Computational effort of BDD-based supervisor synthesis of extended finite automata

Computational Effort of BDD-based Supervisor Synthesis of Extended Finite Automata

Sander Thuijsman*, Dennis Hendriks†, Rolf Theunissen†, Michel Reniers*, Ramon Schiffelers*‡
* Eindhoven University of Technology, Eindhoven, The Netherlands
Email: s.b.thuijsman@tue.nl
† ESI (TNO), Eindhoven, The Netherlands
‡ ASML, Veldhoven, The Netherlands

***Abstract*— We consider supervisor synthesis of Extended Finite Automata that are represented using Binary Decision Diagrams (BDDs). Peak used BDD nodes and BDD operation count are introduced as platform independent and deterministic metrics that quantitatively indicate the computational effort needed to synthesize a supervisor. The use of BDD operation count is novel with respect to expressing supervisor synthesis effort. The (dis-)advantages of using these metrics to state of practice metrics such as wall clock time and worst case state space size are analyzed. The supervisor synthesis algorithm is initiated with a certain event- and variable order. It is already known from literature that variable order influences synthesis performance. We show that the event order is also relevant to consider. We discuss how these orders influence the synthesis effort and, by performing an experiment on a set of models, we show the extent of this influence.**

I. Introduction

Supervisory Control Theory (SCT) [1], [2] is a model-based approach to control (cyber-physical) systems. Given a plant (a model that defines all possible system behavior) and a specification (a model that defines what behavior is allowed), a supervisor can be computed algorithmically (synthesized) that restricts the plant's behavior so that it is in accordance with the specification. Depending on the synthesis algorithm, the supervised system has some useful properties by construction, such as safety, nonblockingness, controllability and maximal permissiveness. There are a number of formal modeling frameworks on which SCT can be applied. The framework of Extended Finite Automata (EFA) [3] is an extension to Finite State Automata by augmenting them with variables, guard expressions and updates, which enables more convenient modeling of systems.

The power of SCT has been demonstrated in literature [4]–[8]. Despite the beneficial properties of SCT, industrial acceptance is scarce. The exponential state space explosion that occurs during supervisor synthesis is a major hurdle [9]. A way to mitigate this is by symbolically representing the EFA using Binary Decision Diagrams (BDDs) [10]–[13]. This approach is considered state of the art to handle industrial sized systems [14].

The synthesis complexity depends on the amount of BDD nodes required to represent the system during synthesis [12]. It is well known that this number is largely dependent on the *variable order* [15]. A contribution of this paper is showing that in addition to the variable order, the *event order* is relevant to consider for the effort that is needed to synthesize a supervisor (supervisor synthesis effort). This is the order in which the synthesis algorithm iterates through the events when performing reachability/fixed point searches. By performing supervisor synthesis to a set of models using a large number of random variable- and event orders, we demonstrate to what extent these orders influence the supervisor synthesis effort.

Even when restricting ourselves to only BDD-based supervisor synthesis of EFA, there is no consensus in literature on how to express the supervisor synthesis effort. Several metrics are used, such as: wall clock time, peak random access memory and state space sizes [6], [8], [14], [16], [17]. These metrics give some intuitive indication on the supervisor synthesis effort, but there is no exact method on how to interpret them. In this paper, we present *peak used BDD nodes* and *BDD operation count* as deterministic, platform independent metrics that provide a quantitative indication of the supervisor synthesis effort. Peak used BDD nodes has been used to express supervisor synthesis effort [12], [16]–[18]. The contribution of BDD operation count is novel in this context. We show the relevance of these BDD-based metrics and compare them to state of practice metrics.

II. BDD-Based Supervisor Synthesis of EFA

A. Supervisor synthesis of EFA

We consider EFA A defined as *7-tuple*

$$A = (L, D, \Sigma, E, L^0, D^0, L^m)$$

where L is the domain of locations, $D{=}D_1{\times}...{\times}D_p$ is the domain of variables and Σ is the set of events, usually called the alphabet. L^0 is the set of initial locations, $D^0{=}D_1^0{\times}...{\times}D_p^0$ is the set of initial variable values and L^m is the set of marked locations. E is a set of edges where an edge $e_k \in E$ is defined as *5-tuple*

$$e_k = (l_k^o, l_k^t, \sigma_k, g_k^e, f_k^e)$$

where l_k^o and l_k^t are the origin and target location in L, σ_k is an event in Σ, $g_k^e : D \to \{false, true\}$ is the guard evaluation function and $f_k^e : D \to D$ is the update function that assigns new values to the variables.

The *state* of the EFA specifies the automaton location and the value of each of the variables as a pair $(l, d) \in L{\times}D$.

Consequently the initial states are pairs $(l^0, d^0) \in L^0 \times D^0$ and the marked states are pairs $(l^m, d) \in L^m \times D$.

An edge e_k is *enabled* if the current location is l_k^o and g_k^e evaluates to $true$ for the current variable values. Only enabled edges can be executed. Upon execution, the state is updated according to l_k^t and f_k^e. Thus we can speak of an origin and target state that relate to an edge.

The set of events Σ is split into two disjoint subsets, Σ_c and Σ_u, denoting *controllable* and *uncontrollable* events. We speak of controllable and uncontrollable edges respective to the event they are labeled with. Controllable edges can be enforced by the supervisor, uncontrollable edges can not; The supervisor can only disable an edge e_k when $\sigma_k \in \Sigma_c$.

This paper is based on the supervisor synthesis algorithm for EFA as presented in [19]. An EFA of the plant that models all possible behavior and an EFA of the specification, where all forbidden behavior of the system ends in blocking states, are given. The algorithm produces a supervisor EFA by iteratively strengthening guards resulting in blocking states finally becoming unreachable. This supervisor EFA contains all behavior of the system that is safe (blocking states can never be reached), nonblocking (a marked state can always be reached), controllable (only controllable edges are restricted by the supervisor) and it is maximally permissive (the restrictions are minimal in order to guarantee safety, nonblockingness and controllability). A high level pseudo code description of the algorithm is given in Algorithm 1. A more rigorous description can be found in [19]. *Flagging* a state by $true$, denotes adapting a predicate so that the new predicate evaluates to $true$ for that state.

Algorithm 1 Supervisor synthesis of EFA

Input: Plant P and specification R represented as EFA
Output: Maximally permissive nonblocking, safe and controllable EFA G respective to P and satisfying R

1: Create refined EFA G that has the same behavior as P, where the disallowed behavior by R ends in a set of blocking states in G
2: Create bad state predicate $\mathcal{B}$: flag all blocking states of G by $true$ and all others by $false$
3: **repeat** Create new nonblocking predicate $\mathcal{N}$: flag all marked states of G by $true$ and all unmarked states by $false$
4: **repeat** Flag all states in $\mathcal{N}$ by $true$ that have an enabled edge to a state already flagged $true$ in $\mathcal{N}$
5: **until** $\mathcal{N}$ did not change
6: Flag all states in $\mathcal{B}$ that are currently set to $false$ in $\mathcal{B}$ by their negation in $\mathcal{N}$
7: **repeat** Flag all states in $\mathcal{B}$ by $true$ that have an enabled uncontrollable edge to a state already flagged $true$ in $\mathcal{B}$
8: **until** $\mathcal{B}$ did not change
9: Set guards in G to $false$ so that all controllable edges that lead to a state flagged $true$ in $\mathcal{B}$ are disabled
10: **until** Guards did not change

B. CIF toolset

There are several tools that allow modelling of plants and requirements with the ability to synthesize a supervisor. Of the tools considered in [20], the tools Supremica [14] and CIF [21] allow for the use of EFA. Both tools base their EFA supervisor synthesis algorithm on the use of BDDs. The syntheses in this paper are performed using the EFA supervisor synthesis tool of CIF[1]. CIF has been used to synthesize supervisors for industrial sized systems [4]–[6], [22], [23]. Its EFA supervisor synthesis tool is based on the supervisor synthesis algorithm sketched in Section II-A, barring some minor differences that are irrelevant to our interpretation of the algorithm. The tool is instrumented to extract the metrics introduced in Section III.

C. Binary Decision Diagrams

Boolean functions can be used to symbolically represent EFA [24]. An efficient manner to store and make adaptations to Boolean functions is using BDDs [10], [11]. These are directed acyclic graphs that contain decision nodes that all have two child nodes which can be reached by taking a low/0 or high/1 decision edge from the parent node. At the bottom of the graph there are two terminal nodes representing $true$ and $false$. (Groups of) nodes represent variable values or automata locations. When referring to BDDs here, we consider Reduced Ordered BDDs [25] which impose some extra restrictions, guaranteeing a canoncial form. This representation is better suited for computer manipulation [12]. After converting the EFA to a BDD structure, the synthesis operations are directly applied to the BDDs. When synthesis is complete, the BDDs can be converted back to the EFA structure. Several synthesis algorithms that employ BDDs exist. Their efficiency usually depends on their handling of the BDDs [12], [13], [17].

D. Variable order

Because *Ordered* BDDs are used, a total ordering is imposed over the set of variables. If in the variable order a precedes b (denoted $a<b$), then no path of decision edges can exist in the BDD from the nodes representing b to a [25]. The variable order has a major effect on the amount of BDD nodes that are used to represent a system [15]. In turn, it has a large influence on the efficiency of supervisor synthesis for EFA [12], [17]. Finding the optimal variable order is NP-hard [26], thus CIF uses the FORCE and sliding window algorithms [27] to come up with a good order, which makes a relatively small BDD representation more likely. These heuristic algorithms are supplied with an initial variable order from which variables are grouped together that have high interaction, meaning they often appear together in guard or update expressions. Note that these algorithms find local optima; Providing them with different initial orders, gives different resulting variable orders. In CIF, the default initial variable order is an alphabetic ordering of the names of all variables and automata locations.

[1]The CIF tooling and documentation is open source and freely available at `cif.se.wtb.tue.nl`

E. Edge order

In lines 4 and 7 of Algorithm 1, a backwards reachability search is performed to find edges of which the target state is flagged $true$ in predicate $\mathcal{N}$ or $\mathcal{B}$, and the origin state is not. If such an edge is found, the predicate will be adapted on the fly; The iteration over edges does not restart, and the new predicate will immediately be considered for the next edge that is checked. Applying the edges in a different order will not influence the resulting predicate when exiting the reachability loop. However, the BDD sizes required to represent the intermediate predicates, and the rate at which states get flagged in the predicate may differ. The construction of the guards in line 9 of Algorithm 1 is also influenced by the edge order. Unlike the variable order, no reordering algorithm is applied to the edge order in CIF.

Example: We consider the EFA of Figure 1(a). This EFA consists out of two locations $L{=}\{l_0, l_1\}$ of which l_1 is marked: $L^m{=}\{l_1\}$, as indicated in Figure 1(a) by a double circle. We have Boolean variables a and b, forming variable space $D{=}\{false, true\}{\times}\{false, true\}$. Both variables are initially set to $false$: $D^0{=}\{(false, false)\}$. Events a_on and b_on can occur at l_0, variables a and b will then respectively update to $true$. These updates are denoted in Figure 1(a) by the keyword 'do'. An edge with event label $continue$ can be taken from origin location l_0 to target location l_1. This can only happen if the guard $a{=}{=}b$ evaluates to $true$. The guard is denoted by the keyword 'if'. The reachable state space of this EFA is displayed in Figure 1(b). The edges have been enumerated e_1 to e_6. For easy reference, we use a or $\neg a$ to respectively denote the values $a{=}true$ or $a{=}false$, same holds for b.

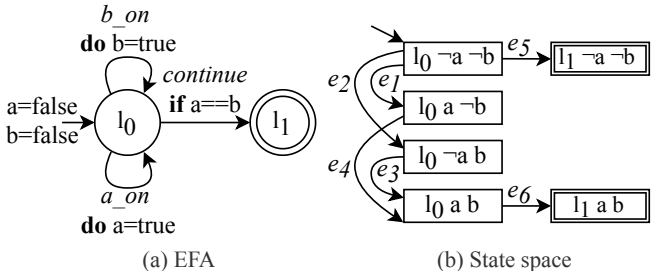


Fig. 1: Example EFA and respective reachable state space

Let us consider the construction of the nonblocking predicate $\mathcal{N}$. Initially, only the marked states are flagged as $true$. As our example only has two locations, a single Boolean variable l_s can be used to specify the location. We define l_s to indicate l_0 and $\neg l_s$ indicates l_1. The BDD representing the initial predicate is shown in Figure 2(a). Solid decision edges denote that the origin node evaluates to $true$, dashed decision edges denote $false$. The square T and F vertices represent the $true$ and $false$ terminal nodes.

Let us examine the case that we continue performing backwards reachability by first considering edge e_6. The target state of this edge is part of the nonblocking predicate, thus we can flag its origin state $(l_s\ a\ b)$ in the predicate, resulting in the BDD of Figure 2(b). All BDDs in Figure 2 have variable order $l_s{<}a{<}b$. A caption 'B.w. $e_x; e_y$' denotes backwards reachability by first applying edge e_x followed by edge e_y. Applying edge order $e_6; e_5; e_3$ results in the same nonblocking predicate as applying $e_6; e_3; e_5$. The resulting BDD is shown in Figure 2(e). However, the intermediate BDDs to represent the predicate after applying $e_6; e_5$ or $e_6; e_3$ use a different amount of nodes, seen when comparing Figure 2(c) to 2(d). This illustrates how edge order can influence the efficiency of the algorithm.

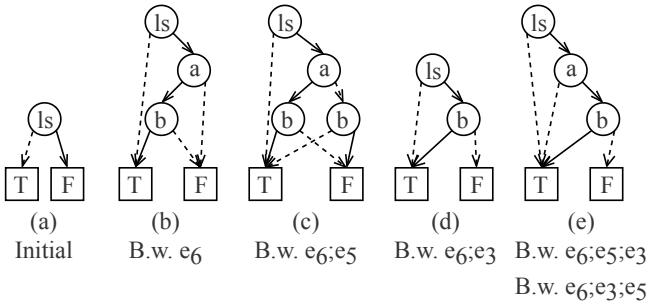


Fig. 2: Different BDD sizes for varying edge orders

F. Ordering granularity

Complex supervisory control models will contain many edges. When analyzing edge orders, instead of working with these large sets, we use a less granular approach and group all edges with the same event label together. In the remainder of this paper we will consequently refer to these orders by *event* orders rather than *edge* orders. Likewise for variable order, we will consider the variables and automata locations as defined in the model, instead of the order of the Boolean variables in the BDD. We also do not interleave [15] variables with each other. This lower granularity enables more intuitive interpretations of why less effort is required when using a certain order compared to some other order.

III. Metrics for Computational Effort

Algorithms are typically judged by their space- and time complexity [28]. We propose two metrics: *peak used BDD nodes* and *BDD operation count* to quantitatively express the space- and time effort required for supervisor synthesis. These metrics have some advantages over peak random access memory and wall clock time: First, they are deterministic; performing a supervisor synthesis twice with the same input and algorithm configuration will give the exact same result. This determinism also holds when doing the synthesis on two different platforms, even if one is a supercomputer and the other is a personal computer. As a result, it becomes easier to compare results from different publications. Second, there is no overhead in the measurement, loaded-in Java classes and other computer processes will not influence the measurement. After introducing these BDD-based metrics, an elaboration on their relation to the conventional metrics will be given in Section III-C.

We distinguish *complexity* from *effort*. Complexity regards classes of problems, and defines the generic trend of the

(space/time) resources a computation requires for inputs of different sizes, often expressed using 'Big O' notation [29]. Effort specifies the amount of resources for one particular computation, where the complete input is considered rather than only its size. This input includes algorithm configuration settings and, in our case, variable- and event order.

A. Peak used BDD nodes

Due to state space explosion [9], the space complexity is a limiting factor when applying supervisor synthesis. During supervisor synthesis, the number of BDD nodes that are used to describe the system generally fluctuates. The space effort can be measured by the peak number of BDD nodes used during synthesis [12], [28]. Since *Reduced* Ordered BDDs are used, which are minimal representations, the peak used BDD nodes is the minimal amount of BDD nodes required to represent the predicates to solve the synthesis problem.

In CIF, BDD nodes are stored in a hash table. Each new node is allocated to an entry in the hash table. Once the hash table reaches a certain fill rate, garbage collection is employed to free no longer used entries. We only count the *used* BDD nodes, i.e., hash table entries that still contain relevant information for the BDDs that are still in use. Garbage collection is performed by means of a standard mark-and-sweep algorithm. Functions from the implementation of this algorithm in the JavaBDD library[2] are reused to count the BDD nodes that are in use.

Peak used BDD nodes is a reproducible metric; Performing a supervisor synthesis twice with the same input yields exactly the same peak used BDD nodes. For the example in Section II-E, the peak used BDD nodes when applying edge order $e_6; e_5; e_3$ is 6 nodes, and for $e_6; e_3; e_5$ it is 5 nodes.

B. BDD operation count

The time complexity can be expressed in the number of steps/operations of an algorithm [28]. The time complexity of performing operations on BDDs is dependent on the number of nodes in the BDDs. For example, performing a Boolean operation $(a \otimes b)$ has a worst case time complexity $\mathcal{O}(\#a \times \#b)$, where $\#a$ denotes the number of nodes in BDD a [30]. As the supervisory synthesis is done by performing operations on BDDs, we use BDD operation count to express the time effort of performing supervisor synthesis. Since BDD operations (such as `and`, `or` and `not`) are implemented as functions that employ structural recursion on BDD nodes, the number of invocations of such functions can be used to express time effort. Since the functions are deterministic, the results are reproducible.

Generally, these functions consist of three parts. First, a few checks are made to see whether the requested calculation is a terminal case. Second, if it is a non-terminal case, it is checked whether the calculation has already been performed, and is still in the cache. Note that we do not mean hardware cache here, but a table actively storing results of previous calculations. If both previous cases did not occur, the function performs recursive expansion over the child nodes. For more details about terminal cases, cache lookup and recursive expansion over child nodes we refer to [31].

Checking whether there is a terminal case, or looking for a solution in the cache requires relatively little computational effort. Thus, the time effort is mainly influenced by the third part of the algorithm, where actual operations are applied to the BDD [31]. Therefore, we only increase the BDD operation count each time we reach this part of the algorithm.

C. Relevance of metrics

In order to compare the BDD-based metrics to the conventional metrics, we perform a number of supervisor syntheses and extract these metrics. The data presented in this paper is acquired by performing supervisor syntheses to the models shown in Table I[3]. The models are selected to have a wide range of model sizes. Table I shows the worst case state space size of the uncontrolled plant for each model, which is the product of all location and variable domain sizes.

TABLE I: Case study models

Name	Worst case state space size
Robotic swarm aggregation [32]	$1.0 \cdot 10^{0}$
Robotic swarm clustering [32]	$1.0 \cdot 10^{0}$
Robotic swarm segregation [32]	$6.4 \cdot 10^{1}$
Robotic swarm formation [32]	$8.0 \cdot 10^{1}$
Power substation system [33]	$2.1 \cdot 10^{10}$
Advanced driver assistance system [23]	$3.4 \cdot 10^{9}$
Multi agent formation [34]	$1.0 \cdot 10^{3}$
Ball sorting system [35]	$7.4 \cdot 10^{4}$
Production cell [36]	$7.5 \cdot 10^{8}$
FESTO production line [6]	$1.3 \cdot 10^{28}$
Waterway lock [22]	$6.0 \cdot 10^{32}$

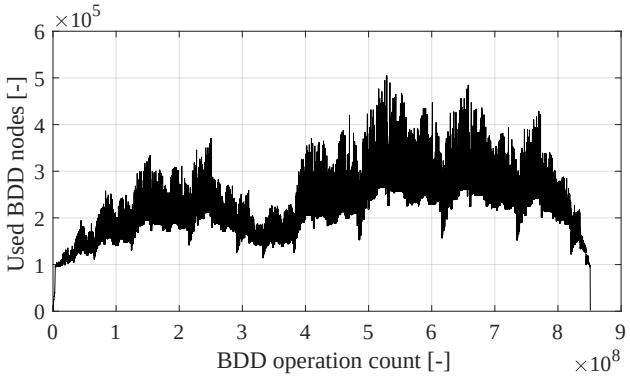
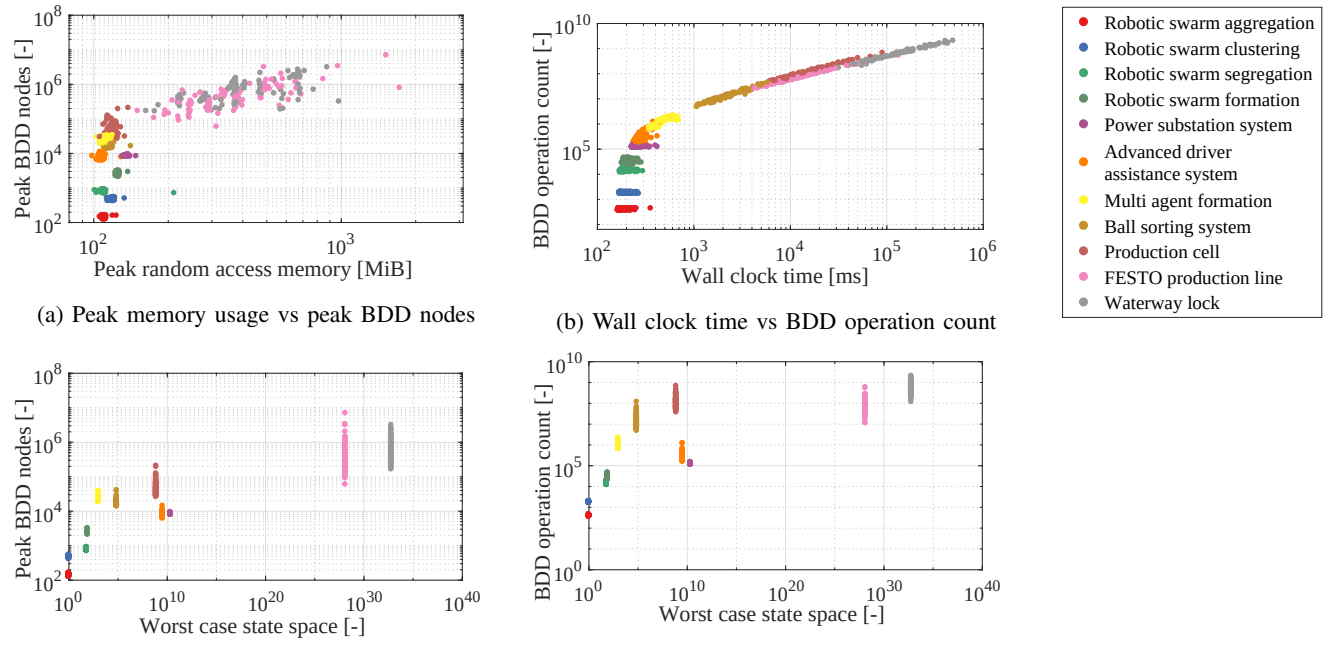


Fig. 3: Evolution of used BDD nodes during synthesis

For a supervisor synthesis of the Waterway lock model, Figure 3 shows how the number of used BDD nodes evolves, as BDD operations are performed during synthesis. Intuitively, the horizontal axis represents the ever-increasing number of operations performed as time progresses, and the vertical axis represents the fluctuating memory usage. The metrics

[2]The JavaBDD library is available at `javabdd.sourceforge.net`

[3]Some models are readily available at `github.com/magoorden/T-AC2018`, others can be extracted from the listed references



(a) Peak memory usage vs peak BDD nodes

(b) Wall clock time vs BDD operation count

(c) Worst case state space size vs peak BDD nodes

(d) Worst case state space size vs BDD operation count

Fig. 4: BDD-based metrics against conventional metrics

presented in this paper are the maxima along both axes in this plot; the peak used BDD nodes and the final BDD operation count.

Figures 4a and 4b show how peak random access memory and wall clock time relate to peak used BDD nodes and BDD operation count. A supervisor was synthesized for each model of Table I for 100 pairs of random variable- and event orders. Note that the heuristic variable ordering algorithms were turned on for this test. The measurements for random access memory and wall clock time were done separately from the measurements of the BDD-based metrics to avoid them from interfering. It can be seen that for small models, peak random access memory and wall clock time can not indicate a difference in synthesis effort, as all results of the small models are grouped around the same result $\sim (1 \cdot 10^2$ MiB, $2 \cdot 10^2$ ms). Influences like loaded-in Java classes for the peak random access memory and reading the input- and writing the output file for wall clock time dominate these metrics. The BDD-based metrics enable a distinction in effort for the actual synthesis part of the computation.

For larger computations, a linear relation is visible between wall clock time and BDD operation count. The threshold at which this relation starts, and its slope, are dependent on the used hardware. The scattering that is seen for larger computations in Figure 4a is a result of the manner in which the BDD space allocation takes place; When the current table is full, it gets doubled in size, the new free entries in this table will have an influence on the memory, but are not measured when counting the used BDD nodes. Also, when performing computations that require more memory, Java (Java Virtual Machine) will perform garbage collection in the background to free memory. For separate measurements this will happen at different times, which impacts the peak random access memory, not the amount of used BDD nodes.

An advantage of wall clock time and peak random access memory is that a user performing supervisor synthesis is more likely to be familiar with these metrics. It gives a better idea whether their computer is able to perform the synthesis in an acceptable amount of time given the available memory.

The advantage of using worst case state space size of the uncontrolled system over BDD-based metrics to indicate the synthesis effort, is that no supervisor synthesis or reachability computations are required to calculate this number. Figures 4c and 4d show how this state space size relates to the BDD-based metrics. There is some general trend of a larger state space size suggesting more supervisor synthesis effort, but due to the high variance it is not a very accurate indicator.

IV. Impact of Variable- and Event Order on Computational Effort

We have presented two metrics that indicate the computational effort of a single supervisor synthesis procedure. These metrics are fairly easy to extract when performing a computation. However, we have to take care when making conclusions on this synthesis effort. The variable- and event order as introduced in Section II have an influence on the results. This can also be seen in Figure 4, where the results are scattered due to using different variable- and event orders. Recall that the BDD-based metrics are deterministic, re-performing a synthesis with the same variable- and event orders would provide the exact same result.

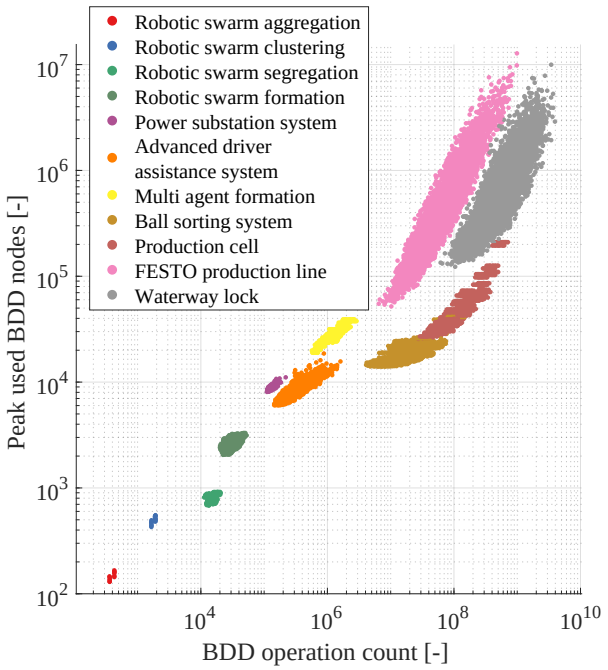


Fig. 5: Supervisor synthesis effort for all combinations of 100 event- and 100 variable orders for each model

A. *Extent of order influence*

We investigate to what extent the initial variable order and event order influence the supervisor synthesis effort. For each of the models of Table I, a supervisor has been synthesized for all combinations of 100 random event orders and 100 random initial variable orders. The effort of performing each synthesis is shown in Figure 5. It can be seen that there are major differences in computational effort by using different orders. For the FESTO production line, the highest peak used BDD nodes is 246 times larger than the lowest peak used BDD nodes. For BDD operations this factor is 153. This is purely a result of changing the variable- and event orders; all other algorithm configurations were the same for all measurements. The results are especially surprising when considering that the heuristic variable order algorithms described in Section II-D are being applied to the initial variable order.

Figure 5 also shows that measuring both peak used BDD nodes and BDD operation count is relevant. It would be difficult to distinguish the computational effort between some of the syntheses, if only one of the metrics was used. For example, if we only measured the peak used BDD nodes, we would not see much difference for the efforts of the Ball sorting system and Multi agent formation. Additionally considering the BDD operation count enables us to differentiate between the efforts.

Figure 6 shows the peak used BDD nodes for all syntheses of the FESTO production line model. Darker squares indicate a higher amount of peak used BDD nodes. All variable- and event orders were given an index. A row shows the peak used BDD nodes of all syntheses that were performed with the same event order and varying variable orders, and a column shows the same for a fixed variable order with varying event orders. In Figure 6, we see rows and columns where the elements are similarly colored, indicating that variable order and event order both have a reasonable impact on the peak used BDD nodes for this particular model. There are other models where only the elements in columns are similarly colored, indicating that the variable order mainly influences their synthesis effort. We observe similar results for the BDD operation count.

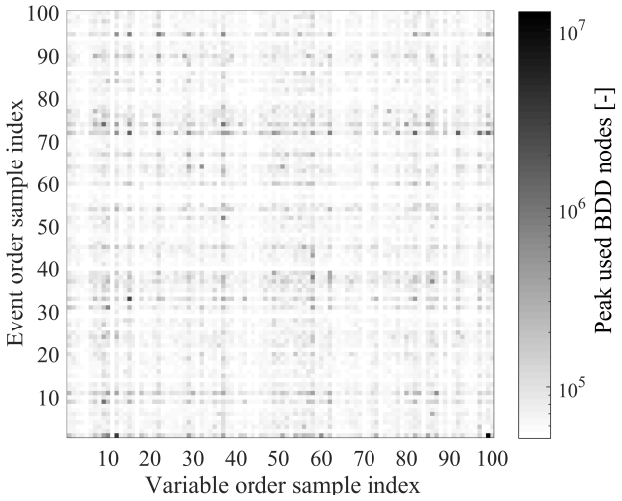


Fig. 6: Peak used BDD nodes for all supervisor syntheses of FESTO production line

If we define the peak used BDD nodes for a certain model as a deterministic function $f(o_{v,i}, o_{e,j})$, where $o_{v,i}$ is the $i'th$ sample random variable order and $o_{e,j}$ the $j'th$ sample random event order, the global sample mean [37] of the peak used BDD nodes $\mu_G(f)$ is given by Equation 1.

$$\mu_G(f) = \frac{1}{N \cdot M} \sum_{i=1}^{N} \sum_{j=1}^{M} f(o_{v,i}, o_{e,j}) \tag{1}$$

where N and M respectively are the total number of sampled variable- and event orders. For our experiment, $N = M = 100$ for each model.

The global (unbiased) sample variance [37] of the peak used BDD nodes $\sigma^2_G(f)$ is given by Equation 2.

$$\sigma^2_G(f) = \frac{1}{N \cdot M - 1} \sum_{i=1}^{N} \sum_{j=1}^{M} \big(f(o_{v,i}, o_{e,j}) - \mu_G(f)\big)^2 \tag{2}$$

The sample variance $\sigma^2_{v,i}(f)$ of the peak used BDD nodes for the event orders tested with a particular variable order $o_{v,i}$, is given by Equation 3.

$$\sigma^2_{v,i}(f) = \frac{1}{M - 1} \sum_{j=1}^{M} \big(f(o_{v,i}, o_{e,j}) - \mu_{v,i}(f)\big)^2 \tag{3}$$

where $\mu_{v,i}(f) = \frac{1}{M} \sum_{j=1}^{M} f(o_{v,i}, o_{e,j})$ is the mean peak used BDD nodes of the event orders tested with variable

TABLE II: Sample means and variances of all models

Name	$\mu_G(f)$	$\sigma^2_G(f)$	$\overline{\sigma^2_v}(f)$	$\overline{\sigma^2_e}(f)$	$\mu_G(g)$	$\sigma^2_G(g)$	$\overline{\sigma^2_v}(g)$	$\overline{\sigma^2_e}(g)$
Robotic swarm aggregation	$1.4 \cdot 10^2$	$1.3 \cdot 10^2$	$3.9 \cdot 10^1$	$4.0 \cdot 10^{-2}$	$4.1 \cdot 10^2$	$1.1 \cdot 10^3$	$9.9 \cdot 10^1$	$1.2 \cdot 10^3$
Robotic swarm clustering	$4.8 \cdot 10^2$	$9.2 \cdot 10^2$	$3.1 \cdot 10^2$	$1.3 \cdot 10^0$	$1.8 \cdot 10^3$	$1.5 \cdot 10^4$	$7.0 \cdot 10^2$	$1.5 \cdot 10^4$
Robotic swarm segregation	$8.1 \cdot 10^2$	$3.5 \cdot 10^3$	$4.1 \cdot 10^2$	$4.9 \cdot 10^5$	$1.4 \cdot 10^4$	$2.9 \cdot 10^6$	$3.4 \cdot 10^3$	$2.7 \cdot 10^6$
Robotic swarm formation	$2.6 \cdot 10^3$	$6.5 \cdot 10^4$	$8.2 \cdot 10^3$	$8.1 \cdot 10^6$	$3.3 \cdot 10^4$	$3.3 \cdot 10^7$	$6.1 \cdot 10^4$	$2.8 \cdot 10^7$
Power substation system	$8.4 \cdot 10^3$	$5.8 \cdot 10^4$	$2.0 \cdot 10^4$	$4.7 \cdot 10^7$	$1.3 \cdot 10^5$	$6.6 \cdot 10^7$	$4.8 \cdot 10^4$	$3.6 \cdot 10^7$
Advanced driver assistance system	$7.9 \cdot 10^3$	$1.4 \cdot 10^6$	$6.6 \cdot 10^5$	$1.2 \cdot 10^{10}$	$3.1 \cdot 10^5$	$1.6 \cdot 10^{10}$	$9.4 \cdot 10^5$	$7.6 \cdot 10^9$
Multi agent formation	$2.6 \cdot 10^4$	$2.4 \cdot 10^7$	$6.8 \cdot 10^1$	$7.4 \cdot 10^9$	$1.2 \cdot 10^6$	$2.1 \cdot 10^{11}$	$2.5 \cdot 10^7$	$2.1 \cdot 10^{11}$
Ball sorting system	$1.7 \cdot 10^4$	$1.9 \cdot 10^7$	$2.3 \cdot 10^5$	$1.5 \cdot 10^{13}$	$1.8 \cdot 10^7$	$2.5 \cdot 10^{14}$	$1.9 \cdot 10^7$	$2.4 \cdot 10^{14}$
Production cell	$5.2 \cdot 10^4$	$8.8 \cdot 10^8$	$4.2 \cdot 10^5$	$2.6 \cdot 10^{14}$	$1.4 \cdot 10^8$	$1.0 \cdot 10^{16}$	$8.8 \cdot 10^8$	$1.0 \cdot 10^{16}$
FESTO production line	$6.2 \cdot 10^5$	$4.8 \cdot 10^{11}$	$3.8 \cdot 10^{11}$	$4.6 \cdot 10^{15}$	$9.0 \cdot 10^7$	$6.3 \cdot 10^{15}$	$3.4 \cdot 10^{11}$	$4.2 \cdot 10^{15}$
Waterway lock	$7.6 \cdot 10^5$	$4.0 \cdot 10^{11}$	$2.6 \cdot 10^{11}$	$1.0 \cdot 10^{17}$	$6.8 \cdot 10^8$	$1.9 \cdot 10^{17}$	$2.8 \cdot 10^{11}$	$1.3 \cdot 10^{17}$

order $o_{v,i}$. The mean sample variance for fixed variable orders $\overline{\sigma^2_v}(f)$ is computed by Equation 4.

$$\overline{\sigma^2_v}(f) = \frac{1}{N} \sum_{i=1}^{N} \sigma^2_{v,i}(f) \tag{4}$$

Equations 3 and 4 can analogously be applied to compute the sample variance of peak used BDD nodes for variable orders tested with particular event orders $\sigma^2_{e,j}(f)$, and the mean sample variance for fixed event orders $\overline{\sigma^2_e}(f)$. Likewise, we can define a function $g(o_{v,i}, o_{e,j})$ for the BDD operation count of a model and apply above computations to this.

When relating these characteristics to what we see in Figure 6, a low mean sample variance for fixed variable orders $\overline{\sigma^2_v}(f)$ would indicate a similar amount of peak used BDD nodes for a given variable order. This would be visible in Figure 6, as elements located in the same column would be similarly colored. This would indicate that the variable order mainly influences the peak used BDD nodes, and the event order has little influence.

For each model, the global sample mean μ_G, global sample variance σ^2_G, mean sample variance for fixed variable orders $\overline{\sigma^2_v}$ and mean sample variance for fixed event orders $\overline{\sigma^2_e}$ are given for peak used BDD nodes (f) and BDD operation count (g) in Table II. For most of the models, the mean sample variance for fixed variable orders is smaller than the mean sample variance for fixed event orders. This indicates that the variable order has a larger influence on the supervisor synthesis effort than the event order. However, the mean variance for fixed variable orders is large enough that the event order is still of considerable influence to the supervisor synthesis effort.

Models that require a relatively large amount of supervisor synthesis effort, also have a relatively large variance in effort. This would also be observed if we were to normalize the variance to the mean values of the models (σ^2/μ). This indicates that applying a good variable- and event order becomes more beneficial when considering models that require more supervisor synthesis effort.

V. Conclusions

We have introduced peak used BDD nodes and BDD operation count as metrics that express computational effort of BDD-based supervisor synthesis of EFA. Unlike wall clock time and peak random access memory, the BDD-based metrics are platform independent, deterministic and include no overhead in their measurements. It has been shown that worst case state space size is not an effective indicator for supervisor synthesis effort.

We have shown why and how variable- and event orders influence the supervisor synthesis effort. We performed supervisor synthesis using a large set of random variable- and event orders on 11 models of different sizes. This experiment has shown that the choice of variable- and event orders can influence the supervisor synthesis effort by multiple orders of magnitude. The variable order has a larger impact on the supervisor synthesis effort than the event order, but both influences are considerable. The influence of the variable- and event orders increases when considering models that require more supervisor synthesis effort.

Because the BDD-based metrics allow for effective indication of effort for small-scale models, the research can be continued by investigating whether conclusions made by applying certain modeling- or abstraction techniques on small-scale models convert well when implementing them on large-scale models. If this is the case, the BDD-metrics can be used on toy examples to investigate these techniques, instead of being constrained to using large-scale examples.

The current variable reordering algorithms leave some room for improvement. Different methods are to be researched. The metrics presented in the paper can be used for this, as it allows for more precise performance comparisons and comparisons on small-scale models. The same holds for finding some heuristic ordering algorithm for the events. The lower granularity orders as presented in this paper, i.e., model variable- and event order instead of Boolean variable- and edge order, can be used to more easily create an intuition on distinguishing good orders from bad orders.

The modeling tool Supremica has a BDD-based supervisor synthesis algorithm for EFA that is similar to CIF's. It can be researched whether the conclusions made in this paper,

based on experiments in CIF, also apply to Supremica. It would be interesting to know which of these tools is able to synthesize supervisors with less effort.

ACKNOWLEDGMENT

Research leading to these results has received funding from the EU ECSEL Joint Undertaking under grant agreement nº 826452 (project Arrowhead Tools) and from the partners national programs/funding authorities.

This research was partially supported by ESI (TNO), ASML Netherlands B.V., and the Netherlands Ministry of Economic Affairs, carried out as part of the TKI-projects 'Concerto' and 'Transposition'.

The authors thank Martijn Goorden for making the CIF case study models available that were used in the experiments presented by this paper.

REFERENCES

[1] P. Ramadge and W. Wonham, "Supervisory control of a class of discrete event processes," *SIAM J. Control*, vol. 25, no. 1, pp. 206–230, 1987.

[2] ——, "The control of discrete event systems," *Proc. IEEE*, vol. 77, no. 1, pp. 81–98, Jan 1989.

[3] M. Skoldstam, K. Åkesson, and M. Fabian, "Modeling of discrete event systems using finite automata with variables," in *46th IEEE Conf. Decision and Control*, Dec 2007, pp. 3387–3392.

[4] R. Theunissen, M. Petreczky, R. Schiffelers, D. van Beek, and J. Rooda, "Application of supervisory control synthesis to a patient support table of a magnetic resonance imaging scanner," *IEEE Trans. Autom. Sci. Eng.*, vol. 11, no. 1, pp. 20–32, Jan 2014.

[5] R. Loose, B. van der Sanden, M. Reniers, and R. Schiffelers, "Component-wise supervisory controller synthesis in a client/server architecture," *14th IFAC Workshop on Discrete Event Syst.*, vol. 51, no. 7, pp. 381 – 387, 2018.

[6] F. Reijnen, M. Goorden, J. van de Mortel-Fronczak, M. Reniers, and J. Rooda, "Application of dependency structure matrices and multilevel synthesis to a production line," in *IEEE Conf. Control Technol. Appl.*, Aug 2018, pp. 458–464.

[7] L. Swartjes, R. Su, and J. Rooda, "A case study on timed supervisory control on a linear cluster tool using aggregated timed synthesis," in *IEEE Int. Conf. Control Autom.*, Dec 2011, pp. 1189–1194.

[8] B. van der Sanden, M. Reniers, M. Geilen, T. Basten, J. Jacobs, J. Voeten, and R. Schiffelers, "Modular model-based supervisory controller design for wafer logistics in lithography machines," in *ACM/IEEE 18th Int. Conf. Model Driven Eng. Languages and Sys.*, Sept 2015, pp. 416–425.

[9] W. Wonham, K. Cai, and K. Rudie, "Supervisory control of discrete-event systems: A brief history," *Annu. Rev. Control*, vol. 45, pp. 250 – 256, 2018.

[10] S. Akers, "Binary decision diagrams," *IEEE Trans. Comput.*, vol. 27, no. 6, pp. 509–516, June 1978.

[11] C. Lee, "Representation of switching circuits by binary-decision programs," *The Bell System Tech. J.*, vol. 38, no. 4, pp. 985–999, July 1959.

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

[13] S. Miremadi, B. Lennartson, and K. Åkesson, "A BDD-based approach for modeling plant and supervisor by extended finite automata," *IEEE Trans. Control Syst. Technol.*, vol. 20, no. 6, pp. 1421–1435, Nov 2012.

[14] R. Malik, K. Åkesson, H. Flordal, and M. Fabian, "Supremica-an efficient tool for large-scale discrete event systems," *Proc. 20th IFAC World Congr.*, vol. 50, no. 1, pp. 5794 – 5799, 2017.

[15] A. Aziz, S. Taşiran, and R. Brayton, "BDD variable ordering for interacting finite state machines," in *Proc. 31st Annu. Des. Autom. Conf.*, 1994, pp. 283–288.

[16] M. Shoaei, L. Feng, and B. Lennartson, "Abstractions for nonblocking supervisory control of extended finite automata," in *IEEE Int. Conf. Autom. Sci. Eng*, Aug 2012, pp. 364–370.

[17] Z. Fei, S. Miremadi, K. Åkesson, and B. Lennartson, "Efficient symbolic supervisor synthesis for extended finite automata," *IEEE Trans. Control Syst. Technol.*, vol. 22, no. 6, pp. 2368–2375, Nov 2014.

[18] S. Miremadi, Z. Fei, K. Åkesson, and B. Lennartson, "Symbolic representation and computation of timed discrete-event systems," *IEEE Trans. Autom. Sci. Eng.*, vol. 11, no. 1, pp. 6–19, Jan 2014.

[19] L. Ouedraogo, R. Kumar, R. Malik, and K. Åkesson, "Nonblocking and safe control of discrete-event systems modeled as extended finite automata," *IEEE Trans. Autom. Sci. Eng.*, vol. 8, no. 3, pp. 560–569, July 2011.

[20] M. Reniers and J. van de Mortel-Fronczak, "An engineering perspective on model-based design of supervisors," *14th IFAC Workshop on Discrete Event Syst.*, vol. 51, no. 7, pp. 257 – 264, 2018.

[21] D. van Beek, W. Fokkink, D. Hendriks, A. Hofkamp, J. Markovski, J. van de Mortel-Fronczak, and M. Reniers, "CIF 3: Model-based engineering of supervisory controllers," *Tools and Algorithms for the Construction and Anal. of Syst.*, vol. 8413, pp. 575–580, 2014.

[22] F. Reijnen, M. Goorden, J. van de Mortel-Fronczak, and J. Rooda, "Supervisory control synthesis for a waterway lock," in *IEEE Conf. Control Technol. Appl.*, Aug 2017, pp. 1562–1563.

[23] T. Korssen, V. Dolk, J. van de Mortel-Fronczak, M. Reniers, and M. Heemels, "Systematic model-based design and implementation of supervisors for advanced driver assistance systems," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 2, pp. 533–544, Feb 2018.

[24] Y. Yang and R. Gohari, "Embedded supervisory control of discrete-event systems," in *IEEE Trans. Autom. Sci. Eng.*, Aug 2005, pp. 410–415.

[25] R. Bryant, "Symbolic boolean manipulation with ordered binary-decision diagrams," *ACM Comput. Surv.*, vol. 24, no. 3, pp. 293–318, Sept. 1992.

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

[27] F. Aloul, I. Markov, and K. Sakallah, "FORCE: A fast and easy-to-implement variable-ordering heuristic," *IEEE Great Lakes Symp. on VLSI*, May 2003.

[28] C. Meinel and T. Theobald, *Algorithms and Data Structures in VLSI Design*, 1st ed. Berlin, Heidelberg: Springer-Verlag, 1998.

[29] D. Knuth, "Big omicron and big omega and big theta," *ACM Sigact News*, vol. 8, no. 2, pp. 18–24, 1976.

[30] I. Wegener, "BDDs-design, analysis, complexity, and applications," *Discrete Appl. Math.*, vol. 138, no. 1-2, pp. 229–251, Mar. 2004.

[31] F. Somenzi, "Binary decision diagrams," *NATO Sci. Ser., Ser. III*, pp. 303–368, 1999.

[32] Y. Lopes, S. Trenkwalder, A. Leal, T. Dodd, and R. Groß, "Supervisory control theory applied to swarm robotics," *Swarm Intell.*, vol. 10, no. 1, pp. 65–97, Mar. 2016.

[33] W. Chao, Z. Tang, G. Lin, J. Guo, W. Lin, and S. Yu, "Modular supervisory control of computer based preventing electric mal-operation system for multiple bays in substation," in *IEEE Inform. Technol., Networking, Electron. and Autom. Control Conf.*, Dec. 2017, pp. 1730–1739.

[34] K. Cai and W. Wonham, "New results on supervisor localization, with case studies," *Discrete Event Dynamic Syst.*, vol. 25, no. 1-2, pp. 203–226, May 2014.

[35] G. Čengić and K. Åkesson, "A control software development method using IEC 61499 function blocks, simulation and formal verification," *IFAC Proc. Vol.*, vol. 41, no. 2, pp. 22–27, Jan. 2008.

[36] L. Feng, K. Cai, and W. Wonham, "A structural approach to the non-blocking supervisory control of discrete-event systems," *Int. J. Adv. Manuf. Technol.*, vol. 41, no. 11-12, pp. 1152–1168, Apr. 2009.

[37] D. Montgomery and G. Runger, *Applied Statistics and Probability for Engineers*. John Wiley and Sons, 2003.