Proceedings of the
44th IEEE Conference on Decision and Control, and
the European Control Conference 2005
Seville, Spain, December 12-15, 2005

Efficient Analysis of Large Discrete-Event Systems with Binary Decision Diagrams

Arash Vahidi, Bengt Lennartson, Martin Fabian
Department of Signals and Systems, Chalmers University of Technology
SE-412 96 Göteborg, Sweden
{ vahidi, bl, fabian }@s2.chalmers.se

***Abstract*— Efficient analysis and controller synthesis in the context of Discrete-Event Systems (DES) is discussed in this paper. We consider efficient reachability search for solving common problems in the Supervisory Control Theory (SCT). The search is based on symbolic computations including crucial partitioning techniques. Finally, the efficiency of the presented algorithms is demonstrated on a set of hand-made and real-world industrial systems.**

***Index Terms*— Discrete-event systems, supervisory control, reachability search, symbolic computation**

I. Introduction

In the Supervisory Control Theory (SCT) of Ramadge and Wonham [20] controller synthesis is known to often suffer from the state explosion problem. This is preventing SCT from having a major breakthrough in industry.

In [19] a verification and synthesis algorithm is presented based on symbolic state and transition relations using Binary Decision Diagrams (BDDs). This is an efficient alternative compared to existing algorithms based on BDDs such as Hoffmann and Wong-Toi [11].

The bottleneck to improve both memory and run-time performance for this kind of analysis and synthesis is to achieve efficient reachability searches. An important part of the algorithm in [19] is a new intelligent search strategy. This strategy based on crucial partitioning techniques is presented and analyzed in more detail in this paper, including proofs and interesting extensions. It is shown to be able to analyze and synthesize controllers for discrete event systems with extremely large state spaces.

II. Preliminaries

A DES is often described as one or more mathematical objects. The common method to represent these objects is to use textual description such as regular expressions or graphical representations such as Petri nets or automata. In this work we will only consider the latter.

A *deterministic finite automaton* is a fivetuple $A = \langle Q, \Sigma, \delta, q_i, Q_m \rangle$ where Q is the finite set of states and Σ is the set of events (the alphabet). Σ is divided into two disjoint subsets, the controllable events Σ_c and the uncontrollable events Σ_u. State transitions are described by the transfer function $\delta : Q \times \Sigma \to Q$, additionally, δ_u denotes the subset of δ associated with uncontrollable events. $q_i \in Q$ is the initial state and $Q_m \subseteq Q$ is the marked-states subset. Furthermore, $\delta(q, \sigma)!$ denotes that δ is defined for the state q and σ, and $\dot{q}$ will be used to denote the next state. When $\delta(q, \sigma)!$ this implies that $\delta(q, \sigma) = \dot{q}$.

A sequence of events in an alphabet form a *trace*, also known as a *string*. Let Σ^* denote the set of all finite strings of elements of Σ (including the empty string ϵ). A *language* L is a subset of Σ^*, furthermore the *closure* of the language L, denoted $\overline{L}$, is the set of all prefixes in L.

If we extend the definition of δ to strings, i.e. $\delta : Q \times \Sigma^* \to Q$, then the language of an automata A can be defined as $L(A) = \{s \in \Sigma^* \mid \delta(q_i, s)!\}$. Similarly, the *marked language* of the same automata is defined as $L_m(A) = \{s \in \Sigma^* \mid \delta(q_i, s) \in Q_m\}$.

In this work, we will also use the *transition relation* as a simplification of the transfer function, for more efficient computations. A transition relation $T : Q \to 2^Q$ is defined as $T = \{(q_1, q_2) \mid \exists \sigma \in \Sigma,\ \delta(q_1, \sigma) = q_2\}$. Furthermore, it is sometimes useful to include only the uncontrollable transitions. For this purpose, the *uncontrollable transition relation* is created. $T_u = \{(q_1, q_2) \mid \exists \sigma \in \Sigma_u,\ \delta(q_1, \sigma) = q_2\}$.

A. Composition

Composition between two or more automata is defined by the *full synchronous operator* $||$, originating from the early work of Hoare [10]. An important property of the full synchronous operator is that $L(A||B) = L(A) \cap L(B)$. Furthermore, the $||$-operator is associative, allowing composition of more than two automata. As a convention, we will use superscript indices to denote members of a composition, for example $A = A^1||...||A^n$.

More specifically, the composition of two automata $A^1 = \langle Q^1, \Sigma^1, \delta^1, q_i^1, Q_m^1 \rangle$ and $A^2 = \langle Q^2, \Sigma^2, \delta^2, q_i^2, Q_m^2 \rangle$ results in the composite system $A^1||A^2 = \langle Q, \Sigma^1 \cup \Sigma^2, \delta, q_i, Q_m^1 \times Q_m^2 \rangle$ where $Q \subseteq Q^1 \times Q^2$ and $q_i = \langle q_i^1, q_i^2 \rangle$. The composite transfer function δ is defined as follows.

$$\delta((q^1, q^2), \sigma) = \begin{cases} \delta^1(q^1, \sigma) \times \delta^2(q^2, \sigma) & \text{if } \delta^1(q^1, \sigma)! \wedge \delta^2(q^2, \sigma)! \\ \delta^1(q^1, \sigma) \times \{q^2\} & \text{if } \delta^1(q^1, \sigma)! \wedge \sigma \in \Sigma^1 - \Sigma^2 \\ \{q^1\} \times \delta^2(q^2, \sigma) & \text{if } \delta^2(q^2, \sigma)! \wedge \sigma \in \Sigma^2 - \Sigma^1 \\ \text{undefined} & \text{otherwise} \end{cases} \tag{1}$$

When more than two automata are involved in a composition, we will use the *conjunctive* representation instead of (1). To do this, a *conjunctive transfer functions* $\hat{\delta^i}$ must first be created.

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

This is simple *event-compensation* (by introducing self-loops) and is used to transform a set of automata with possibly different alphabets into another set of automata with equal alphabets. After that, composition is a simple matter of computing the conjunction of the transfer functions:

$$\delta = \{ \langle (q^1, q^2, \ldots, q^n), \sigma, (\dot{q}^1, \dot{q}^2, \ldots, \dot{q}^n) \rangle \mid \bigwedge_{1 \le i \le n} \hat{\delta^i}(q^i,\sigma) = \dot{q}^i \}$$

B. Communication in Composite Automata

It is sometimes interesting to investigate how the automata in a composition are related. To do this, we make use of *Process Communication Graphs*, adapted from [3]. A PCG is a weighted undirected graph $PCG = \langle V, E, w \rangle$, where V is the set of vertices with a one to one mapping to the automata. $E \subseteq V \times V$ is the set of edges and $w : E \to \mathbb{R}^+$ is the weight associated with each edge.

The communication complexity is measured with help of *dependency sets*. Given a set of automata $\{A^1, \cdots, A^n\}$, let the dependency set of an automaton A^i, denoted $D(A^i)$ be A^i itself plus the set of automata that are directly connected to A^i in its corresponding PCG. The cardinality of such set equals $d(A^i) + 1$, that is, the degree of the PCG node A^i plus one. We also define the *exclusive dependency set* as $D^+(A^i) = D(A^i) - A^i$.

The *level-2 dependency set* of an automaton A^i, denoted $D^2(A^i)$ is defined as $D^2(A^i) = \{A^j | \exists A^k \in D(A^i).\ A^j \in D(A^k)\}$

C. Reachability

A common operation when working with discrete-event systems is *reachability analysis*. An example of reachability analysis is to find all states in an automaton that can be reached from the initial state by a series of events. Finding these states can be done using the reachability algorithm that follows.

Algorithm 1: ForwardReachable

```
input: $Q_0$, $T$ (if not the default $T$ is used)
let $k := 0$
repeat
    $k := k + 1$
    $Q_k := Q_{k-1} \cup \{\dot{q} |\ \exists q \in Q_{k-1}.\ (q, \dot{q}) \in T\}$
until $Q_k = Q_{k-1}$
return $Q_k$
```

When the inverse of T is used, set of all states *leading to* some state in the automata is found instead. This is equivalent to Algorithm 2.

Algorithm 2: BackwardReachable

```
input: $Q_0$, $T$ (if not the default $T$ is used)
let $k := 0$
repeat
    $k := k + 1$
    $Q_k := Q_{k-1} \cup \{q |\ \exists \dot{q} \in Q_{k-1}.\ (q, \dot{q}) \in T\}$
until $Q_k = Q_{k-1}$
return $Q_k$
```

Notice that the presented algorithms are based on *breadth-first* search. This is in contrast to classic search algorithms, which are often based on depth-first traversal.

III. Synthesis

Assume that in a system, automata are divided into logical groups such as *plants*, *specifications* and *controllers*. Let P, Sp and C denote the total composition of these three groups, respectively. Note that $L(P) \cap L(C)$, which denotes the behavior of the plant under the control of C is, interestingly, also the language of $P||C$.

It would be interesting to see if the plant meets the specification, or if the plant under the control of C does that. More formally, if $L(P) \subseteq L(Sp)$ or $L(P||C) \subseteq L(Sp)$. Due to the existence of the uncontrollable events, one must also verify that the plant can actually be *forced* to fulfil the specification. This is known as the *controllability* test $L(Sp)\Sigma_u \cap L(P) \subseteq L(Sp)$.

Another interesting property is whether the system is *live*. More specifically, if the system $P||C$ is *nonblocking*. In a nonblocking system, marked states can always be reached in zero or more transitions from any reachable states $\overline{L_m(P||C)} = L(P||C)$.

Verification of the controllability and nonblocking properties both require reachability analysis. Let

$$Q_{puc} = \{(q_1, q_2,) \mid \exists \sigma \in \Sigma_u.\ \delta^P(q_1,\sigma)! \wedge \neg\delta^{Sp}(q_2,\sigma)!\} \tag{2}$$

be the set of all possible uncontrollable states. Then in the composition $P||Sp$, the state set $(Q_{puc} \cap ForwardReachable(\{q_i\}))$ contains all proven uncontrollable states. If (and only if) this set is empty, then $L(Sp)\Sigma_u \cap L(P) \subseteq L(Sp)$.

Furthermore, it can trivially be shown that iff in $P||C$

$$ForwardReachable(\{q_i\}) \subseteq BackwardReachable(Q_m)$$

then $\overline{L_m(P||C)} = L(P||C)$.

The ultimate goal in *Supervisory Control Theory* is to build a controlling device, a *supervisor* S, which by limiting the behavior of plants guarantees that the specification plus some additional requirements, such as controllability and nonblocking, are met [20]. Notice that S corresponds to C in the text above, but has been renamed to emphasize that it has been automatically generated by some synthesis procedure.

A. Safe-State Synthesis

It can be shown that a minimally restrictive supervisor that is both controllable and non-blocking, can be constructed as an automaton S such that $Q^S = Q^{P||Sp}$ while $L(S) \subseteq L(P||Sp)$ (some transitions are removed) [14], [20].

In this work, we will use the slightly different approach to controller synthesis where bad states are removed. As long as the system is in one of the remaining states, controllability and nonblocking is guaranteed. The remaining states are therefore called the *safe states*. A *safe-state supervisor* can be created by building a supervisor candidate $S0 = P||Sp$, then removing states from Q^{S0} until it is both controllable and nonblocking. Notice that in such case, $L(S0) \subseteq L(Sp)$, meaning that $S0$ already meets the specification. This also applies to any subsets of $S0$ (including the supervisor S).

Let $Q^S \subseteq Q^{S0}$ denote the set of safe states. In practice, this set is the only part of the supervisor needed to be represented, $S0$ and S are never created. The transfer function δ^S and thus the language $L(S)$ can be constructed *online* by processing $\delta^{P||Sp}$ on the fly. We call this approach an *online control scheme*,

In [19], a set of algorithms are presented that given P and Sp compute S using only a series of backward reachability searches. It is also noticed that the reachability searches are the bottleneck of the synthesis algorithms. Therefore, in the following, we will concentrate on the performance of the reachability search procedures.

IV. Efficient Synthesis by Efficient Reachability Search

While the supervisor control theory is very clear and elegant, there exist a few practical limitations which complicate the use of SCT in real applications The ones that we consider in this work are:

A The sets of states (e.g. Q and Q^S) are usually far larger than the amount of available computation space. We will use Binary Decision Diagrams to partially solve this problem.
B The transfer functions δ is also very large. Again, we will use Binary Decision Diagrams for efficient representation. However, in this case, the transfer function is a such complicated function that even BDDs are not able to represent it efficiently. To deal with this problem, we will suggest use of partitioning techniques.
C The temporary BDDs representing the intermediate states sets in a reachability search (Q_k in Algorithm 1) are often significantly larger than the final BDDs (representing the fixed point $Q_k = Q_{k+1}$). We suggest a set of specialized reachability algorithms to address this.

These complications are addressed in the rest of this work.

A. Symbolic Computation

A *symbolic representation* of an object (for example a set or a relation) is a non-numeric representation that is more suited for computers than humans. *Symbolic computation* is the process of manipulating an object directly in its symbolic form. Throughout the rest of this paper, we will transparently use a *symbolic* representation of automata. We will use *Binary Decision Diagrams* (BDDs) to represent sets of states, transfer functions etc. for better performance. Therefore, when we discuss the size or the complexity of some object, we always refer to its symbolic form.

In short, a BDD is a directed acyclic graph representing a boolean function $f : 2^V \rightarrow \{0, 1\}$, where V is a set of boolean variables. Using BDDs, sets, functions and relations can be represented as binary predicates. These predicates are recalled as *characteristic functions* in the literature.

What makes BDDs interesting is that using simple rules it is possible to remove redundant nodes in their graphs, hence achieving substantial compression of the boolean function they represent. Furthermore, if the reduced BDD is following a total order on V, it is also a *canonical* representation [5].

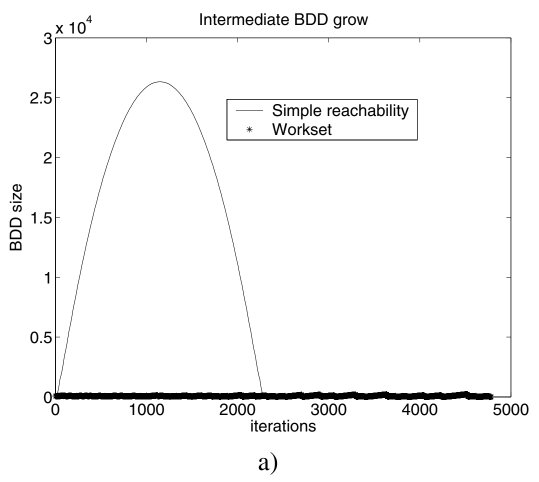
In addition, binary operations on functions can be done directly on their BDDs while retaining the reduction and order, and their canonicity. This is in fact the main advantage of BDDs.

An important subject on BDDs that the reader must know in order to understand the rest of this paper is that a binary BDD operation $x \otimes y$ has under normal conditions the time complexity $O(|x| \times |y|)$, where $|x|$ and $|y|$ are the number of nodes in the BDDs x and y, respectively [5]. Therefore, in general, algorithms and operations that involve smaller BDDs are preferred in symbolic computation. Behind this, no further knowledge about BDDs is required for reading this paper. Readers unfamiliar with BDDs and symbolic tools may still be interested in consulting [5], [13] for some background information.

B. Non-monolithic Representation of Complex Functions

A problem with composition of large sets of automata is that the total transition relation T becomes extremely complex. If an algorithm uses a traditional state enumeration based approach, this is usually not a problem since the set of global states Q is considered to be a larger obstacle and the real bottleneck. However, if a symbolic approach is taken, the set of states is often compressed to such degree that representing the states is not a problem anymore (to some limit, of course). This does however not apply to T, due to its complex structure.

Similar problems have been studied in the field of (symbolic) formal verification, where the corresponding transition relation is often too large to be represented as a single *monolithic* relation. It has been suggested that by partitioning methods, one may split T into a set of less complex relations with a clear connection in between,



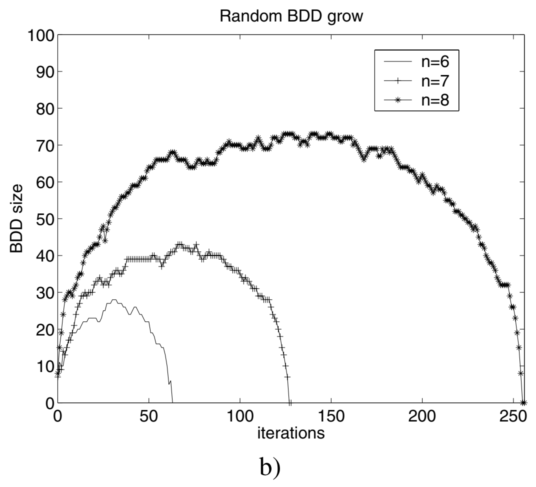


Fig. 1. Intermediate BDD grow in (a) the transfer line experiment, and (b) the random BDD growth experiment.

e.g. $T = T^1 \otimes \cdots \otimes T^n$ [6], [13], [4], [12]. We have already presented the conjunctive transfer function $\hat{\delta}$, where $\delta = \bigwedge_i \hat{\delta}^i$ (similarly for T), which is a form a conjunctive partitioning. Another useful method for partitioning is the *disjunctive* partitioning.

A *disjunctive* transfer function, denoted $\tilde{\delta}^i$, is a transfer function such that

$$\delta(q,\sigma) = \bigcup_{A^i.\ \sigma\in\Sigma^i} \tilde{\delta}^i(q,\sigma)$$

Assuming that $q = \langle q^1, \cdots, q^n \rangle$, let

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

the disjunctive transfer function is

$$\begin{aligned} \tilde{\delta}^i(q,\sigma) \;=\; & \{(\dot{q}^1, \dot{q}^2, \cdots) \mid [\bigwedge_{A^j\in D(A^i)} \zeta^{\hat{i},j}(q^j,\sigma) \leftrightarrow \dot{q}^j] \\ & \wedge \underbrace{[\bigwedge_{A^k\notin D(A^i)} q^k \leftrightarrow \dot{q}^k]}_{\text{"}keep\text{"}} \} \end{aligned}$$

from which the disjunctive transition relation $\tilde{T}^i$ can be computed. In terms of BDD size, the disjunctive transition relations are significantly smaller than the monolithic T. Interestingly, $\tilde{T}^i$ tends to grow towards the size of T as the part marked by the word "keep" in the corresponding $\tilde{\delta}^i$ decreses in size.

A partitioned transition relation can be used for reachability search without the need for creating a monolithic δ or T. For disjunctive transition relations, this is shown in Algorithm 3.

Algorithm 3: DisjunctiveForwardReachable

```
input: $Q_0$, $\Delta = \{\tilde{T}^i, \cdots, \tilde{T}^n\}$
let $k := 0$
repeat
    $k := k + 1$
    $Q' := \bigcup_{1\le i\le n}\{\dot{q} \mid \exists q \in Q_{k-1}.\ (q,\dot{q}) \in \tilde{T}^i\}$
    $Q_k := Q_{k-1} \cup Q'$
until $Q_k = Q_{k-1}$
return $Q_k$
```

C. The Large intermediate BDD problem

The disjunctive partitioning and Algorithm 3 usually solves the problem with large BDD representation of δ and T. However, when analyzing the running time of that algorithm another problem is revealed. If we compute the forward and backward reachable states and plot the BDD size of the intermediate variables Q_k, see Figure 1.a we will notice the intermediate reachable state sets have very large BDDs.

The observed intermediate BDD size explosion is an obstacle to efficient BDD-based computation. For example, during the forward reachability of the central lock model the intermediate BDDs grow very fast behind the memory limit causing the reachability search to fail. In addition, as BDD operations have a time complexity directly influenced by the size of the involved BDDs, large intermediate BDDs will result in very long running-times.

Experiment 4.1: **Random BDD Growth**

The main reason behind the intermediate BDD size explosion is that new elements are added to the set Q_k in such pseudo-random order that its BDD can not be reduced efficiently. If we draw the BDDs for a monotonic but randomly growing set of 2^n elements, we will observe the same behavior. See Figure 1.b for an experiment with $n = 6, 7, 8$. In addition, in a reachability search that utilizes breadth-first traversal this effect is amplified in several dimensions. □

One way to reduce the intermediate BDDs in reachability search is to introduce a type of guided search that avoids this randomness. This subjects has been discussed in [8], [2], [15]. Here, we will present a more relevant and general approach.

Having access to a set of transfer functions and the 'in between dependency map' (as a PCG), the reachability problem can be formulated as the problem of calculating a *global fixpoint* of a set of functions. One way to do so is to use a *monolithic* approach, that is, to compute the global fixpoint by applying all the transfer functions at the same time. A more efficient solution is to compute the global fixpoint by calculating a series of dependent local fixed points using the disjunctive transfer functions. This problem dates back to the classic work of the Polish logician Alfred Tarski in lattice theory [17], and chaotic fixpoint computation [7]. Based on these works and the static dependency map introduced by PCGs, we suggest a simple iteration strategy for reachability computation. Assuming that the average level-1 complexity is low, the disjunctive transfer function can be used for efficient reachability search by the *Workset* algorithm (Algorithm 4).

Algorithm 4: WorksetBackwardReachable

```
input: $Q_m, \Delta =$ set of $\tilde{T}^i$
let $W := \Delta,\ Q_0 := Q_m,\ K := 0$
repeat
    $\mathbb{H}$ : Pick and remove a transition $\tilde{T}^i \in W$
    $k := k + 1$
    $Q_k = Q_{k-1} \cup BackwardReachability(Q_{k-1}, \tilde{T}^i)$
    if $Q_k \neq Q_{k-1}$ then $W := W \cup D^+(A^i)$
until $W = \emptyset$
return $Q_k$
```

The Workset algorithm was inspired by a reachable marking vector exploration algorithm of Petri nets which uses a "work-list" of possibly enabled transitions (hence the name). This algorithm will try to saturate each disjunctive transfer function in order to achieve node reduction in each region of the BDD Q_k before another region is targeted. In some situations, it is however more efficient to not strive for this local saturation. Algorithm 5, the *Step-stone* algorithm, is a variation of the Workset algorithm which works without saturation.

In both algorithms, in the part marked by $\mathbb{H}$ a heuristic decision procedure is applied to choose the next automaton from the working set W. For simple problems with low level-1 dependency, we found a simple procedure based on *reinforcement learning* to be very reliable for the workset algorithm [16]. In short, one "awards" the automata that have been able to find new states in past by selecting them more often than the others.

The step-stone on the other hand works best with a selection stragey which minimizes the cardinality of the union of the number of events in automata in W as follows

$$\text{choose } \tilde{T}^i \text{ s.t. } min\{\ |\ [\bigcup_{\tilde{T}^j \in W} \Sigma^j] \cup [\bigcup_{A^k \in D^+(A^i)} \Sigma^k]\ |\ \}$$

In practice, this strategy will make the step-stone algorithm to work very similar to the workset algorithm. Note also that both strategies will try to select A^i from a limited part of the model, resembling efficient modular state traversal.

Algorithm 5: StepstoneForwardReachable

```
input: $q_i, \Delta =$ set of $\tilde{T}^i$
let $W := \Delta,\ Q_0 := \{q_i\},\ K := 0$
repeat
    $\mathbb{H}$ : Pick a transition $\tilde{T}^i \in W$
    $k := k + 1$
    $Q_k = Q_{k-1} \cup \{\dot{q} \mid \exists q \in Q_{k-1}.\ (q, \dot{q}) \in \tilde{T}^i)\}$
    if $Q_k = Q_{k-1}$ then
        $W := W - \{A^i\}$
    else
        $W := W \cup D^+(A^i)$
    end
until $W = \emptyset$
return $Q_k$
```

V. Algorithm Efficiency

In Figure 1.a, it is clearly seen that the workset algorithms are able to avoid the large intermediate BDD problem. This, in conjunction with the disjunctive representation of the transfer function is the key to efficient state space search.

In Table I, the reachability search performance for a set of selected SCT problems are shown (refer to [18] for more details). In addition Table II presents performance data for synthesis of a maximally permissive non-blocking and controllable supervisor for some of these problems[1]. In all tables, a "-" sign indicates out of memory (256 MB) or out of time (15 minutes). The given values are the average of multiple runs.

As clearly seen in these tables, the workset reachability algorithm is very efficient. Even without combining this algorithm with more sophisticated verification or synthesis algorithm (such as modular and hierarchical ones), it is capable of solving very large verification and synthesis problems.

Finally, it was noticed in [19] that the average size of the dependency sets $D(A^i)$ gives a better indication about the performance of the workset algorithm than for example the state space size. This explains why the state space of hand-made academical examples such as the transfer line (TL in Table I) with very small dependency sets are much easier to search.

[1]These numbers were acquired by running the DES tool Supremica on a standard desktop PC (2.5 GHz P4) with the SUN JDK 5.0 running on Microsoft XP. Supremica used the JBDD interface to communicate with the BuDDy BDD library. The variable ordering algorithm used in these examples was a modification of the FORCE heuristic [1] which works with process communication graphs instead of hypergraphs extracted from CNF formulas.

TABLE I

RUNNING TIMES FOR REACHABILITY SEARCH BY DIFFERENT ALGORITHMS.

Model	**States**	**Reachability Time [s]**							
		Monolithic		**Disjunctive**		**Workset**		**Step-stone**	
		Backward	Forward	B.	F.	B.	F.	B.	F.
TL(10)	1.18×10^{21}	0.70	1.49	0.95	1.79	0.15	0.24	0.24	0.55
TL(50)	2.29×10^{105}	-	-	-	-	33	15	48	15
TL(100)	5.26×10^{210}	-	-	-	-	448	128	118	-
PHP(7,6)	5.16×10^{11}	0.25	0.47	0.27	0.40	0.79	0.30	1.06	0.34
PHP(8,7)	8.02×10^{13}	3.69	14.20	4.01	9.21	6.16	4.99	188	3.27
PHP(9,8)	3.87×10^{16}	147	207	149	259.12	125	53	336	128
Robot Cell	7.52×10^{8}	0.06	0.06	0.05	0.04	0.01	0.01	0.01	0.01
AGV	5.16×10^{10}	0.15	0.61	6.91	0.46	0.02	0.07	0.02	0.07
Parallel Man.	9.65×10^{23}	-	-	9.12	37.68	0.19	0.92	0.13	0.14
Central Lock.	1.18×10^{26}	-	-	-	-	12	5.87	4.81	32
Shoe Factory	2.03×10^{27}	-	-	10.47	12.33	2.01	2.44	2.89	3.80
Shoe F. + 3 shoes	2.84×10^{32}	-	-	72	80	5.60	4.42	2.93	2.79

TABLE II

RUNNING TIMES FOR SUPERVISORY SYNTHESIS.

Model	**Iterations**	**Total time [s]**		
	(during synthesis)	**Supremica**	**Monolithic**	**Workset**
Robot Cell	2	1.10	0.22	< 0.02
AGV	4	-	0.78	0.11
Parallel Man.	2	-	-	0.24
Shoe Factory + 3 shoes	2	-	-	3.31
PHP(5,6)	2	-	0.10	0.15
PHP(6,7)	2	-	0.33	1.00
Central Locking	3	-	-	61

VI. SUMMARY AND CONCLUSIONS

While symbolic tools such as Binary Decision Diagrams and SAT solvers has for long been known to the DES and Supervisory Control Theory communities, their use have been very limited. The main reason to this is probably that a straight-forward usage often leads to disappointedly bad performance. In this work, we have pointed out some very simple methods for achieving better performance during reachability search, which result in much more efficient verification and synthesis procedures.

REFERENCES

[1] Fadi Aloul, Igor Markov, and Karem Sakallah. FORCE: A fast and easy-to-implement variable-ordering heuristic. In *Proceedings of the Great Lakes Symposium on VLSI*, 2003.

[2] Dennis Arkeryd. Efficient utilization of BDD:s for resource booking problems. Master thesis EX052/2000, Chalmers University of Technology, 2000.

[3] Adnan Aziz and Serdar Taziran. BDD variable ordering for interacting finite state machines. In *Proceedings of 31th Design Automation Conference*, pages 283–288, 1994.

[4] Z. Brezocnik, A. Casar, and T. Kapus. Efficient symbolic traversal algorithms using partitioned transition relations. In *Proceedings of COST 247 International Workshop on Applied Formal Methods in System Design*, pages 146–155, Slovenia, 1996.

[5] Randal E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[6] Jerry R. Burch, Edmund M. Clarke, and David E. Long. Symbolic model checking with partitioned transition relations. In A. Halaas and P. B. Denyer, editors, *Proceedings of 1991 Intl. Conf. on VLSI*, 1991.

[7] P. Cousot and R. Cousot. Constructive versions of Tarski's fixed point theorems. *Pacific Journal of Mathematics*, 82:43–57, 1979.

[8] Jaco Geldenhuys and Anti Valmari. Techniques for smaller intermediary BDDs. In *The 12th International Conference on Concurrency Theory, Lecture Notes in Computer Science 2154*, pages 233–247. Springer-Verlag, 2001.

[9] A. Geser, J. Knoop, G. Luettgen, B. Steffen, and O. Ruething. Chaotic fixed point iterations. Technical Report MIP-9403, University of Passau, 1994.

[10] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International Series in Computer Science, 1985.

[11] Gerard Hoffmann and Howard Wong-Toi. Symbolic synthesis of supervisory controllers. In *Proceedings of 1992 American Control Conference*, pages 2789–2793, Chicago, IL, USA, 1992.

[12] R. Hojati, S. Krishnan, and R. Brayton. Early quantification and partitioned transition relation. In *Proceedings of IEEE International Conference on Computer Design*, pages 12–19, 1996.

[13] Alan John Hu. *Techniques for Efficient Formal Verification Using Binary Decision Diagrams*. PhD thesis, Carnegie Mellon University, 1995.

[14] Ratnesh Kumar, V. K. Garg, and S. I. Marcus. On controllability and normality of discrete event dynamical systems. *Systems and Control Letters*, 17:157–168, 1991.

[15] Kavita Ravi and Fabio Somenzi. Hints to accelerate symbolic traversal. In *Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, pages 250–264, 1999.

[16] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, 1998.

[17] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[18] Arash Vahidi. Efficient analysis of discrete systems. supervisor synthesis with binary decision diagrams. PhD thesis 487, Department of Signals and Systems, Chalmers University of Technology, 2004.

[19] Arash Vahidi, Bengt Lennartson, and Martin Fabian. Efficient supervisory synthesis of large systems. In *Proceedings of the International Workshop on Discrete Event Systems (WODES'04)*, September 2004.

[20] William Murray Wonham. *Notes on Control of Discrete-Event Systems*. University of Toronto, 2002.