Symbolic reachability computation using the disjunctive partitioning technique in Supervisory Control Theory

Symbolic Reachability Computation Using the Disjunctive Partitioning Technique in Supervisory Control Theory

Z. Fei, K. Åkesson and B. Lennartson

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 a given specification is fulfilled. A problem, which prevents SCT from having a major breakthrough industrially, is that the supervisory synthesis often 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 efficient symbolic state-space traversal approach, depending on the disjunctive partitioning technique, is presented and the correctness of it is proved. Finally, the efficiency of the presented approach is demonstrated on a set of benchmark examples.**

I. INTRODUCTION

In *Supervisory Control Theory* (SCT) [1], [2], [3], given the model of a discrete event system (DES) to be controlled, the *plant*, and the intended behavior, the *specification*, a supervisor can be automatically synthesized, guaranteeing that the closed-loop system always fulfills the given specification. Here the aforementioned supervisor is said to be *minimally restrictive*, meaning the plant is given the greatest amount of freedom to generate events without violating the specification.

A typical issue in SCT is how to efficiently synthesize such minimally restrictive supervisor from a given modular description of the plant and the specification. Since the synthesis task involves a series of reachability computations, traditional explicit state-space traversal algorithms are often intractable due to *the space-state explosion problem*. To alleviate this problem, a well-know approach is to utilize binary decision diagrams (BDDs) [4], [5], to represent system models symbolically and compute supervisors monolithically [6], [7]. However, the monolithic state-space which is the prerequisite for the supervisor computation, might be too large to be constructed due to the intermediate explosion of BDD nodes, even though the final size is manageable. In [8], a state-space traversal algorithm which depends on the disjunctive partitioning technique is presented. An advantage of this approach is that the searchable state-space can be significantly increased, and thus industrially more interesting applications can be handled. However, when the state-space exploration involves a set of forbidden states, this traversal approach does not always search the state-space exhaustively, thus resulting in a supervisor which is not minimally restrictive.

The main contribution of this paper consists of three parts. First, we extend the prior state-space traversal approach in [8], to make it always perform the exhaustive exploration when a set of forbidden states is involved in. Secondly, the correctness of the modified traversal algorithm is formally proved. Additionally, the approach is demonstrated on a set of benchmark examples to compare the efficiency with the other approach.

II. PRELIMINARIES

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

A. Deterministic Finite Automata

Generally, a DES can be modeled either by textual languages such as regular expressions or graphically by for instance Petri nets or automata. Since we are only concerned about the deterministic systems in this paper, deterministic finite automata will be utilized as a framework to model discrete event systems.

Definition II.1 (Deterministic Finite Automaton): *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;
- $\delta : Q \times \Sigma \to Q$ 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 state transitions of an automaton are defined by the partial function $\delta : Q \times \Sigma \to Q$, which takes a *source-state* $q \in Q$ and an event $\sigma \in \Sigma$ and outputs a *target-state* $\acute{q} \in Q$. However, for computational purposes, it might be preferable to reformulate the transitions of an automaton as *relations*. The element $\langle q, \sigma, \acute{q} \rangle$ is said to be related if and only if there is a transition from q to $\acute{q}$ by the event σ. Therefore, the transition relation T of an automaton can be defined as follows:

$$T = \{\langle q, \sigma, \acute{q} \rangle \text{ if } \delta(q, \sigma) = \acute{q}\}.$$

In addition, for the symbolic reachability computation, it is not always necessary to include the event in the transition relation T. Leaving σ out will reduce the number of variables needed for the symbolic representation. In the rest of the paper, σ will be written out when needed.

The composition of two or more automata is realized by the *full synchronous composition* [9]. More specifically, the full synchronous composition of two automata $A^1 = (Q^1, \Sigma^1, T^1, q^1_{init}, Q^1_m)$ and $A^2 = (Q^2, \Sigma^2, T^2, q^2_{init}, Q^2_m)$ results in $A^1 \parallel A^2 = (Q, \Sigma^1 \cup \Sigma^2, T^{1\|2}, q_{init}, Q^1_m \times Q^2_m)$, where $Q \subseteq Q^1 \times Q^2$ and $q_{init} = (q^1_{init}, q^2_{init})$. The composite transition relation $T^{1\|2}$ is defined as:

$$\begin{cases} \langle (q^1,q^2),\sigma,(\acute{q}^1,\acute{q}^2)\rangle & \text{if } \langle q^1,\sigma,\acute{q}^1\rangle \in T^1 \wedge \langle q^2,\sigma,\acute{q}^2\rangle \in T^2 \\ \langle (q^1,q^2),\sigma,(\acute{q}^1,q^2)\rangle & \text{if } \langle q^1,\sigma,\acute{q}^1\rangle \in T^1 \wedge \sigma \notin \Sigma^2 \\ \langle (q^1,q^2),\sigma,(q^1,\acute{q}^2)\rangle & \text{if } \langle q^2,\sigma,\acute{q}^2\rangle \in T^2 \wedge \sigma \notin \Sigma^1 \\ \text{undefined} & \text{otherwise} \end{cases}$$

B. Supervisory Control theory

As described in Section I, 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. 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, $Sp = Sp_1 \parallel \ldots \parallel Sp_n$. Note that for each sub-specification Sp_i, $\Sigma^{Sp_i} \subseteq \Sigma^P$ and for the composed specification Sp, $\Sigma^{Sp} \subseteq \Sigma^P$, which means the specification (sub-specifications) can not specify more than what the plant can achieve.

In SCT, 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. Furthermore, δ_u denotes the transition function which is only associated with the uncontrollable events. The reason that an event is modeled as being uncontrollable is that either it is inherently unpreventable (”spontaneous”) or it has the high priority (”imperative”).

In addition, there are two properties [1], [2] 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.
- *Non-blocking*: The supervisor S guarantees that at least one marked state can be reached from every state in the closed-loop system $S \parallel P$.

It can be shown that a minimally restrictive supervisor that is both nonblocking and controllable can be constructed as an automaton S such that $Q^S = Q^{P\|Sp}$, with some transitions being removed [1], [2]. In [8], a slightly different approach to the supervisory synthesis, where invalid states are excluded, is proposed. As long as the considered system is in one of the remaining states, controllability and nonblocking is guaranteed. A *safe-state supervisor* can be acquired by first building the candidate $S_0 = P \parallel Sp$. Then states are iteratively excluded from Q^{S0} until the remaining states are both controllable and nonblocking. The remaining states are hereby called *safe states*, denoted by Q^S. Algorithm 1 shows the algorithm for computing these nonblocking and controllable safe states. Taking a set of explicit forbidden states Q_x as the input, the algorithm first computes a set of co-reachable states Q', without passing the forbidden states (restrictedBackward). Then a set of uncontrollable states Q'' is computed from the blocking state set $Q^{S_0} \backslash Q'$ (uncontrollableBackward). Those uncontrollable states in Q'' would reach the blocking states in $Q^{S_0} \backslash Q'$ on the occurrence of the uncontrollable events in Σ_u. The algorithm iteratively extends the forbidden state set by adding those blocking and uncontrollable states until there is no new forbidden state found. In practice, only Q^S of the supervisor needs representing. The transition function δ^S (T^S) can be constructed online. More detailed explanation can be found in [8].

Algorithm 1 Safe-state Supervisory Synthesis

```
 1: input : $Q_x \subseteq Q^{S_0}$
 2: let $k := 0, X_0 := Q_x$
 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: let $Q^S := Q^{S_0} \backslash X_k$
10: return $Q^S$
```

C. 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 [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.

1) 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:

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

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 (relation) δ

(T) follows the same rule but with the difference that the transition function (relation) 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.

III. Efficient Reachability Computation

Not surprisingly, reachability (co-reachability) searches 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 $T^{Sp\|P}$ might be extremely large to be constructed due to the immense intermediate size of BDD nodes. In this section, an efficient approach is proposed to further alleviate the state-space explosion problem. The approach is based on the partitioning technique and some heuristics to perform the reachability search intelligently.

A. Partitioning of the Full Synchronous Composition

To tackle the complexity of the transition relation $T^{Sp\|P}$, it is natural to split it into a set of less complex relations with a connection between them. Such methods are based on the conjunctive or disjunctive partitioning techniques.

1) Conjunctive representation: Conjunctive partitioning, introduced in [13], is an approach to efficiently 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 outside their alphabets. This leads to a situation where all automata have equal alphabets. With this conjunctive partitioning defined for each automaton, one can search the state-space without constructing the full monolithic transition relation. To see more detailed explanation, refer to [13].

2) 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, sometimes the *existential quantification* operation results in the size of BDD nodes even larger than the monolithic transition relation. The disjunctive partitioning technique, explained subsequently, on the other hand, is then shown to be an appropriate partitioning technique for SCT.

Given a set of automata $\{A_1,\ldots,A_n\}$, an automaton-based disjunctive transition relation $\check{T}^i$ of an automaton A^i is the set of source state-event-target state triplets:

$$\check{T}^i = \{\langle (q^1,\ldots,q^n),\sigma,(\acute{q}^1,\ldots,\acute{q}^n)\rangle : \\ \sigma\in\Sigma^i \wedge \langle (q^1,\ldots,q^n),\sigma,(\acute{q}^1,\ldots,\acute{q}^n)\rangle \in \widetilde{T}\}$$

where $\widetilde{T}$ denotes the composite transition relation of a complete system $A = A^1 \parallel \ldots \parallel A^n$, where n is the number of automata. From the above definition, we clearly have

$$\widetilde{T} = \bigvee_{1\le i\le n} \check{T}^i.$$

However, the problem is that we might have no access to the global transition relation $\widetilde{T}$. In [14], an approach, based on Definition III.1 and Claim III.1, is presented. The approach constructs the disjunctive transition relation $\check{T}^i$ of A^i directly without generating $\widetilde{T}$. The proof of Claim III.1 can be found in [14].

Definition III.1 (Dependency Set): The *dependency set* of an automaton A^i is defined as:

$$D(A^i) = \{A^j \mid 1\le j\le n \wedge j\ne i \wedge \Sigma^i\cap\Sigma^j \ne \emptyset\}$$

Claim III.1: The disjunctive transition relation $\check{T}^i$ is equal to the set of all transitions $\langle (q^1,\ldots,q^n),\sigma(\acute{q}^1,\ldots,\acute{q}^n)\rangle \in Q\times\Sigma\times Q$ obeying the following three conditions:

- $\sigma\in\Sigma^i$
- $\forall A^j \in D(A^i)$: $[\sigma\in\Sigma^j \wedge \langle q^j,\sigma,\acute{q}^j\rangle \in T^j] \vee [\sigma\notin\Sigma^j \wedge (\acute{q}^j = q^j)]$
- $\forall A^j \notin D(A^i)$: $\acute{q}^j = q^j$

Example III.1: Consider a simple manufacturing process [2] that involves two machines, M_1 and M_2, and a buffer B in between. When a part has been processed by M_1, it is placed in B, which has a capacity of one part only. The part is subsequently processed by M_2. Fig. 1 shows the models of two machines as plants and the buffer as the specification.

The construction of the dependency set for each automaton can be obtained through calculating which automaton shares any event with it. For simplicity, the transition relations and the states are all explicitly enumerated. Taking B as an example, B shares the event u_1 with the automaton M_1 and l_2 with M_2. Therefore, $D(B) = \{M_2, M_1\}$. Besides, the disjunctive transition relation $\check{T}^B$ can be constructed as:

$$\{\langle (W_1,E,I_2),(I_1,F,I_2)\rangle ; \langle (W_1,E,W_2),(I_1,F,W_2)\rangle ; \\ \{\langle (I_1,F,I_2),(I_1,E,W_2)\rangle ; \langle (W_1,F,I_2),(W_1,E,W_2)\rangle\}.$$

The dependency sets and the partial transition relations of the other automata can be constructed similarly.

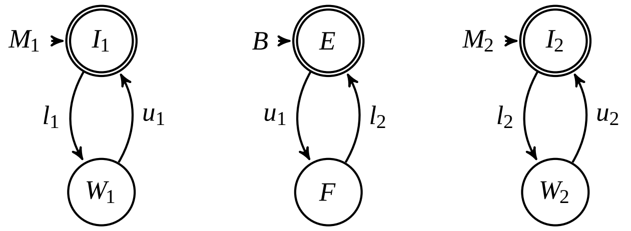

Fig. 1. The automata corresponding to example III.1. Two events u_1 and u_2 are modeled as the uncontrollable events.

B. Workset Strategy

Similar to the conjunctive partitioning technique, partitioning the state-space disjunctively also suffers from the intermediate BDD explosion problem. In order to substantially reduce the intermediate size of BDD nodes, in [8], an algorithm, referred to as the workset algorithm is presented to explore the state-space structurally. The algorithm maintains a set of active disjunctive transition relations W_k. These active transition relations are selected one at a time for the reachability search. If there is any new state found for the currently selected transition relation, then all of its *dependent transition relations,* $D(\check{T}^i) = \{\check{T}^j \mid A^j \in D(A^i)\}$, will be added in W_k. However, this algorithm does not realize the exhaustive search when a set of forbidden states is involved. In this section, we start with a counter example, analyze the problem and propose a way to modify the algorithm. Besides, the correctness of the modified workset algorithm is formally proved. Finally, a set of heuristic decisions is presented. Those heuristics are used to keep the intermediate size of BDD nodes as small as possible during the state-space exploration.

Algorithm 2 Workset Restricted Forward Reachability

```
 1: input $q_{init} := ({q^1}_{init} \times \ldots \times {q^n}_{init}), Q_x \subseteq Q^1 \times \ldots \times Q^n$,
    $W_0$ :=the set of all disjunctive transition relations;
 2: $Q_0 := \{q_{init}\}, k := 0$
 3: repeat
 4:     $\mathbb{H}$:  Pick and remove $\check{T}^i \in W_k$
 5:     $k := k+1$
 6:     $Q_k := Q_{k-1} \cup \{\acute{q} \mid \exists q \in Q_{k-1}, \acute{q} \notin Q_x, \langle q, \acute{q}\rangle \in \check{T}^i\}$
 7:     if $Q_k \neq Q_{k-1}$ then
 8:         $W_k := W_{k-1} \cup D(\check{T}^i)$
 9:     end if
10: until $W = \emptyset$
11: return  $Q_k$
```

1) Restricted Reachability: The problem of the workset algorithm in [8] lies in the restricted reachability algorithm which is part of the synthesis procedure (restrictedBackward). For the normal reachability algorithm, such as searching all reachable states from the initial state or all coreachable states from the marked states, the workset algorithm, which exhaustively explores the state-space works fine. The formal proof can be found in [14]. The restricted reachability algorithm which involves a set of forbidden states, on the other hand, can not realize the exhaustive exploration. In Example III.2, the counter example shows that when performing the restricted forward reachability on the disjunctive transition relations with forbidden states, the workset algorithm does not produce the desired result. The incorrectness of the restricted backward reachability search can be proved symmetrically.

Example III.2: Consider Example III.1, a controllable supervisor is required to be synthesized after verifying that the closed-loop system is uncontrollable and having found two uncontrollable states, (W_1, F, W_2) and (W_1, F, I_2). To guarantee that all of the states in the controllable supervisor are reachable from the initial state, the restricted forward reachability search is required to exclude the unreachable states. Suppose that the sequence of disjunctive transition relations selected by the workset algorithm is $\{\check{T}^{M_1}$, $\check{T}^{M_2}$, $\check{T}^B\}$, the original workset algorithm (as Algorithm 2 shows) performs the reachability search as follows.

Step 1: From the initial state (I_1, E, I_2), three new states (W_1, E, I_2), (I_1, F, I_2), (W_1, F, I_2) are found through $\check{T}^{M_1}$. Among these states, (W_1, F, I_2) is the forbidden state and should be excluded. Due to the newly found states, the dependent transition relations of M_1 ought to be added into the workset. In this case, the workset is unchanged since the only dependent transition relation of M_1 is $\check{T}^B$, and it is already in the workset.

Step 2: Next, the transition relation of M_2, $\check{T}^{M_2}$ is used for searching more states. One more state is found, (I_1, E, W_2). Now the workset only has one transition relation $\check{T}^B$ left.

Step 3: The transition relation $\check{T}^B$ is selected and removed from the workset for performing the reachability search. No more new state is found and the workset is empty. Hence the algorithm terminates and the number of reachable states is 4. However the correct number of reachable states should be 6, which can be acquired by removing forbidden states and associated transitions from the composed automaton.

From the above steps, it can be observed that after finding the new state (I_1, E, W_2), it is possible to find two more new states on $\check{T}^{M_1}$. Unfortunately, $\check{T}^{M_1}$ does not belong to the dependent transition relations of M_2 (no shared events). Consequently, the algorithm does not add $\check{T}^{M_1}$ in the workset for searching again.

Algorithm 3 Modified Restricted Forward Reachability

```
 1: input $q_{init} := ({q^1}_{init} \times \ldots \times {q^n}_{init}), Q_x \subseteq Q^1 \times \ldots \times Q^n$,
    $W_0$ :=the set of all disjunctive transition relations;
 2: let $S_u$ :=Set of disjunctive transition relations containing
    states in $Q_x$ as target states;
 3: $Q_0 := \{q_{init}\}, k := 0$
 4: repeat
 5:     $\mathbb{H}$:  Pick and remove $\check{T}^i \in W_k$
 6:     $k := k+1$
 7:     $Q_k := Q_{k-1} \cup \{\acute{q} \mid \exists q \in Q_{k-1}, \acute{q} \notin Q_x, \langle q, \acute{q}\rangle \in \check{T}^i\}$
 8:     if $Q_k \neq Q_{k-1}$ then
 9:         $W_k := W_{k-1} \cup D(\check{T}^i)$
10:     end if
11: until $W = \emptyset$
12: repeat
13:     $k := k+1$
14:     for all $j$ such that $\check{T}^j \in S_u$ do
15:         $Q_k := Q_{k-1} \cup \{\acute{q} \mid \exists q \in Q_{k-1}, \acute{q} \notin Q_x, \langle q, \acute{q}\rangle \in \check{T}^j\}$
16:     end for
17: until $Q_k = Q_{k-1}$
18: return  $Q_k$
```

2) Modified Workset Algorithm: The workset algorithm needs to be modified to take into account the disjunctive transition relations which have not been searched exhaustively. One way to fix the problem is to repeatedly use the current reachable states to perform the reachability computation on all of the disjunctive transition relations, until there is no new state found. Although the algorithm modified in this way is correct, the main problem is the performance penalty. Here we present a different but simple modification to the workset algorithm. As Algorithm 3 shows, instead of repeatedly performing the reachability computation for all the disjunctive transition relations, the modified algorithm only considers and performs the reachability search on the relations which contain the forbidden states as target states. The proof of correctness of the presented algorithm is shown as follows.

Theorem III.1: Given a set of deterministic finite automata $A^1, A^2, \ldots, A^n$ and a set of forbidden states $Q_x \subseteq Q^1 \times \ldots \times Q^n$. Let Q_R be the set of states reachable from the initial state q_{init} without passing any state in Q_x. Let Q_{Alg} be the result of Algorithm 3. Then $Q_R = Q_{Alg}$ holds.

Proof: In order to prove $Q_R = Q_{Alg}$, we prove two set inclusions $Q_{Alg} \subseteq Q_R$ and $Q_R \subseteq Q_{Alg}$ respectively.

$Q_{Alg} \subseteq Q_R$: The first set inclusion is straightforward. Algorithm 3 takes the given initial state q_{init} as the argument to perform the reachability searches on the set of transition relations. During the execution, some target states will be found by the initial state. Then those found states including the initial state become the source states and continue to search more target states. The algorithm terminates when there is no target state found. Therefore, all the states found by Algorithm 3 are reachable.

$Q_R \subseteq Q_{Alg}$: This set inclusion means that there exists no state $\acute{q} \in Q_R \setminus Q_{Alg}$. In other words, there doesn't exist such a state $\acute{q}$ which is reachable but not found by Algorithm 3. We prove it by contradiction. Here we separate Algorithm 3 into two parts: loop 1 (line number $4-11$) and loop 2 (line number $12-18$). Note that loop 2 is an exhaustive reachability search of $S_u \subseteq W_0$.

Case 1: If there exists $q_c \in Q_{Alg}$, for some j, $\langle q_c, \acute{q} \rangle \in \check{T}^{A^j}$ and $\check{T}^{A^j} \in S_u$. Since the reachability search of loop 2 is exhaustive, $\acute{q}$ must be found, which leads to a contradiction.

Case 2: If there does not exist such $q_c \in Q_{Alg}$ that $\langle q_c, \acute{q} \rangle \in \check{T}^{A^j}$ and $\check{T}^{A^j} \in S_u$, by assumption, $\acute{q}$ must not be found in loop 1. Since $\acute{q} \in Q_R$, there exists a string $s \in \Sigma^*$, such that $q_{init} \to \ldots q_e \to \acute{q}$, where $q_e \in Q_{Alg}$. Hereby, $\langle q_e, \acute{q} \rangle$ should not be executed in loop 1. Here we claim that q_e must not be found in loop 1 but found in loop 2. That is because for all partial transition relations which contain $\langle q_e, \acute{q} \rangle$, they are not in S_u. If q_e can be found in loop 1, $\acute{q}$ must be found in loop 1 as well. Since now, q_e can only be found in loop 2, there exists $q_r \in Q_{Alg}$ such that $\langle q_r, q_e \rangle \in \check{T}^{A^j}$ where $\check{T}^{A^j} \in S_u$ for some j. Similarly, we could prove that q_r must be found in loop 2 but not in loop 1. Recursively, it can be deduced that q_{init} must be found in loop 2 but not in loop 1, which leads to a contradiction, since q_{init} is in Q_0.

Since we have $Q_{Alg} \subseteq Q_R$ and $Q_R \subseteq Q_{Alg}$, then $Q_R = Q_{Alg}$ holds. ■

Correspondingly, the workset restricted backward reachability algorithm can be modified in the same way[1]. The correctness of the modified backward reachability algorithm is proved as follows.

Theorem III.2: Given a set of deterministic finite automata $A^1, A^2, \ldots, A^n$ and a set forbidden states $Q_x \subseteq Q^1 \times \ldots \times Q^n$. Let Q_{Co} be the set of states coreachable from the marked states Q_m without passing any state in Q_x. Let Q_{Alg} be the resultant set of states found by the modified restricted backward reachability algorithm. $Q_{Co} = Q_{Alg}$.

Proof: The proof of this theorem follows the same strategy as the previous one. Actually, the only difference is that when proving $Q_{Co} \subseteq Q_{Alg}$, we show that at least one of the marked states has not been found in loop 1 but found in loop 2, which leads to a contradiction. ■

3) Heuristic Decisions: In Algorithm 3, $\mathbb{H}$ denotes the heuristics of choosing the next disjunctive transition relation for the reachability search such that the intermediate size of BDDs is computed as small as possible. How a transition $\check{T}^i$ is chosen among those in the work set has a 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.

To find a good heuristic, a two-stage reference heuristic was implemented, see Fig. 2. 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:

- *MaxFollowers*: Choose the automata with the largest dependency set cardinality.
- *MinFollowers*: 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 intermediate size of BDD nodes:

- *Reinforcement learning*[15]: Choose the best transition relation based on the previous activity record.
- *Reinforcement learning + Tabu*[16]: Utilize *tabu search* for the selection policy in the reinforcement learning.

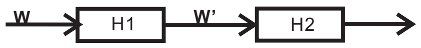

Fig. 2. The two stage selection heuristics for the Workset algorithm

IV. ALGORITHM EFFICIENCY

What we have discussed in the previous sections has been implemented and integrated in the supervisory control tool Supremica [17], [18] which uses *JavaBDD* [19] as the BDD package. In this section, the implemented program is applied to a set of relatively complicated examples including *The Transfer Line* [2], *The Extended Cat and Mouse* [20], *Automated Guided Vehicles*(AGV) [21], *Parallel Manufacturing Example* [22]. The comparison is made between the time-efficiency and space-efficiency of the conjunctive and disjunctive partitioning techniques.

Table I[2] shows the result of applying two partitioning techniques for the benchmark examples above. In the disjunctive partitioning based synthesis, the modified reachability algorithm together with a combination of two heuristics (H1: MaxFollowers, H2: Reinforcement learning + Tabu) is used to explore the state-space. The supervisors synthesized for

[1]We forego the relevant details due to space limitations.

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

TABLE I

COMPARISON BETWEEN TWO PARTITIONING TECHNIQUES

			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)	1.07×10^9	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

these examples are both nonblocking and controllable. It is observed that both of the partitioning based algorithms can handle AGV and Parallel Manufacturing Example, for which the number of reachable states is up to 10^7.

However, with DESs getting larger (The Transfer Line) and more complicated (The Extended Cat and Mouse), the conjunctive partitioning based synthesis is not capable of synthesizing nonblocking and controllable supervisors due to the intermediate BDD node explosion (the "BDD Peak" column). The disjunctive partitioning based synthesis, on the other hand, could successfully explore the state-space and synthesize the supervisors within an acceptable time. The comparison that how different heuristics are chosen influences the reachability algorithm performance can be found in [23].

V. CONCLUSIONS

In this paper, we have given a theoretical analysis of a set of algorithms designed to perform efficient reachability computation on composite discrete event systems. After identifying the problem in the original restricted reachability algorithm, a modified version is proposed and shown to be correct. Furthermore, a set of heuristic decisions is presented to guarantee the state space is explored in a structured way and keep the intermediate size of BDD nodes as small as possible.

Finally, we demonstrated the performance of the reachability algorithms on a set of benchmark examples. We conclude that compared with the conjunctive way of partitioning the composite transition relation, the disjunctive partitioning is more efficient for solving large supervisory problems.

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] W. M. Wonham, "Notes on control of discrete event systems," University of Toronto, Tech. Rep., 1999.

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

[4] S. B. Akers, "Binary decision diagrams," *IEEE Transactions on Computers*, vol. 27, pp. 509–516, JUN 1978.

[5] R. E. Bryant, "Symbolic manipulation with ordered binary decision diagrams," *ACM Computing Surveys 24*, vol. 24, pp. 293–318, 1992.

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

[7] G. Hoffmann and H. Wong-Toi, "Symbolic Synthesis of Supervisory Controllers," in *Proceedings Of The American Control Conference*, 1992, pp. 2789–2793.

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

[9] C. A. R. Hoare, "Communicating sequential processes," *Communications of the ACM*, vol. 21, no. 8, pp. 666–677, 1985. [Online]. Available: http://portal.acm.org/citation.cfm?doid=357980.358021

[10] C. E. Shannon and W. Weaver, *The Mathematical Theory of Communication*. University of Illinois Press, 1949, vol. 181.

[11] B. Bollig and I. Wegener, "Improving the variable ordering of obdds is np-complete," *IEEE Trans. Comput.*, vol. 45, no. 9, pp. 993–1002, 1996.

[12] F. A. Aloul, I. L. Markov, and K. A. Sakallah, "Force: a fast and easy-to-implement variable-ordering heuristic," in *ACM Great Lakes Symposium on VLSI*, 2003, pp. 116–119.

[13] J. R. Burch, E. M. Clarke, and D. E. Long, "Symbolic Model Checking with Partitioned Transition Relations," in *Proceedings of the International Conference on Very Large Scale Integration*, ser. IFIP Transactions, vol. A-1. North-Holland, 1991, pp. 49–58.

[14] M. Byröd, B. Lennartson, A. Vahidi, and K. Åkesson, "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*, July 2006, pp. 288–293.

[15] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[16] F. Glover and M. Laguna, *Tabu search*, ser. Wiley-Interscience Series in Discrete Mathematics and Optimization. Kluwer Academic Publishers, 1997, no. 1. [Online]. Available: http://www.springerlink.com/index/x4t27q703t428x46.pdf

[17] K. Åkesson, M. Fabian, H. Flordal, and A. Vahidi, "Supremica—a Tool for Verification and Synthesis of Discrete Event Supervisors," in *11th Mediterranean Conference on Control and Automation*, 2003.

[18] 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 WODES08*, 2006, pp. 384–385.

[19] JavaBDD, avilable online. [Online]. Available: http://javabdd.sourceforge.net

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

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

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

[23] Z. Fei, S. Miremadi, K. Åkesson, and B. Lennartson, "Efficient Symbolic Supervisory Synthesis and Guard Generation," in *Proceedings of 3rd International Conference on Agents and Artificial Intelligence*, vol. 1, 2011, pp. 106 – 115.