding. The BDD variables in the figure are labeled with indices as follows:

$$\begin{aligned} b^\Sigma &= (b_1^\Sigma, b_0^\Sigma) = (\text{`1'}, \text{`0'}),\\ b^L &= (b_0^L) = (\text{`2'}),\ \acute{b}^L = (\acute{b}_0^L) = (\text{`3'}),\\ b^{vR_1} &= b_0^{vR_1} = (\text{`4'}),\ \acute{b}^{vR_1} = (\acute{b}_0^{vR_1}) = (\text{`5'}),\\ b^{vR_2} &= b_0^{vR_2} = (\text{`6'}),\ \acute{b}^{vR_2} = (\acute{b}_0^{vR_2}) = (\text{`7'}),\\ b^{vR_3} &= b_0^{vR_3} = (\text{`8'}),\ \acute{b}^{vR_3} = (\acute{b}_0^{vR_3}) = (\text{`9'}),\\ b^{v_{11}} &= b_0^{v_{11}} = (\text{`10'}),\ \acute{b}^{v_{11}} = (\acute{b}_0^{v_{11}}) = (\text{`11'}),\\ b^{v_{12}} &= b_0^{v_{12}} = (\text{`12'}),\ \acute{b}^{v_{12}} = (\acute{b}_0^{v_{12}}) = (\text{`13'}),\end{aligned}$$

where b_0 is the least significant bit.

Note that for all of the variables defined within the model, the Boolean variables representing the sign-bit of variable values are omitted since they are all non-negative. Besides, because of (7), the BDD does not contain the cases where variable values are beyond the domains.

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.

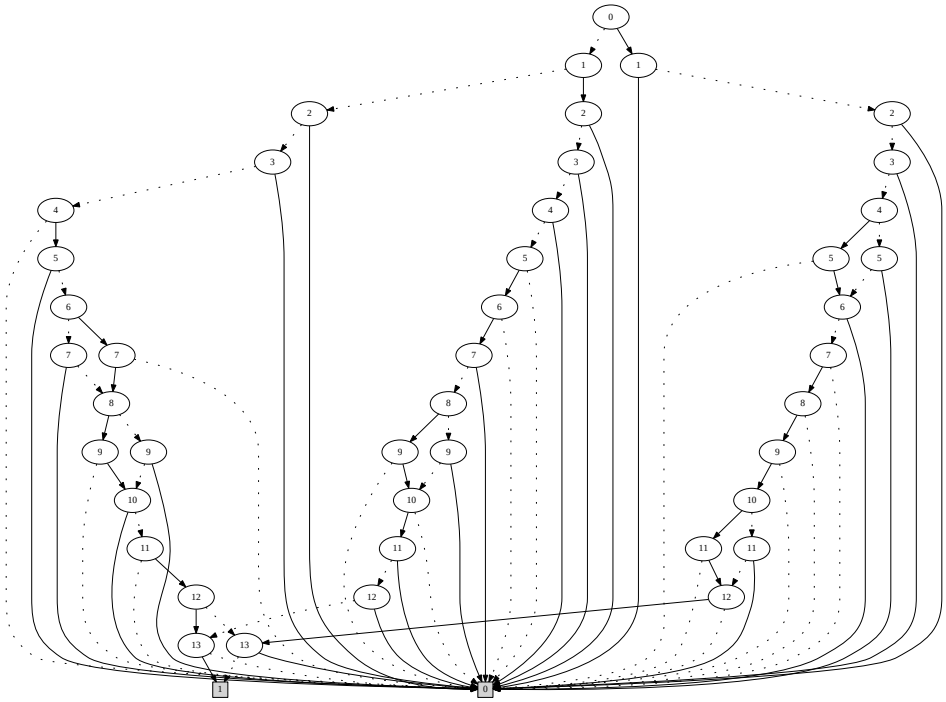


Figure 3: The corresponding BDD for the transition relation of the first EFA in Fig. 2

Table 1: Event and current location encoding for the first EFA in Fig. 2

Event	$b_1^\Sigma b_0^\Sigma$	Location	b_0^L
$\texttt{P}_1\texttt{\_book\_R}_1$	0 0	P_1	0
$\texttt{P}_1\texttt{\_book\_R}_2\texttt{\_release\_R}_1$	1 0		
$\texttt{P}_1\texttt{\_book\_R}_3\texttt{\_release\_R}_2$	0 1		

4 Partitioning of the full synchronous composition

Not surprisingly, reachability (co-reachability) computations turn out to be the bottle-neck of the SCT synthesis algorithm. Adopting the symbolic representation using binary decision diagrams, we can partially solve this problem. However, with more complicated DESs, the BDD representation of the monolithic transition relation, $\chi_{\mapsto_{S_0}}$, might be extremely large to be constructed. More importantly, even though such BDD representing the monolithic transition relation is managed to be constructed, the reachability computation may still suffer from the state-space explosion due to the large intermediate BDDs. In this section, we present a way to partition DESs modeled by EFAs by using the disjunctive partitioning technique and in Section 5, a straightforward but nontrivial algorithm, based on the partitioned BDDs, is presented to guide the state-space exploration.

Since S_0 is the synchronization of a number of sub-plants and sub-specifications in the form of EFAs, in all of the following computations we focus on $N \geq 2$ EFAs and let $\mathsf{E} = \{E_1, E_2, \ldots, E_N\}$.

Partitioning of the transition relation as introduced by [11] has become the standard guideline to alleviate the state-space problem. By splitting the transition relation into a set of *partial transition relations*, connected by either disjunction or conjunction. In [14, 10], an adaption of the disjunctive partitioning technique to finite automata under full synchronous composition was introduced. Informally, this automaton-based approach constructs the disjunctive partial transition relation of each sub-automaton directly from its transition relation and dependency set that can be derived from the PCG graph. For the formal definition and proof, refer to [10]. However, when it comes to the systems modeled by EFAs. The automaton-based partitioning approach does not work any more, because the interaction of automata not only depends on the shared events in the alphabet, but also the updates of global variables. In addition, the automaton-based approach cannot be able to keep track of the variables that are not updated. By definition, these variables need to remain the previous values during the transitions.

In this paper, we take a slightly different way and construct the partial transition relation based on events instead of automata. For each event $\sigma \in \Sigma^{\mathbf{E}}$ where $\mathbf{E} = E_1 \parallel \ldots \parallel E_N$, the corresponding disjunctive partial transition relation $\chi_{\overset{\sigma}{\mapsto}_{\mathbf{E}}}$ under full synchronous composition can be constructed in the following steps:

1. Compute a characteristic function of $\overset{\sigma}{\mapsto}_{\mathbf{E}^\dagger}$, denoted by $\chi_{\overset{\sigma}{\mapsto}_{\mathbf{E}^\dagger}}$ where $\mathbf{E}^\dagger = E_1^\dagger \parallel \ldots \parallel E_m^\dagger$ and $\sigma \in \Sigma^{E_1^\dagger} \cap \ldots \cap \Sigma^{E_m^\dagger}$.
2. Compute a characteristic function of $\overset{\sigma}{\mapsto}_{\mathbf{E}^\ddagger}$, denoted by $\chi_{\overset{\sigma}{\mapsto}_{\mathbf{E}^\ddagger}}$ where $\mathbf{E}^\ddagger = E_1^\ddagger \parallel \ldots \parallel E_{m'}^\ddagger$ and $\{E_1^\ddagger, \ldots, E_{m'}^\ddagger\} = \mathsf{E} \backslash \{E_1^\dagger, \ldots, E_m^\dagger\}$.

Regarding step 1, computing $\overset{\sigma}{\mapsto}_{\mathbf{E}^\dagger}$, two further steps need to be performed in advance:

- Compute $\chi'_{\overset{\sigma}{\mapsto}_{\mathbf{E}^\dagger}}$, which denotes the characteristic function of $\overset{\sigma}{\mapsto}_{\mathbf{E}^\dagger}$ excluding the action functions of EFA variables,
- Compute $\chi_{\overset{\sigma}{\mapsto}^{v}_{\mathbf{E}^\dagger}}$ denoting the update of EFA variables.

To compute $\chi'_{\overset{\sigma}{\mapsto}_{\mathbf{E}^\dagger}}$, we make use of the following two propositions.

Proposition 1. *For an EFA $E^\dagger$ and an event $\sigma \in \Sigma^{E^\dagger}$, the characteristic function representing the explicit transition relation through σ of $E^\dagger$, denoted by*

$$\chi_{\overset{\sigma}{\mapsto}_{E^\dagger}} = \chi_{\mapsto_{E^\dagger}} \wedge \chi_\sigma,$$

where χ_σ is the characteristic function of the event σ and $\chi_{\mapsto_{E^\dagger}}$ is the transition relation of the EFA $E^\dagger$.

Proposition 2. *Let $E_1^\dagger, \ldots, E_m^\dagger$ be $m \geq 2$ EFAs and $\sigma \in \Sigma^{E_1^\dagger} \cap \ldots \cap \Sigma^{E_m^\dagger}$. Then*

$$\chi'_{\overset{\sigma}{\mapsto}_{E_1^\dagger \parallel \ldots \parallel E_m^\dagger}} = \bigwedge_{k=1}^{m} (\exists\, \acute{b}^{V^i} . \chi_{\overset{\sigma}{\mapsto}_{E_k^\dagger}}). \tag{8}$$

Subsequently, we compute $\chi_{\overset{\sigma}{\mapsto}_{\mathbf{E}^\dagger}^{v}}$, which represents the update of EFA variables after the occurrence of σ. In the following computations, we focus on the update of a single variable between two EFAs and extend it to all variables for all EFAs in the model.

Definition 4.1 (Updated Transition Relation Through σ)**.** For an EFA E and a single variable v^i, the updated transition relation for v^i through σ, denoted by $\overset{\sigma}{\mapsto}_{v^i,E}$, can be defined as

$$\overset{\sigma}{\mapsto}_{v^i,E} = \{(\ell, v, \sigma, \acute{\ell}, \acute{v}) \mid \forall (\ell, v, \sigma, \acute{\ell}, \acute{v}) \in \overset{\sigma}{\mapsto}_{E} \wedge \acute{v}^i \neq v^i\}.$$

Recall that,from Definition 2.5, 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: $a_1^i = a_2^i$; both actions update v^i to the same value.
- T_1: $a^i(v) = a_1^i$ or $a^i(v) = a_2^i$.
- I_2: $a_2^i = \xi$; the first action updates v^i but not the second action.
- T_2: $a^i(v) = a_1^i$.
- I_3: $a_1^i = \xi$; the second action updates v^i but not the first action.
- T_3: $a^i(v) = a_2^i$.
- I_4: *otherwise*; either none of the actions updates v^i, or the actions update the variable to different values.
- T_4: $a^i(v) = v^i$.

Definition 4.2 (Interaction Transition Relation Through σ)**.** For two EFAs E_1 and E_2, and a variable v^i, the interaction transition relation through the event σ, denoted by $C_j(\overset{\sigma}{\mapsto}_{v^i,E_1\|E_2})$, can be defined as

$$\begin{aligned} C_1(\overset{\sigma}{\mapsto}_{v^i,E_1\|E_2}) \triangleq \{((\ell^{E_1}, \ell^{E_2}), v, \sigma, (\acute{\ell}^{E_1}, \acute{\ell}^{E_2}), \acute{v}) \mid \\ (\ell^{E_1}, v, \sigma, \acute{\ell}^{E_1}, \acute{v}) \in \overset{\sigma}{\mapsto}_{v^i,E_1} \wedge \\ (\ell^{E_2}, v, \sigma, \acute{\ell}^{E_2}, \acute{v}) \in \overset{\sigma}{\mapsto}_{v^i,E_2}\}, \end{aligned}$$

$$\begin{aligned} C_2(\overset{\sigma}{\mapsto}_{v^i,E_1\|E_2}) \triangleq \{((\ell^{E_1}, \ell^{E_2}), v, \sigma, (\acute{\ell}^{E_1}, \acute{\ell}^{E_2}), \acute{v}) \mid \\ (\ell^{E_1}, v, \sigma, \acute{\ell}^{E_1}, \acute{v}) \in \overset{\sigma}{\mapsto}_{v^i,E_1} \wedge \\ (\ell^{E_2}, v, \sigma, \acute{\ell}^{E_2}, \acute{\nu}) \in \overset{\sigma}{\mapsto}_{E_2} \setminus \overset{\sigma}{\mapsto}_{v^i,E_2}\}, \end{aligned}$$

$$\begin{aligned} C_3(\overset{\sigma}{\mapsto}_{v^i,E_1\|E_2}) \triangleq \{((\ell^{E_1}, \ell^{E_2}), v, \sigma, (\acute{\ell}^{E_1}, \acute{\ell}^{E_2}), \acute{v}) \mid \\ (\ell^{E_1}, v, \sigma, \acute{\ell}^{E_1}, \acute{\nu}) \in \overset{\sigma}{\mapsto}_{E_1} \setminus \overset{\sigma}{\mapsto}_{v^i,E_1} \wedge \\ (\ell^{E_2}, v, \sigma, \acute{\ell}^{E_2}, \acute{v}) \in \overset{\sigma}{\mapsto}_{v^i,E_2}\}, \end{aligned}$$

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

where $\acute{\nu} = (\acute{v}^1,\ldots,\acute{v}^{i-1},\xi,\acute{v}^{i+1},\ldots,\acute{v}^n)$.

Hence, by definition we have:

$$\chi_{\overset{\sigma}{\mapsto}_{v^i,E_1\|E_2}} = \bigvee_{j=1}^{4} \chi_{C_j(\overset{\sigma}{\mapsto}_{v^i,E_1\|E_2})}.$$

Based on Definition 4.2, $\chi_{\overset{\sigma}{\mapsto}_{\mathbf{E}^\dagger}^{v}}$ can be computed as follows:

$$\chi_{\overset{\sigma}{\mapsto}_{\mathbf{E}^\dagger}^{v}} = \bigwedge_{i=1}^{n} \chi_{\overset{\sigma}{\mapsto}_{v^i,\mathbf{E}^\dagger}}. \tag{9}$$

Moreover, $\chi_{\overset{\sigma}{\mapsto}_{\mathbf{E}^\dagger}}$ can be computed according to (8) and (9):

$$\chi_{\overset{\sigma}{\mapsto}_{\mathbf{E}^\dagger}} = \chi'_{\overset{\sigma}{\mapsto}_{\mathbf{E}^\dagger}} \wedge \chi_{\overset{\sigma}{\mapsto}_{\mathbf{E}^\dagger}^{v}}. \tag{10}$$

At this stage, we are done with step 1.

Remark. Recall from Definition 2.5, that if there exists an event σ, such that $\sigma \in \Sigma^{E_1}\backslash\Sigma^{E_2}$, on the occurrence of σ, E_2 would remain the previous location, i.e. $\forall \ell,\acute{\ell} \in L^{E_2}, \ell = \acute{\ell}$. On the other hand, the values of variables are updated according to the transitions labeled by σ in E_1.

Definition 4.3 (Remained Transition Relation of σ)**.** For an EFA E and an event $\sigma \notin \Sigma^E$, the remained transition relation of σ for E, denoted by $\overset{\sigma}{\curvearrowright}_E$ can be defined by

$$\overset{\sigma}{\curvearrowright}_E = \{(\ell,\sigma,\acute{\ell}) \mid \forall \ell,\acute{\ell} \in L^E \wedge \ell = \acute{\ell}\}.$$

Therefore, the characteristic function representing $\overset{\sigma}{\mapsto}_{\mathbf{E}^\ddagger}$ can be computed according to the following proposition.

Proposition 3. *Let $E_1^\ddagger,\ldots,E_{m'}^\ddagger$ be $m' \geq 2$ EFAs and $\sigma \notin \Sigma^{E_1^\ddagger} \cup \ldots \cup \Sigma^{E_{m'}^\ddagger}$, then*

$$\chi_{\overset{\sigma}{\mapsto}_{\mathbf{E}^\ddagger}} = \bigwedge_{k=1}^{m'} \chi_{\overset{\sigma}{\curvearrowright}_{E_k^\ddagger}}, \tag{11}$$

where $\mathbf{E}^\ddagger = E_1^\ddagger \parallel \ldots \parallel E_{m'}^\ddagger$.

At this moment, we have done the first two steps. Sequentially, computing the characteristic function of $\overset{\sigma}{\mapsto}_{\mathbf{E}}$, can be performed based on (10) and (11):

$$\chi_{\overset{\sigma}{\mapsto}_{\mathbf{E}}} = \chi_{\overset{\sigma}{\mapsto}_{\mathbf{E}^\dagger}} \wedge \chi_{\overset{\sigma}{\mapsto}_{\mathbf{E}^\ddagger}} \tag{12}$$

As the final step, we prove that the union of these partitioned characteristic functions $\chi_{\overset{\sigma}{\mapsto}_{\mathbf{E}}}$ based on each event $\sigma \in \Sigma^{\mathbf{E}}$ is equivalent to the characteristic function $\chi_{\mapsto_{\mathbf{E}}}$.

Here we introduce Definition 4.4 and 4.5, which were defined in [6].

Definition 4.4 (Extended Explicit Transition Relation $\looparrowright_{E_k}$)**.** For $N \geq 2$ EFAs $E_1, \ldots, E_N$, the extended explicit transition relation of E_k, denoted by $\looparrowright_{E_k}$, represents the explicit transition relation of E_k together with self-loops on all states with events that are not in the alphabet of E_k

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

Definition 4.5 (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 transition relations in E on which the variable v^i is updated:

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

Remark. In [6], for the symbolic monolithic approach, authors have proved that

$$\chi_{\looparrowright_{\mathbf{E}}} = \bigwedge_{i=1}^{n} \chi_{\looparrowright_{v^i,\mathbf{E}}} \wedge \chi'_{\looparrowright_{\mathbf{E}}}. \tag{13}$$

where n is the number of EFA variables of a model and $\mathbf{E} = E_1 \parallel \ldots \parallel E_N$.

Thus, to prove Theorem 4.1, we can prove that

$$\begin{aligned} \chi_{\looparrowright_{\mathbf{E}}} &= \bigwedge_{i=1}^{n} \chi_{\looparrowright_{v^i,\mathbf{E}}} \wedge \chi'_{\looparrowright_{\mathbf{E}}} \\ &= \bigvee_{\sigma \in \bigcup_{k=1}^{N} \Sigma^{E_k}} \chi_{\overset{\sigma}{\mapsto}_{\mathbf{E}}}. \end{aligned}$$

Lemma 4.1. *By Definition 4.4 and 4.5, (13) can be rewritten as*

$$\chi_{\mapsto_{\mathbf{E}}} = \bigwedge_{i=1}^{n} \chi_{\mapsto_{v^i,\mathbf{E}}} \wedge \chi'_{\mapsto_{\mathbf{E}}} \tag{14}$$

Proof. Since $E_1, \ldots, E_N$ are defined as the set of EFAs of a model, the second term (disjunct) of the definition of $\looparrowright_{\mathbf{E}}$ is omitted. Therefore we have.

$$\looparrowright_{\mathbf{E}} = \mapsto_{\mathbf{E}} .$$

The other two cases can be similarly proved. □

Theorem 4.1. *For $N \geq 2$ EFAs $E_1, \ldots, E_N$ and an n-tuple of variables $v^i, \ldots, v^n$, the following statement holds:*

$$\chi_{\mapsto_{\mathbf{E}}} = \bigvee_{\sigma \in \bigcup_{k=1}^{N} \Sigma^{E_k}} \chi_{\overset{\sigma}{\mapsto}_{\mathbf{E}}}, \tag{15}$$

where $\mathbf{E} = E_1 \parallel \ldots \parallel E_N$.

Proof. Based on Lemma 4.1, in order to prove (15), we can instead prove

$$\bigwedge_{i=1}^{n} \chi_{\mapsto_{v^i,\mathbf{E}}} \wedge \chi'_{\mapsto_{\mathbf{E}}} = \bigvee_{\sigma \in \bigcup_{k=1}^{N} \Sigma^{E_k}} \chi_{\overset{\sigma}{\mapsto}_{\mathbf{E}}}$$

Since

$$\begin{aligned}
&\bigwedge_{i=1}^{n} \chi_{\mapsto_{v^i,\mathbf{E}}} \wedge \chi'_{\mapsto_{\mathbf{E}}} \\
&= \bigvee_{\sigma \in \bigcup_{k=1}^{N} \Sigma^{E_k}} (\bigwedge_{i=1}^{n} \chi_{\overset{\sigma}{\mapsto}_{v^i,\mathbf{E}}}) \wedge \bigvee_{\sigma \in \bigcup_{k=1}^{N} \Sigma^{E_k}} \chi'_{\overset{\sigma}{\mapsto}_{\mathbf{E}}} \\
&= \bigvee_{\sigma \in \bigcup_{k=1}^{N} \Sigma^{E_k}} (\bigwedge_{i=1}^{n} \chi_{\overset{\sigma}{\mapsto}_{v^i,\mathbf{E}}} \wedge \chi'_{\overset{\sigma}{\mapsto}_{\mathbf{E}}}),
\end{aligned} \tag{16}$$

for each $\sigma \in \Sigma^{E_1} \cup \ldots \cup \Sigma^{E_N}$, there exists a subset of EFAs $\mathsf{E}^\dagger = \{E_1^\dagger, \ldots, E_m^\dagger\} \subseteq \{E_1, \ldots, E_N\}$ such that $\sigma \in \Sigma^{E_1^\dagger} \cap \ldots \cap \Sigma^{E_m^\dagger}$. In addition, we denote $\mathsf{E}^\ddagger = \{E_1^\ddagger, \ldots, E_{m'}^\ddagger\} = \{E_1, \ldots, E_N\} \backslash \{E_1^\dagger, \ldots, E_m^\dagger\}$. Let $\mathbf{E}^\dagger = E_1^\dagger \parallel \ldots \parallel E_m^\dagger$ and $\mathbf{E}^\ddagger = E_1^\ddagger \parallel \ldots \parallel E_{m'}^\ddagger$, by (12), (11), (10) and (9), we have

$$\begin{aligned}
&\bigvee_{\sigma \in \bigcup_{k=1}^{N} \Sigma^{E_k}} \chi_{\overset{\sigma}{\mapsto}_{\mathbf{E}}} \\
&= \bigvee_{\sigma \in \bigcup_{k=1}^{N} \Sigma^{E_k}} \big(\chi_{\overset{\sigma}{\mapsto}_{\mathbf{E}^\dagger}} \wedge \chi_{\overset{\sigma}{\mapsto}_{\mathbf{E}^\ddagger}}\big) \\
&= \bigvee_{\sigma \in \bigcup_{k=1}^{N} \Sigma^{E_k}} \big(\chi_{\overset{\sigma}{\mapsto}_{\mathbf{E}^\dagger}} \wedge \bigwedge_{k=1}^{m'} \chi_{\overset{\sigma}{\curvearrowright}_{E_k}}\big) \\
&= \bigvee_{\sigma \in \bigcup_{k=1}^{N} \Sigma^{E_k}} \big(\chi'_{\overset{\sigma}{\mapsto}_{\mathbf{E}^\dagger}} \wedge \chi_{\overset{\sigma}{\mapsto}^{v}_{\mathbf{E}^\dagger}} \wedge \bigwedge_{k=1}^{m'} \chi_{\overset{\sigma}{\curvearrowright}_{E_k}}\big) \\
&= \bigvee_{\sigma \in \bigcup_{k=1}^{N} \Sigma^{E_k}} \big(\chi'_{\overset{\sigma}{\mapsto}_{\mathbf{E}^\dagger}} \wedge \bigwedge_{i=1}^{n} \chi_{\overset{\sigma}{\mapsto}_{v^i,\mathbf{E}^\dagger}} \wedge \bigwedge_{k=1}^{m'} \chi_{\overset{\sigma}{\curvearrowright}_{E_k}}\big) \\
&= \bigvee_{\sigma \in \bigcup_{k=1}^{N} \Sigma^{E_k}} \Big(\big(\chi'_{\overset{\sigma}{\mapsto}_{\mathbf{E}^\dagger}} \wedge \bigwedge_{k=1}^{m'} \chi_{\overset{\sigma}{\curvearrowright}_{E_k}}\big) \wedge \\
&= \qquad\qquad \big(\bigwedge_{i=1}^{n} \chi_{\overset{\sigma}{\mapsto}_{v^i,\mathbf{E}^\dagger}} \wedge \bigwedge_{k=1}^{m'} \chi_{\overset{\sigma}{\curvearrowright}_{E_k}}\big)\Big) \\
&= \bigvee_{\sigma \in \bigcup_{k=1}^{N} \Sigma^{E_k}} \big(\chi'_{\overset{\sigma}{\mapsto}_{\mathbf{E}}} \wedge \bigwedge_{i=1}^{n} \chi_{\overset{\sigma}{\mapsto}_{v^i,\mathbf{E}}}\big)
\end{aligned} \tag{17}$$

Because of (16) and (17), Theorem 4.1 is proved. □

5 Efficient Reachability Computation

Following the previous section, we conclude that in order to design successful BDD-based reachability algorithms for large-scaled systems, it is vital to traverse the state-space in a structural way. As mentioned before, the interaction between two EFAs is not only affected by the shared events, but also the update of EFA variables. For instance, after an occurrence of an event, the values of variables are updated. These updated variables may lead to the guard of another transition from the second EFA to be evaluated to be true, even though this transition is labeled by a different event. When designing the algorithm, this issue should be taken into account. Otherwise, the algorithm might either explore the state-space in an incorrect way or is not an exhaustive exploration.

In this section, we present another algorithm which is structurally similar to the work-set algorithm with the difference that it works for the systems modeled as extended finite automata. Sequentially, the correctness of the algorithm is formally proved. Finally, we conclude the section by briefly commenting on alternative variations on this algorithm.

5.1 An Event-based Forward Reachability Algorithm

Recall from Section 4 that the system under full synchronous composition is split into a set of event-based BDDs by applying the disjunctive partitioning technique. As Algorithm 1 shows, taking as input the initial state and the set of partial transition relations of which each corresponds to each event, the algorithm maintains a set of active partial transition relations, W_k. For each iteration, one partial transition relation is selected and a saturated reachability search (Algorithm 2) is performed on it. If more reachable states are found, the *event and variable dependent transition relation sets* of σ, defined as follows, are appended to the workset. The algorithm terminates as long as there is no transition relation in W_k.

Definition 5.1 (Event Dependent Transition Relation Set of σ)**.** For $N \geq 2$ EFAs, $\mathsf{E} = \{E_1, \ldots, E_N\}$, the event dependent transition relation sets of σ, denoted by $D^e(\overset{\sigma}{\mapsto}_{E_1\|\ldots\|E_N})$ is defined as:

$$D^e(\overset{\sigma}{\mapsto}_{E_1\|\ldots\|E_N}) = \{\overset{\sigma'}{\mapsto}_{E_1\|\ldots\|E_N} |\ \sigma' \in D^e(\sigma) \wedge \sigma' \neq \sigma\},$$

where

$$\begin{aligned} D^e(\sigma) = \{\sigma' \mid \exists E_i \in \mathsf{E}, \ell, \acute{\ell}, \check{\ell} \in L^{E_i}, v, \acute{v}, \check{v} \in\ & V \\ \text{such that } (\ell, v, \sigma, \acute{\ell}, \acute{v}) \in \mapsto_{E_i} \wedge & (\acute{\ell}, \acute{v}, \sigma', \check{\ell}, \check{v}) \in \mapsto_{E_i}\}. \end{aligned}$$

Definition 5.2 (Variable Dependent Transition Relation Set of σ)**.** For $N \geq 2$ EFAs $E_1, \ldots, E_N$ and a n-tuple of variables $v^1, \ldots, v^n$, the variable dependent transition relation sets of σ, denoted by $D^v(\overset{\sigma}{\mapsto}_{E_1\|\ldots\|E_N})$, is defined as:

$$D^v(\overset{\sigma}{\mapsto}_{E_1\|\ldots\|E_N}) = \{\overset{\sigma'}{\mapsto}_{E_1\|\ldots\|E_N} |\ \sigma' \in D^v(\sigma) \wedge \sigma' \neq \sigma\},$$

where

$$\begin{aligned} D^v(\sigma) = \{\sigma' \mid \exists (\ell, \sigma', g, a, \acute{\ell}) \in \rightarrow_{E_1\|\ldots\|E_N}, \forall \chi_{v^i} \in g \\ \text{such that } \exists (\ell, v, \sigma, \acute{\ell}, \acute{v}) \in \overset{\sigma}{\mapsto}_{E_1\|\ldots\|E_N} \wedge v^i \neq \acute{v}^i)\} \end{aligned}$$

Algorithm 1 Event-based Forward Reachability

input $q_0 := (\ell_0^{E_1} \times \ldots \times \ell_0^{E_N} \times v_0)$,
$W_0 := \{\overset{\sigma}{\mapsto}_{E_1\|\ldots\|E_N} | \ \forall \sigma \in \Sigma^{E_1} \cup \ldots \cup \Sigma^{E_N}\}$
let $Q_0 := \{q_0\}, k := 0$
repeat
  Pick and remove $\overset{\sigma}{\mapsto}_{E_1\|\ldots\|E_N} \in W_k$
  $k := k + 1$
  $Q_k := Q_{k-1} \cup \texttt{Reachability}(Q_{k-1}, \overset{\sigma}{\mapsto}_{E_1\|\ldots\|E_N})$
  if $Q_k \neq Q_{k-1}$ **then**
    $W_k := W_{k-1} \cup D^e(\overset{\sigma}{\mapsto}_{E_1\|\ldots\|E_N}) \cup D^v(\overset{\sigma}{\mapsto}_{E_1\|\ldots\|E_N})$
  end if
until $W_k = \emptyset$
return Q_k

Algorithm 2 Reachability

input $Q, \overset{\sigma}{\mapsto}_{E_1\|\ldots\|E_N}$
let $Q_0 := Q, k := 0$
repeat
  $k := k + 1$
  $Q_k := Q_{k-1} \cup \{(\acute{q}, \acute{v}) \mid (\acute{q}, \acute{v}) \in Q_k$ such that
    $\exists (q, v) \in Q_{k-1} \wedge (q, v, \sigma, \acute{q}, \acute{v}) \in \overset{\sigma}{\mapsto}_{E_1\|\ldots\|E_N}\}$
until $Q_k = Q_{k-1}$
return Q

5.2 The Proposed Algorithm is Correct

In this sub-section, we establish the correctness of the algorithm. The algorithm is based on a rather involved scheme of enabling and disabling a subset of partial transition relations. To prove the algorithm correctness, the important point is to be able to guarantee that what is output by the algorithm really is the set of all reachable states. The proof strategy of this event-based algorithm is similar to that of the original workset algorithm [10], which is automaton-based.

Lemma 5.1. *At iteration k of the event-based forward reachability algorithm, the set W_k contains all active transition relations which are sufficient for the current reachable state set Q_k to reach more states in one step.*

Proof. The lemma can be proved by induction.

The basic step: We start by showing that the lemma holds before the first loop iteration, when $k = 0$. The current reachable state set Q_0 consists of only the initial state while W_0 contains all of partial transition relations. Therefore, the partial transition relations in W_k contains all partial transition relations sufficient to reach more states from Q_0 in one step (trivially).

The inductive step: We assume that at iteration k, the lemma holds. Next we prove that the lemma still holds at the iteration $k+1$. For this step, we can prove by contradiction. We assume that at iteration $k+1$, there exists one transition relation $\overset{\sigma}{\mapsto}_{E_1\|...\|E_N} \notin W_k$ but more reachable states can be found from Q_k with $\overset{\sigma}{\mapsto}_{E_1\|...\|E_N}$. And then we prove such assumption can lead to a contradiction. At the iteration $k+1$, $\overset{\sigma}{\mapsto}_{E_1\|...\|E_N} \notin W_k$ means that the transition relation has been removed before and has not been put back into the transition relation set. Since at iteration k, the lemma holds, that is W_k contains all transition relations leading to states reachable from Q_k. Here we separate the relations in W_k into two sets. The first set consists of all relations where more reachable states can be found by Q_k, denoted by S^R while the second set is $W_k \backslash S^R$.

(i) Firstly, let us consider the case that $\overset{\sigma}{\mapsto}_{E_1\|...\|E_N}$ is selected and removed before the iteration k. At the iteration k of the algorithm, suppose a relation $\overset{\sigma'}{\mapsto}_{E_1\|...\|E_N}$ is removed at line 4 of Algorithm 1. Next the value of k is increased by 1 and the current reachable state is now Q_{k-1}. Then an exhaustive reachability search is performed on the relation. If $\overset{\sigma'}{\mapsto}_{E_1\|...\|E_N} \in S^R$, new reachable states can be found by Algorithm 2 and then the algorithm puts $D^v(\overset{\sigma'}{\mapsto}_{E_1\|...\|E_N})$ and $D^e(\overset{\sigma'}{\mapsto}_{E_1\|...\|E_N})$ into the relation set. Otherwise, the reachable state set is unchanged, i.e. $Q_k = Q_{k-1}$. By assumption, the relation $\overset{\sigma}{\mapsto}_{E_1\|...\|E_N}$ is not in W_{k+1}. If $\overset{\sigma'}{\mapsto}_{E_1\|...\|E_N} \in S^R$, only the transition relations in $D^e(\overset{\sigma'}{\mapsto}_{E_1\|...\|E_N})$ or $D^e(\overset{\sigma'}{\mapsto}_{E_1\|...\|E_N})$ might be qualified and added in S^R at the iteration $k+1$ but $\overset{\sigma}{\mapsto}_{E_1\|...\|E_N}$ is in neither one of them. In this case, it can be deduce for $\overset{\sigma}{\mapsto}_{E_1\|...\|E_N} \notin W_{k+1}$, there does not exist more states reachable from Q_{k+1}, which contradicts the assumption. If $\overset{\sigma'}{\mapsto}_{E_1\|...\|E_N} \notin S^R$, the reachable state set is unchanged since the iteration k. Then all transition relations leading to states reachable from Q_{k+1} are all in W_{k+1}. This also contradicts the assumption we made before.

(ii) Secondly, we consider the case that $\overset{\sigma}{\mapsto}_{E_1\|...\|E_N}$ is selected and removed at the iteration k. Similar as the first case, here W_k is separated as two set: S^R and $W_k \backslash S^R$. If $\overset{\sigma}{\mapsto}_{E_1\|...\|E_N} \in S^R$, there exists some states which can be reached in one step from Q_k. But for the next

iteration $k+1$, there cannot be more states found in $\overset{\sigma}{\mapsto}_{E_1\|\ldots\|E_N} \in S^R$, since Algorithm 2 is a exhaustive search. This contradicts the assumption that for $\overset{\sigma}{\mapsto}_{E_1\|\ldots\|E_N} \notin W_{k+1}$, there exists more reachable states leading to states reachable from Q_{k+1} in one step. If $\overset{\sigma}{\mapsto}_{E_1\|\ldots\|E_N} \notin S^R$, then this transition relation is removed at iteration k and the reachable state set is unchanged. At iteration $k+1$, $\overset{\sigma}{\mapsto}_{E_1\|\ldots\|E_N} \notin W_{k+1}$ cannot be used to find more reachable states from the old reachable state set from the iteration k, which again contradicts the assumption.

To sum up, based on the two cases above. We can prove that the lemma holds at the iteration $k+1$. □

Theorem 5.1. *The event-based forward reachability algorithm terminates in a finite time and the output is the complete set of reachable states.*

Proof. In each step of the event-based forward reachability algorithm, zero or more states are added to the set of reachable states. The total set of states is finite, so the set of reachable states can grow only finitely many times. In each iteration when no new states are found, exactly one transition relation is removed from the set of active relations W_k. The set W_k is also finite and hence the algorithm will terminate in finite time.

Assume now that the algorithm has terminated. By Lemma 5.1, we know that W_k holds all transition relations which can be used to find all states reachable from Q_k. However, since the algorithm has terminated, W_k is empty and thus no more states are reachable from Q_k, i.e. Q_k is the set of all reachable states.states. □

5.3 Several Variants Of The Algorithm

One drawback of the aforementioned event-based algorithm is that in some cases more iterations are needed to reach the fixed point in the reachability task, especially for large-scaled systems involving a large number of events in the alphabet. To make such reachability computations more efficient, thus reducing the number of iterations, it is possible to combine multiple partial transition relations. This idea is originally presented in the literature [11], referring to as *clustering*. Based on this general principle, here we briefly introduce and comment on three intuitive variants of the algorithm, which have been implemented:

- Greedy clustering with a static threshold: in this variant, a static threshold is set manually to denote the maximal number of BDD nodes for a cluster. As soon as the construction of partial transition relations is completed, multiple partial transition relations are randomly chosen and combined until the number of nodes in the joint BDD exceeds this threshold. The performance of this approach is largely dependent on the value of the static threshold.

- Clustering based on the alphabet of each EFA: in this approach, the combination of multiple partial transition relations is performed based on the alphabet of each EFA. In other words, corresponding to each EFA $E_i \in \mathsf{E}, \mathsf{E} = \{E_1, \ldots, E_N\}$ in the model, one cluster, denoted by $C(E_i)$, is constructed, which is defined is as follows:

  $$C(E_i) = \{\overset{\sigma}{\mapsto}_{E_1\|\ldots\|E_N} |\ E_i \in \mathsf{E} \wedge \forall\sigma \in \Sigma^{E_i}\}.$$

  Empirically, this variant of the algorithm seems work well for real-world applications and shows a better performance than others.

- Clustering based on the update of variables: for each EFA variable v^i in the n-tuple variables $v^i, \ldots, v^n$ of a system $\mathsf{E} = \{E_1, \ldots, E_N\}$, a cluster, denoted by $C(v^i)$, can be constructed based on the update of v^i at any EFA $E_i \in \mathsf{E}$:

$$C(v^i) = \{\overset{\sigma}{\mapsto}_{E_1 \| \ldots \| E_N} | \ \exists E_i \in \mathsf{E}, \exists (\ell, \sigma, g, a, \acute{\ell}) \in \rightarrow_{E_i} \text{ such that } \forall a^i(v) \in a \wedge a^i(v) \neq \xi\}.$$

6 Case Studies

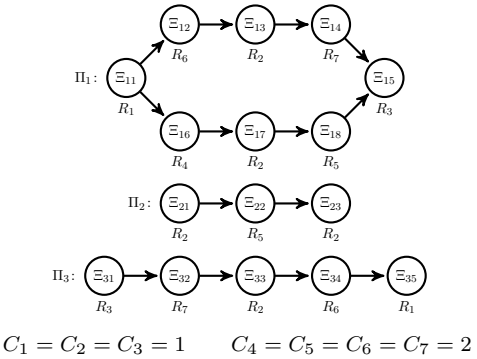
In this section, the proposed algorithm is applied to a set of academic and industrial benchmark examples to demonstrate the efficiency.

6.1 Benchmark Examples

We first present the benchmark examples to be analyzed. Since the proposed partitioning method and reachability algorithm also applies to systems modeled as ordinary automata, we also include some examples of this category.

6.1.1 Resource Allocation System (RAS)

Consider a flexible manufacturing system, shown in Fig. 4, introduced in [32]. The RAS is constituted of three process types and seven resource types $\mathcal{R} = \{R_1, \ldots, R_7\}$ with the corresponding capacities and resource request shown in Fig. 4. Such RAS can be modeled as three EFAs of which each represents one process type $\Pi_i, i = 1, 2, 3$. For more exposition of the modeling approach, refer to [30]. The task is to synthesize a maximally permissive non-blocking supervisor from such EFA model to avoid dead-lock situation.



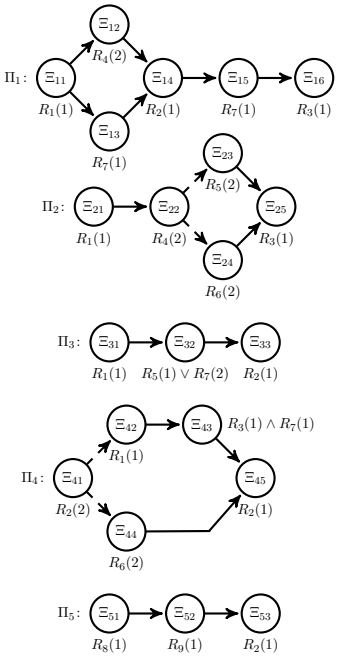
$C_1 = C_2 = C_3 = 1 \qquad C_4 = C_5 = C_6 = C_7 = 2$

Figure 4: The flexible manufacturing system configuration in [32]

6.1.2 Resource Allocation Systems with Error Handling (RAS-EH)

Because in many contemporary resource allocation applications, it is necessary to have some form of error handling. As a second RAS example, shown in Fig. 5, we consider the situation where errors can occur at some processing stages. In particular, we suppose that errors might occur at Ξ_{22} or Ξ_{41} and thus the repair needs to be performed. To model the error handling mechanism, alternative branches labeled by uncontrollable events are introduced after the

necessary processing stages. Being different from the above RAS where all of the resource allocation events are controllable and can be disabled by the supervisor, the supervisor of the RAS with the error handling cannot influence which branch to choose. Hereby, it must assume that there exists a non-blocking path for every branch it may take dynamically.



$C_1 = 1, C_2 = 4, C_3 = C_5 = 2,$
$C_4 = C_6 = C_7 = C_8 = C_9 = 2$

Figure 5: A extension of RAS of Fig. 4 with error handling

6.1.3 Ball Sorting Process (BSP)

The process under control of this example is shown in Fig. 6. The goal is to sort steel balls that are placed in a queue at the process gate. Each ball enters the process one the gate is open and then it moves to the first lift. The lift brings the ball to the measuring station. Once the size of the ball is measured it is pushed out and moves up to the second lift. Depending on the size, the ball is transported to the first or second level. At the destination level, each ball is pushed out of the lift and moves towards the position where it can be picked up by the rotating hand. The robot hand is responsible for transporting balls back to the home position in front of the gate. Two types of balls are handled by the process, the small and the big ones. The small balls should be moved to the first level and the big balls to the second level. In [33], an EFA model for its execution model in the IEC 61449 standard has been generated and used to developed the control logic.

6.1.4 Automated Guided Vehicles (AGV)

A model of a flexible manufacturing cell was introduced in [34]. The cell consists of three workstations, two input stations and one output station and five Automated Guided Vehicles (AGVs), each one responsible to route some parts through the cell by following certain paths.

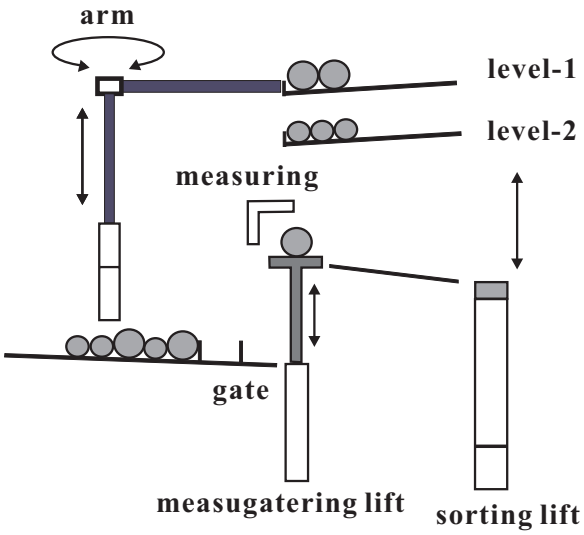


Figure 6: Ball Sorting Process

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

6.1.5 Parallel Manufacturing Example (PME)

The Parallel Manufacturing Example, introduced in [35] consists of three manufacturing units running in parallel. The system is modeled in three layers in a hierarchical interface-based manner. See the thesis [35] for more information.

6.1.6 Cat and Mouse Tower (CMT)

Consider the cat and mouse problem presented by [1]. Assume that these five rooms maze is just the first level of a tower composed by n identical levels. A controllable bidirectional passageway connects room j of level $5 * i + j$ to room j of $5 * i + j + 1$ for ($i = 0, 1, 2, 3, \ldots$ and $j = 1, 2, 3, 4, 5$). The first level is only connected with the second and the last only with the last-but-one. There are initially k cats in room 1 of the first level and k mice in room 5 of the last level.

There are various ways to model this problem but they are beyond the scope of this paper. Here we adopt the model presented in [22]. With a few modifications, the presented model can be converted to the corresponding EFA model, which results in a compact representation comparing to its ordinary automata counterpart. For example, Fig. 7 shows the EFA which models room 2 of level 2, for $n = 3$ and $k = 3$. Two variables $c\_\ell_2 r_2$ and $m\_\ell_2 r_2$ are declared to denote the number of cats and mouse of this room respectively.

6.1.7 Extended Dinning Philosophers (EDP)

The case generalizes the classic dining philosophers problem by allowing philosophers to go through a number of intermediate states after picking up the left fork but before picking up the right one. Assuming that there are n philosophers sitting around the table and each philosopher needs to go through k intermediate states to start to dine. Besides, it is assumed

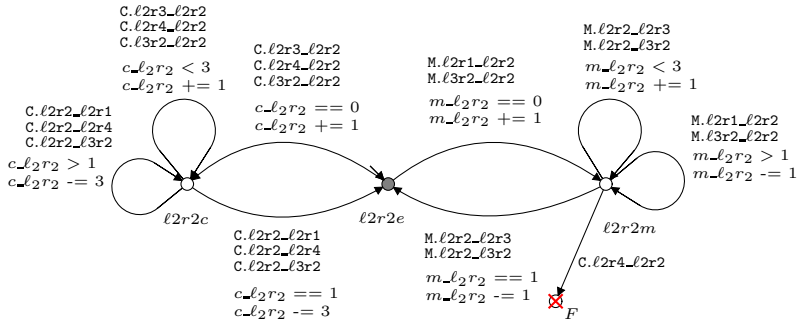
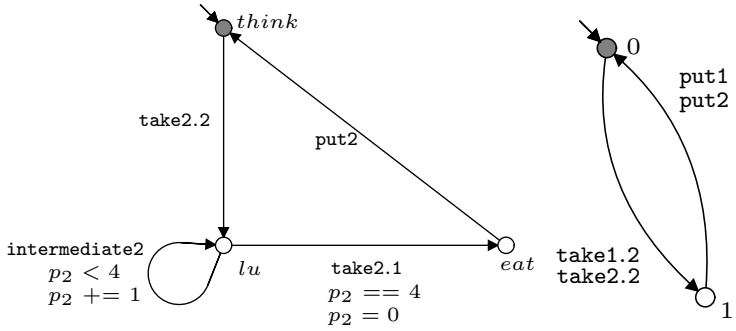


Figure 7: EFA model for room 2 of level 2 ($k = 3$, $n = 3$)

that the event "philosopher i takes the left fork" is modeled as an uncontrollable event if i is even. Design a maximally permissive non-blocking supervisor.

For this problem, we consider an EFA (plant) for each philosopher and an ordinary automaton (specification) for each fork. Fig. 8 shows the EFA of Philosopher 2 (plant) and an automaton of Fork 2 (specification), for 2 philosophers and 4 intermediate steps. It can be observed that for each philosopher, the number of intermediate steps are modeled as a variables and corresponding guards and actions are constructed and attached to the transitions. The EFA models shows the same behavior as the ordinary automata in [22] but with the compact and trackable representation.



(a) Philosopher 2 (b) Fork 2

Figure 8: Models of Philosopher 2 (plant) and Fork 2 (specification)

6.2 Results

The proposed partitioning and traversal algorithm in this paper has been implemented and integrated in the supervisory control tool *Supremica* [36] which uses *JavaBDD* [37] as the BDD package. Experiments are carried out on a standard PC (Intel Core 2 Quad CPU @ 2.4 GHz and 3GB RAM) running Windows 7 and the result is shown in Table 2. For each benchmark example, the maximally permissive supervisor generated by the algorithm is reachable, non-blocking and controllable.

It can be observed that both the monolithic and partitioning approaches can handle AGV, for which the number of reachable states is up to 10^7. However, by comparing the maximal number of BDD nodes during the reachability computation, which can express the maximal

Table 2: Comparison Between Two Symbolic Synthesis Approaches

			BDD Monolithic Approach		BDD Partitioning Approach	
Model	Reachable States	Supervisor states	BDD Peak (R)	Computation Time (s)	BDD Peak (R)	Computation Time (s)
RAS	1.19×10^4	0.88×10^4	2826	0.49	215	0.13
RAS-EH	1.84×10^6	0.68×10^6	42314	18.67	2275	0.87
BSP	706	706	M.O.	–	16640	10.48
AGV	2.29×10^7	1.15×10^7	9663	3.60	1001	0.87
PME	8.13×10^5	0.46×10^5	1022	0.24	225	0.14
CMT (1,5)	605	579	447	0.01	255	0.02
CMT (5,1)	1056	76	635	0.06	590	0.04
CMT (1,7)	1198	1156	801	0.10	321	0.39
CMT (7,1)	2710	155	1074	0.15	974	0.06
CMT (3,3)	2.96×10^5	1.64×10^5	16770	24	5070	4.1
CMT (5,5)	1.07×10^{10}	3.15×10^9	M.O.	–	65102	79
EDP (5,10)	167761	1596	1157	0.5	134	0.4
EDP (5,50)	3.46×10^8	1.38×10^5	7743	1.25	178	0.55
EDP (5,100)	1.05×10^{10}	1.05×10^6	–	T.O.	192	1.3
EDP (5,200)	3.28×10^{11}	8.20×10^6	–	T.O.	206	6.5

M.O. indicates memory out during reachability search (due to large intermediate BDDs) and T.O. indicates time out (10 min).

memory usage, the monolithic approach needs 9 times more memory than the partitioning approach. Regarding the example Ball Sorting Process, event though the final number of supervisor states is only 706, the intermediate BDDs during the state-space exploration, on the other hand, are large due to the high interactive complexity of the system. The monolithic approach fails to explore the state-space while the partitioning approach can survive and synthesize the supervisor within 11 seconds. As mentioned before, since the proposed partitioning algorithm is based on the alphabet which might contain a large number of events, more iterations than the standard algorithm are needed to reach the final fixed point. However, the intermediate BDDs produced during the computation are smaller, leading to improved memory and runtime efficiency. Finally, with respect to the last two benchmark examples, Cat and Mouse Tower and Extended Dining Philosophers, the partitioning approach can also handle some relatively large problem instances with the acceptable time. However, with the values of parameters growing, both the computation time and memory used increase rapidly.

7 Conclusions

In this paper, we presented an alternative symbolic approach to large-scaled systems modeled as extended finite automata. The proposed approach first partitions the closed-loop system under full synchronous composition based on the disjunctive partitioning technique and then depends on an efficient algorithm to explore the state-space in a structural way.

The proposed approach has been implemented and integrated into the prior work. Besides, it is applied to a set of academic and industrial examples to demonstrate the efficiency. Overall, the whole framework provides the convenience for users to model systems and obtain control functions in the same model domain. All computations are performed symbolically by BDDs, which are transparent and the only interface users deal with is the EFA framework.

There are several directions towards which we could extend and improve the framework in future. For instance, there is a potential to improve the BDD variable ordering. It is

believed that a sub-optimal but well-functioning BDD variable ordering can still dramatically enhance the performance of the symbolic algorithm proposed in this paper and thus larger and more complicated systems can be handled. Moreover, it is possible to combine our symbolic approach with some sophisticated synthesis techniques, such as compositional techniques, to improve the efficiency of the synthesis task further.

References

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

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

[3] K. Andersson, J. Richardsson, B. Lennartson, and M. Fabian, "Hierarchical Control Applying Supervisor Synthesis and relation Extraction," in *International Congress ANIPLA 2006 - Methodologies for Emerging Technologies in Automation*, Rome, Italy, Nov. 2006.

[4] S. Balemi, G. Hoffmann, P. Gyugyi, H. Wong-Toi, and G. Franklin, "Supervisory control of a rapid thermal multiprocessor," *IEEE Transactions on Automatic Control*, vol. 38, no. 7, pp. 1040–1059, 1993.

[5] 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.

[6] S. Miremadi, B. Lennartson, and K. Åkesson, "A BDD-based Approach for Modeling Plant and Supervisor by Extended Finite Automata," *accepted for IEEE Transactions on Control Systems Technology*, 2011.

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

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

[9] R. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Transactions on Computers*, vol. 35, no. 8, pp. 677–691, 1986.

[10] M. Byröd, B. Lennartson, A. Vahidi, and K. Åkesson, "Efficient Reachability analysis on Modular Discrete-Event Systems using Binary Decision Diagrams," in *8th international Workshop on Discrete Event Systems, WODES'06*, Ann Arbor, MI, USA, July 2006, pp. 288–293.

[11] J. R. Burch, E. M. Clarke, D. E. Long, K. L. Mcmillan, and D. L. Dill, "Symbolic Model Checking for Sequential Circuit Verification," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, vol. 13, no. 4, pp. 401–424, 1994.

[12] J. Geldenhuys and A. Valmari, "Techniques for smaller intermediary bdds," in *12th International Conference on Concurrency Theory*, ser. Lecture Notes in Computer Science, K. Larsen and M. Nielsen, Eds. Springer Berlin / Heidelberg, 2001, vol. 2154, pp. 233–247.

[13] G. Cabodi, P. Camurati, L. Lavagno, and S. Quer, "Disjunctive Partitioning and Partial Iterative Squaring: An Effective Approach for Symbolic Traversal of Large Circuits," in *34th Design Automation Conference*. ACM Press, 1997, pp. 728–733.

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

[15] Z. Fei, K. Åkesson, and B. Lennartson, "Symbolic reachability computation using the disjunctive partitioning technique in supervisory control theory," in *IEEE International Conference on Robotics and Automation*, 2011, pp. 4364–4369.

[16] Y.-L. Chen and F. Lin, "Modeling of discrete event systems using finite state machines with parameters," in *IEEE International Conference on Control Applications, CCA'00*, Sept. 2000, pp. 941–946.

[17] B. Gaudin and P. H. Deussen, "Supervisory Control on Concurrent Discrete Event Systems with Variables," in *American Control Conference, 2007. ACC '07.* New York, NY, USA: IEEE, 2007, pp. 4274 – 4279.

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

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

[20] A. Voronov and K. Akesson, "Verification of process operations using model checking," in *5th IEEE International Conference on Automation Science and Engineering*, aug. 2009, pp. 415 –420.

[21] 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 *5th IEEE Symposium on Logic in Computer Science, 1990. LICS '90,*, June 1990, pp. 428–439.

[22] S. Miremadi, K. Åkesson, M. Fabian, A. Vahidi, and B. Lennartson, "Solving two supervisory control benchmark problems using Supremica," in *9th International Workshop on Discrete Event Systems, 2008, WODES 08.*, May 2008, pp. 131–136.

[23] C. Ma and W. M. Wonham, "STSLib and its application to two benchmarks," in *9th International Workshop on Discrete Event Systems, 2008, WODES'08.*, May 2008, pp. 119–124.

[24] C. E. Shannon, "A Mathematical Theory of Communication," *The Bell System Technical Journal*, vol. 27, pp. 379–423,625–656, 1948.

[25] H. Andersen, "An introduction to binary decision diagrams," Department of Information Technology, Technical University of Denmark, Tech. Rep., 1999.

[26] B. Bollig and I. Wegener, "Improving the Variable Ordering of OBDDs Is NP-Complete," *IEEE Transactions on Computers*, vol. 45, no. 9, pp. 993–1002, 1996.

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

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

[29] S. A. Reveliotis, *Real-Time Management of Resource Allocation Systems: A Discrete Event Systems Approach (International Series in Operations Research & Management Science)*. Springer, December 2004.

[30] Z. Fei, S. Miremadi, and K. Åkesson, "Modeling sequential resource allocation systems using extended finite automata," in *7th IEEE International Conference on Automation Science and Engineering*, Trieste, 2011, pp. 444–449.

[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. Ezpeleta, J. Colom, and J. Martinez, "A petri net based deadlock prevention policy for flexible manufacturing systems," *IEEE Transactions on Robotics and Automation*, vol. 11, no. 2, pp. 173 –184, 1995.

[33] G. Cengi, "A Control Software Development Method Using IEC 61499 Function Blocks , Simulation and Formal Verification," in *the 17th IFAC World Congress*, 2008.

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

[35] R. J. Leduc, "Hierarchical interface-based supervisory control," Ph.D. dissertation, Department of Electrical and Computer Engineering, University of Toronto, 2002.

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

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