# EFFICIENT SYMBOLIC SUPERVISORY SYNTHESIS AND GUARD GENERATION

## *Evaluating Partitioning Techniques for the State-space Exploration*

Z. Fei, S. Miremadi
*Department of Signals and Systems, Chalmers University of Technology, Gothenburg, Sweden*

K. Åkesson, B. Lennartson
*Department of Signals and Systems, Chalmers University of Technology, Gothenburg, Sweden*

Keywords: Supervisory control theory, Deterministic finite automata, Symbolic representation, Reachability search, Propositional formula.

Abstract: The supervisory control theory (SCT) is a model-based framework, which automatically synthesizes a supervisor that restricts a plant to be controlled based on specifications to be fulfilled. Two main problems, typically encountered in industrial applications, prevent SCT from having a major breakthrough. First, the supervisor which is synthesized automatically from the given plant and specification models might be incomprehensible to the users. To tackle this problem, an approach was recently presented to extract compact propositional formulae (guards) from the supervisor, represented symbolically by binary decision diagrams (BDD). These guards are then attached to the original models, which results in a modular and comprehensible representation of the supervisor. However, this approach, which computes the supervisor symbolically in the conjunctive way, might lead to another problem: the state-space explosion, because of the large number of intermediate BDD nodes during computation. To alleviate this problem, we introduce in this paper an alternative approach that is based on the disjunctive partitioning technique, including a set of selection heuristics. Then this approach is adapted to the guard generation procedure. Finally, the efficiency of the presented approach is demonstrated on a set of benchmark examples.

## 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 system always fulfill specifications. Nevertheless, from the control engineering point of view, instead of verifying the correctness of the system, a controller which guarantees that the system behaves according to specifications is preferred. The Supervisory Control Theory (SCT) (Ramadge and Wonham, 1987; Ramadge and Wonham, 1989; Cassandras and Lafortune, 2008) provides such a control-theoretic framework to design a device, called the *supervisor*, for reactive systems referred as discrete event systems (DESs). Given a model of the system 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 (Balemi et al., 1993; Feng et al., 2007; Shoaei et al., 2010).

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 the DES becoming more complicated, the traditional explicit state-space traversal algorithm may be intractable due to *the space-state explosion problem*. By using binary decision diagrams (BDDs) (Akers, 1978; Bryant, 1992), 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 stat-


Fei Z., Miremadi S., Åkesson K. and Lennartson B..
EFFICIENT SYMBOLIC SUPERVISORY SYNTHESIS AND GUARD GENERATION - Evaluating Partitioning Techniques for the State-space Exploration.
DOI: 10.5220/0003178801060115
In *Proceedings of the 3rd International Conference on Agents and Artificial Intelligence* (ICAART-2011), pages 106-115
ISBN: 978-989-8425-40-9

space traversal algorithm into a BDD-based computation scheme does not guarantee that the algorithm will become remarkably efficient. Thus numerous research has 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.

Everything has its pros and cons. 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 (Miremadi, 2010), 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, because of the large number of intermediate BDD nodes during computation.

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 splitted 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 the 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 the evaluation of two used partitioning techniques in Section 6.2 and some conclusions in Section 7.

## 2 MOTIVATING EXAMPLE

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

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.

Fig. 1 shows one way to model the system as the state machines, or *automata*. Fig.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 $S0$ (Fig. 1e), where unreachable states have been excluded. 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 $\langle q^A_2, q^B_2, q^C_2, q^D_2 \rangle$, 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 deadlock situation. After removing the blocking state together with the associated transitions, a nonblocking supervisor is produced.

It can be observed that for such a simple example, the composed automaton contains 12 states (unreachable states from the initial state have been removed). With the system 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* (BDDs). In (Miremadi, 2010), 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

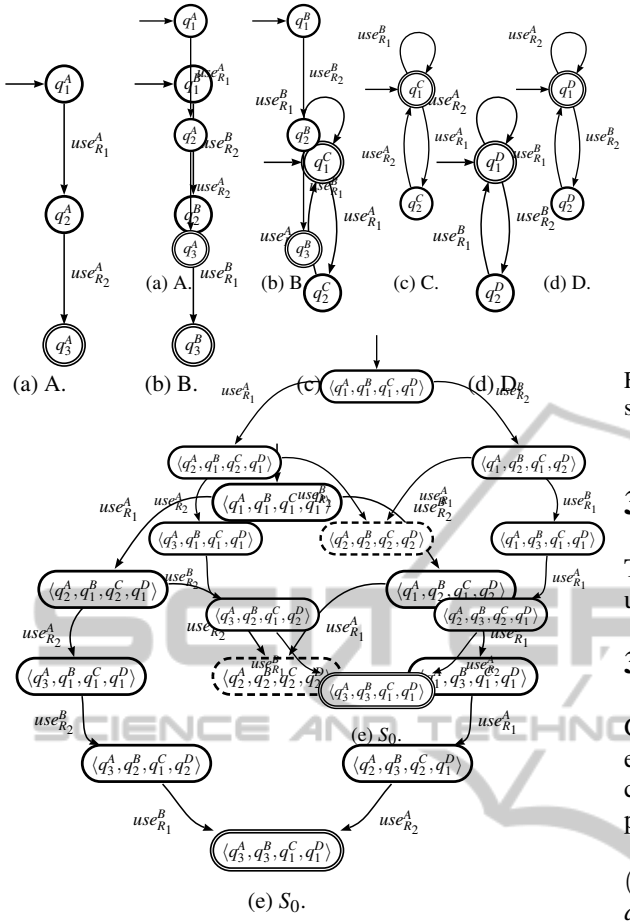


Figure 1: Example.1a-1b) Robot automata $A$ and $B$, 1c-1d) resource automata $C$ and $D$, and 1e) a supervisor candidate $S_0$.

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. Fig.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.

Figure 2: Guards representing the behavior of the supervisor for the example.

# 3 PRELIMINARIES

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

## 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. For this paper, we focus on deterministic finite automata.

*A deterministic finite automaton* is a five-tuple $(Q, \Sigma, \delta, q_{init}, Q_m)$, where $Q$ is a finite set of states, $q_{init} \in Q$ is the initial state, $Q_m \subseteq Q$ is the set of marked or accepting states, and the *alphabet* $\Sigma$ is the finite set of events. The transitions of the states are expressed by the partial transition *function* $\delta$, where $\delta(q, \sigma) = \acute{q}$ means that there exists a transition labeled by event $\sigma \in \Sigma$ from the *source-state* state $q \in Q$ to the *target-state* state $\acute{q} \in Q$.

In SCT, the events in the alphabet, $\Sigma$ can either be controllable or uncontrollable. Therefore, $\Sigma$ can be divided into two disjoint subsets, the controllable event set $\Sigma_c$, and the uncontrollable event set, $\Sigma_u$. The supervisor is only allowed to restrict controllable events from occurring in the plant.

The goal of SCT is to automatically synthesize a maximally permissive supervisor, restricting the behavior of the plant to fulfill the given specification. Generally, the plant or the specification could be modularly expressed by a number of sub-plants or sub-specifications. The composition of two or more automata is realized by the full synchronous operator $\|$, defined in (Hoare, 1985). For instance, if the plant is given as a number of sub-plants $P_1, \ldots, P_n$, the plant $P$ is computed by synchronizing the sub-plants $P = P_1 \| \ldots \| P_n$. More specifically, let $A^i = (Q^i, \Sigma^i, \delta^i, q^i_{init}, Q^i_m), i = 1,2$ be two automata. The full synchronous composition of $A^1$ and $A^2$ results in $A^1 \| A^2 = (Q^{1\|2}, \Sigma^1 \cup \Sigma^2, \delta^{1\|2}, q^{1\|2}_{init}, Q^1_m \times Q^2_m)$,

where $Q^{1\|2} \subseteq Q^1 \times Q^2$ and $q^{1\|2}_{init} = (q^1_{init}, q^2_{init})$. The composite transition function $\delta^{1\|2}$ is defined as follows.

$$\delta^{1\|2}((q^1,q^2),\sigma) = \begin{cases} \delta^1(q^1,\sigma) \times \delta^2(q^2,\sigma) & \text{if } \sigma \in \Sigma^1 \cap \Sigma^2 \\ \delta^1(q^1,\sigma) \times \{q^2\} & \text{if } \sigma \in \Sigma^1 \setminus \Sigma^2 \\ \{q^1\} \times \delta^2(q^2,\sigma) & \text{if } \sigma \in \Sigma^2 \setminus \Sigma^1 \\ \text{undefined} & \text{otherwise} \end{cases} \quad (1)$$

Additionally, given a plant $P$ and a specification $Sp$, two properties (Ramadge and Wonham, 1987; Ramadge and Wonham, 1989; Cassandras and Lafortune, 2008) that the supervisor has or should have are:

1. *Controllability*: The supervisor $S$ is never allowed to disable any uncontrollable event that might be generated by the plant $P$. Let $\Sigma_u$ be the set of uncontrollable event set. A state $q_u$ is said to be an uncontrollable state if it is enabled by the plant $P$, while disabled by the supervisor $S$.
2. *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

*Binary Decision Diagrams* (BDDs), 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 (Shannon and Weaver, 1949). 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 an NP-complete problem (Bollig and Wegener, 1996). In this paper, a simple but powerful heuristic based on Aloul's Force algorithm (Aloul et al., 2003) 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*.

Table 1: Set operations on the characteristic functions.

| Sets/Operations | Characteristic function |
|---|---|
| $\emptyset$ | 0 |
| $U$ | 1 |
| $t \in S$ | $\chi^t_S \leftrightarrow 1$ |
| $S_1 \subseteq S_2$ | $(\chi_{S_1} \rightarrow \chi_{S_2}) \leftrightarrow 1$ |
| $U \setminus S$ | $\neg\chi_S$ |
| $S_1 \cup S_2$ | $\chi_{S_1} \vee \chi_{S_2}$ |
| $S_1 \cap S_2$ | $\chi_{S_1} \wedge \chi_{S_2}$ |

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

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

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 $S_1 \cup S_2 = \{\alpha \in U \mid \alpha \in S_1 \vee \alpha \in S_2\}$. Table 1 shows more operations on characteristic functions.

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 $\delta$ 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

As mentioned above, one of the challenges to apply SCT in industry is the state-space explosion problem when synthesizing the supervisor. After adopting Binary Decision Diagrams, the problem can be alleviated to some extent. But that's not enough, since the intuitive total representation of the transition function might still be huge. Two partitioning techniques are discussed in this section.

## 4.1 Safe-state Synthesis

The *safe-state* algorithm, an efficient supervisor synthesis algorithm, formally defined in (Vahidi et al., 2006), is used in this paper. The algorithm creates the supervisor by first building the candidate $S0 = P \parallel Sp$, then removing states from $Q^{S0}$ until the remaining safe states are both nonblocking 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 (Vahidi et al., 2006).

**Algorithm 1:** The Safe State Synthesis.

1: **input** : $Q_x, Q^{S0}$
2: **let** $X_0 := Q_x, k := 0$
3: **repeat**
4: $\quad k := k+1$
5: $\quad Q' := RestrictedBackward(Q_m, X_{k-1})$
6: $\quad Q'' := UncontrollableBackward(Q^{S0} \backslash Q')$
7: $\quad X_k := X_{k-1} \cup (Q'')$
8: **until** $X_k = X_{k-1}$
9: **return** $Q^{S0} \backslash X_k$

## 4.2 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 the large and complicated modular DES, the BDD representation of the total transition function $\delta^{Sp\|P}$ 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 (Burch et al., 1991; Burch et al., 1994), 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:

$$\hat{\delta}^i(q^i,\sigma) = \begin{cases} \delta^i(q^i,\sigma) & \text{if } \delta^i(q^i,\sigma) \\ q^i & \text{if } \sigma \notin \Sigma^i \\ \text{undefied} & \text{otherwise} \end{cases} \quad (2)$$

$$\delta = \bigwedge_{1 \le i \le n} \hat{\delta}^i \quad (3)$$

By making use of the above equations (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 $\acute{q}^i$.

**Algorithm 2:** ConjForwardReachability.

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

**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 in SCT, the intermediate states ($Q_{k-1}$) can still cause the explosion problem when performing the reachability search, which prevents the conjunctive partitioning technique from being applied to large systems in SCT. The disjunctive partitioning of the full synchronous composition, explained subsequently, on the other hand, is then shown to be an appropriate partitioning.

Assuming the automaton set $A = \{A_1, \ldots, A_n\}$ and the state $q = \langle q^1, q^2, \ldots, q^n \rangle$, a disjunctive transition function, the partial transition function $\check{\delta}^i$, is defined based on the event $\sigma \in \Sigma^i$ and the dependent set $D(A^i)$:

$$\check{\delta}^i(q,\sigma) = \left( \bigwedge_{A^j \in D(A^i)} \zeta^{i,j}(q^j,\sigma) \right) \wedge \left( \bigwedge_{A^k \notin D(A^i)} q^k \overset{\sigma}{\leftrightarrow} q^k \right) \quad (4)$$

$$\zeta^{i,j}(q^j,\sigma) = \begin{cases} \delta^j(q^j,\sigma) & \text{if } \sigma \in \Sigma^i \cap \Sigma^j \\ q^j & \text{otherwise} \end{cases} \quad (5)$$

and

$$D(A^i) = \{A^j \in A \mid \exists A^i \in A \text{ where } \Sigma^i \cap \Sigma^j \neq \emptyset\} \quad (6)$$

The total transition function is defined as

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

The construction of the dependent set for each automaton can be obtained through calculating which automaton shares any event with it. Taking Fig. 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 $\Sigma^A$ and $\Sigma^C$. By using equations (4) and (5), 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.

## 4.3 Workset based Strategies

In Section 4.2, 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 (Byröd et al., 2006), 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 (Vahidi et al., 2006). 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, "·" 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:** WorksetForwardReachbility.

```
1: input : Q_init, {δ̌^1, ..., δ̌^n}
2: let W_0 := {δ̌^1, ..., δ̌^n}, Q_0 := Q_init
3: repeat
4:    ℍ: Pick and remove a transition δ̌^i ∈ W_k
5:    k := k + 1
6:    Q_k := Q_{k-1} ∪ {q̂ | ∃q ∈ Q_{k-1}, δ̌^i(q, ·) = q̂}
7:    if Q_k ≠ Q_{k-1} then
8:       W_k := W_{k-1} ∪ E(δ̌^i)
9:    end if
10: until W = ∅
11: return Q_k
```

**Selection Heuristics**

In Algorithm 3, $\mathbb{H}$ denotes the heuristics of selecting the next transition relation for the reachability search such that the number of intermediate BDD nodes is computed as small as possible. How a transition $\check{T}^i$ is chosen among those in the working set $W$ 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 on 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, see Fig.3. Using this method, a complex selection procedure can be described as a combination of two selection rules. In the current implementation, the first stage $H1$ selects a subset $W' \subset W$ to be sent to $H2$ using one of the following rules:

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

In case $W'$ is not a singleton, the second stage $H2$ is used to choose a single transition relation $\check{T}^i$ among $W'$. In the experiment, the following shown heuristics can significantly reduce the number of intermediate BDD nodes for the simple problems.

1. *Reinforcement learning* (R) (Kaelbling et al., 1996): Choose the best transition relation based on the previous activity record.
2. *Reinforcement learning + Tabu* (RT) (Glover and Laguna, 1997): Same as the reinforcement learning with the difference that using *tabu search* for the selection policy.

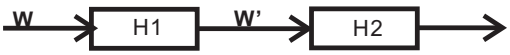


Figure 3: The two stage selection heuristics for the workset algorithm.

# 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, an approach, the guard generation procedure which originates from (Miremadi, 2010), is discussed and combined with the BDD-based disjunctive partitioning approach.

The guard generation procedure, being dependent on three kinds of state sets (explained below), extracts a set of conditional propositional formulae, referred to as *guards* indicating under which conditions the event can be executed without violating the specifications and attaches them to the original models. In (Miremadi, 2010), the computation of the monolithic transition function is the prerequisite for generating these state sets. Here an alternative way which is based on the disjunctive partitioning approach is presented.

## 5.1 Computation of the Basic State Set

Concerning the states that are retained or removed after the synthesis process, the states that enable an arbitrary event $\sigma$ 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 $Q_{\mathbf{f}}^{\sigma}$, is the set of states in the supervisor where the execution of $\sigma$ is defined for $S_0$, but not for the supervisor. The allowed state set, denoted by $Q_{\mathbf{a}}^{\sigma}$, is the set of states in the supervisor where the execution of $\sigma$ is defined for the supervisor. In other word, for each event $\sigma$ in $S_0$'s alphabet, $Q_{\mathbf{a}}^{\sigma}$ represents the set of states where event $\sigma$ 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 $Q_{\mathbf{dc}}^{\sigma}$, which describes a situation where executing $\sigma$ 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 $Q_{\mathbf{f}}^{\sigma}$ and the allowed states $Q_{\mathbf{a}}^{\sigma}$ 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 the synthesis algorithm in Section 4.1. The don't care state set, $Q_{\mathbf{dc}}^{\sigma}$ can be defined as the complement of the union of $Q_{\mathbf{a}}^{\sigma}$ and $Q_{\mathbf{f}}^{\sigma}$ (9). The proof can be found in (Miremadi, 2010).

$$Q_{\mathbf{dc}}^{\sigma} = C(Q_{\mathbf{a}}^{\sigma} \cup Q_{\mathbf{f}}^{\sigma}) \tag{9}$$

**Algorithm 4:** Compute $Q_{\mathbf{f}}^{\sigma}$.

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

**Algorithm 5:** Compute $Q_{\mathbf{a}}^{\sigma}$.

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

## 5.2 Guard Generation

Based on the basic state sets, some logic restrictions can be extracted, expressing under which conditions the events can be executed without violating the specifications. For every automaton in the DES, a new variable $v$ is introduced to hold the current state of the automaton. The following propositional function for the event $\sigma$, $G^{\sigma}: Q^{A^1} \times Q^{A^2} \times \ldots \times Q^{A^n} \rightarrow \mathbb{B}$, referred to as *guards*, is defined as:

$$G^{\sigma}\langle v_{A^1}, v_{A^2}, \ldots v_{A^n}\rangle = \begin{cases} \text{true} & \langle v_{A^1}, v_{A^2}, \ldots v_{A^n}\rangle \in Q_{\mathbf{a}}^{\sigma} \\ \text{false} & \langle v_{A^1}, v_{A^2}, \ldots v_{A^n}\rangle \in Q_{\mathbf{f}}^{\sigma} \\ \text{don't care} & \text{otherwise} \end{cases} \tag{10}$$

where $\mathbb{B}$ is the set of Boolean values and $v_{A^i}$ represents the current state of automaton $A^i$. In particular, $\sigma$ 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 (Miremadi, 2010).

Table 2: Nonblocking and controllability synthesis.

| | | | Conjunctive Synthesis | | Disjunctive Synthesis | |
|---|---|---|---|---|---|---|
| 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) | $1.07 \times 10^9$ | $8.49 \times 10^4$ | 2352 | 1.69 | 299 | 0.59 |
| Transfer line (10,3) | $1.15 \times 10^{18}$ | $6.13 \times 10^{13}$ | 31022 | 48.36 | 1257 | 3.89 |
| Transfer line (15,3) | $1.23 \times 10^{27}$ | $4.42 \times 10^{20}$ | – | – | 3032 | 12.80 |
| Cat&mouse (1,1) | 20 | 6 | 43 | 0.02 | 31 | 0.05 |
| Cat&mouse (1,5) | 605 | 579 | 2343 | 0.08 | 273 | 0.09 |
| Cat&mouse (5,1) | 1056 | 76 | 848 | 0.30 | 305 | 0.30 |
| Cat&mouse (5,5) | $6.91 \times 10^9$ | $3.15 \times 10^9$ | – | – | 15964 | 20.86 |

* - denotes memory out.

# 6 CASE STUDIES

What we have discussed in the previous sections has been implemented and integrated in the supervisory control tool Supremica (Åkesson et al., 2003; Åkesson et al., 2006) which uses *JavaBDD* (JavaBDD, 2007) as BDD package. In this section, the implemented program will be applied to a set of relatively complicated examples [1]. The examples are presented including references in the right column.

## 6.1 Benchmark Examples

A set of benchmark examples is briefly described as follows.

**Automated Guided Vehicles**

An AGV system, described in (Holloway and Krogh, 1990), 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 (Leduc, 2002), 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 (Wonham, 1999), defines a very simple factory consisting of a series of n 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. Fig. 4 shows the single cell in the transfer line. The capacity of each buffer is *m*, which is usually chosen to be either 1 or 3.

Figure 4: A single cell in the transfer line.

**The Extended Cat and Mouse**

An extended cat and mouse problem (Miremadi et al., 2008), which is more complicated than the transfer line model, generalizes the classic one presented in (Ramadge and Wonham, 1987). 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. 2. 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**

Fig. 2 shows the result of applying two partitioning techniques for the examples explained above. The

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

Table 3: Computing time for the nonblocking supervisor with different heuristics.

| | Computation Time | | | |
|---|---|---|---|---|
| $(n,k)$ | 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 |

supervisors synthesized for these examples are both nonblocking 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 $10^7$.

However, with DESs getting larger and more complicated, the conjunctive partitioning technique is not capable of synthesizing nonblocking 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, in the figure, shows that the disjunctive partitioning together with heuristic decisions can effectively reduce the number of intermediate BDD nodes.

**Heuristics**

Table 3 shows the computing time for synthesizing the nonblocking supervisor from different instances. Different combinations of heuristics, presented in Section 4.3, 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 3, 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.

# 7 CONCLUSIONS

In this paper, we improved and extended our previous work, the guard generation procedure, by applying an approach to efficiently performing symbolic reachability exploration on composite discrete event systems. More specifically, the content of the paper can be summarized as follows:

1. Introduce the partitioning techniques to split the BDD representation of $\delta^{Sp\|P}$ into a set of smaller but structural components.
2. 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.
3. 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.
4. 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

Akers, S. B. (1978). Binary decision diagrams. *IEEE Transactions on Computers*, 27:509–516.

Åkesson, K., Fabian, M., Flordal, H., and Malik, R. (2006). Supremica—an integrated environment for verification, synthesis and simulation of discrete event systems. In *Proceedings of the 8th international Workshop on Discrete Event Systems WODES08*, pages 384–385.

Åkesson, K., Fabian, M., Flordal, H., and Vahidi, A. (2003). Supremica—a tool for verification and synthesis of discrete event supervisors. In *11th Mediterranean Conference on Control and Automation*.

Aloul, F. A., Markov, I. L., and Sakallah, K. A. (2003). Force: a fast and easy-to-implement variable-ordering heuristic. In *in ACM Great Lakes Symposium on VLSI*, pages 116–119.

Balemi, S., Hoffmann, G. J., Gyugyi, P., Wong-Toi, H., and Franklin, G. F. (1993). Supervisory control of a rapid thermal multiprocessor. *IEEE Transactions on Automatic Control*, 38(7):1040–1059.

Bollig, B. and Wegener, I. (1996). Improving the variable ordering of OBDDs is NP-complete. *IEEE Trans. Comput.*, 45(9):993–1002.

Bryant, R. E. (1992). Symbolic manipulation with ordered binary decision diagrams. *ACM Computing Surveys 24*, 24:293–318.

Burch, J. R., Clarke, E. M., and Long, D. E. (1991). Symbolic model checking with partitioned transition relations. In *Proceedings of the International Conference on Very Large Scale Integration*, volume A-1 of *IFIP Transactions*, pages 49–58. North-Holland.

Burch, J. R., Clarke, E. M., Long, D. E., Mcmillan, K. L., and Dill, D. L. (1994). Symbolic model checking for sequential circuit verification. *IEEE Transactions on ComputerAided Design of Integrated Circuits and Systems*, 13(4):401–424.

Byröd, M., Lennartson, B., Vahidi, A., and Åkesson, K. (2006). Efficient reachability analysis on modular discrete-event systems using binary decision diagrams. In *Proceedings of the 8th international Workshop on Discrete Event Systems, WODES'06*, pages 288–293.

Cassandras, C. G. and Lafortune, S. (2008). *Introduction to Discrete Event Systems*. Springer, 2nd edition.

Feng, L., Wonham, W. M., and Thiagarajan, P. S. (2007). Designing communicating transaction processes by supervisory control theory. *Form. Methods Syst. Des.*, 30(2):117–141.

Glover, F. and Laguna, M. (1997). *Tabu search*. Number 1 in Wiley-Interscience Series in Discrete Mathematics and Optimization. Kluwer Academic Publishers.

Hoare, C. A. R. (1985). Communicating sequential processes. *Communications of the ACM*, 21(8):666–677.

Holloway, L. E. and Krogh, B. H. (1990). Synthesis of feedback control logic for a class of controlled Petri Nets. *IEEE Transactions on Automatic Control*, 35(5):514–523.

JavaBDD (2007). available online.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, 4:237–285.

Leduc, R. J. (2002). *Hierarchical interface-based supervisory control*. PhD thesis, Department of Electrical and Computer Engineering, University of Toronto.

Miremadi, S. (2010). *Symbolic Computation of Control Functions Modeled by Automata with Variables*. Licentiate thesis, Chalmers University of Technology, Gothenburg, Sweden.

Miremadi, S., Åkesson, K., Fabian, M., Vahidi, A., and Lennartson, B. (2008). Solving two supervisory control benchmark problems using supremica. In *Discrete Event Systems, 2008. WODES 2008. 9th International Workshop on*, pages 131–136.

Ramadge, P. J. G. and Wonham, W. M. (1987). Supervisory control of a class of discrete event processes. *SIAM Journal of Control and Optimization*, 25(1):206–230.

Ramadge, P. J. G. and Wonham, W. M. (1989). The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98.

Shannon, C. E. and Weaver, W. (1949). *The mathematical theory of communication*, volume 181. University of Illinois Press.

Shoaei, M. R., Lennartson, B., and Sajed, M. (2010). Automatic Generation of Controllers for Collision-Free Flexible Manufacturing Systems. In *CASE 2010 6th IEEE Conference on Automation Science and Engineering*, page 7.

Vahidi, A., Fabian, M., and Lennartson, B. (2006). Efficient supervisory synthesis of large systems. *Control Engineering Practice*, 14(10):1157–1167.

Wonham, W. M. (1999). Notes on control of discrete event systems. Technical report, University of Toronto.