DSM-based variable ordering heuristic for reduced computational effort of symbolic supervisor synthesis

Lousberg, S.A.J.

Award date:
2019

Department of Mechanical Engineering
Control Systems Technology

DSM-based variable ordering heuristic for reduced computational effort of symbolic supervisor synthesis

Master Thesis

Sam Anne Josco Lousberg
0805332

Supervisors:
dr.ir. Michel Reniers TU/e
Sander Thuijsman, MSc TU/e
dr.ir. Ramon Schiffelers ASML

Eindhoven, November 2019
CST2019.086

Abstract

In this research we consider the effects of the variable ordering in Binary Decision Diagrams (BDDs) on the computational effort required for symbolic supervisor synthesis. In recent research it has been shown that improving the variable ordering can result in a substantial further decrease of synthesis effort required compared to current ordering techniques. Therefore, we propose a novel variable ordering heuristic for the reduction of computational effort of BDD-based symbolic supervisor synthesis. By performing an analysis of the backwards reachability search, we show how variables can be ordered to reduce symbolic synthesis effort. By placing variables that often appear together in transition relations near each other, the effort can be reduced by orders of magnitude. We achieve this by utilizing a Dependency Structure Matrix (DSM) to store the number of times pairs of BDD-variables appear together in transitions. Subsequently, the DSM is manipulated by two matrix reordering heuristics, resulting in several feasible variable orderings. We utilize a metric that has been shown to be able to predict if variable orders perform well for effort reduction in other decision diagram-based applications, to choose which of the computed orders should be used for synthesis. We perform an experiment on a set of benchmark models to measure the achieved effort reduction of our proposed heuristic. Finally, we show that our approach is competitive in reducing computational effort compared to a state of practice variable ordering heuristic named FORCE, commonly applied to symbolic supervisor synthesis. Moreover, the best improvements in effort reduction are shown for the computationally most demanding models tested.

Preface

This is my master thesis "DSM-based variable ordering heuristic for reduced computational effort of symbolic supervisor synthesis". This thesis describes the research conducted at ASML from March to November 2019, as part of my graduation project for the master Mechanical Engineering at the Control Systems Technology sector of the University of Technology Eindhoven.

Hereby, I would like to thank my supervisors Michel, Sander and Ramon. Ramon, thank you for providing me with the opportunity to conduct this research at ASML, it has been a joy to get an insight in this company. Sander, I would like to thank you for always being able to make some time to discuss my research. Michel, thank you for the (bi-)weekly meetings at ASML, your enthusiasm about my research has motivated me immensely. I also would like to thank my colleagues at ASML for the pleasant and motivating work environment. Furthermore, I would like to thank my friends from Bacchus, my parents Rob and Karin, brother Kas and Marjolein for the everlasting support during my studies.

I hope you to enjoy the following pages.

Sam Lousberg

Contents

Chapter 1

Introduction

1.1 Background and related work

Supervisory Control Theory (SCT) [24] is a model-based approach to control Discrete Event Systems (DESs). In the framework of SCT we compute a supervisor that is safe (unsafe or undesirable states are not reachable), non-blocking (marked states are reachable), controllable (only controllable events are disabled) and maximally permissive (restrictions are minimal with regards to the three aforementioned criteria). The supervisor is computed by synthesis of plants (models of the uncontrolled system) with respect to requirements (specifications of allowed or desired behavior). In this report the plants and requirements are modeled as automata.

Despite the fact that SCT has been successfully applied to some examples of industrial size, e.g. in a magnetic resonance imaging scanner [34] and a waterway lock [27], industrial acceptance is yet scarce. This is mainly due to the computational complexity of synthesis for industrial sized models. The exponential state-space explosion that occurs is a limiting factor [38]. To partially overcome this, the plants and requirements can be represented by Extended Finite Automata (EFAs) that are symbolically expressed by Binary Decision Diagrams (BDDs) [22]. Synthesis can be directly applied to the symbolic representation of EFAs. Essential to utilizing BDDs is finding an efficient variable ordering. This ordering has a critical influence on the amount of computation time and computer memory required [21,32]. Computing the most efficient order is unfortunately NP-complete [4] and therefore heuristic algorithms are often used to find a decent ordering. This type of algorithm is designed to find an approximate solution to a problem when finding the exact solution is deemed to be computationally too expensive.

The importance of the variable ordering has been well described in literature. More specifically, in recent research [35] it has been proposed that improving the variable ordering can reduce the computational effort required for BDD-based supervisor synthesis of EFAs by orders of magnitude, even if FORCE [2] is applied, a state-of-the-art [20] variable ordering heuristic.

1.2 Report goal

This research proposes a new heuristic algorithm to find a variable ordering that results in a further reduction of computational effort required for supervisor synthesis compared to current implementations. More precisely, the plants and requirements are linearized, resulting in a single EFA. Subsequently, a variable ordering is computed and the linearized EFA is converted to BDDs. Synthesis is directly applied to the symbolic description of the model to result in a supervisor. Thereafter, the symbolic supervisor is converted to Disjunctive Normal Form/Conjunctive Normal Form (DNF/CNF), in which the representation in BDDs is transformed back to a comprehensible form, such that the user can better understand the computed supervisor. In these forms the supervisor is expressed in a predicate solely consisting of conjunctions, disjunctions and negations. The magnitude in which the variable ordering affects the computational effort required for symbolic

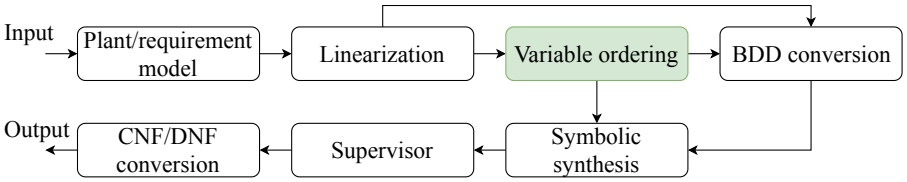


Figure 1.1: Flowchart of symbolic supervisor synthesis and the pre- and post-processing steps required.

supervisor synthesis is known from literature [35]. In this report we conduct further research to point out why the variable ordering influences the required computational effort for synthesis and how an ordering can be chosen such that the effort reduces. Therefore, the variable ordering is the main topic of this report. A flowchart of symbolic synthesis and its pre- and post-processing steps is shown in Figure 1.1.

1.3 Outline

The outline of this report is as follows: Chapter 2 elaborates about symbolic synthesis and preliminary knowledge the reader may be unfamiliar with. Subsequently, an analysis of the backwards reachability search is performed. This analysis and effect of the variable ordering on this essential part of synthesis is described in Chapter 3. This leads to the proposition of a new variable ordering heuristic for reduced computational effort, as described in Chapter 4. The proposed heuristic is applied to a set of benchmark models to compare its effectiveness with FORCE, this benchmark experiment is described in Chapter 5. The conclusions and recommendations for future work that result from this study are found in Chapter 6.

Chapter 2

Symbolic synthesis of Extended Finite Automata

In this research we present a heuristic to reduce the computational effort of symbolic supervisor synthesis. In the context of this report we perform synthesis of Extended Finite Automata (EFAs) represented by Binary Decision Diagrams (BDDs). This chapter elaborates on some preliminary knowledge about this type of synthesis, BDDs and related subjects.

2.1 Extended Finite Automata

The modeling framework of Extended Finite Automata (EFAs) [30] is a more compact representation to model DESs compared to regular Finite Automata (FAs). An EFA is found by appending discrete variables, guard expressions and update functions to FAs, defined as follows.

Definition 2.1: An EFA is a 7-tuple $(L, V, \Sigma, E, L_m, L_0, V_0)$ with the set of locations L, domain of variables $V = V^1 \times ... \times V^n$ where n is the number of discrete variables, set of events Σ, set of edges E, set of marked locations $L_m \subseteq L$, set of initial locations $L_0 \subseteq L$ and set of initial variable valuations $V_0 = V_0^1 \times ... \times V_0^n$. Each edge $e \in E$ is a 5-tuple $e = (o_e, t_e, \sigma_e, g_e, u_e)$ with the origin location $o_e \in L$, target location $t_e \in L$, event label $\sigma_e \in \Sigma$, guard $g_e : V \to \{true, false\}$ and finally the variable update function $u_e : V \to V$ [23].

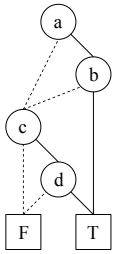
Events are divided into two types: controllable- and uncontrollable events. The first type can be disabled while the second type cannot be disabled by the supervisor. We denote the set of controllable events by $\Sigma_c \subseteq \Sigma$ and uncontrollable events by $\Sigma_u \subseteq \Sigma$, additionally $\Sigma_c \cap \Sigma_u = \emptyset$ and $\Sigma_c \cup \Sigma_u = \Sigma$. When referring to controllable or uncontrollable *edges*, we refer to whether $\sigma_e \in \Sigma$ in e is controllable or uncontrollable. Furthermore, we use the notation $P[u]$ for a predicate P where all occurrences of variables are replaced by the assignment of the update function u. As an example, let $P : x > 2$ and $u : x := x + 1$, such that $(x > 2)[x := x + 1]$ becomes $x > 1$. Moreover,

$$o_e \xrightarrow{g_e \ \sigma_e \ u_e} t_e$$

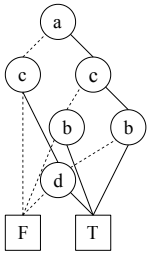
indicates an edge $e \in E$ from origin location o_e to the target location t_e with guard g_e, event σ_e and update function u_e. This edge is enabled only if the guard is evaluated to *true* for the origin location and current variable values. The variable valuations are updated thereafter according to u_e. An edge can only be taken if g_e is evaluated to true for the current variable valuations.

2.2 Binary Decision Diagrams

An EFA can be symbolically described by Boolean functions [22]. Any Boolean function $f : B \to \{true, false\}$, where B is the set of Boolean variables and $b \in B$ a single Boolean variable, can be



(a) Variable ordering $a < b < c < d$.



(b) Variable ordering $a < c < b < d$.

Figure 2.1: Two variable orderings for $f : (a \wedge b) \vee (c \wedge d)$.

expressed as

$$f = (b \wedge f|_{b=true}) \vee (\neg b \wedge f|_{b=false}), \tag{2.1}$$

where $f|_{b=true}$ denotes assigning *true* to b. By recursively applying above expression for all $b \in B$ a Binary Decision Diagram (BDD) [1] can be constructed to express f. BDDs are directed acyclic graphs that consist out of two types of nodes: decision- and terminal nodes. Each decision node is labeled by a Boolean variable $b \in B$ and has two edges leading to child nodes, one edge labeled *true* and the other *false*. When evaluating b to *true* or *false* we take aforementioned edge, respectively. Visually we represent a *true* (*false*) edge by a solid (dashed) line. At the leaves of the BDD are placed the terminal nodes, these can either be labeled by *true* or *false*.

The BDDs used for the synthesis of EFAs are reduced ordered BDDs [7]. As a result of some reduction rules these are minimal in the number of decision nodes and in canonical form for a given ordering. A total ordering over the set of Boolean variables is imposed, such that the reduced ordered BDD only has the same Boolean variables placed at each level. This ordering is referred to as the *variable ordering*. Moreover, there is only one terminal node for *true* and one for *false*. In this report we refer to reduced ordered BDDs as BDDs.

The variable ordering is denoted by $<$, where $b_1 < b_2$ indicates that decision node b_1 is placed closer to the root than decision node b_2. Furthermore, the ordering can have a major influence on the number of decision nodes that are required to represent the same function. The size of the BDD is defined by the number of decision nodes, in the following example we show the effect of the variable ordering on the size.

Example 2.1: Let the function $f : (a \wedge b) \vee (c \wedge d)$ be a logical expression, two BDDs corresponding to f are shown in Figure 2.1 for orderings $a < b < c < d$ and $a < c < b < d$. While both BDDs describe the same Boolean formula, one ordering requires four and the other six decision nodes. For BDDs describing more variables this effect is even more noticeable and in worst-case can lead to a size exponential in the number of Boolean variables [7].

2.2.1 BDD operations

The BDD size affects the amount of computer memory required. Furthermore, it also has a major effect on the computation time of logical manipulations applied to the BDD. Common operations are those of computing the conjunction (`And` operation) and the disjunction (`Or` operation) of two BDDs. These operations are both based on the recursive expansion following from Equation 2.1 of BDDs f and g with operation **op** for variable b

$$f \ \mathbf{op} \ g = [b \wedge (f|_{b=true} \ \mathbf{op} \ g|_{b=true})] \vee [\neg b \wedge (f|_{b=false} \ \mathbf{op} \ g|_{b=false})] .$$

The two sub-operations are recursively being expanded according to the formula shown above, starting from the top node(s), as imposed by the variable ordering. The operations are recursively applied until they lead to a terminal case [32]. Important to applying BDD operations is the use of the unique table guaranteeing their canonicity and a hash table, where recent results are stored and can be retrieved later. Thereby, it is prevented that often performed operations are repeatedly

Algorithm 1 Supervisor synthesis of Extended Finite Automata [23]

Input Plant P and requirement R modeled as EFAs
Output EFA G with restricted guards that is safe, non-blocking, controllable and maximally permissive w.r.t. P and R

1: Create refined G with the allowed behavior of P where the disallowed behavior of R results in the set of forbidden locations $L_f \subseteq L$
2: **while** Guards changing **do**
3: Set the initial non-blocking predicate N for marked locations $l \in L_m$ to *true* and for all other locations $l \notin L_m$ to *false*
4: **while** Non-blocking predicates changing **do**
5: Compute the new non-blocking predicate for every origin location o_e w.r.t. every outgoing edge to the target location t_e

$$N_{o_e} := N_{o_e} \vee \bigvee_{o_e \xrightarrow{g_e\ \sigma_e\ u_e} t_e} (g_e \wedge N_{t_e}[u_e])$$

6: **end while**
7: **if** Non-blocking predicate B not yet initialized **then**
8: Set the initial bad-state predicate for locations $l \in L_f$ to *true* and for all other locations $l \notin L_f$ to $B_l := \neg N_l$
9: **else**
10: Set the bad-state predicate for locations $l \in L_f$ to *true* and for all other locations $l \notin L_f$ to $B_l := \neg N_l \vee B_l$
11: **end if**
12: **while** Bad-state predicates changing **do**
13: Compute the new bad-state predicate for every origin location o_e w.r.t. every outgoing edge to the target location t_e where $\sigma_e \in \Sigma_u$

$$B_{o_e} := B_{o_e} \vee \bigvee_{o_e \xrightarrow{g_e\ \sigma_e\ u_e} t_e,\ \sigma_e \in \Sigma_u} (g_e \wedge B_{t_e}[u_e])$$

14: **end while**
15: Adapt the guards for all edges where $\sigma_e \in \Sigma_c$ w.r.t. to the bad-state predicate of target location t_e

$$g_e := g_e \wedge \neg B_{t_e}[u_e]$$

16: **end while**

being computed [32]. Applying operations to smaller BDDs results in fewer recursive operations and thereby to a reduction of computation time.

2.3 Compositional Interchange Format

In this report we utilize the Compositional Interchange Format (CIF) [36] for symbolic synthesis. CIF is an automata-based modeling language that allows for BDD-based synthesis of EFAs, based on the backwards reachability search [23] shown in Algorithm 1.

In CIF, there are two types of variables: the first type is used as a location pointer for automata and the second type to describe the value of a discrete variable. Both variables are described by BDD-variables during synthesis. The BDD-variables as used in CIF are separated into two types: current-state $x \in X$ and next-state $x^+ \in X^+$ variables. A more thorough elaboration follows in Section 3.1. Each BDD-variable only belongs to a single automaton or discrete variable. In this report we group all BDD-variables that belong to a single automaton or discrete variable and refer

to these groups of BDD-variables as CIF-variables. Furthermore, in the ordering of BDD-variables we do not interleave [3] BDD-variables belonging to separate CIF-variables. Moreover, each next-state BDD-variable is always placed adjacent its current-state variable in the order. For automata with only one location, no location pointer is required and therefore no CIF- and BDD-variable is utilized for these.

By default, CIF automatically orders the CIF-variables alphabetically by the name of the automaton or discrete variable. Subsequently, the variable ordering heuristic FORCE [2] is utilized to find a more efficient order. FORCE minimizes a so called *span* by trying to place highly dependent variables near each other. Afterwards, a window is slid over the order produced by FORCE where the variables within the window are reordered locally according to the same placement criteria. In this report we refer to applying FORCE as applying both reorderings sequentially.

Moreover, the modelling tool Supremica [18] allows for BDD-based synthesis of EFAs comparable to CIF's implementation. Supremica also utilizes FORCE as variable ordering heuristic. However, Supremica is not regarded in this report.

2.4 Metrics for computational effort

In this report the computational effort required for synthesis is expressed by two BDD-based metrics: peak used BDD nodes and total operation count [35]. The first metric is the peak size of all BDDs combined during synthesis. As computer memory is always finite, this is the main limiting factor for successful synthesis. The latter is the number of times a recursive call is made to any BDD operation and mainly expresses the computation time of synthesis. These metrics allow to measure the computational effort required in a deterministic, platform-independent way and include no overhead in their measurements, opposed to more traditional metrics such as computer memory usage and wall-clock time. Performing synthesis with constant parameter settings results in the same measurement each time for these BDD-based metrics.

Chapter 3

Analysis of the backwards reachability search

In this chapter we analyze the symbolic computations of non-blocking and bad-state predicates. These predicates are required during the backwards reachability search based on Algorithm 1 and take a considerable amount of computational resources during symbolic synthesis. By an analysis of the computations of these predicates we show how the variable ordering affects the BDD size and by what extent this contributes to the total computational effort of synthesis.

3.1 The backwards reachability search

The non-blocking N and bad-state B predicates are essential during symbolic synthesis. The non-blocking predicate expresses what locations and discrete variable values can be reached backwards from the marked states, with respect to the current guards and updates. The bad-state predicate expresses what locations and discrete variable values lead to undesirable states [23]. By continuously adapting these predicates based on possible transitions in the model a stage is reached where these predicates do not change anymore. Subsequently, the initial guards are restricted based on the final non-blocking predicate to result in the supervisor.

To be more precise, synthesis is applied to a single EFA described by BDDs, where each transition is determined by an edge $e \in E$. This single EFA results from the linearization of a set of automata, a more thorough elaboration about this follows in Section 3.3. Edges are applied in a backwards manner such that we start at the target location t_e towards the origin location o_e. We utilize current-state variables $x \in X$ and next-state variables $x^+ \in X^+$, to allow BDDs to express predicates before and after a transition, respectively. Therefore, the BDD of each update $u_e(X, X^+)$ is expressed in both current- and next-state variables (the update adapts the state after a transition based on the state before a transition). The BDD of each guard $g_e(X)$ is only expressed in current-state variables (the evaluation of the guard only depends on the state before the transition).

Symbolic synthesis of EFAs in CIF is based on Algorithm 1. However, it slightly differs from the original algorithm as the states are symbolically encoded by BDDs. The main differences are that there is only one non-blocking and one bad-state predicate combining all states. Instead of a predicate of forbidden states, a predicate of safe states $P_s(X)$ is used, that allows all states of the plants to be reached, except the forbidden states as imposed by the requirements, see [23] for a more thorough elaboration. Controllable edges that lead from forbidden to safe states are disallowed by restriction of their guards. Uncontrollable edges lead to a restriction of P_s if the edge is enabled in the plants but disabled by the requirements.

The backwards reachability search commences from the initial non-blocking predicate $N_0(X)$, this is the marked state predicate $N_m(X)$ set to *true* for every marked state and *false* for all other states, conjoined to $P_s(X)$, such that the search commences from $N_0(X) = N_m(X) \wedge P_s(X)$.

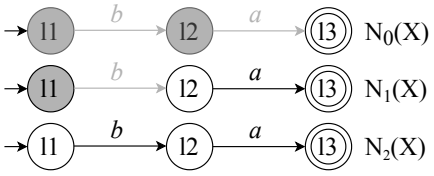


Figure 3.1: Non-blocking predicate during the backwards reachability search for first applying edge a and second edge b. Grey states are not (yet) part of the corresponding predicate as the grey edges leading from those states have not (yet) been searched backwards.

Next, a copy is made of $N_0(X)$ where all $x \in X$ are replaced by their $x^+ \in X^+$, resulting in $N_0(X^+)$. The conjunction of the guard and update is called the transition relation $T_e(X, X^+) = g_e(X) \wedge u_e(X, X^+)$ and is expressed in both X and X^+ for aforementioned reasons. The non-blocking predicate backwards-reachable in at most one step $N_1(X)$ [8] follows

$$N_1(X) = N_0(X) \vee \exists_{X^+} \left[N_0(X^+) \wedge T_e(X, X^+) \right].$$

Subsequently, repeatedly apply above equation for k steps

$$N_k(X) = N_{k-1}(X) \vee \exists_{X^+} \left[N_{k-1}(X^+) \wedge T_e(X, X^+) \right], \tag{3.1}$$

until the fixed-point $N_k(X) = N_{k-1}(X)$ is reached. This results in the states backwards-reachable from the marked state, an example is shown in Figure 3.1. For the bad-state predicate we perform a similar search, however, in this case only uncontrollable edges are regarded. We set the BDD of the initial bad-state predicate to $B_0(X) = \neg N_k(X)$ and perform a similar sequence of computations until we find the fixed-point $B_j(X) = B_{j-1}(X)$, using

$$B_j(X) = B_{j-1}(X) \vee \exists_{X^+} \left[B_{j-1}(X^+) \wedge T_e(X, X^+) \right], \tag{3.2}$$

by only applying $T_e(X, X^+)$ corresponding to uncontrollable edges. Next, the initial non-blocking predicate is set equal to $N_0(X) = N_m(X) \wedge \neg B_j(X)$ and we repeat from the first step at least once. If the bad-state predicate is unchanged after the new iteration, the backwards reachability search is finished and the predicate of the controlled system $P_c(X)$ is set equal to the most recently computed $N_k(X)$, otherwise repeat. Finally, it is verified whether the initial state is still present in the predicate and the supervisor is computed by strengthening the guards of all controllable edges using

$$g_e(X) := g_e(X) \wedge \exists_{X^+} \left[P_c(X^+) \wedge T_e(X, X^+) \right].$$

3.2 The relational product

Next to the `And` and `Or` operations, the existential quantification is an operation that can be applied to a BDD [7]. Given a BDD f and variable x, the quantification of x out of f results from recursively applying

$$\exists_x f = f|_{x=true} \vee f|_{x=false},$$

starting from the top variable as imposed by the variable ordering. This leads to a frequently applied operation during the backwards reachability search

$$\exists_v \left[f \wedge g \right],$$

where f and g are both BDDs and v the set of variables we want to existentially quantify. This operation can be executed by first computing the `And` between f and g and later quantifying over v. However, this results in a large intermediate result of $f \wedge g$. Therefore, we compute both the conjunction and existential quantification in a single recursive pass over f and g by utilizing the relational product [8] operation. The idea behind this operation is to avoid computing the entire

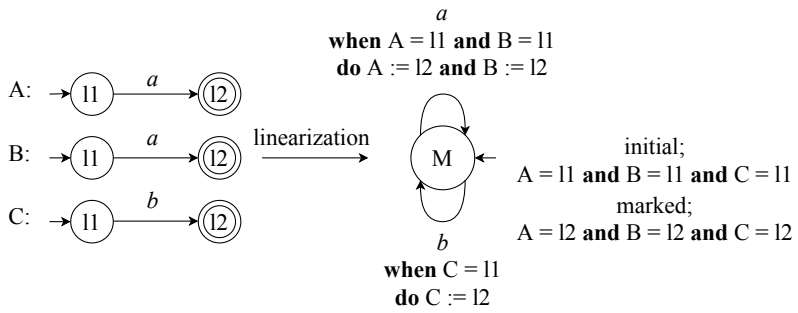


Figure 3.2: Linearization of a set of automata. The initial and marked predicates are given under *initial;* and *marked;*, respectively.

BDD $f \wedge g$, quantifying early over v and thereby reducing memory usage and number of required operations. The worst-case complexity [19] of the relational product is

$$\mathcal{O}(|f| \times |g| \times 2^{2|b|}), \tag{3.3}$$

where $|f|$ and $|g|$ are the sizes of the BDDs and $|b|$ the number of total BDD-variables. Nonetheless, computing the relational product is known to be an expensive computation during the backwards reachability search [8].

3.3 Linearization of automata

In CIF, synthesis is applied to a linearization of plant and requirement automata, which is only thereafter converted to BDDs. For linearization[1], a single EFA M is introduced with the same alphabet as the union of all alphabets of original automata. For each automaton a discrete pointer variable is generated to enumerate the locations. Next, all edges are assigned to automaton M as self-loops where the update is used to assign the target location t_e that is reached when the edge is taken and the guards are evaluated *true* for origin location o_e, for which the edge can be taken. Discrete variable updates of the original automata are conjoined to the updates of the linearized EFA. Finally, M is converted to BDDs and symbolic synthesis is directly applied to this single EFA. See the following example where a set of automata is linearized.

Example 3.1: Let A, B and C be three automata with each two locations denoted by *l1* and *l2* and a single edge leading from the initial state to the marked state. The edges of automata A and B have a synchronized event a and automaton C event b. After applying the linearization, we find automaton M with an initial and marked state predicate described by location pointers and two self-loops with a guard (when) and update (do), see Figure 3.2 for the linearized automaton M. Note that even though A, B and C are regular FAs in this example, the resulting M is an EFA.

3.4 Propagation of transition relations

Notice that in Equations (3.1) and (3.2) each computation depends on the previously computed result. As the number of BDD-variables $|b|$ and the size of the transition relation for each edge is equal throughout synthesis, we know that the BDD that mainly determines the worst-case complexity of the following relational product operation is the previously computed non-blocking or bad-state predicate, recall Equation (3.3). Thus, if $N_{k-1}(X)$ is small, the effort required for the computation of $N_k(X)$ is low. Furthermore, note that during every step of the computation of the non-blocking predicate we first compute the following predicate

$$\exists_{X^+} \left[N_{k-1}(X^+) \wedge T_e(X, X^+) \right].$$

[1]CIF documentation about linearize-product is available at: `http://cif.se.wtb.tue.nl/tools/cif2cif/linearize-product.html`

The same applies to the bad-state predicate

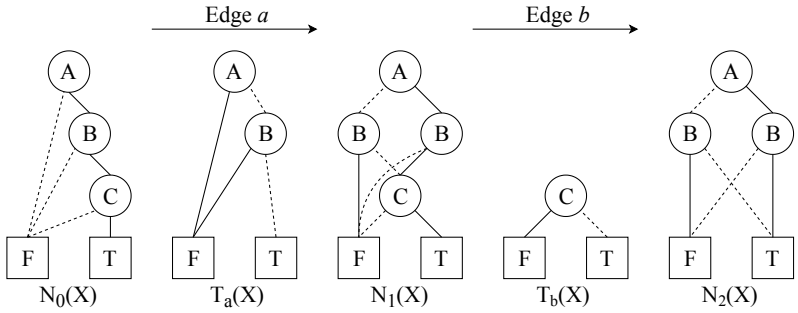
$$\exists_{X^+} \left[B_{j-1}(X^+) \wedge T_e(X, X^+) \right].$$

The results of these equations only consist out of current-state variables, as the next-state variables are existentially quantified. In a sense, we add assignments that are imposed by the current-state variables of the transition relation. Furthermore, we know that the variables in each transition relation are strongly related in the Boolean functions the BDDs represent. Moreover, BDDs are overall small if strongly related BDD-variables are placed near each other [21, 32]. An example of this effect is shown in previous chapter in Figure 2.1, where keeping BDD-variables a and b as well as c and d near each other results in a smaller BDD.

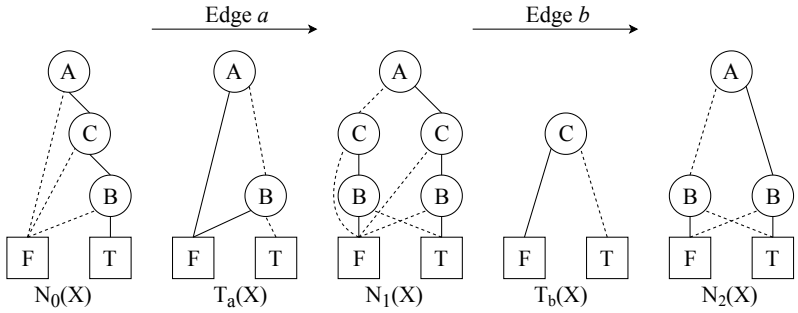
To summarize our observation, if we keep all current-state variables of each transition relation near each other, it is likely that the resulting non-blocking and bad-state predicates are small, as we know that these variables are strongly related. Placing these variables near each other in the ordering should result in less synthesis effort. In the following section we show an example of this effect on the BDD size of the non-blocking and bad-state predicates and confirm with an experiment that this indeed is the case.

3.5 Relation between the transition relations, variable ordering and the computational effort required

We observe that variable orderings that result in low computational effort are ordered such that BDD-variables appearing together in transition relations are placed near each other, opposed to far apart for orderings resulting in high computational effort. This observation can best be described by the following example where we utilize the linearized model from Example 3.1.



(a) Variable ordering $A < B < C$.



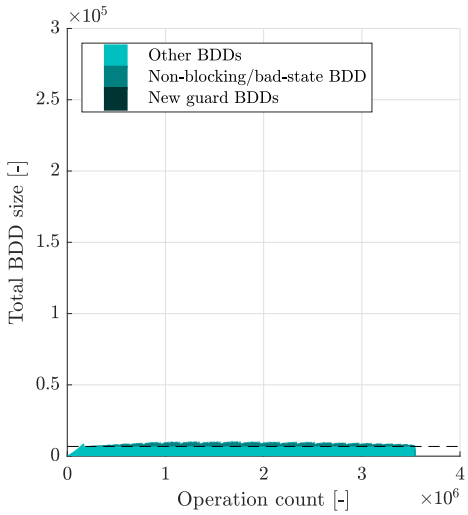
(b) Variable ordering $A < C < B$.

Figure 3.3: BDDs of $N(X)$ and $T(X)$ for two different variable orderings during the first two steps of computing the non-blocking predicate for the linearized automaton M shown in Figure 3.2.

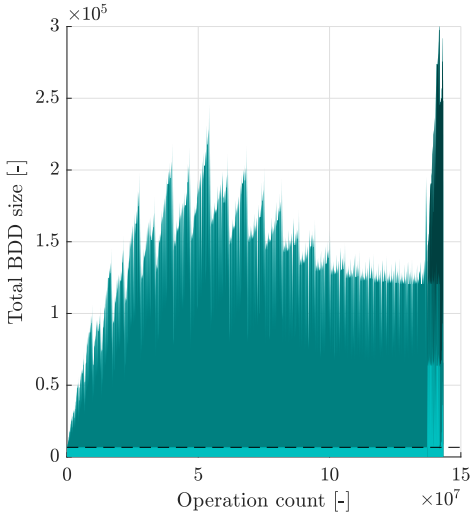
Example 3.2: Figure 3.3 shows the first two steps of computing the non-blocking predicate for variable orderings $A < B < C$ and $A < C < B$ for the linearized automaton shown in Figure 3.2. To encode automaton M we require three BDD-variables: A, B and C. For each variable it holds that taking the *false* path relates to *l1* and the *true* path to *l2*. After applying edge a in reverse, it results in a non-blocking predicate BDD of four decision nodes for the first, but five decision nodes for the second order. Next, edge b is applied in reverse resulting in the final non-blocking predicate of this synthesis, for both orders resulting in a non-blocking predicate of three decision nodes.

Strictly keeping A and B near each other in the ordering results in less peak used BDD nodes as the intermediate non-blocking predicate $N_1(X)$ is smaller. To further analyze this behavior we execute the same experiment for a model of relevant complexity. We measure the size of all BDDs combined, non-blocking/bad-state predicates and new guard predicates during synthesis. Two different variable orderings are chosen: one such that variables appearing in transition relations are placed near each other and the other such that variables appearing in transition relations are placed apart from each other. In Figure 3.4 the results of this experiment are shown for the Cluster tool model [33]. We notice that the total size of the BDDs prior to the backwards reachability search is relatively equal for both orderings. For the first ordering the size of the non-blocking/bad-state predicate only becomes slightly larger during synthesis, for the second ordering this predicate becomes substantially larger, more so, during the computation of the new guards the total BDD size increases even more. For the first ordering this effect is hardly noticeable. Naturally, applying operations to larger BDDs also results in more operations as this requires more recursive calls.

As previous results show, we conclude that the non-blocking and bad-state predicates are of major importance in synthesis. The predicates of newly computed guards increase as well, however, this is a result of the size of the final predicate of the controlled system, as can be seen in Equation (3.1). All other BDD predicates do hardly differ in size depending on the chosen variable ordering during the major part of synthesis. This can be explained as all other BDDs describe relatively few Boolean variables. Recall that the worst-case size of a BDD is exponential in the number of Boolean variables and therefore, the variable ordering has less effect on the total



(a) Total BDD size and operation count for a variable ordering where variables appearing in the same transition relations are placed near each other.



(b) Total BDD size and operation count for a variable ordering where variables appearing in the same transition relations are placed apart from each other.

Figure 3.4: The total BDD size and operation count for two variable orderings applied to the Cluster tool model. The dashed line indicates the size of the BDDs prior to the backwards-reachability search. Note that in (a) this is $6.8 \cdot 10^3$ nodes and in (b) $6.7 \cdot 10^3$ nodes.

size of these BDDs.

To conclude, we have shown that placing variables that appear in the same transition relation near each other can give an enormous improvement to the computational effort required. In next chapter we show how to find these groups of variables and propose a heuristic approach to reorder the variable ordering such that these variables are placed near each other in the order.

Chapter 4

Variable ordering heuristic

The previous chapter shows how placing the variables within the ordering influences the size of the intermediate predicates and that this placement has a substantial effect on the computational effort required. It is shown that variables appearing together in transition relations should be kept near each other in the ordering, to reduce the computational effort required.

In this chapter we introduce a novel variable ordering heuristic to achieve the aforementioned ordering. This heuristic requires a Dependency Structure Matrix (DSM) to store pairs of CIF-variables that often appear together in transition relations. Subsequently, the DSM is manipulated utilizing two matrix reordering heuristics, resulting in several viable orders. By utilizing the Weighted Event Span (WES) metric we estimate which ordering should be applied to symbolic synthesis to result in the most effort reduction. To efficiently compute the WES, we introduce the Transition Relation Matrix (TRM) to store current-state BDD-variables appearing in each transition relation.

4.1 Dependency Structure Matrix

A Dependency Structure Matrix (DSM) is an $n \times n$ matrix utilized to represent dependencies between n aspects of a system or model [5, 6]. The aspects of a system are captured along the rows and columns, both ordered as $(1, ..., n)$, where each index represents a unique aspect of the system. The diagonal elements are always equal to zero. The values of the off-diagonal elements indicate whether there is a dependency between the aspects that the row and column indices represent.

There are two main types of DSMs: binary DSMs and Numerical DSMs (NDSMs). The first have off-diagonal elements that can either be valued 0 (no dependency) or 1 (a dependency). The latter can have off-diagonal elements valued either 0 or a positive integer, where a higher value indicates a stronger dependency in relation to lower valued elements. Furthermore, aforementioned types of DSMs can either be static or dynamic. The first indicates that all dependencies are regarded as undirected and accordingly this DSM is always symmetric. The second has directed dependencies and therefore can be asymmetric.

The framework of DSMs is useful for analyzing and manipulating the structure of a system or model. For this purpose many algorithms exist to cluster (find clusters of related aspects) or sequence (remove feedback marks) DSMs, see [6] for an overview. However, in this report no conventional DSM manipulation algorithms are used. We utilize static NDSMs to store which pairs of CIF-variables often appear together in transition relations. Subsequently, the NDSM is manipulated by two matrix ordering heuristics to result in a variable ordering where CIF-variables of aforementioned pairs are placed near each other in the order. The construction of the NDSM is shown in Section 4.3 and the manipulation of the constructed NDSM in Section 4.4.

4.2 Transition Relation Matrix

In this section the Transition Relation Matrix (TRM) is introduced. This is a matrix with $|E|$ rows and $|x|$ columns, where $|E|$ indicates the number of edges in a model and $|x|$ the number of current-state BDD-variables. The row index represents the transition relation of $e \in E$ and the column index the current-state variables $x \in X$ that appear in the transition relation. This is indicated by the elements of each row as 0 (not appearing) or 1 (appearing). Each BDD-variable is counted only once per edge, as the number of decision nodes in a BDD depends on the variable ordering. The construction of the TRM is shown in the next section. We utilize the TRM to compute the Weighted Event Span (WES) for a given variable ordering, shown in Section 4.5.

4.3 Constructing the NDSM and TRM

To compute the NDSM and TRM we require the transition relation of current-state variables $T_e(X)$ for each $e \in E$. For each $T_e(X, X^+)$ we can find $T_e(X)$ by existential quantification of the set of next-state variables

$$T_e(X) = \exists_{X^+} T_e(X, X^+).$$

The reasoning behind this is that we are only interested in the current-state variables that remain after the existential quantification of $T_e(X, X^+)$. These variables are the only ones that remain in the non-blocking and bad-state predicate during the backwards reachability search.

Subsequently, we extract all BDD-variables that appear in $T_e(X)$ for each edge and extract the related CIF-variables. We denote by $T_e(X) = \{b_1, b_2\}$ that current-state BDD-variables b_1 and b_2 appear in $T_e(X)$. To describe which BDD-variables belong to a CIF-variable a similar notation is used.

For each row of the TRM depicting $T_e(X)$ an element is set to 1 for each BDD-variable that appears, otherwise to 0. In the NDSM we increment an off-diagonal entry by 1 for each time the corresponding pair of CIF-variables appears together in a transition relation, this is done in a way that the NDSM remains symmetric. We construct the NDSM and TRM together, as these are both constructed based on the transition relations. Therefore, it is more efficient to construct these at the same time. However, the NDSM stores pairs of CIF-variables that appear together in transition relations and the TRM the BDD-variables that appear in transition relations. See the following example.

Example 4.1: We apply synthesis to a model with CIF-variables A to C and current-state BDD-variables b_1 to b_4 where $A = \{b_1\}$, $B = \{b_2, b_3\}$ and $C = \{b_4\}$. The model has two edges $e_1, e_2 \in E$ and corresponding transition relations T_1 and T_2, furthermore $T_1(X) = \{b_1, b_2, b_3\}$ and $T_2(X) = \{b_2, b_3, b_4\}$. We start with an initial empty

$$\text{NDSM}_0 = \begin{array}{c} \begin{matrix} A & B & C \end{matrix} \\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{array} \begin{matrix} A \\ B \\ C \end{matrix}$$

and initial empty

$$\text{TRM}_0 = \begin{array}{c} \begin{matrix} b_1 & b_2 & b_3 & b_4 \end{matrix} \\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \begin{matrix} T_1 \\ T_2 \end{matrix} .$$

Subsequently the transition relations are used to manipulate both matrices. In this example we first apply $T_1(X)$ and next $T_2(X)$, however, note that the order in which the transition relations are applied does not matter for the final result. First, we manipulate the matrices based on $T_1(X)$. This results in an increment of 1 to elements (A, B) and (B, A) in the NDSM. Thus, the NDSM

results in

$$\text{NDSM}_1 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

Subsequently, elements (T_1, b_1), (T_1, b_2) and (T_1, b_3) in the TRM are set to 1

$$\text{TRM}_1 = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Subsequently, by repeating the same matrix manipulations, only now for $T_2(X)$ we result in the next and final

$$\text{NDSM}_2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

and

$$\text{TRM}_2 = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix}.$$

Applying these steps for each transition relation of a model results in an NDSM where the value of each non-diagonal element indicates the number of times the represented pair of CIF-variables appears together in a transition relation. The TRM describes all BDD-variables that appear in each $T_e(X)$, this matrix is utilized in Section 4.5 to efficiently compute the WES for a given variable ordering. The use of DSMs in SCT is not novel, in [16] clustering is applied to a DSM containing the dependencies between plants and requirements, applied to multilevel synthesis. For our application however, the NDSM is reordered based on the values of each element. By placing higher valued elements near the diagonal and zero values far away from the diagonal, we can find a variable ordering where groups of variables that often appear together in transition relations are placed near each other.

To achieve this matrix reordering we utilize existing so called node ordering heuristics. These heuristics have been designed for bandwidth, profile and/or wavefront reduction of symmetric sparse matrices, for an elaboration on these metrics we refer the reader to [12] for bandwidth and [31] for profile and wavefront. We do not directly utilize these metrics, however, minimizing one (or more) of the three aforementioned metrics results in a matrix where non-zero elements are placed near the diagonal. Note that the NDSMs in our approach are also sparse symmetric matrices. Moreover, the effective use of these heuristics for static variable ordering optimization for decision diagrams is shown in [20], where several node ordering heuristics have been compared on their effectiveness in effort reduction for symbolic reachability analysis. We analyze two node ordering heuristics and compare the effectiveness of orderings computed using these.

4.4 Weighted adjacency graph and node reordering

The node ordering heuristics are applied to the weighted adjacency graph of the NDSM. For an NDSM with row index i, column index j and size n we denote elements by $e_{i,j}$. For each row i we generate a node labeled by i. Subsequently, each non-zero element $e_{i,j}$ results in an undirected edge with weight $e_{i,j}$ between nodes i and j. This results in a weighted adjacency graph where the node labels are reordered using the heuristic approaches described in Sections 4.4.1 and 4.4.2. Some notations and definitions that are of importance regarding weighted adjacency graphs are the following: order R is a reordering of the list of initial node labels $L = (1, ..., n)$ such that the index of nodes in the initial order is replaced by the integer value of the same index in R. For a matrix this is simply reordering both the column and row indices in a similar manner. We denote the reordering of a list L by R as $L(R)$ and of a matrix M by R as $M(R, R)$. The degree of a node is the number of direct neighbors. A peripheral node is a node whose shortest path to the node furthest away is of maximal length. See the following clarifying example.

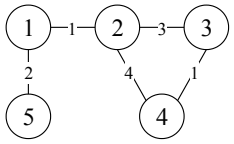


Figure 4.1: Weighted adjacency graph of Equation (4.1).

Example 4.1: The weighted adjacency graph of matrix

$$M = \begin{bmatrix} 0 & 1 & 0 & 0 & 2 \\ 1 & 0 & 3 & 4 & 0 \\ 0 & 3 & 0 & 1 & 0 \\ 0 & 4 & 1 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.1}$$

is shown in Figure 4.1. Node 2 has the maximum degree of 3, and nodes 3, 4 and 5 are peripheral nodes as each of these nodes has a shortest path of length 3 to the node furthest away.

4.4.1 Cuthill-McKee ordering

The Cuthill-McKee (CM) ordering [12] is a bandwidth reducing heuristic algorithm for the reordering of sparse symmetric matrices. The basics of the standard CM ordering are as follows: to begin with, the NDSM is converted to a weighted adjacency graph. Subsequently, a starting node must be chosen, the choice of this starting node is of great importance for the results of the CM ordering. A peripheral node is usually a good choice, however, finding a true peripheral node is computationally expensive for large graphs. A pseudo-peripheral node finder can find an approximate solution p that is sufficient for this application [15]. Subsequently, all direct neighbors of the initial node p are marked and appended in ascending degree to (an initial empty) list R. Afterwards, the next node in R is selected as p and repeatedly all unmarked neighbors of p are appended to R in ascending order of degree and marked. This breadth first search is recursively applied until all nodes are marked, the resulting list R is the resulting order.

We implemented the CM ordering based on the queue implementation [28] shown in Algorithm 2. Furthermore, we notice that with a small adjustment of the standard algorithm we allow a bias for appending nodes with a higher weight first. With the weight of a node we mean the weight of the edge that connects the parent node p to this (child) node in list C in the weighted adjacency graph. Instead of appending nodes to R based on degree, we append based on descending weight. However, if two or more nodes to be appended have equal weight, we first append nodes in ascending degree. This adjustment is shown in lines 9 - 11 in Algorithm 2. Nodes that are appended earlier during each expansion over neighbors have a bias to be placed closer to the diagonal compared to nodes visited afterwards. Despite losing some of the bandwidth reducing characteristics of the CM ordering, we found that this adjustment gave improved results for the intended use. Furthermore, it might occur that unconnected CIF-variables appear, these result in a node without any neighbors in the adjacency graph. These are excluded from the reordering as applied by the node ordering heuristics and are always placed at the front of the ordering, for reasons that are given in Section 4.5.

4.4.2 Sloan's ordering

Sloan's ordering [31] is a profile and wavefront reducing heuristic for symmetric sparse matrices. This heuristic reorders the adjacency graph starting at two pseudo-peripheral nodes. However, unlike the CM ordering, no adjustment is made to Sloan's ordering such that it can sort numeric DSMs, as this is out of the scope of this research. Nevertheless, this might be of interest for

Algorithm 2 Weighted Cuthill-McKee ordering

Input NDSM M
Output Ordering R

1: Initialize empty list R, compute weighted adjacency graph A of M and initialize list of unconnected nodes E in any order
2: Compute pseudo-peripheral node p of A
3: Mark p and append p to R
4: **while** Unmarked nodes exist in A **do**
5: Find list of unmarked neighbors C of p and sort C in descending weight
6: Sort nodes in C with equal weight in ascending degree
7: Append C to R and mark all nodes in C
8: Set the next node in R as p
9: **end while**
10: Set R equal to R appended to E

future work. Sloan's ordering is implemented based on a Matlab implementation[2] of the original FORTRAN code in [31]. Sloan's ordering reorders the NDSM based on zero and non-zero values.

4.5 Weighted Event Span

Besides two heuristic algorithms, reversing the computed order shows (sometimes significant) varying results for the computational effort required. As we want to find an efficient variable ordering in advance of synthesis, we use the Weighted Event Span (WES) [29]. Siminiceanu and Ciardo noticed that placing variables of more costly operations to the rear of the ordering (lower in the BDD) improved results for a symbolic reachability search. Furthermore, the WES has extensively been benchmarked in [20] which showed that the WES is a valuable metric regarding the reduction of peak decision diagram nodes for symbolic methods utilizing several types of decision diagrams. To compute the WES for a given ordering we utilize the computed TRM. We reorder the row corresponding to each $e \in E$ of the TRM according to ordering R, as the resulting value depends on the ordering applied. Subsequently, we compute the WES by

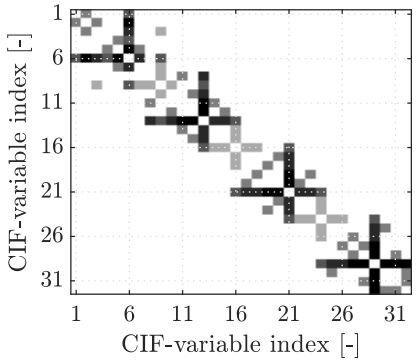
$$\text{WES} = \sum_{e \in E} \frac{2x_l}{|x|} \cdot \frac{x_l - x_f + 1}{|x||E|}, \tag{4.2}$$

where $|x|$ is the number of current-state BDD-variables, $|E|$ the number of edges (equal to the number of rows in the TRM) and x_f (x_l) the index of the first (last) non-zero element in row e of the TRM reordered according to R [29]. Note that the first term in the WES results in a lower score when the lowest BDD-node that appears in an edge is lower and vice versa. The second term results in a lower score when the first (highest) and last (lowest) BDD-variables are placed closer near each other after ordering R is applied.

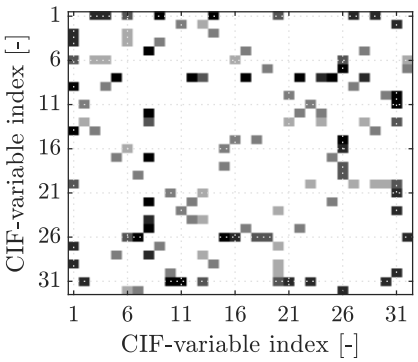
4.6 Variable ordering heuristic

The aforementioned ordering heuristics and WES result in the proposed heuristic algorithm. For ease of reference we name this heuristic DSM-based Cuthill-McKee-Sloan variable ordering Heuristic (DCSH). For a given NDSM we compute both the weighted CM ordering R_{CM} and Sloan's ordering R_S. As reversing the order can have a noticeable impact on the computational effort, we also reverse both orders denoted by R^r_{CM} and R^r_S, respectively. When reversing the order we make sure that if any unconnected nodes in the adjacency graph appear, they are always placed

[2]Sloan's ordering Matlab implementation is found at: `https://mathworks.com/matlabcentral/fileexchange/71934-reduceprofile`



(a) NDSM of the Cluster tool model that resulted in small non-blocking and bad-state predicates.



(b) NDSM of the Cluster tool model that resulted in large non-blocking and bad-state predicates.

Figure 4.2: Two NDSMs for an efficient (a) and inefficient (b) ordering for the Cluster tool model.

Algorithm 3 DSM-based Cuthill-McKee-Sloan variable ordering Heuristic (DCSH)

Input Transition relations $T(X)$ for all $e \in E$
Output Ordering R

1: Initialize empty TRM and NDSM
2: **for** all $e \in E$ **do**
3: Find all BDD-variables in $T_e(X)$ and add these to the TRM
4: Find all CIF-variables in $T_e(X)$ and add these to the NDSM
5: **end for**
6: Compute weighted adjacency graph A of the NDSM
7: Compute weighted CM ordering R_{CM} and reversed order R^r_{CM}
8: Compute Sloan's ordering R_S and reversed order R^r_S
9: Compute WES for all four orderings using the TRM
10: Set R to the ordering with minimal WES

at the front of the ordering, as this reduces the WES. Subsequently, we compute the WES for all four orderings and choose the ordering that results in minimal WES.

A summary of DCSH is shown in Algorithm 3. Computing two different orderings and WES for four orderings can give an intricate impression, despite, the computation time required is marginal and the possible improvements on the computational effort of synthesis is received to be significant in some cases. For instance, executing the code for Algorithm 3 for the FESTO model [25], which is the largest model encountered in this report based on worst-case peak used BDD nodes, has a computation time of < 400 ms while the reduction of effort for synthesis is considerably larger. Further optimization of DCSH's implementation is expected to even reduce the computation time. In Figure 4.2 the NDSMs are shown corresponding to the orderings used for the experiment of which the resulting predicate sizes are shown in Figure 3.4. It can clearly be seen that the ordering that resulted in small predicate sizes has all CIF-variables near the diagonal. For the order that resulted in large predicate sizes we can see the opposite.

Note that the computed order R is used to order CIF-variables. We always keep the BDD-variables belonging to CIF-variables adjacent to each other in the ordering. It is likely that BDD-variables occurring in the same CIF-variables appear often together in transition relations and therefore keeping these near each other is often effective. In the next chapter we analyze DCSH and compare it to FORCE in a benchmark experiment based on the computational effort that is required for synthesis when using the heuristics.

Chapter 5

Benchmark experiment

We perform a benchmark experiment applied to a set of CIF-models to measure DCSH's effectiveness in reduction of computational effort required for synthesis. The computational effort is expressed in peak used BDD nodes and total operation count (recall Section 2.4), denoted by PBN an TOC for ease of reference, respectively. Moreover, a comparison of DCSH against FORCE is made based on their ability in effort reduction. The models used for this experiment and their references are given in Table 5.1. We execute two experiments: one to solely measure DCSH's performance and a second experiment to compare DCSH to FORCE. A description of each set-up for these experiments is shown in the flowcharts in Figure 5.1. The edge ordering is an additional ordering that has been shown to influence the computational effort [35]. This ordering is kept constant for each model and is out of the scope of this research. Furthermore, all experiments are performed using CIF.

Table 5.1: Set of CIF-models used for the benchmark experiments.

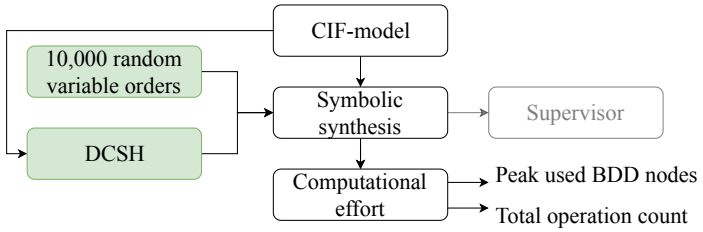
Model
Advanced Driver Assistance System (ADAS) [17]
Power substation [11]
Theme park [14]
Automated Vehicle Guidance (AVG) [37]
Multi Agent Formation (MAF) [9]
Cluster tool [33]
Ball system [10]
Bridge [26]
Production cell [13]
Waterway lock [27]
FESTO [25]

5.1 Experiments

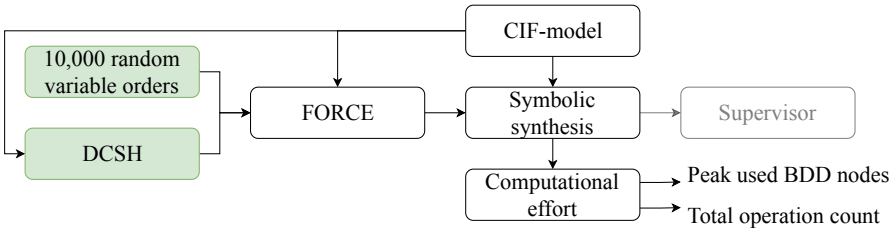
5.1.1 Experiments without applying FORCE

To measure the effectiveness of DCSH in reducing computational effort, we perform an experiment using the benchmark models. For each model shown in Table 5.1, 10,000 random variable orders are generated that are directly used for synthesis without applying any heuristics. We extract the PBN and TOC for each measurement, the means per model are denoted by μ_B and μ_O, respectively. The results of the Bridge and FESTO models are excluded from this experiment, as synthesis was unsuccessful due to out of memory errors for some of the orders with 16 GB of computer memory allocated. We use aforementioned measurements as a baseline to determine how much effort is required for synthesis without applying any heuristics.

Next, we compute a variable ordering for each model using DCSH as described in Algorithm



(a) Flowchart describing the experiment set-up without applying FORCE.



(b) Flowchart describing the experiment set-up with applying FORCE.

Figure 5.1: Flowcharts of both experiments. DCSH and FORCE compute an order based on the characteristics of the CIF-model, however, FORCE is highly dependent on the initial order provided. For both experiments the same set of 10,000 random variable orders is used per model.

3. Subsequently, this order is used to perform one synthesis per model. The PBN h_B and TOC h_O that result from applying DCSH are extracted accordingly.

5.1.2 Experiments with applying FORCE

To compare the effectiveness in effort reduction of DCSH with FORCE we repeat previous experiment. The same random orders as previous experiment are used for each model. These are used as initial order for FORCE and the order that results from FORCE is used for synthesis. Again, we extract the PBN and TOC for each measurement, the means are denoted by μ_B^F and μ_O^F, respectively. These measurements are used as a baseline to determine the effort that is required when FORCE is applied as variable ordering heuristic for synthesis.

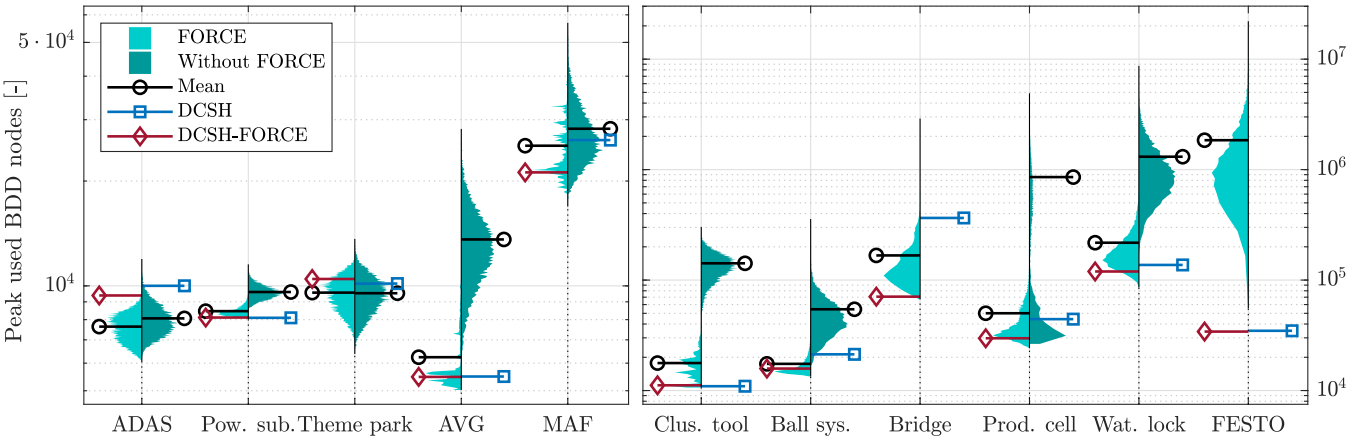
We use a large set of initial orders for FORCE as it is known from literature that FORCE is highly dependent on the initial order provided and this has a significant effect on the results [35]. Moreover, we notice that utilizing DCSH to provide an initial order for FORCE can result in a further decrease of effort required, the order that results from applying FORCE is used for synthesis. Therefore, we also compute an order using this method for each model and apply it to synthesis. The PBN and TOC are extracted, denoted by h_B^F and h_O^F, respectively. We apply FORCE as it is currently implemented[3] in CIF, thus FORCE applies the reordering based on other dependencies than DCSH. For ease of reference, we denote by DCSH-FORCE that first an order is computed by DCSH and that this order is used as initial order for FORCE.

The computations for both experiments are performed on a high performance cluster, however, as the used metrics are platform-independent, the utilized hardware has no influence on the measurements [35].

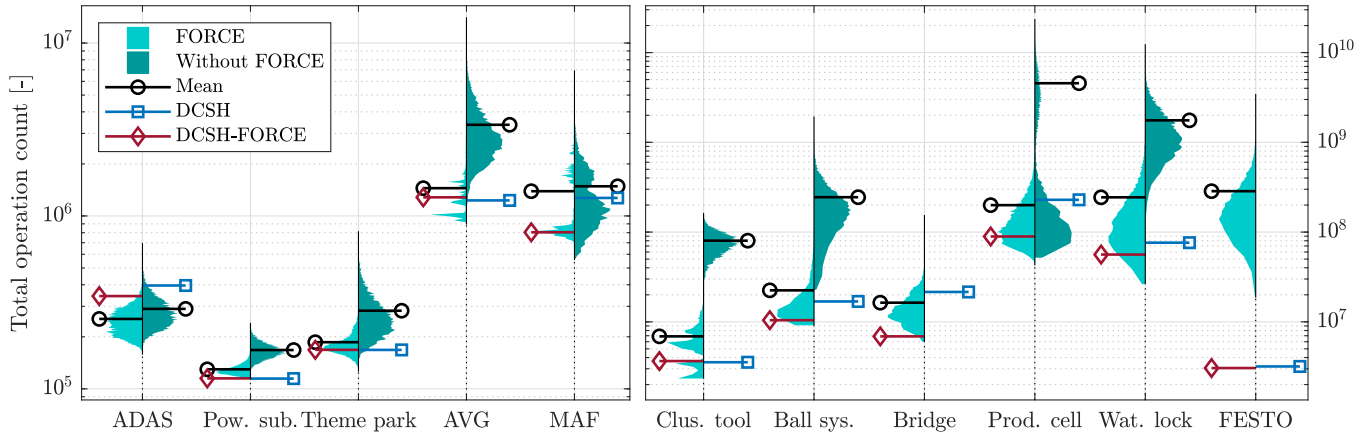
5.2 Results of the experiments

For each model a normalized histogram of measured PBN and TOC is derived, one for each experiment. These histograms along with the means and effort of applying DCSH and DCSH-

[3]See CIF's documentation about the BDD variable order: `http://cif.se.wtb.tue.nl/tools/datasynth.html`



(a) Peak used BDD nodes for both experiments.



(b) Total operation count for both experiments.

Figure 5.2: The computational effort measured for both experiments executed on the eleven benchmark models. In (a) the peak used BDD nodes are shown and in (b) the total operation count. For each model a normalized histogram is shown for both experiments. The results of the Bridge and FESTO models are excluded from the experiment without using FORCE, as this resulted in out of memory errors for some of the random orders. The top (bottom) of the solid line indicates the worst-case (best-case) of measured peak used BDD nodes and total operation count. Furthermore, the markers indicate the mean of the effort with FORCE applied when pointing towards the left and without FORCE applied when pointing towards the right.

FORCE are shown in Figure 5.2. The worst- and best-case measured effort of each model is indicated by the top and bottom of the solid part of the bold lines. In the following part of this report we refer to the models as small if shown in the boxes on the left and large when shown in the boxes on the right in Figure 5.2. This division is based on the worst-case PBN.

Moreover, to discuss the effectiveness of DCSH and DCSH-FORCE in reducing the effort compared to FORCE on average, we compute the reduction of PBN $r_B = \mu_B^F / h_B$ and reduction of TOC $r_O = \mu_O^F / h_O$ of comparing DCSH to μ_B^F and μ_O^F. Furthermore, we denote the reduction of PBN by $r_B^F = \mu_B^F / h_B^F$ and reduction of TOC by $r_O^F = \mu_O^F / h_O^F$ for DCSH-FORCE against μ_B^F and μ_O^F. We do not compute the ratios of effort reduction for DCSH and DCSH-FORCE against

synthesis with no heuristics applied, as these are of less interest, however, these results can be seen in Figure 5.2. The aforementioned ratios along with the mean PBN and mean TOC for both experiments are shown in Table 5.2.

Lastly, we compute the fraction f of random initial variable orders for FORCE that resulted in less synthesis effort compared to DCSH and DCSH-FORCE as follows

$$f = \frac{\text{Number of measurements that resulted in less effort than heuristic}}{\text{Number of total measurements}}.$$

These fractions are computed for DCSH against random initial orders for FORCE, denoted by f_B and f_O for the PBN and TOC, respectively. Furthermore, these fractions are also computed for DCSH-FORCE against random initial orders for FORCE, denoted by f_B^F for PBN and f_O^F for TOC.

Table 5.2: Results of the experiments. The column indicated by R is the ordering with minimal WES and thus applied to synthesis. Out of Memory (OoM) indicates that some of the random orderings resulted in an OoM error with 16 GB of memory allocated and are therefore omitted from these results. Values printed in bold are the best results.

Model	Mean effort				Effort reduction of DCSH/DCSH-FORCE				R
	μ_B^F	μ_O^F	μ_B	μ_O	r_B^F	r_O^F	r_B	r_O	
ADAS	$\mathbf{7.64 \cdot 10^3}$	$\mathbf{2.54 \cdot 10^5}$	$8.07 \cdot 10^3$	$2.90 \cdot 10^5$	**0.81**	**0.74**	0.76	0.64	R_{CM}^r
Pow. sub.	$\mathbf{8.46 \cdot 10^3}$	$\mathbf{1.30 \cdot 10^5}$	$9.60 \cdot 10^3$	$1.68 \cdot 10^5$	1.04	**1.13**	**1.05**	**1.13**	R_S
Theme park	$9.56 \cdot 10^3$	$\mathbf{1.86 \cdot 10^5}$	$\mathbf{9.53 \cdot 10^3}$	$2.83 \cdot 10^5$	0.91	**1.11**	**0.94**	**1.11**	R_S
AVG	$\mathbf{6.24 \cdot 10^3}$	$\mathbf{1.45 \cdot 10^6}$	$1.36 \cdot 10^4$	$3.37 \cdot 10^6$	**1.14**	1.13	**1.14**	**1.18**	R_{CM}^r
MAF	$\mathbf{2.53 \cdot 10^4}$	$\mathbf{1.39 \cdot 10^6}$	$2.83 \cdot 10^4$	$1.48 \cdot 10^6$	**1.19**	**1.73**	0.96	1.09	R_{CM}^r
Cluster tool	$\mathbf{1.77 \cdot 10^4}$	$\mathbf{6.89 \cdot 10^6}$	$1.42 \cdot 10^5$	$8.02 \cdot 10^7$	1.59	1.88	**1.62**	**1.94**	R_S
Ball system	$\mathbf{1.74 \cdot 10^4}$	$\mathbf{2.24 \cdot 10^7}$	$5.44 \cdot 10^4$	$2.45 \cdot 10^8$	**1.10**	**2.15**	0.82	1.33	R_S
Bridge	$\mathbf{1.67 \cdot 10^5}$	$\mathbf{1.63 \cdot 10^7}$	OoM	OoM	**2.36**	**2.37**	0.46	0.76	R_S
Prod. cell	$\mathbf{5.01 \cdot 10^4}$	$\mathbf{2.00 \cdot 10^8}$	$8.56 \cdot 10^5$	$4.55 \cdot 10^9$	**1.68**	**2.23**	1.13	0.87	R_{CM}^r
Wat. lock	$\mathbf{2.18 \cdot 10^5}$	$\mathbf{2.44 \cdot 10^8}$	$1.31 \cdot 10^6$	$1.76 \cdot 10^9$	**1.82**	**4.34**	1.59	3.20	R_{CM}^r
FESTO	$\mathbf{1.85 \cdot 10^6}$	$\mathbf{2.86 \cdot 10^8}$	OoM	OoM	**54.01**	**93.46**	53.12	89.69	R_{CM}^r

Table 5.3: Fractions of random initial variable orders for FORCE that performed better than DCSH and DCSH-FORCE and the WES values for the order computed by DCSH and DCSH-FORCE. Values printed in bold are the best results.

Model	DCSH			DCSH-FORCE		
	f_B	f_O	WES	f_B^F	f_O^F	WES
ADAS	**1.00**	1.00	**0.044**	**1.00**	**0.97**	0.061
Pow. sub.	**0.01**	**0.00**	**0.063**	0.02	**0.00**	0.065
Theme park	**0.70**	**0.28**	**0.025**	0.80	**0.28**	0.080
AVG	**0.37**	**0.45**	**0.166**	**0.37**	0.51	0.170
MAF	0.60	0.50	**0.017**	**0.19**	**0.19**	0.037
Cluster tool	**0.06**	**0.17**	0.055	0.12	0.18	**0.050**
Ball system	0.87	0.54	0.043	**0.50**	**0.10**	**0.042**
Bridge	0.97	0.81	0.090	**0.02**	**0.01**	**0.086**
Prod. cell	0.55	0.72	0.070	**0.09**	**0.16**	**0.065**
Wat. lock	0.16	0.13	0.078	**0.05**	**0.05**	**0.075**
FESTO	**0.00**	**0.00**	**0.041**	**0.00**	**0.00**	**0.041**

5.2.1 Results of the experiment without applying FORCE

For two models, ADAS and Theme park, DCSH was on average unable to reduce synthesis effort compared to the mean of the measured effort, as shown in Figure 5.2. We suspect that for these small models the placement of CIF-variables based on their appearance in transition relations is not an effective way to reduce their effort. However, note that reducing the computation time and

memory usage is not very noticeable to the user, as for these two models the difference of best- against worst-case effort is relatively small. Overall, for seven out of nine models on which the experiment was successfully conducted, the effort reduced noticeably compared to the mean of the random orders, as shown in Figure 5.2.

5.2.2 Results of the experiment with applying FORCE

Applying FORCE as variable ordering heuristic to random orders reduced the mean effort in most cases significantly. Only for the Theme park model we see that the mean PBN increased after applying FORCE, although this is only marginal as can be seen in Table 5.2. All other results show a reduction in mean effort required. However, the results did not show a consistent reduction of effort. The individual measurement results fluctuate noticeably, as can be seen in Figure 5.2.

We notice that DCSH-FORCE indeed mostly resulted in a further decrease of computational effort required compared to DCSH. Only for the Power substation, Theme park and Cluster tool models the PBN increased and for the AVG and Cluster tool models the TOC increased, see Table 5.2 and Figure 5.2. Furthermore, the decrease of effort was mostly noticeable for the Bridge, Production cell and Waterway lock models, which are all three regarded as large models.

Lastly, the final comparison made is whether DCSH is able to reduce effort compared to FORCE for more than half of the random measurements per model, see Table 5.3. If the fraction corresponding to both metrics for a model is < 0.50, this is indeed the case.

For DCSH against FORCE we see that DCSH performed better for five out of eleven models. However, some of the improvements, especially for the two large models Waterway lock and FESTO, are substantial. For the second experiment, where DCSH-FORCE is applied, effort is reduced in seven out of eleven cases.

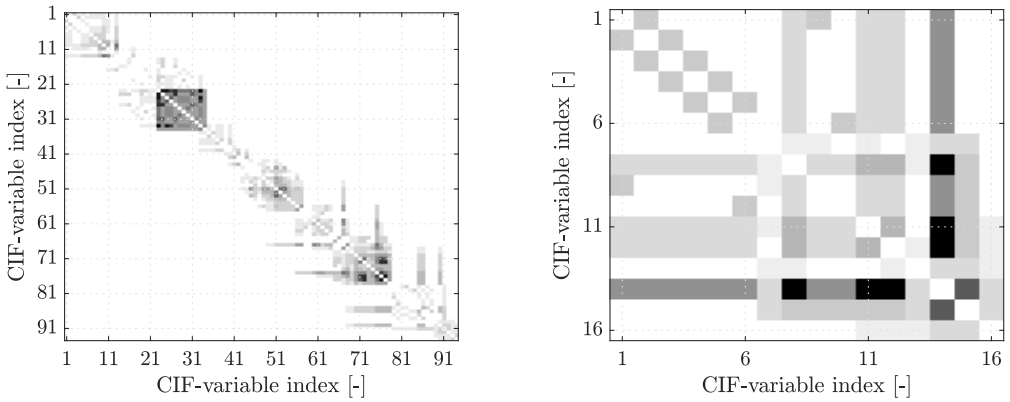
FORCE minimizes a so called span [2] of transition groups that is related to the WES [20]. We suspect that FORCE gets stuck in a local optimum when minimizing the span, and therefore is highly dependent on the initial order provided. Because the WES and span are related, DCSH already provides an order with a low WES and by using FORCE the WES reduces even more. This is indeed the case for most models, most noticeable for five out of the six large models, see Table 5.3. For the most cases where the WES increased, the effort also increased, only for the ADAS, MAF and Cluster tool models this correlation did not hold. As both DCSH and FORCE require minimal computational effort, it is a viable option to utilize DCSH-FORCE, noting that the further reduction in synthesis effort can be significant.

We noticed that DCSH is mostly independent of the initial ordering provided. As both the weighted CM and Sloan's algorithm start reordering from pseudo-peripheral nodes. The true peripheral nodes in a (weighted) adjacency graph are constant for each possible labeling of the nodes. Therefore, the resulting order is largely dependent on the initial approximation of the peripheral node for weighted CM or nodes for Sloan. Any non-determinism that follows is due to having two or more nodes with the same degree for Sloan and same degree and weight for weighted CM. The exact order in which these nodes are appended to the order is therefore undecided. However, to exactly quantify the effect of this, future experiments should be conducted.

5.2.3 Discussion of the experiment results

A noticeable outlier can be seen in the effort reduction of applying DCSH to the FESTO model, the computationally most complex model based on PBN. This significant reduction can be explained as for this specific model every CIF-variable in the transition relations only appears with few other variables. Therefore, it is possible to perform a better optimization of the effort for models that have a similar modular structure, based on our findings. We suspect that for models that have a monolithic structure (where variables appear in transition relations together with many other variables), there is a certain limit in possible effort reduction that can be achieved by computing an effective variable ordering.

Figure 5.3 shows the ordered NDSM of the FESTO and MAF models, the modular nature of the FESTO model allows an order in which non-zero values are placed close to the diagonal and



(a) Ordered DSM of the FESTO model.

(b) Ordered DSM of the MAF model.

Figure 5.3: Ordered DSMs for the FESTO and MAF model.

a modular clustering can be seen. All non-zero values are placed near the diagonal. The MAF model, however, has a monolithic structure and therefore applying DCSH resulted in relatively less effort reduction.

To conclude, we have shown that DCSH is competitive with the state-of-the-art heuristic FORCE, most noticeable for computationally more complex models. However, FORCE is shown to be highly dependent on the initial order. When comparing the effort resulting from applying DCSH against FORCE, it can be seen that both heuristics perform about equal in reducing effort. Both FORCE and DCSH were able to reduce total effort for five out of eleven cases. For the Theme park model this is undecided, as FORCE resulted in lower TOC and DCSH in less PBN, see the computed fractions in Table 5.3. However, the most significant reductions in effort when DCSH is applied were made on the largest models, most noticeable the Waterway lock and FESTO models. Note that for the effective use of FORCE the user must provide a decent order as input, we noticed that for DCSH this is not the case. Performing DCSH as pre-ordering for FORCE showed, in some cases significant, further reduction of effort and thereby also is the most effective way in reducing computational effort as shown in this report.

Chapter 6

Conclusions

This report describes a novel DSM-based variable ordering heuristic applied to symbolic supervisor synthesis, named DSM-based Cuthill-McKee-Sloan variable ordering Heuristic (DCSH). We have shown by what extent and why the variable ordering influences the backwards reachability search. Moreover, we have shown the significance of the computational effort required for this search and thereby for total symbolic synthesis.

By reordering the variables based on the transition relations of edges, synthesis effort can be reduced by orders of magnitude. Even though the relationship among peak used BDD nodes and transition relations have been described in literature, no explanation of the exact reasons behind this behavior is described. In this research we show by means of an example and empirical results why this takes place. Moreover, we show how much this contributes to the total synthesis effort and thereby is a novel contribution of this research.

DCSH relies on a Numerical Dependency Structure Matrix (NDSM) and Transition Relation Matrix (TRM) that are computed before synthesis. Two node ordering heuristics, weighted Cuthill-McKee and Sloan's ordering, are directly applied to the (weighted) adjacency graph of the NDSM to result in a reordering of the initial CIF-variables. The Weighted Event Span (WES) metric is subsequently utilized to predict which of the found orders should be applied to synthesis, besides whether reversing the order could be beneficial. DCSH is compared to a state-of-the-art variable ordering heuristic FORCE in an experiment, applied to a set of benchmark models. The results show that DCSH is competitive with FORCE in effort reduction. Moreover, DCSH showed significantly better performance for the two largest models tested. Furthermore, effective use of FORCE requires the user to provide a suitable initial variable ordering. Utilizing DCSH to compute an initial order for FORCE is shown to further decrease the effort in most cases. As the computational effort required for applying this additional heuristic is insignificant compared to the potential reduction of effort for symbolic synthesis, this is a feasible option. Hence, this method is the recommended method of applying DCSH.

Lastly, the DSM-based approach allows visualization of the ordering, which can be useful if one wants to compare other matrix reordering schemes to the schemes used in DCSH to see whether they provide the required reordering, as the DSM can be a useful method to visualize how related variables appear in a variable ordering. The WES can be used to estimate whether the ordering could result in decreased effort without applying synthesis, this could be especially useful for computationally complex models.

For future work it is recommended to study whether the conclusions made in this context hold for BDD-based algorithms utilized for alternative, albeit similar applications, such as model checking. Furthermore, it could be beneficial to study whether adjusting Sloan's ordering to sort on weight results in a further decrease of effort when DCSH is applied. Moreover, the effectiveness of other ordering algorithms should be studied. We experienced that DCSH was relatively insensitive to the initial order provided due to the behavior of computing an order based on the adjacency graph. It could be valuable to conduct further research whether this experienced behavior holds.

DCSH is a so called static variable ordering, it computes an order before synthesis commences

based on model characteristics. It could be beneficial to study whether a dynamic ordering, that updates the variable order during synthesis, could further reduce the effort required. Furthermore, in this report no interleaving of BDD-variables belonging to separate CIF-variables is applied. It could be of interest to study whether finding orders where BDD-variables are interleaved results in less effort.

Bibliography

[1] S. B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, 27(6):509–516, June 1978. 4

[2] F. Aloul, I. Markov, and K. A. Sakallah. FORCE: A fast and easy-to-implement variable-ordering heuristic. *IEEE Great Lakes Symposium on VLSI*, May 2003. 1, 6, 23

[3] A. Aziz, S. Tasiran, and R. K. Brayton. BDD variable ordering for interacting finite state machines. In *31st Design Automation Conference*, pages 283–288, June 1994. 6

[4] B. Bollig and I. Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on Computers*, 45(9):993–1002, Sep 1996. 1

[5] T. Browning. Applying the design structure matrix to system decomposition and integration problems: A review and new directions. *IEEE Transactions on Engineering Management*, 48:292 – 306, Aug 2001. 13

[6] T. Browning. Design structure matrix extensions and innovations: A survey and new opportunities. *IEEE Transactions on Engineering Management*, 63(1):27–52, Feb 2016. 13

[7] R. E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, Sept. 1992. 4, 8

[8] J. R. Burch, E. M. Clarke, D. E. Long, K. L. McMillan, and D. L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(4):401–424, April 1994. 8, 9

[9] K. Cai and W. M. Wonham. New results on supervisor localization, with case studies. *Discrete Event Dynamic Systems*, 25(1-2):203–226, June 2015. 19

[10] G. Čengić and K. Åkesson. A control software development method using IEC 61499 function blocks, simulation and formal verification. *IFAC Proceedings Volumes*, 41(2):22 – 27, 2008. 17th IFAC World Congress. 19

[11] 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 *2017 IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pages 1730–1739, Dec 2017. 19

[12] E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th National Conference*, ACM '69, pages 157–172, New York, NY, USA, 1969. ACM. 15, 16

[13] L. Feng, K. Cai, and W. Wonham. A structural approach to the non-blocking supervisory control of discrete-event system. *The International Journal of Advanced Manufacturing Technology*, 41:1152–1168, April 2009. 19

[14] S. T. J. Forschelen, J. M. van de Mortel-Fronczak, R. Su, and J. E. Rooda. Application of supervisory control theory to theme park vehicles. *Discrete Event Dynamic Systems*, 22(4):511–540, Dec 2012. 19

[15] A. George and J. W. H. Liu. An implementation of a pseudoperipheral node finder. *ACM Transactions on Mathematical Software*, 5(3):284–295, Sept. 1979. 16

[16] M. Goorden, J. van de Mortel-Fronczak, M. Reniers, and J. Rooda. Structuring multilevel discrete-event systems with dependency structure matrices. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 558–564, Dec 2017. 15

[17] 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 Transactions on Intelligent Transportation Systems*, 19(2):533–544, Feb 2018. 19

[18] R. Malik, K. Åkesson, H. Flordal, and M. Fabian. Supremica - An efficient tool for large-scale discrete event systems. *IFAC-PapersOnLine*, 50(1):5794 – 5799, 2017. 20th IFAC World Congress. 6

[19] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Norwell, MA, USA, 1993. 9

[20] J. Meijer and J. van de Pol. Bandwidth and wavefront reduction for static variable ordering in symbolic reachability analysis. In *NASA Formal Methods*, Lecture Notes in Computer Science, pages 255–271. Springer, June 2016. 1, 15, 17, 23

[21] S. Minato. *Binary Decision Diagrrams and Applications for VLSI CAD*, volume 342 of *The Springer International Series in Engineering and Computer Science*. Springer US, 1st edition, 1996. 1, 10

[22] S. Miremadi, B. Lennartson, and K. Åkesson. A BDD-based approach for modeling plant and supervisor by extended finite automata. *IEEE Transactions on Control Systems Technology*, 20(6):1421–1435, 2012. 1, 3

[23] L. Ouedraogo, R. Kumar, R. Malik, and K. Åkesson. Nonblocking and safe control of discrete-event systems modeled as extended finite automata. *IEEE Transactions on Automation Science and Engineering*, 8:560 – 569, Aug 2011. 3, 5, 7

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

[25] 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 *2018 IEEE Conference on Control Technology and Applications, CCTA 2018*, pages 458–464, Oct 2018. 18, 19

[26] F. Reijnen, M. Reniers, J. van de Mortel-Fronczak, and J. Rooda. Structured synthesis of fault-tolerant supervisory controllers. *IFAC-PapersOnLine*, 51(24):894 – 901, 2018. 10th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes SAFEPROCESS 2018. 19

[27] F. F. H. Reijnen, M. A. Goorden, J. M. van de Mortel-Fronczak, and J. E. Rooda. Supervisory control synthesis for a waterway lock. In *2017 IEEE Conference on Control Technology and Applications (CCTA)*, pages 1562–1563, Aug 2017. 1, 19

[28] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2003. 16

[29] R. I. Siminiceanu and G. Ciardo. New metrics for static variable ordering in decision diagrams. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 90–104, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. 17

[30] M. Skoldstam, K. Åkesson, and M. Fabian. Modeling of discrete event systems using finite automata with variables. In *2007 46th IEEE Conference on Decision and Control*, pages 3387–3392, 2007. 3

[31] S. W. Sloan. A FORTRAN program for profile and wavefront reduction. *International Journal for Numerical Methods in Engineering*, 28:2651–2679, 1989. 15, 16, 17

[32] F. Somenzi. Binary decision diagrams. In *Calculational System Design, volume 173 of NATO Science Series F: Computer and Systems Sciences*, pages 303–366. IOS Press, 1999. 1, 4, 5, 10

[33] R. Su, J. H. van Schuppen, and J. E. Rooda. Aggregative synthesis of distributed supervisors based on automaton abstraction. *IEEE Transactions on Automatic Control*, 55(7):1627–1640, July 2010. 11, 19

[34] R. J. M. Theunissen, M. Petreczky, R. R. H. Schiffelers, D. A. van Beek, and J. E. Rooda. Application of supervisory control synthesis to a patient support table of a magnetic resonance imaging scanner. *IEEE Transactions on Automation Science and Engineering*, 11(1):20–32, Jan 2014. 1

[35] S. Thuijsman, D. Hendriks, R. Theunissen, M. Reniers, and R. Schiffelers. Computational effort of BDD-based supervisor synthesis of extended finite automata. In *2019 IEEE 15th International Conference on Automation Science and Engineering*, 2019. 1, 2, 6, 19, 20

[36] D. A. Van Beek, W. J. Fokkink, D. Hendriks, A. Hofkamp, J. Markovski, J. M. van de Mortel-Fronczak, and M. A. Reniers. CIF 3: Model-based engineering of supervisory controllers. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8413 LNCS:575–580, 2014. 5

[37] W. M. Wonham and K. Cai. *Supervisory Control of Discrete-Event Systems*. Springer International Publishing, 1st edition, 2019. 19

[38] W. M. Wonham, K. Cai, and K. Rudie. Supervisory control of discrete-event systems: A brief history. *Annual Reviews in Control*, 45:250–256, 2018. 1

TU/e Technische Universiteit Eindhoven University of Technology

Declaration concerning the TU/e Code of Scientific Conduct for the Master's thesis

I have read the TU/e Code of Scientific Conduct[i].

I hereby declare that my Master's thesis has been carried out in accordance with the rules of the TU/e Code of Scientific Conduct

Date

13-11-2019

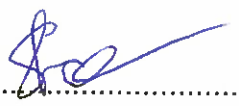
Name

Sam Lousberg

ID-number

0805332

Signature



Submit the signed declaration to the student administration of your department.

[i] See: http://www.tue.nl/en/university/about-the-university/integrity/scientific-integrity/
The Netherlands Code of Conduct for Academic Practice of the VSNU can be found here also.
More information about scientific integrity is published on the websites of TU/e and VSNU