Symbolic State-Space Exploration and Guard Generation in Supervisory Control Theory

Zhennan Fei, Sajed Miremadi, Knut Åkesson, and Bengt Lennartson

Automation Research Group, Department of Signals and Systems
Chalmers University of Technology, Göteborg, Sweden
{zhennan,miremads,knut,bengt.lennartson}@chalmers.se

Abstract. Supervisory Control Theory (SCT) is a model-based framework for automatically synthesizing a supervisor that minimally restricts the behavior of a plant such that given specifications is fulfilled. The main obstacle which prevents SCT from having a major industrial breakthrough is that the supervisory synthesis, consisting of a series of reachability tasks, suffers from the state-space explosion problem. To alleviate this problem, a well-known strategy is to represent and explore the state-space symbolically by using Binary Decision Diagrams. Based on this principle, an alternative symbolic state-space traversal approach, depending on the disjunctive partitioning technique, is presented in this paper. In addition, the approach is adapted to the prior work, the guard generation procedure, to extract compact propositional formulae from a symbolically represented supervisor. These propositional formulae, referred to as guards, are then attached to the original model, resulting in a modular and comprehensible representation of the supervisor.

Keywords: Supervisory control theory, State-space exploration, Binary decision diagrams, Partitioning techniques, Propositional formulae.

1 Introduction

The analysis of reactive systems has been paid much attention by researchers and scientists in the computer science community. One of the classic methods to analyze reactive systems is utilizing formal verification techniques, such as model checking, to verify whether the considered system fulfills specifications. Nevertheless, from the control engineering point of view, instead of verifying the correctness of the system, a controller which guarantees that the system always behaves according to specifications is preferred. Supervisory Control Theory (SCT) [1,2] provides such a control-theoretic framework to design a device, called the *supervisor*, for reactive systems, referred to as Discrete Event Systems (DESs). Given the model of a DES to be controlled, the *plant*, and the intended behavior, the *specification*, the supervisor can be automatically synthesized, guaranteeing that the closed-loop system fulfills given specifications. SCT has been applied for various applications in different areas such as automated manufacturing lines and embedded systems [3,4,5].

Generally, a supervisor is a function that, given a set of events, restricts the plant to execute desired events according to the specification. A typical issue is how to realize such a control function efficiently and represent it appropriately. Since the synthesis task involves a series of reachability computations, as DESs becoming more complicated, the traditional explicit state-space traversal algorithm may be intractable due to *the state-space explosion problem*. By using binary decision diagrams (BDD) [6,7], the supervisor can be represented and computed symbolically such that the state-space explosion problem is alleviated to some extent. However, the symbolic computation is not a silver bullet. Transforming from the traditional explicit state-space traversal algorithm into a BDD-based computation scheme does not guarantee that the algorithm will become remarkably efficient. Thus numerous researches have been performed to improve the efficiency of symbolic computations. In this paper, we mainly focus on partitioning techniques, which decompose the state-space into a set of structural components and utilize these partitioned components to realize efficient reachability computations.

With BDD-based traversal algorithms, some larger DESs could be solved without causing the state-space explosion. Meanwhile, another problem is arising from the BDD representation of the resultant supervisor. Since the original models have been reformulated and encoded, it is cumbersome for the users to relate each state with the corresponding BDD variables. Therefore, it is more convenient and natural to represent the supervisor in a form similar to the models. In [8], a promising approach is presented, where a set of minimal and tractable logic expressions, referred to as guards, are extracted from the supervisor and attached to the original models of the closed-loop system. However, this approach computes the supervisor symbolically based on the conjunctive partitioning technique. This might lead to the state-space explosion, due to the large number of intermediate BDD nodes.

The main contribution of this paper is adapting a symbolic supervisory synthesis approach to the guard generation procedure, to make it applicable for industrially interesting applications. The approach automatically synthesizes a supervisor by taking the advantage of the disjunctive partitioning technique. The monolithic state-space is then split into a set of simpler components and the reachability search is performed structurally with a set of heuristic decisions. Moreover, the guard generation procedure is tailored to use the partitioned structure to extract the simplified guards and attach them to the original models. Finally, a comparison of algorithm efficiency between two partitioning techniques is made by applying them to a set of benchmark examples.

The paper is organized as follows: For the readers who might be unfamiliar with Supervisory Control Theory, Section 2 gives an informal and brief explanation. Section 3 provides some preliminaries that are used throughout the paper. The symbolic supervisory synthesis and the guard generation procedure will be discussed in detail in Section 4 and 5. In Section 6, we apply what we have discussed and implemented to several real case studies. Finally, we end up with some conclusions in Section 7.

2 Motivating Example

For readers who might be unfamiliar with SCT, the following simple example gives a brief overview and states what the exact problem this paper is about to solve.

Example 1. Consider a resource booking problem where two industrial robots need to book two resources in opposite order to carry out their tasks. To avoid collisions, a constraint requires that two robots are not allowed to occupy two zones simultaneously.

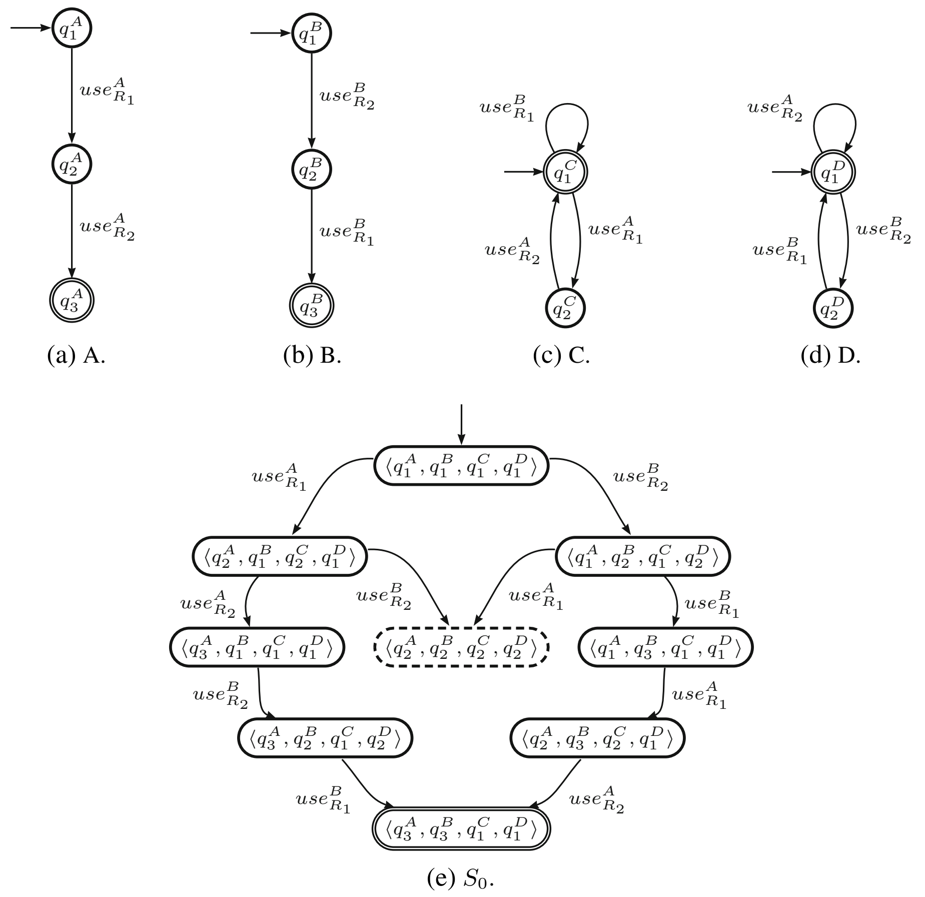
Figure 1 shows one way to model the system as the state machines, or *deterministic finite automata*. Figure 1a and 1b depict the robot (plant) models and Fig. 1c and 1d depict the resource (specification) models. The states having an incoming arrow from outside denote the beginning of the task, while the states having double circles, called *marked states*, denote the accomplishment of the task. The event $use^A_{R_1}$ means that Robot A uses Resource 1. The other events can be interpreted similarly. The goal of the SCT is to automatically synthesize a *minimally restrictive* supervisor from these modular models. Traditionally, to do this, the algorithm starts with the composition (formally described in Section 3.1) of all the automata as the initial candidate supervisor S_0 (Fig. 1e). Then the undesirable states will be removed iteratively. Generally, undesirable states can either be blocking or uncontrollable. A state is blocking when no marked state can be reached, while uncontrollable states are defined in Section 3.1. In Fig. 1e, we have one blocking state $\langle q_2^A, q_2^B, q_2^C, q_2^D \rangle$, which depicts the situation where Robot A has booked Resource 1 and is trying to book Resource 2, while Robot B has booked Resource 2 and is trying to book Resource 1. In such case, none of the robots can do other movements, which is a blocking situation. After removing the blocking state together with the associated transitions, a non-blocking supervisor is produced.

It can be observed that for such a simple example, the composed automaton contains 9 states. With a DES getting more complicated, the composed automaton will become significantly larger. To alleviate this problem, a well known strategy is to represent the state space symbolically by using *Binary Decision Diagrams* (BDD). In [8], based on this principle, an alternative approach is presented, where guards are generated to prevent the controlled system to reach undesirable states. The advantage of this approach is that it never constructs the composed automaton, which means that an incomprehensible BDD representation of the supervisor is avoided. Instead, the approach characterizes a supervisor by a set of minimal guards that are attached to the original models to represent the supervisor behavior. Figure 2 shows the application of the guard generation to the example, where the variables v_A, v_B, v_C, v_D are introduced to hold the current states of the corresponding automata.

The intention of this paper is to improve the guard generation procedure by introducing an alternative symbolic approach. This approach, which is based on the disjunctive partitioning technique, partitions the transition function into a set of simple but structural components. These components, having the disjunctive connection relation between each other, therefore can be used to search the state-space without constructing a total transition function for the composed automaton. Besides, to keep the intermediate number of BDD nodes as small as possible, the approach includes a set of selection heuristics to search the state-space in a structural way.

3 Preliminaries

This section provides some preliminaries which are used throughout the rest of the paper.



q_1^A
$use^A_{R_1}$
q_2^A
$use^A_{R_2}$
q_3^A
(a) A.
q_1^B
$use^B_{R_2}$
q_2^B
$use^B_{R_1}$
q_3^B
(b) B.
$use^B_{R_1}$
q_1^C
$use^A_{R_1}$
$use^A_{R_2}$
q_2^C
(c) C.
$use^A_{R_2}$
q_1^D
$use^B_{R_2}$
$use^B_{R_1}$
q_2^D
(d) D.
$\langle q_1^A, q_1^B, q_1^C, q_1^D\rangle$
$use^A_{R_1}$
$use^B_{R_2}$
$\langle q_2^A, q_1^B, q_2^C, q_1^D\rangle$
$\langle q_1^A, q_2^B, q_1^C, q_2^D\rangle$
$use^A_{R_2}$
$use^B_{R_2}$
$use^A_{R_1}$
$use^B_{R_1}$
$\langle q_3^A, q_1^B, q_1^C, q_1^D\rangle$
$\langle q_2^A, q_2^B, q_2^C, q_2^D\rangle$
$\langle q_1^A, q_3^B, q_1^C, q_1^D\rangle$
$use^B_{R_2}$
$use^A_{R_1}$
$\langle q_3^A, q_2^B, q_1^C, q_2^D\rangle$
$\langle q_2^A, q_3^B, q_2^C, q_1^D\rangle$
$use^B_{R_1}$
$use^A_{R_2}$
$\langle q_3^A, q_3^B, q_1^C, q_1^D\rangle$
(e) S_0.

Fig. 1. Example 1. 1a-1b) Robot automata A and B, 1c-1d) resource automata C and D, and (1e) a supervisor candidate S_0.

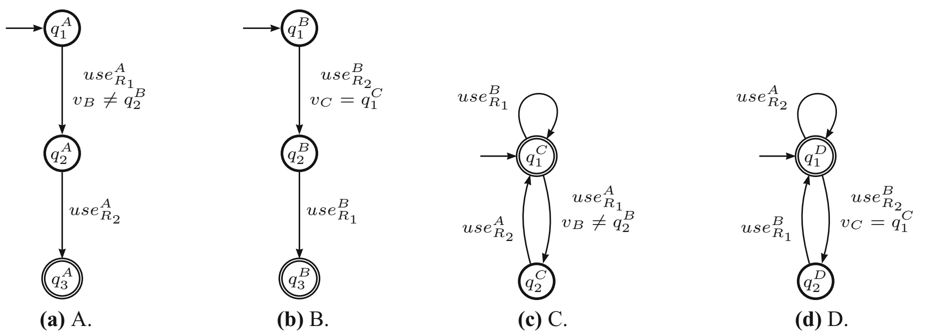


Fig. 2. Guards representing the behavior of the supervisor for Example 1

3.1 Supervisory Control Theory

Generally, a DES can either be described by textual expressions, such as regular expressions or graphically by for instance Petri nets or automata. In this paper, we focus on deterministic finite automata.

Definition 1. *A deterministic finite automaton (DFA), is a five-tuple:*

$$(Q, \Sigma, \delta, q_{init}, Q_m)$$

where:

- *Q is a finite set of states;*
- *Σ is a non-empty finite set of events;*
- *$\delta \colon Q \times \Sigma \to Q$ is a partial transition function which expresses the state transitions;*
- *$q_{init} \in Q$ is the initial state;*
- *$Q_m \subseteq Q$ is a set of marked or accepting states.*

The composition of two or more automata is realized by the *full synchronous composition* [9].

Definition 2. *Let $A^i = (Q^i, \Sigma^i, \delta^i, q^i_{init}, Q^i_m), i = 1, 2$ be two DFAs. The full synchronous composition of A^1 and A^2 is*

$$A^1 \parallel A^2 = (Q^{1\|2}, \Sigma^1 \cup \Sigma^2, \delta^{1\|2}, q^{1\|2}_{init}, Q^1_m \times Q^2_m)$$

where:

- *$Q^{1\|2} \subseteq Q^1 \times Q^2$;*
- *$q^{1\|2}_{init} = \langle q^1_{init}, q^2_{init} \rangle$;*
- $\delta^{1\|2}(\langle q^1, q^2 \rangle, \sigma) = \begin{cases} \delta^1(q^1, \sigma) \times \delta^2(q^2, \sigma) & \text{if } \sigma \in \Sigma^1 \cap \Sigma^2 \\ \delta^1(q^1, \sigma) \times \{q^2\} & \text{if } \sigma \in \Sigma^1 \setminus \Sigma^2 \\ \{q^1\} \times \delta^2(q^2, \sigma) & \text{if } \sigma \in \Sigma^2 \setminus \Sigma^1 \\ \text{undefined} & \text{otherwise.} \end{cases}$

As described in Section 1, the goal of SCT is to automatically synthesize a minimally restrictive supervisor S, which guarantees the behavior of the plant P always fulfills the given specification Sp. Here if the plant is given as a number of sub-plants $P_1, \ldots, P_n$, the plant can be obtained by performing the full synchronous composition operation on these sub-plants. Thus $P = P_1 \parallel \ldots \parallel P_n$. Similarly, $Sp = Sp_1 \parallel \ldots \parallel Sp_m$.

In SCT, events in the alphabet Σ can either be controllable or uncontrollable. Hence, Σ can be divided into two disjoint subsets, the controllable event set Σ_c and the uncontrollable event set Σ_u. The supervisor is only allowed to restrict controllable events from occurring in the plant.

Additionally, given a plant P and a specification Sp, two properties [1,2] that the supervisor ought to have are:

- *Controllability*: Let Σ_u be the set of uncontrollable event set. The supervisor S is never allowed to disable any uncontrollable event that might be generated by the plant P.
- *Non-blocking*: This is a progress property enforced by the supervisor S, which guarantees that at least one marked state is always reachable in the closed-loop system, $S \parallel P$.

3.2 Binary Decision Diagrams (BDD)

Binary decision diagrams (BDD), used for representing Boolean functions, can be extended to symbolically represent states, events and transitions of automata. In contrast to explicit representations, which might be computationally expensive in terms of time and memory, BDDs often generate compact and operation-efficient representations.

A binary decision diagram is a directed acyclic graph (DAG) consisting of two kinds of nodes: *decision nodes* and *terminal nodes*. Given a set of Boolean variables V, a BDD is a Boolean function $f\colon 2^V \to \{0,1\}$ which can be recursively expressed using Shannon's decomposition [10]. Besides, a variable v_1 has a lower (higher) *order* than variable v_2 if v_1 is closer (further) to the root and is denoted by $v_1 \prec v_2$. The variable ordering will impact the number of BDD nodes. However, finding an optimal variable ordering of a BDD is a NP-complete problem [11]. In this paper, a simple but powerful heuristic based on Aloul's Force algorithm [12] is used to compute a suitable static variable ordering.

Symbolic Representation of Automata. The BDD data structure can be extended to also represent models such as automata. The key point is to make use of *characteristic functions*.

Given a finite state set U as universe, for every $S \subseteq U$, the characteristic function can be defined as follows:

$$\chi_S(\alpha) = \begin{cases} 1 & \alpha \in S \\ 0 & \alpha \notin S \end{cases} . \tag{1}$$

Set operations can be equivalently carried on corresponding characteristic functions. For example, $S_1 \cup S_2$, $(S_1, S_2 \subseteq U)$ can be mapped equivalently to $\chi_{S_1} \vee \chi_{S_2}$, since $S_1 \cup S_2 = \{\alpha \in U \mid \alpha \in S_1 \vee \alpha \in S_2\}$.

The elements of a finite set can be expressed as a Boolean vector. So a set with n elements, requires a Boolean vector of length $\lceil \log_2 n \rceil$. Just like the case of coding the states in a set, binary encoding of the transition function δ follows the same rule but with the difference that the transition function distinguishes between source-states and target states. Hence, we need two Boolean vectors with different sets of Boolean variables to express the domain of source-states and target-states respectively.

4 BDD-Based Partitioning Computation

The *safe-state* algorithm, an efficient supervisor synthesis algorithm, formally defined in [13], is used in this paper. The algorithm creates the supervisor by first building the candidate $S_0 = P \parallel Sp$, then removing states from Q^{S_0} until the remaining safe states are both non-blocking and controllable.

As Algorithm 1 shows, given a set of forbidden states Q_x, the algorithm computes the set of safe states Q^S by iteratively removing the blocking states (RestrictedBackward in line 5) and the uncontrollable states (UncontrollableBackward in line 6). Note that after the termination of the algorithm, not all of the safe states are reachable from the initial state. Therefore, a forward reachability search is needed to exclude the safe states which are not reachable. The safe-state algorithm is discussed in more detail in [13].

Algorithm 1. The safe-state synthesis algorithm.

```
1: input : $Q_x, Q^{S_0}$
2: let $X_0 := Q_x, k := 0$;
3: repeat
4:     $k := k + 1$;
5:     $Q' :=$ RestrictedBackward$(Q_m, X_{k-1})$;
6:     $Q'' :=$ UncontrollableBackward$(Q^{S_0} \backslash Q')$;
7:     $X_k := X_{k-1} \cup (Q'')$;
8: until $X_k = X_{k-1}$
9: return $Q^{S_0} \backslash X_k$
```

4.1 Efficient State Space Search

Not surprisingly, the backward and forward reachability searches turn out to be the bottle-neck of the algorithm presented above. The problem with the intuitive reachability is that for a large and complicated modular DES, the BDD representation of the total transition function δ^{S_0} is often too large to be constructed. The natural way to tackle the complexity of the transfer function is to split it into a set of less complex partial functions with a connection between them. Such methods are based on conjunctive and disjunctive partitioning techniques.

Conjunctive Representation. Conjunctive partitioning, introduced in [14,15], is an approach to represent synchronous digital circuits where all transitions happen simultaneously. In the context of DES, the conjunctive partitioning of the full synchronous composition can be achieved by adding self-loops to the automata for events that are not included in their original alphabets. This leads to a situation where all automata have equal alphabet. Therefore, the conjunctive transition function $\hat{\delta}^i$ for the automaton A^i and the total transition function can be defined as follows:

$$\hat{\delta}^i(q^i, \sigma) = \begin{cases} \delta^i(q^i, \sigma) & \text{if } \delta^i(q^i, \sigma) \text{ is defined} \\ q^i & \text{if } \sigma \notin \Sigma^i \\ \text{undefined otherwise} & . \end{cases} \tag{2}$$

$$\delta = \bigwedge_{1 \le i \le n} \hat{\delta}^i \ . \tag{3}$$

By making use of (2) and (3), we can search the state-space without constructing the total transition function. Algorithm 2 applies this technique for the forward reachability search. Assuming that the automaton set $A = \{A_1, \ldots, A_n\}$ and the state $q = \langle q^1, q^2, \ldots, q^n \rangle$, the algorithm explores the target state $\acute{q}$ by performing each conjunctive transition function $\hat{\delta}^i$ with arguments (the local state q^i and the event $\sigma \in \Sigma$) to get each local target state $\acute{q}^i$.

Disjunctive Representation. The conjunctive partitioning of the transition relation works well for formal verification of synchronous digital circuits. However, because of the asynchronous feature of the full synchronous composition, the intermediate states (Q_{k-1}) can still cause the explosion problem when performing the reachability search,

Algorithm 2. Conjunctive forward reachability algorithm.

```
1: input : $Q_{init}, \{\hat{\delta}^1, \dots, \hat{\delta}^n\}, \Sigma$
2: let $Q_0 := Q_{init}, k := 0$;
3: repeat
4:     $k := k + 1$;
5:     $Q_k := Q_{k-1} \cup \{\acute{q} \mid \exists q \in Q_{k-1}, \exists \sigma \in \Sigma, \forall i \in \{1, \dots, n\}$ such that $\hat{\delta}^i(q^i, \sigma) = \acute{q}^i\}$;
6: until $Q_k = Q_{k-1}$
7: return $Q_k$
```

which prevents the conjunctive partitioning technique from being applied to large systems. The disjunctive partitioning, explained subsequently, on the other hand, is then shown to be an appropriate partitioning technique for SCT.

Assuming $A = \{A_1, \dots, A_n\}$ and $q = \langle q^1, \dots, q^n \rangle$, the disjunctive transition function $\check{\delta}^i$ of A^i, is defined based on the event $\sigma \in \Sigma^i$ and the dependency set $D(A^i)$:

$$D(A^i) = \{A^j \in A \mid \exists A^i \in A \text{ where } \Sigma^i \cap \Sigma^j \neq \emptyset\} \ . \tag{4}$$

$$\check{\delta}^i(q, \sigma) = \left(\bigwedge_{A^j \in D(A^i)} \zeta^{i,j}(q^j, \sigma) \right) \wedge \left(\bigwedge_{A^k \notin D(A^i)} q^k \overset{\sigma}{\leftrightarrow} q^k \right) \ . \tag{5}$$

$$\zeta^{i,j}(q^j, \sigma) = \begin{cases} \delta^j(q^j, \sigma) & \text{if } \sigma \in \Sigma^i \cap \Sigma^j \\ q^j & \text{otherwise} \ . \end{cases} \tag{6}$$

Additionally, the total transition function is defined as:

$$\delta = \bigvee_{1 \le i \le n} \check{\delta}^i \ . \tag{7}$$

The construction of the dependency set for each automaton can be obtained through calculating which automaton shares any event with it. Taking Example 1 as an example, for the automaton A, since it shares the events $use^A_{R_1}$, $use^A_{R_2}$ with the automaton C and the event $use^A_{R_2}$ with the automaton D, $D(A)$ can be constructed as follows:

$$D(A) = \{A, C, D\}.$$

Besides, the total transition function defined for the state $\langle q^A_1, q^B_1, q^C_1, q^D_1 \rangle$ and the event $use^A_{R_1}$ can be obtained by computing $\check{\delta}^A$ and $\check{\delta}^C$, since $use^A_{R_1}$ only belongs to Σ^A and Σ^C. By using (5) and (6), it can be inferred that

$$\delta(\langle q^A_1, q^B_1, q^C_1, q^D_1 \rangle, use^A_{R_1}) = \check{\delta}^A(\langle q^A_1, q^B_1, q^C_1, q^D_1 \rangle, use^A_{R_1})$$

$$= \check{\delta}^C(\langle q^A_1, q^B_1, q^C_1, q^D_1 \rangle, use^A_{R_1}) = \langle q^A_2, q^B_1, q^C_2, q^D_1 \rangle.$$

Notice that the disjunctive transition function represented in BDDs, is shown explicitly here to easily understand.

4.2 Workset Based Strategies

In Section 4.1, we suggested the use of partitioning techniques to deal with the large number of intermediate BDD nodes. However, using partitioning techniques alone is not enough to yield efficient BDD-based reachability algorithms. In [16], it has been shown that random structural reachability search yields poor compression of intermediate BDD nodes. In order to improve these algorithms to substantially reduce the number of intermediate BDD nodes, it is vital to search the state space in a structural and efficient way. Here we introduce a simple algorithm, Algorithm 3, which is formally defined in [13]. The workset algorithm maintains a set of active disjunctive transition functions W_k. These active transition functions are selected one at a time for the local reachability search. If there is any new state found for the currently selected transition relation, then all of its *dependent transition functions* (8) will be added in W_k. Notice that in Algorithm 3, "$\cdot$" can be any event, since we don't care about the specific events as long as it is defined in $\check{\delta}^i$.

$$E(\check{\delta}^i) = \{\check{\delta}^j \mid A^j \in D(A^i)\backslash\{A^i\}\} \ . \tag{8}$$

Algorithm 3. Workset forward reachbility algorithm.

```
1: input : $Q_{init}, \{\check{\delta}^1, \dots, \check{\delta}^n\}$
2: let $W_0 := \{\check{\delta}^1, \dots, \check{\delta}^n\}, Q_0 := Q_{init}, k := 0$;
3: repeat
4:     $\mathbb{H}$: Pick and remove a transition $\check{\delta}^i \in W_k$;
5:     $k := k + 1$;
6:     $Q_k := Q_{k-1} \cup \{\acute{q} \mid \exists q \in Q_{k-1}, \check{\delta}^i(q, \cdot) = \acute{q}\}$;
7:     if $Q_k \neq Q_{k-1}$ then
8:         $W_k := W_{k-1} \cup E(\check{\delta}^i)$;
9:     end if
10: until $W = \emptyset$
11: return $Q_k$
```

Selection Heuristics. In Algorithm 3, $\mathbb{H}$ denotes the heuristics of selecting the next disjunctive transition function for the reachability search such that the number of intermediate BDD nodes is computed as small as possible. How a disjunctive transition function $\check{\delta}^i$ is chosen among those in the working set has great influence on the performance of the algorithm. Here we suggest a series of simple heuristics that have been implemented and seem to work well for real-world problems. In Section 6, those heuristics will be applied to a benchmark example to compare how they influence the performance of the workset algorithm.

To find a good heuristic, a two-stage selection rule was implemented, as Fig. 3 shows. Using this method, a complex selection procedure can be described as a combination of two selection rules. In the current implementation, the first stage H_1 selects a subset $\mathrm{W}' \subset \mathrm{W}$ to be sent to H_2 using one of the following rules:

- MaxF: Choose the automata with the largest dependency set cardinality.
- MinF: The opposite of above.

In case W′ is not a singleton, the second stage H_2 is used to choose a single disjunctive transition function $\check{\delta}^i$ among W′. In the experiment, the following shown heuristics can significantly reduce the number of intermediate BDD nodes for some relatively large problems.

- Reinforcement learning (R) [17]: Choose the best transition relation based on the previous activity record.
- Reinforcement learning + Tabu (RT) [18]: Same as the reinforcement learning with the difference that using tabu search for the selection policy.

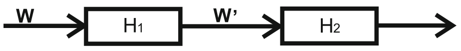


Fig. 3. The two stage selection heuristics for the workset algorithm

5 Supervisor as Guards

As mentioned in Section 1, given a supervisor represented as a BDD, it is cumbersome for the users to relate each state to the corresponding BDD variables. Therefore, it is more convenient and natural to represent the supervisor in a form similar to the original models. In this section, the guard generation procedure, originating from [8], is discussed and combined with the BDD-based disjunctive partitioning approach in Section 4.1.

The guard generation procedure, being dependent on three kinds of state sets, extracts a set of compact guards indicating under which conditions the event can be executed without violating the specifications. These guards are then attached to the original model to represent the supervisor.

5.1 Computation of the Basic State Sets

Concerning the states that are retained or removed after the synthesis process, the states that enable an arbitrary event σ can be divided into three *basic state sets*: forbidden state set, allowed state set and don't care state set.

The forbidden state set, denoted by Q_{f}^{σ}, is the set of states in the supervisor where the execution of σ is defined for S_0, but not for the supervisor. The allowed state set, denoted by Q_{a}^{σ}, is the set of states in the supervisor where the execution of σ is defined for the supervisor. In other word, for each event σ, Q_{a}^{σ} represents the set of states where event σ must be allowed to be executed in order to end up in states belonging to the supervisor.

In order to obtain compact and simplified guards, inspired from the Boolean minimization techniques, another set of states, denoted by Q_{dc}^{σ}, which describes a situation where executing σ will not impact the result of the synthesis, is utilized to minimize the guards.

Algorithms 4 and 5 presented below show how to compute the forbidden states Q^{σ}_{f} and the allowed states Q^{σ}_{a} by making use of the disjunctive transition functions. Note that Q^S and Q_x denote the resultant supervisor states and all the forbidden states yielded from Algorithm 1. The don't care state set, Q^{σ}_{dc} can be defined as the complement of the union of Q^{σ}_{a} and Q^{σ}_{f}. The proof can be found in [8].

5.2 Guard Generation

Based on the basic state sets, guards can be extracted. For every automaton in the DES, a new variable v is introduced to hold the current state of the automaton. For each event σ, the following propositional function, $G^{\sigma}\colon Q^{A^1} \times Q^{A^2} \times \ldots \times Q^{A^n} \to \mathbb{B}$ is defined as:

$$G^{\sigma}\langle v_{A^1}, v_{A^2}, \ldots v_{A^n}\rangle = \begin{cases} \text{true} & \langle v_{A^1}, v_{A^2}, \ldots v_{A^n}\rangle \in Q^{\sigma}_{\mathrm{a}} \\ \text{false} & \langle v_{A^1}, v_{A^2}, \ldots v_{A^n}\rangle \in Q^{\sigma}_{\mathrm{f}} \\ \text{don't care otherwise} & . \end{cases} \tag{9}$$

where $\mathbb{B}$ is the set of Boolean values and v_{A^i} represents the current state of automaton A^i. In particular, σ is allowed to be executed from the state $\langle v_{A^1}, v_{A^2}, \ldots v_{A^n}\rangle$ if the guard is true.

By applying minimization methods of Boolean functions (utilizing the don't care state set) and certain heuristics, the generated guards can be simplified. The procedure is discussed in details in [8].

Algorithm 4. Computation of Q^{σ}_{f}.

```
1: input : $\sigma, Q_x, Q^S, \{\check{\delta}^1, \ldots, \check{\delta}^n\}$
2: let $Q^{\sigma}_{\mathrm{f}} := \emptyset$;
3: for all $A^i$ if $\sigma \in \Sigma^i$ do
4:     $Q^{\sigma}_{\mathrm{f}} := Q^{\sigma}_{\mathrm{f}} \cup \{q \mid \exists \acute{q} \in Q_x, \check{\delta}^i(q,\sigma) = \acute{q}\}$;
5: end for
6: let $Q^{\sigma}_{\mathrm{f}} := Q^{\sigma}_{\mathrm{f}} \cap Q^S$;
7: return $Q^{\sigma}_{\mathrm{f}}$
```

Algorithm 5. Computation of Q^{σ}_{a}.

```
1: input : $\sigma, Q^S, \{\check{\delta}^1, \ldots, \check{\delta}^n\}$
2: let $Q^{\sigma}_{\mathrm{a}} := \emptyset$;
3: for all $A^i$ if $\sigma \in \Sigma^i$ do
4:     $Q^{\sigma}_{\mathrm{a}} := Q^{\sigma}_{\mathrm{a}} \cup \{q \mid \exists \acute{q} \in Q^S, \check{\delta}^i(q,\sigma) = \acute{q}\}$;
5: end for
6: let $Q^{\sigma}_{\mathrm{a}} := Q^{\sigma}_{\mathrm{a}} \cap Q^S$;
7: return $Q^{\sigma}_{\mathrm{a}}$
```

6 Case Studies

What we have discussed in the previous sections has been implemented and integrated in the supervisory control tool Supremica [19] which uses JavaBDD [20] as BDD package. In this section, the implemented program will be applied to a set of relatively complicated examples[1].

6.1 Benchmark Examples

A set of benchmark examples is briefly described as follows.

Automated Guided Vehicles. An AGV system, described in [21], is a simple manufacturing system where five automated guided vehicles transport material between stations. As the routes of the vehicles cross each other, single-access zones are introduced to avoid collisions.

Parallel Manufacturing Example. The Parallel Manufacturing Example, introduced in [22], consists of three manufacturing units running in parallel. The system is modeled in three layers in a hierarchical interface-based manner.

The Transfer Line. The Transfer Line TL(n, m), introduced as a tutorial example in [23], defines a very simple factory consisting of a series of identical cells. Each cell contains two machines and two buffers, one between the machines and one before a testing unit which decides whether the work piece should be sent back to the first machine for further processing, or if it should be passed to the next cell. The capacity of each buffer is m, which is usually chosen to be either 1 or 3.

The Extended Cat and Mouse. An extended cat and mouse problem [8], which is more complicated than the transfer line model, generalizes the classic one presented in [1]. The extended version makes it possible to generate problem instances of arbitrary size, where n and k denote the number of levels and cats respectively.

6.2 Approach Evaluation

In this section, we evaluate the approach from two aspects. First, a comparison between two partitioning techniques is made by analyzing the statistical data from Fig. 1. In addition, the extended cat and mouse example with multiple instances is utilized to investigate how the choice of heuristics in the workset algorithm influences the time efficiency.

Conjunctive vs. Disjunctive. Figure 1 shows the result of applying two partitioning techniques for the examples explained above. The supervisors synthesized for these examples are both non-blocking and controllable and the safe states are reachable. It is observed that both of the partitioning based algorithms can handle the AGV and the Parallel Manufacturing example, for which the number of reachable states is up to 10^7.

[1] The experiment was carried out on a standard Laptop (Core 2 Duo processor, 2.4 GHz, 2GB RAM) running Ubuntu 10.04.

Table 1. Non-blocking and controllability synthesis

			Conjunctive Synthesis		Workset Algorithm	
Model	Reachable States	Supervisor states	BDD Peak	Computation Time (s)	BDD Peak	Computation Time (s)
AGV	22929408	1148928	9890	6.50	2850	0.87
Parallel Man	5702550	5702550	12363	2.47	2334	1.57
Transfer line (1,3)	64	28	17	0.05	13	0.10
Transfer line (5,3)	1.07×10^9	8.49×10^4	2352	1.69	299	0.59
Transfer line (10,3)	1.15×10^{18}	6.13×10^{13}	31022	48.36	1257	3.89
Transfer line (15,3)	1.23×10^{27}	4.42×10^{20}	–	–	3032	12.80
Cat and mouse (1,1)	20	6	43	0.02	31	0.05
Cat and mouse (1,5)	605	579	2343	0.08	273	0.09
Cat and mouse (5,1)	1056	76	848	0.30	305	0.30
Cat and mouse (5,5)	6.91×10^9	3.15×10^9	–	–	15964	20.86

* - denotes memory out.

However, with DESs getting larger and more complicated, the conjunctive partitioning technique is not capable of synthesizing non-blocking and controllable supervisors any more. The disjunctive partitioning, on the other hand, could successfully explore the state space within acceptable time. In addition, the column ”BDD Peak”, the maximal number of BDD nodes during the reachability computation shows that the disjunctive partitioning together with heuristic decisions can effectively reduce the number of intermediate BDD nodes.

Heuristics. Table 2 shows the computation time for synthesizing non-blocking supervisors of the extended cat and mouse with different instances. Different combinations of heuristics, presented in Section 4.2, are chosen to test the performance of the workset algorithm. Empirically, for the models with relatively large dependency sets, the heuristic pair (MaxF,RT) seems to be a good choice, although it hasn't been formally proved. Observing the results from Table 2, the workset algorithm can handle problem instances with either a large number of levels n or cats k rather well. However, with both numbers increasing, the computation time increases rapidly no matter which heuristic pair is chosen.

Table 2. Computation time for non-blocking supervisors with different heuristics

	Computation Time (s)			
Cat and mouse (n, k)	Workset(MaxF,R)	Workset(MaxF,RT)	Workset(MinF,R)	Workset(MinF,RT)
$(1, 1)$	0.04	0.06	0.05	0.05
$(1, 5)$	0.30	0.27	0.33	0.36
$(5, 1)$	0.08	0.08	0.09	0.08
$(5, 5)$	3.15	2.90	3.85	3.42
$(1, 10)$	0.67	0.66	0.75	0.73
$(7, 7)$	21.4	17.6	25.5	22.9
$(10, 1)$	0.23	0.20	0.24	0.23
$(10, 7)$	100.3	88.5	136.4	138.0

7 Conclusions

In this paper, we improved and extended our previous work, the guard generation procedure to make it applicable for industrially interesting applications. More specifically, the content of the paper can be summarized as follows:

- Introduce the partitioning techniques to split the BDD representation of $\delta^{Sp\|P}$ into a set of smaller but structural components.
- To alleviate the problem that the intermediate number of BDD nodes might still be huge during the reachability exploration, we introduce the workset algorithm together with a set of simple heuristics to search the state-space in a structured and efficient way.
- The guard generation procedure is tailored to make use of the partitioned transition functions and the synthesized supervisor to compute the basic state sets for an event.
- The presented approach is applied to a set of benchmark examples to be evaluated.

It is concluded that the disjunctive partitioning, with appropriate heuristics, is suitable for solving large modular supervisory control problems. There are several directions towards which we could extend our approach. For instance, additional heuristics could be applied to the workset algorithm, to further decrease the number of intermediate BDD nodes. Moreover, it is possible to combine with more sophisticated synthesis techniques, such as compositional techniques, to substantially improve the algorithm efficiency.

References

1. Ramadge, P.J.G., Wonham, W.M.: The Control of Discrete Event Systems. Proceedings of the IEEE 77, 81–98 (1989)
2. Cassandras, C.G., Lafortune, S.: Introduction to Discrete Event Systems, 2nd edn. Springer, Heidelberg (2008)
3. Balemi, S., Hoffmann, G.J., Gyugyi, P., Wong-Toi, H., Franklin, G.F.: Supervisory Control of a Rapid Thermal Multiprocessor. IEEE Transactions on Automatic Control 38, 1040–1059 (1993)
4. Feng, L., Wonham, W.M., Thiagarajan, P.S.: Designing Communicating Transaction Processes by Supervisory Control Theory. Formal Methods in System Design 30, 117–141 (2007)
5. Shoaei, M.R., Lennartson, B., Miremadi, S.: Automatic Generation of Controllers for Collision-Free Flexible Manufacturing Systems. In: 6th IEEE Conference on Automation Science and Engineering, pp. 368–373 (2010)
6. Akers, S.B.: Binary Decision Diagrams. IEEE Transactions on Computers 27, 509–516 (1978)
7. Bryant, R.E.: Symbolic Manipulation with Ordered Binary Decision Diagrams. ACM Computing Surveys 24, 293–318 (1992)
8. Miremadi, S., Akesson, K., Lennartson, B.: Extraction and Representation of a Supervisor Using Guards in Extended Finite Automata. In: 9th International Workshop on Discrete Event Systems, pp. 193–199 (2008)
9. Hoare, C.A.R.: Communicating Sequential Processes. Communications of the ACM 21, 666–677 (1985)

10. Shannon, C.E., Weaver, W.: The Mathematical Theory of Communication. University of Illinois Press (1949)
11. Bollig, B., Wegener, I.: Improving the Variable Ordering of OBDDs Is NP-Complete. IEEE Transactions on Computers 45, 993–1002 (1996)
12. Aloul, F.A., Markov, I.L., Sakallah, K.A.: Force: A Fast and Easy-To-Implement Variable-Ordering Heuristic. In: 13th ACM Great Lakes symposium on VLSI, pp. 116–119 (2003)
13. Vahidi, A., Fabian, M., Lennartson, B.: Efficient Supervisory Synthesis of Large Systems. Control Engineering Practice 14, 1157–1167 (2006)
14. Burch, J.R., Clarke, E.M., Long, D.E.: Symbolic Model Checking with Partitioned Transition Relations. In: International Conference on Very Large Scale Integration, vol. A-1, pp. 49–58 (1991)
15. Burch, J.R., Clarke, E.M., Long, D.E., Mcmillan, K.L., Dill, D.L.: Symbolic Model Checking for Sequential Circuit Verification. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 13, 401–424 (1994)
16. Byröd, M., Lennartson, B., Vahidi, A., Åkesson, K.: Efficient Reachability analysis on Modular Discrete-Event Systems using Binary Decision Diagrams. In: 8th International Workshop on Discrete Event Systems, pp. 288–293 (2006)
17. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement Learning: A Survey. Journal of Artificial Intelligence Research 4, 237–285 (1996)
18. Glover, F., Laguna, M.: Tabu Search. Journal of the Operational Research Society 5 (1997)
19. Åkesson, K., Fabian, M., Flordal, H., Malik, R.: Supremica – An Integrated Environment for Verification, Synthesis and Simulation of Discrete Event Systems. In: 8th International Workshop on Discrete Event Systems, pp. 384–385 (2006)
20. JavaBDD, `http://javabdd.sourceforge.net`
21. Holloway, L.E., Krogh, B.H.: Synthesis of Feedback Control Logic for a Class of Controlled Petri Nets. IEEE Transactions on Automatic Control 35, 514–523 (1990)
22. Leduc, R.J.: Hierarchical Interface-Based Supervisory Control. In: 40th IEEE Conference on Decision and Control, pp. 4116–4121 (2002)
23. Murray Wonham, W.: Notes on Control of Discrete Event Systems. University of Toronto (1999)