

Efficient Reachability Analysis on Modular Discrete-Event Systems using Binary Decision Diagrams

Martin Byröd, Bengt Lennartson, Arash Vahidi, Knut Åkesson
Automation Research Group,
Department of Signals and Systems, Chalmers University of Technology
SE-412 96 Göteborg, Sweden
{martin.byrod, bl, vahidi, knut}@s2.chalmers.se

Abstract—A well known strategy for handling the exponential complexity of modular discrete event systems is to represent the state space symbolically rather than explicitly, using binary decision diagrams (BDDs). In this paper, key success factors in the design of efficient BDD-based reachability algorithms for synthesis and verification are discussed. Furthermore, it is shown how the modular structure of a discrete event system (DES) can be utilized by taking advantage of the *process communication graph* and *partitioning techniques*. Finally, a recently introduced reachability algorithm based on these principles is discussed and a proof of correctness for this algorithm is given.

I. INTRODUCTION

A general problem with computations in the supervisory control theory of Ramadge and Wonham [1] is the exponential growth of states in a modular discrete event system (DES) as the number of sub-systems increases. With explicit representation of all reachable states this quickly leads to unmanageable time and memory complexity.

We fight the exponential growth of states by employing a symbolic representation of the state space. More specifically, we make use of binary decision diagrams (BDDs) introduced by Akers [2] and later extended by Bryant [3]. A BDD is an efficient data structure, which under the right conditions can reach logarithmic compression of the state space [3]. Generally, BDD-based synthesis and verification rely on reachability calculations on the state space. We therefore focus on designing efficient algorithms for reachability analysis.

In general, adapting existing algorithms for use with BDDs is not entirely straightforward. As noted by Hu [4] among others, special attention has to be paid to certain issues in order to bring out the full potential of the BDDs. Of these issues, finding a good variable ordering is arguably the most important concern [3]. Variable ordering has been studied in some detail by a number of researchers [5], [6], [7]. Unfortunately the problem of finding the optimal variable ordering is NP-complete [8]. Still though, a sub-optimal, but well-functioning variable ordering can be found by dividing the problem into two sub-problems; First find a good *internal* ordering for each sub-automaton, thereafter find an ordering *between* the different sub-automata [5], [9].

For systems of realistic size, though, variable ordering alone is not enough. A well known issue with BDD-based reachability searches is the size of the BDDs. Commonly, the intermediate BDDs become very large. This can quickly become a serious impediment to efficient computation. As discovered by Burch et al. [10] and also mentioned in e.g. [11], naive breadth first reachability exploration is in general ill-suited for use with BDDs. To reach significant BDD reduction it is crucial to explore the search space in an intelligent way. The key is to impose *structure* on the state space exploration. In this paper we give the idea of structured reachability a deeper information theoretic meaning. Furthermore, we illustrate how the earlier proposed *workset* algorithm [12] utilize the principle of structured search to facilitate BDD-compression.

The workset algorithm guides the search by making use of the *process communication graph* (PCG) introduced by Aziz et al. [5]. The PCG is a graphical representation of the interaction of automata under full synchronous composition and provides valuable information both for variable ordering and for guiding the state space exploration. In addition to this, we make use of *heuristics* to guide the search. In [13], the workset algorithm is shown to be useable for synthesis and verification of discrete event systems with extremely large state spaces ($> 10^{200}$ states in some cases).

Different ways of guiding the search have been studied by e.g. Ravi et al. who use user-specified *hints* to alter the exploration order [14]. In contrast, the approach in this paper is fully automatic.

A further important ingredient in the workset algorithm is the use of *partitioning techniques* introduced by Burch et al. [10]. BDD-based algorithms rely on a relational formulation of the DES, replacing the transition function with a *transition relation*. Burch et al. partition the transition relation into a set of conjuncts or disjuncts. These partitioning techniques have since been used by others in a number of contexts [11], [15].

In [16] we introduced a straightforward but non-trivial adaptation of these techniques to work with finite automata under full synchronous composition. Whereas the general concept of partitioning is rigorously defined in [10], the actual

construction of the partial transition relations can be done in many ways. Our method works by constructing one partial transition relation for each automaton. In particular, we make use of *disjunctive partitioning*, i.e. we represent the monolithic transition relation T by a collection of subsets that form T by union (disjunction in the language of logic). Here, the term *monolithic* refers to the complete, fully synchronized DES.

In this paper we introduce a reformulation of the disjunctive partitioning to clarify the mechanisms of the earlier proposed BDD-based synthesis and verification algorithms [17], [13], [18]. Moreover, we give an in-depth discussion and analysis of these algorithms. Specifically, we discuss how it is possible to avoid excessive BDD growth during intermediate phases of the reachability search.

Central to the workset algorithm is a mechanism that improves search efficiency by switching the partial transition relations on and off during the search. Here, it is crucial to make use of the information in the PCG to be able to ensure that all reachable states are found (even though the transition relation is only partially enabled). To formalize the fact that this mechanism works, we give a proof of correctness of the workset algorithm in this paper.

II. PRELIMINARIES AND NOTATION

We focus on supervisor synthesis based on reachability calculations on finite automata (FA). An FA is defined as a quintuple $A = \langle Q, \Sigma, \delta, q_i, Q_m \rangle$, where Q is the set of states, Σ is the alphabet, $\delta : Q \times \Sigma \rightarrow Q$ is a partial transition function, q_i is the initial state and Q_m is a set of marked states.

In this paper, we pay special attention to modular systems $A = A^1 \parallel \dots \parallel A^i \parallel \dots \parallel A^N$ consisting of a number of finite automata (FA) under under Hoare's full synchronous composition [19].

Below, some specific concepts on which this work is based are presented briefly.

A. Relational Formulation of a Finite Automaton

A finite automaton is usually defined with a transition function $\delta : Q \times \Sigma \rightarrow Q$, that takes a from-state $q \in Q$ and an event $\sigma \in \Sigma$ and outputs a to-state $q^+ = \delta(q, \sigma)$. However, for computational purposes it might be preferable to reformulate the transitions of the automaton as relations, using the language of set theory.

A *relation* is a rule that says whether a certain number of elements $\omega_1, \dots, \omega_n$ in a set Ω are *related*. A relation on Ω is represented by a subset $R \subseteq \Omega^n$ of Ω^n by saying that the elements $\omega_1, \dots, \omega_n$ are related iff $\langle \omega_1, \dots, \omega_n \rangle \in R$.

In the context of automata, relation basically means transition. Here, the elements $\langle q, \sigma, q^+ \rangle$ are said to be related iff there is a transition from q to q^+ by the event σ . This allows us to define the transition relation T of an automaton A . We write T as a subset of $Q \times \Sigma \times Q$.

$$T = \{ \langle q, \sigma, q^+ \rangle : \exists \text{ transition from } q \text{ to } q^+ \text{ by } \sigma \}$$

It is not always needed to include the event in the transition relation. Later on, when the partial transition relations have been constructed and are represented by BDDs, the event will no longer be strictly necessary. Leaving σ out will reduce the number of variables needed for the BDD representation, which is beneficial for computations. In the following σ will be written out as needed.

B. Communication in Composite Automata

For computational purposes it is interesting to investigate how the automata in a composition are related. The *process communication graph* (PCG), adapted from [5], serves this purpose. A PCG is an undirected graph used to represent the interaction between automata under synchronization. In the PCG, each node represents one automaton and there is an edge between two automata iff the corresponding alphabets share one or more events.

The communication complexity in a system of automata can be estimated by making use of the concept of *dependency sets*. The dependency set $D(A^i)$ of an automaton is defined as the set of automata that share events with A^i excluding A^i itself, i.e. $D(A^i)$ is the set of neighbors of A in the PCG. As it turns out, the cardinalities of these sets, the *level-1 dependencies*, yield a surprisingly good estimate of how difficult a particular problem is in terms of reachability analyses [18].

For the discussion in the following sections it will also be useful to define the dependency set of a partial transition relation. To this end, we make use of the above defined automaton dependency set and define $D(\tilde{T}^i)$ as

$$D(\tilde{T}^i) = \{ \tilde{T}^j : A^j \in D(A^i) \}$$

C. Binary Decision Diagrams

Building on the relational formulation of automata given above, we will use *binary decision diagrams* (BDDs) to represent sets of states and transitions in an efficient way [20], [2]. A binary decision diagram is a directed acyclic graph used to represent a boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ in a compact way. The power of the BDD representation lies in the fact that it can be reduced according to certain rules often yielding very efficient data representation.

If the variables in a BDD respect a total order, the BDD is said to be *ordered*. Bryant showed that for a given ordering of the variables, a reduced ordered BDD is a canonical representation of a boolean function [3]. This is a very important property of BDDs and account for much of their popularity. In the following, we will assume that the BDDs are ordered and reduced. For a thorough introduction to BDDs, refer to [3].

Binary decision diagrams are computationally useful for representing subsets embedded in a large set or *universe*, e.g. the state-space of a finite automaton. A standard way of representing subsets is by using a characteristic function. A characteristic function is a function on the elements of the universe which evaluates to one for elements belonging to the particular subset and to zero for all other elements.

By introducing a binary encoding of the elements of the universe and defining the characteristic function on this encoding we obtain a function fit to be represented by a BDD. For a large number of subsets this BDD can then be significantly reduced. In practice this often yields a much more compact representation than an explicit enumeration of the elements.

Representing the subsets of a universe in a slightly different way provides another useful way of looking at BDDs. Consider a universe U consisting of n elements. A particular subset $S \subseteq U$ of this universe can be represented by an n -digit binary string where each digit corresponds to one element of the universe. This is done by indicating members of the subset by ones and all other elements by zeros. For example consider the universe $U = \{A,B,C,D,E\}$. Here, the subset $S = \{B,E\}$ would be represented by the string 01001.

The point here is that BDDs can be seen as a means of compressing binary strings. Whereas there are other more straightforward ways of compressing binary strings, the advantage here is that logical operations can be performed easily and efficiently directly on the BDD-representations.

III. REACHABILITY CALCULATIONS WITH BINARY DECISION DIAGRAMS

A reachability search is typically performed as an iterative fixed point calculation; The reachable set Q_k is repeatedly expanded by adding all states reachable in one step from Q_k . This is continued until no more new states are found - the global fixed point is reached.

Written out in set-notation, the iteration step is

$$Q_{k+1} = Q_k \cup \{q : \exists q' \in Q_k : \langle q', q \rangle \in T\} \quad (1)$$

By utilizing the fact that the set-notation carries over nicely to the language of logic we obtain an expression suited for use with BDDs.

The first step is to replace Q_k with the characteristic function $\chi_k(q)$ defined as

$$\chi_k(q) = \begin{cases} 1 & q \in Q_k \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

The transition relation T is a subset of $Q \times Q$ and can be replaced by a characteristic function $\tau(q', q)$ in the same way.

Using these definitions, (1) now becomes

$$\chi_{k+1}(q) = \chi_k(q) \vee [\exists q' : \chi_k(q') \wedge \tau(q', q)] \quad (3)$$

The characteristic functions χ and τ can both be directly represented by BDDs. Expression (3), is a purely logical expression and therefore constitutes a well-defined operation on BDDs. This means that, in principle, all the pieces are in place for a BDD-based reachability analysis. However, to obtain efficient BDD-based algorithms, special care has to be taken to avoid producing overly large BDDs. This issue will be discussed in detail in the following sections.

A. Partitioning of the Transition Relation

Partitioning of the transition relation as introduced by Burch et al. [21], [22], [10] has become something of a standard tool for alleviating problems with large BDDs [11]. Burch et al. introduced splitting of the transition relation into a set of *partial transition relations*, joined by either disjunction or conjunction. In [16] we have previously introduced an adaptation of these partitioning techniques for use with automata under full synchronous composition. We now give a reformulation of these ideas that clarifies the connection between automata, partial relations and BDDs.

A complex discrete event system can often in a natural and efficient way be represented modularly. In our case, this means dealing with automata consisting of a number of synchronized smaller automata. The transition relation T can then in a natural way be decomposed into partial transition relations by directly considering the modular structure of the system.

Just as the monolithic transition relation can be seen as a set of elements of type $\langle q, \sigma, q^+ \rangle$, the partial transition relations will be sets of the same type. The monolithic set T can then be written as a union or intersection of these partial sets, depending on how they are defined yielding disjunctive or conjunctive partitioning respectively. In this document we will leave out the definition of the conjunctive partitioning and refer instead to [16] for details.

The automaton-based disjunctive partial transition relation \tilde{T}^i of an automaton A^i is defined in the context of a complete system $A = A^1 \parallel \dots \parallel A^N$ and depends on the monolithic transition relation T . The disjunctive transition relation is the set of state-event-state triplets such that the event is in the alphabet of A^i and such that all partial states q^i are updated correctly given this event. In other words, the disjunctive partial relation \tilde{T}^i picks out precisely those transitions from T that occur by events in Σ^i ;

$$\tilde{T}^i = \{ \langle q^1 \dots q^N, \sigma, q^{1+} \dots q^{N+} \rangle : \sigma \in \Sigma^i \wedge \langle q^1 \dots q^N, \sigma, q^{1+} \dots q^{N+} \rangle \in T \}. \quad (4)$$

Here, we assume that in general $\Sigma^i \neq \Sigma^j$. In the case $\Sigma^i = \Sigma^j$ for all i, j this definition simply yields the monolithic T .

From this definition we clearly have $T = \bigcup \tilde{T}^i$. By inspecting (4), we see that we use T to generate the \tilde{T}^i 's. The problem is that we might not have access to the global transition relation T and so we would like to construct the partial relations directly from the transition relations of the sub-automata (T^i). Claim 1 allows us to do precisely that. By considering the augmented dependency set $D(A^i) = D(A^i) \cup \{A^i\}$ it is possible to generate \tilde{T}^i from T^i .

Claim 1: The set \tilde{T}^i defined in (4) is equal to the set of all transitions $\langle q^1 \dots q^N, \sigma, q^{1+} \dots q^{N+} \rangle \in Q \times \Sigma \times Q$ obeying the following three conditions

- 1) $\sigma \in \Sigma^i$
- 2) $\forall A^j \in D(A^i) \cup \{A^i\} : \left[\sigma \in \Sigma^j \wedge \langle q^j, \sigma, q^{j+} \rangle \in T^j \right] \vee \left[\sigma \notin \Sigma^j \wedge (q^{j+} = q^j) \right]$
- 3) $\forall A^j \notin D(A^i) \cup \{A^i\} : q^{j+} = q^j$

Proof: We show that \tilde{T}^i consists of precisely those transitions in T occurring by events in Σ^i . The first condition is trivially required for this.

Note that automata not in $D(A^i) \cup \{A^i\}$ are not affected by $\sigma \in \Sigma^i$. This is expressed by the third condition. We now consider automata in $D(A^i) \cup \{A^i\}$. For automata $A^j \in D(A^i) \cup \{A^i\}$ we have two possibilities:

1) *Either* σ is in the alphabet of A^j and then q^j has to be updated

$$\sigma \in \Sigma^j \wedge \langle q^j, \sigma, q^{j+} \rangle \in T^j,$$

2) *or* σ is not in the alphabet of A^j and then q^j must remain unchanged

$$\sigma \notin \Sigma^j \wedge (q^{j+} = q^j).$$

Putting these conditions together yields the claim statement. ■

B. The Large Intermediate BDD Problem

A partitioned representation of the transition relation is alone not enough to yield efficient BDD-based reachability algorithms. This becomes evident when one studies the BDDs representing the intermediate sets of reachable states during reachability analysis.

Given the partitioning of T and the BDD representation of Q and the \tilde{T}^i s, it is natural to try a standard breadth first reachability search. This is however bound to deliver disappointing results. Here, it is important to understand that in order to reach any substantial reduction of the BDDs, structure in the represented subsets is vital.

The reason that BDD reduction is so important relates to the time-complexity of certain BDD operations. Under normal conditions, binary BDD operations have time complexity $\mathcal{O}(|x| \times |y|)$, where $|x|$ and $|y|$ are the number of nodes in the BDDs x and y [3]. Therefore, in general, smaller BDDs yield faster, more efficient algorithms.

Consider a system consisting of a number of loosely coupled, synchronized automata and assume that we wish to perform a reachability analysis on this system. By traversing the system in a breadth first way we add new states in a near-random fashion, exploring all sub-automata more or less at the same time. During the intermediate phases of the search the set of reachable states will contain newly discovered states from most of the sub-automata, but probably only few of the automata will be fully exhausted.

This leads to very little structure in the intermediate sets of reachable states. As mentioned above, poor structure yields poor reducibility or compression of the underlying BDDs. To clarify what happens, we will now study a simple experiment in some detail.

1) *Experiment of the Randomly Growing Set:* Consider a set/universe consisting of N elements. We construct a sequence of growing subsets of this universe by selecting new elements from the universe at random. As mentioned in section II-C we can represent these subsets by BDDs. In the beginning, when only few elements are selected as well

as towards the end when most of the elements are selected, these BDD representations will be very compact. However, during the middle phase of this process, approximately half of the elements of the universe will have been selected and without any particular structure. As can be seen in Figure 1, the corresponding BDDs are very large.

2) *Information Theoretic Considerations:* The phenomenon outlined in the above experiment can best be understood by studying the binary string representation of the subsets as mentioned in section II-C. When exactly half of the elements have been selected, the binary string that represents this subset will be a completely random binary string consisting of equally many zeros and ones. From information theory it is well known that this condition renders compression theoretically impossible [23].

Moreover, information theory provides us with a tool for estimating the compressibility of a binary string. The amount of compression that can be performed is bounded by the entropy of the string. The entropy is defined as a function on a random variable and we therefore view the N -digit binary string as a set of N boolean random variables $X = \{X_1, \dots, X_N\}$. Assuming that the string contains p ones we get N random variables with $\Pr[X_i = 1] = p/N$. We calculate the entropy of X as

$$\begin{aligned} H(X) &= \sum_{i=1}^N H(X^i) \\ &= \sum_{i=1}^N \sum_{x^i \in X^i} \Pr(x^i) \log_2 \left(\frac{1}{\Pr(x^i)} \right) \\ &= N \left(\frac{p}{N} \log_2 \frac{N}{p} + \left(1 - \frac{p}{N}\right) \log_2 \frac{1}{1 - \frac{p}{N}} \right), \end{aligned}$$

where we make use of the additivity of the entropy to split $H(X)$ over the N variables [23].

The BDD-sizes for the subsets are plotted together with this entropic information content in Figure 1 for $N = 1024$. It is difficult to estimate the information content of a BDD, but it seems reasonable to count the number of nodes as a measure of this. Even so it is hard to give an exact number. We have therefore taken the liberty to plot the BDD-size graph on a different scale to fit the entropy curve. This is mainly to show the close correspondence in shape between BDD growth and entropy growth.

IV. WORKSET BASED STRATEGIES

Following the previous section, we conclude that in order to design successful BDD-based reachability algorithms for large scale systems it is vital to traverse the search space in a structured way. What we need to do is to modify the order of traversal of straightforward breadth first search to fight the induced pseudo-randomness of this approach. Based on the PCG of a system of automata and the disjunctive partitioning of the transition relation, we have in previous papers proposed a set of algorithms for this purpose [18], [13], [17]. These algorithms are specifically designed to iteratively expand the

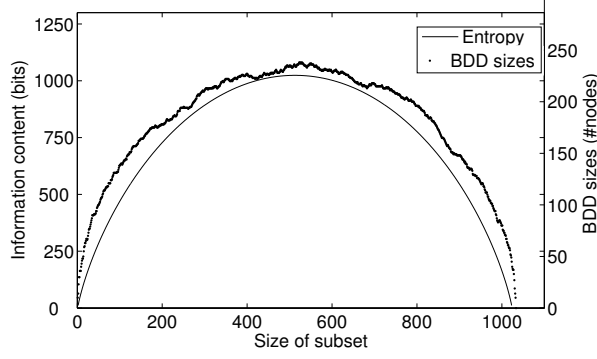


Fig. 1. Information content of a monotonically growing subset of a set of $N = 1024$ elements. Elements are selected at random and added to the subset one by one. This graph shows BDD sizes together with the calculated entropy on different scales.

set of reachable states in a way that is advantageous for BDD-compressibility.

In this section we give an in-depth discussion and explanation of these algorithms as well as a proof of correctness. For clarity and brevity we focus on one algorithm, the workset algorithm (Algorithm 1) introduced in [17] and conclude by briefly commenting on variations on this algorithm.

The workset algorithm operates by maintaining a set W_k of active partial transition relations - the workset. These transition relations are selected one at a time and a saturating "local" reachability search is performed by trying to reach node reduction in the region of the BDD representing states in the corresponding sub-automaton.

Algorithm 1: Workset Forward Reachability

input: q_i
let $W_0 :=$ the set of disjunctive partial transition relations,
 $Q_0 := q_i, k := 0$
repeat
 Pick and remove a transition $\tilde{T}^i \in W_k$
 $k := k + 1$
 $Q_k = Q_{k-1} \cup \text{ForwardReachability}(Q_{k-1}, \tilde{T}^i)$
 if $Q_k \neq Q_{k-1}$ **then** $W_k := W_{k-1} \cup D(\tilde{T}^i)$
until $W = \emptyset$
return Q_k

Note here the nested use of a forward reachability search marked by *ForwardReachability*. This local traversal is a standard breadth first search, which performs a full reachability analysis on the states locally reachable by \tilde{T}^i .

A. Exploring the State Space in the Right Order

It is not immediately clear that we gain anything by exploring the state space in the particular way of the workset algorithm. Here, the key is the more structured strategy employed by the workset algorithm. As discussed in section III-B, straightforward breadth first search (BFS) reachability yields poor reducibility of the underlying BDDs.

We will now give a simple example to illustrate how the mechanism of the workset algorithm works to prevent intermediate BDD blowup. Consider the automata representations of the two independent processes A and B illustrated in figure 2.

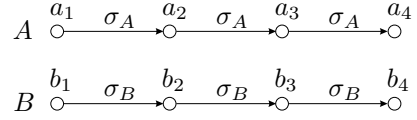


Fig. 2. Two independent processes A and B with four states each. A and B are represented by finite automata.

A reachability analysis on $A||B$ will demonstrate how the workset algorithm explores the search space. This will be compared with standard BFS reachability. For the two automata A and B , we have the following partial transition relations

$$\begin{aligned} \tilde{T}^A &= \{ \langle a_1 * _1, a_2 * _1 \rangle, \langle a_2 * _2, a_3 * _2 \rangle, \langle a_3 * _3, a_4 * _3 \rangle \} \\ \tilde{T}^B &= \{ \langle * _4 b_1, * _4 b_2 \rangle, \langle * _5 b_2, * _5 b_3 \rangle, \langle * _6 b_3, * _6 b_4 \rangle \} \end{aligned}$$

Note that each partial transition relation here has 12 elements. To save space we use a placeholder $*_i$ to denote an arbitrary state not affected by this transition relation. We have that e.g. $*_1$ can be any of the states $\{b_1, b_2, b_3, b_4\}$.

The workset algorithm performs exhaustive BFS reachability based on these partial relations one at a time. Note that the workset algorithm is based on two nested loops. In each outer iteration a partial relation \tilde{T}^i is selected and removed. Thereafter a BFS reachability analysis based on \tilde{T}^i is performed in an inner loop by the function *ForwardReachability*.

In this case, since A and B are completely independent, the workset algorithm will traverse the automata exactly one time each. Starting with \tilde{T}^A , the states $\{a_1 b_1, a_2 b_1, a_3 b_1, a_4 b_1\}$ are discovered. Here, no more new states are found by \tilde{T}^A and we switch to \tilde{T}^B yielding the rest of the reachable states. Instead of writing these out explicitly we refer to figure 3, where this process is illustrated (right) and compared to the corresponding BFS exploration (left).

To get a clearer picture of what happens we study the progress of the reachability search by using a binary string representation of the reachable states. Each position in the

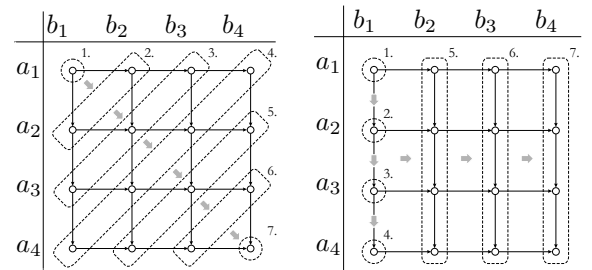


Fig. 3. Reachability search on $A||B$ by two different strategies (A and B are defined in Fig. 2). The diagram to the right shows the workset algorithm exploring the state space by first considering \tilde{T}^A and then \tilde{T}^B . The diagram to the left shows a standard breadth first reachability search.

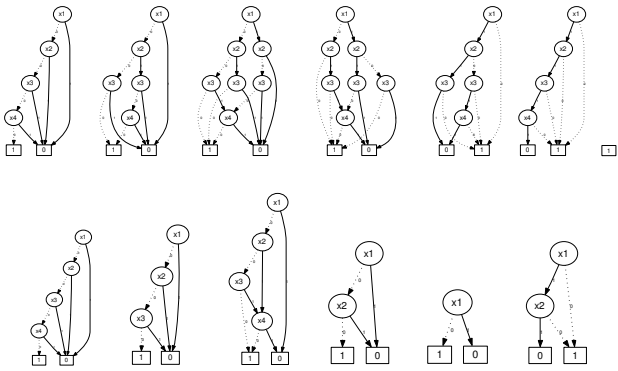


Fig. 4. BDDs representing the intermediate sets of reachable states for each step of the workset algorithm (bottom) and standard breadth first search (top). As can be seen, the most complex BDD obtained by the workset algorithm during the reachability search has four nodes. By contrast, during straightforward BFS reachability, BDDs with as many as seven nodes are produced twice.

string indicates whether or not the respective state is present in the particular subset. The following table gives the progress for the workset algorithm compared to breadth first search.

step	reachable states: BFS	reachable states: workset
1	1000000000000000	1000000000000000
2	1100100000000000	1100000000000000
3	1110110010000000	1110000000000000
4	1111111011001000	1111000000000000
5	1111111111101100	1111111100000000
6	1111111111111110	1111111111100000
7	1111111111111111	1111111111111111

By inspecting this output we see that the workset algorithm adds states in a structured way from left to right. This is to be compared with the evolution of the BFS exploration, which contains more of an element of randomness.

Intuitively it should be clear that the workset algorithm produces a more agreeable output. The main point of this section is however provided by building the actual BDDs corresponding to the intermediate subsets produced during the reachability search.

The BDDs representing the intermediate sets of reachable states for each step of the example in Figure 3 are given in Figure 4. As can be seen, the most complex BDD obtained by the workset algorithm during the reachability search has four nodes. By contrast, during straight forward BFS reachability, BDDs with as many as seven nodes are produced twice.

B. Variable Ordering

Naturally, the ideas presented here relate very much to variable ordering. In fact BFS could probably be made to work just as well as the workset algorithm by choosing a different variable ordering. The problem here is that this variable ordering is not known a priori. In fact, as mentioned before, the problem of finding the optimal variable ordering is NP [8]. On the other hand, the workset algorithm works very well with the natural variable ordering induced by encoding the states of each automaton in turn. What is essential is

that the variable ordering agrees with the order in which the state space is explored so that a compact BDD representation can be given in each step. Basically this is accomplished by minimizing a cost function based on the PCG of the system. Refer to [16] for details of the encoding scheme.

C. Dealing With Interactions Between Automata

The ideas presented in section IV-A build on the fact that the workset algorithm operates on one automaton at a time and iteratively tries to perform locally exhaustive reachability searches. However, in most cases, due to interacting automata it is inevitable that during the reachability analysis the transitions in some automaton will be exhausted without all states in that automaton having been found. The algorithm will then be forced to proceed with some other automaton or more correctly some other partial transition relation and then later come back to the incompletely explored automaton.

The following example illustrates this situation and how the workset algorithm behaves in this case. We consider, again, two processes A and B , this time interacting via the event σ_C . The automata are given in Figure 5.

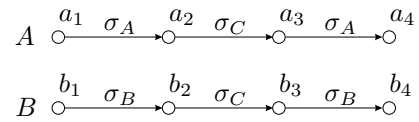


Fig. 5. Two interacting processes A and B with four states each.

Initially the algorithm starts out from the state a_1b_1 . Assuming that we begin by choosing the transition relation \tilde{T}^A the additional state a_2b_1 is found, but no further states. We are forced to switch to \tilde{T}^B and then add $\{a_1b_2, a_2b_2\}$. Now, since new states were found, the relations contained in $D(\tilde{T}^B)$ are added to the workset W . In this case this is precisely \tilde{T}^A , which is again selected to continue the reachability search. We refrain from accounting for the remaining steps and instead refer to Figure 6 where the complete process is illustrated.

D. The Workset Algorithm is Correct

In this section we will establish the correctness of the workset algorithm. The workset algorithm is based on a rather involved scheme of enabling and disabling the partial transition relations. It is not immediately clear that this mechanism

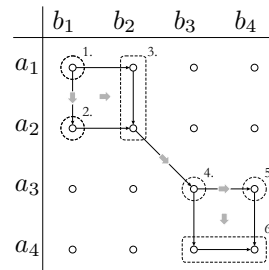


Fig. 6. Reachability search on the system $A||B$ of interacting automata (A and B are defined in Fig. 5). On a system of interacting automata the workset algorithm potentially has to visit the same transition relation several times.

works as intended. The important point is to be able to guarantee that what is output by the workset algorithm really is the set of *all* reachable states. How the workset algorithm works to ensure this was suggested by the example in Section IV-C and will here be formalized.

The analysis is rather straightforward to do, but the argument will run more smoothly if we make use of the following lemma.

Lemma 2: At iteration k of the workset algorithm, the set W_k of active transition relations contains all transitions leading to states reachable in one step from Q_k .

Proof: We need to show that at any iteration k , the image of Q_k under T equals the image of Q_k under W_k . Note that at the outset, W contains all partial transition relations \tilde{T}^i , thus giving a complete representation of T . Now an inductive argument will cover the general case.

Assume that at iteration k , the claim of Lemma 2 holds. At iteration $k + 1$, a transition relation \tilde{T}^i is now selected and an exhaustive reachability search is performed with this relation. Since the search is exhaustive, no further states can be reached by \tilde{T}^i , which can thus be safely removed from W_k . During the iteration $k + 1$ *either* no new states are found and then the image of Q_k is unchanged *or* new states are found and then the set $D(\tilde{T}^i)$ of all transition relations that might contain transitions from the new states are added to W_k . Thus the union of the relations in W_k will be a subset of T , however still containing all transitions leading from Q_k . ■

Theorem 3: The workset algorithm terminates in finite time and the output is the complete set of reachable states.

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

Assume now that the algorithm has terminated. By Lemma 2 we know that W_k holds transitions to all states reachable from Q_k . However, since the algorithm has terminated, W_k is empty and thus no more states are reachable from Q_k , i.e. Q_k is the set of all reachable states. ■

E. Variations of The Workset Algorithm

One weakness of the workset algorithm is that in some cases even the partitioned transition relations can become very large. The sizes of these partitions are decided by the dependency set $D(A)$ of each automaton. Even if the level of interaction between automata is in general small, this set might be large for a small subset of the automata, causing the corresponding transition relations to become very large. For example in the Profisafe model treated in [18], the largest dependency set contains as many as 48 automata, yielding transition relations of unmanageable size. This problem can be overcome by using a different type of partitioning. Instead of partitioning over automata, we have experimented with partitioning over events leading to an *event-based* workset algorithm. In other

words, this event-based algorithm is essentially equivalent to the workset algorithm, apart from building on a different partitioning technique.

While the event-based transition relations are often much smaller than their automaton-based counterparts, more iterations are generally needed to reach fixed point [18]. To some extent, this can be compensated for by combining multiple event-based partitions into larger ones. This is often referred to as clustering, see for instance [10].

The workset algorithm performs an exhaustive reachability search in each step before selecting a new transition relation, referred to as local saturation. In some cases it is, however, not the best strategy to go for complete saturation in each step. For this, we have introduced the *step-stone* algorithm [13] as a variation of the workset algorithm which works without complete saturation. Instead a new transition relation is selected for each expansion of the reachable set. The step-stone algorithm uses an automaton selection heuristic in the same way as the workset algorithm does. It is interesting to note that if the step-stone algorithm is combined with a certain type of selection heuristic, i.e. a heuristic with memory which will only switch \tilde{T}^i upon exhaustion, it becomes precisely the workset algorithm.

The proofs of correctness of the event-based workset algorithm and the step-stone algorithm are essentially analogous to that of the workset algorithm. For the event-based workset algorithm, the crucial step is to ensure correctness of the event-based partitioning, i.e. to show that this partitioning behaves essentially in the same way as automaton-based partitioning. Except for the difference in partitioning, the event-based workset algorithm is equivalent to the automaton-based counterpart.

For the step-stone algorithm, it principally suffices to note that the rule for removing and adding automata to the workset W is the same as for the workset algorithm. This means that the argument in the proof of Lemma 2 is easily adapted to work for the step-stone algorithm.

F. Related Work

To the authors knowledge, the concept of locally exhaustive reachability search was introduced by Burch et al. in [10], using the term modified breadth first strategy. Burch et al. developed this approach in the context of digital circuits to be used for model checking purposes.

Burch et al. utilize the principle of local exhaustion of transition relations to prevent BDD blowup. The set of reachable states is calculated by repeatedly calculating local fixed points until a global fixed point is found.

In [11] Valmari and Geldenhuys generalize the ideas introduced in [10] and formulate them in the general setting of states and transition relations. Based on this they introduce a number of similar schemes designed to avoid BDD growth. For transition relation selection they use a simple scheme of cycling through the partial transition relations one by one until no more new states are found.

In our work, we develop this last part of the scheme further and make use of the dependency set $D(\tilde{T}^i)$ to keep track

of exactly which \tilde{T}^i might need to be considered again. Moreover, we note the importance of the order in which the transition relations are selected. We suggest a scheme based on heuristics such as reinforcement learning for dealing with the problem of transition relation selection.

V. SUMMARY AND CONCLUSION

In this paper, we have given a theoretical analysis of a set of algorithms designed to perform efficient reachability analysis on composite discrete event systems. By carefully chosen examples, we have tried to provide a better understanding of the problem with large BDDs and how the workset- and related algorithms acts to avoid this problem. Especially, we have noted the importance of exploring the state space in a structured way or rather in a way that yields structured intermediate sets of reachable states.

Furthermore, a reformulation of disjunctive partitioning in the context of full synchronous composition has been presented. Finally, we have proven the correctness of the workset algorithm and suggested how these arguments carries over to related algorithms such as the step-stone algorithm and the event-based workset algorithm.

REFERENCES

- [1] W. M. Wonham, *Notes on Control of Discrete-Event Systems*. University of Toronto, 2002.
- [2] S. B. Akers, "Binary decision diagrams," *IEEE Transactions on Computers*, vol. 27, no. 6, pp. 509–516, 1978.
- [3] R. E. Bryant, "Symbolic Boolean manipulation with ordered binary-decision diagrams," *ACM Computing Surveys*, vol. 24, no. 3, pp. 293–318, 1992. [Online]. Available: citeseer.nj.nec.com/article/bryant92symbolic.html
- [4] A. J. Hu, "Techniques for efficient formal verification using binary decision diagrams," Ph.D. dissertation, Carnegie Mellon University, 1995.
- [5] A. Aziz and S. Taziran, "BDD variable ordering for interacting finite state machines," in *Proceedings of 31th Design Automation Conference*, 1994, pp. 283–288.
- [6] S. J. Friendman and K. J. Supowit, "Finding the optimal variable ordering for binary decision diagrams," *IEEE Trans. Comput.*, vol. C-43, no. 11, pp. 1262–1269, 1994.
- [7] N. Ishiura, H. Sawada, and S. Yajima, "Minimization of binary decision diagrams based on exchanges of variable," in *Proceedings of IEEE Int. Conf. Comput.-Aided Design*, 1991, pp. 472–475.
- [8] B. Bollig and I. Wegener, "Improving the variable ordering of OBDDs is NP-complete," *IEEE Transactions on Computers*, vol. 45, no. 9, 1996.
- [9] Z. H. Zhang and W. M. Wonham, "STCT: An efficient algorithm for supervisory control design," *Symposium on Supervisory Control of Discrete Event Systems*, 2001.
- [10] J. R. Burch, E. M. Clarke, D. E. Long, K. L. MacMillan, and D. L. Dill, "Symbolic model checking for sequential circuit verification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 4, pp. 401–424, 1994. [Online]. Available: citeseer.nj.nec.com/burch93symbolic.html
- [11] J. Geldenhuys and A. Valmari, "Techniques for smaller intermediary BDDs," in *The 12th International Conference on Concurrency Theory, Lecture Notes in Computer Science 2154*. Springer-Verlag, 2001, pp. 233–247.
- [12] A. Vahidi, M. Fabian, and B. Lennartson, "Early termination by local string in incremental language containment tests," in *Proceedings of the International Workshop on Discrete Event Systems (WODES'04)*, Sep 2004.
- [13] A. Vahidi, B. Lennartson, and M. Fabian, "Efficient analysis of large discrete-event systems with binary decision diagrams," in *Proceedings of the 44th IEEE Conference on Decision and Control*, Dec 2005.
- [14] K. Ravi and F. Somenzi, "Hints to accelerate symbolic traversal," in *Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, 1999, pp. 250–264.
- [15] G. Cabodi, P. Camurati, L. Lavagno, and S. Quer, "Disjunctive partitioning and partial iterative squaring: an effective approach for symbolic traversal of large circuits," in *DAC '97: Proceedings of the 34th annual conference on Design automation*. New York, NY, USA: ACM Press, 1997, pp. 728–733.
- [16] A. Vahidi, "Efficient analysis of discrete systems. supervisor synthesis with binary decision diagrams," Department of Signals and Systems, Chalmers University of Technology, PhD thesis 487, 2004.
- [17] A. Vahidi, B. Lennartson, and M. Fabian, "Efficient supervisory synthesis of large systems," in *Proceedings of the International Workshop on Discrete Event Systems (WODES'04)*, Sept. 2004.
- [18] —, "Efficient supervisory synthesis of large systems," *accepted for publication in Control Engineering Practice*, 2006.
- [19] C. A. R. Hoare, "Communicating sequential processes," in *On the construction of programs – an advanced course*, R. M. McKeag and A. M. Macnaghten, Eds. Cambridge University Press, 1980, pp. 229–254.
- [20] C. Lee, "Representation of switching circuits by binary decision programs," *Bell System Technology Journal*, vol. 38, pp. 985–999, 1959.
- [21] J. R. Burch, E. M. Clarke, and D. E. Long, "Representing circuits more efficiently in symbolic model checking," in *Proceedings of 28th ACM/IEEE Design Automation Conf.*, 1991.
- [22] —, "Symbolic model checking with partitioned transition relations," in *Proceedings of 1991 Intl. Conf. on VLSI*, A. Halaas and P. B. Denyer, Eds., 1991.
- [23] T. M. Cover and J. A. Thomas, "Elements of information theory," 1991.