Speeding Up Symbolic Model Checking by Accelerating Dynamic Variable Reordering

Christoph Meinel
FB IV - Informatik
Universität Trier, GER
meinel@uni-trier.de

Christian Stangier
FB IV - Informatik
Universität Trier, GER
stangier@uni-trier.de

ABSTRACT

Symbolic Model checking is a widely used technique in sequential verification. As the size of the OBDDs and also the computation time depends on the order of the input variables, the verification may only succeed if a well suited variable order is chosen. Since the characteristics of the represented functions are changing, the variable order has to be adapted dynamically. Unfortunately, dynamic reordering strategies are often very time consuming and sometimes do not provide any improvement of the OBDD representation. This paper presents adaptions of reordering techniques originally intended for combinatorial verification to the specific requirements of symbolic model checking. The techniques are orthogonal in the way that they use either structural information about the OBDDs or semantical information about the represented functions. The application of these techniques substantially accelerates the reordering process and makes it possible to finish computations, that are too time consuming, otherwise.

1. INTRODUCTION

Model checking has been proven to be a powerful tool in the verification of sequential circuits, reactive systems, protocols, etc. The model checking of systems with huge state spaces is possible only if there is a very efficient representation of the model. Reduced Ordered Binary Decision Diagrams (shortly: OBDDs) [1] allow an efficient *symbolic* representation of the model [4].

Due to the huge number of operations applied to the OBDDs during symbolic model checking, the computation time is strongly related to the size of the OBDDs. As the order of the input variables has a strong influence on the size of the OBDDs, well suited variable orders have to be found. Since it is NP-hard to find the optimal variable order for a given function, much effort is spent on finding reasonable good orders or improving given ones. In practice, techniques that improve the size of a given OBDD by changing the variable order dynamically during the computation have been proven to be most powerful. Many common *dynamic reordering* approaches are based on swapping the position of neighboured variables in a given OBDD. This operation can be performed locally and thus, can be computed efficiently. The *sifting* algorithm [6] that is based on this idea moves each variable to the top and to the bottom of the order to find its best position. This algorithm has been proven to be one of the most efficient reordering strategies.

Dynamic reordering strategies are especially useful for symbolic model checking, since the represented functions (e.g. reachable state sets) are changing during computation. As a consequence, the variable order has to be adapted to fulfill the new requirements.

Although, dynamic variable reordering may drastically reduce the OBDD size, often it is very time consuming and sometimes does not lead to substantially smaller OBDD sizes. Also, recent research [9; 2] has shown the need for improvement of variable reordering during model checking. The authors of [9] claim that sometimes variable reordering is invoked too frequently, but many computations would not finish without reordering. It is a nontrivial task to decide whether OBDD sizes grow due to unappropriate variable orders or due to changes of functions that require more OBDD nodes for representation.

Our goal is to speed up the reordering process. There are two techniques for acceleration of variable reordering, called *block restricted sifting* (BRS) [5] and *sample sifting* [7; 3]. These techniques originally intended for combinatorial circuits are orthogonal in the way that they use either *structural* information about the OBDDs or *semantical* information about the represented functions. Unfortunately these techniques produce insufficient results if applied to variable reordering during symbolic model checking. For one half of our benchmark set they perform worse than standard sifting. We made necessary adaptions, that take into account the special needs of symbolic model checking. The major improvement of these techniques is a significant reduction of the computation time with only a small penalty in size.

2. SPEEDING UP MODEL CHECKING

OBDD based model checking tools like SMV [4] use variable reordering techniques for the reduction of OBDD sizes. Indeed, the huge amount of operations necessary for iterations or fixpoint computations requires too much time or is impossible due to memory limitations, if the underlying OBDDs are large. Also, the represented functions, like reachable state sets are changing during computation and thus, the variable order has to be adapted to avoid an exponential growth of OBDD sizes.

Although, sifting is the fastest common reordering technique, it is often too time consuming to be applied during model checking. It emerges that sometimes a large fraction of the computation time is spent on sifting without any gain in OBDD size. Another fact is that sifting is too costly to be invoked whenever functions are changing.

Unlike in usual applications that require variable reordering we have to fulfill the following two requirements in symbolic model checking simulteanously.

TIME: Reordering during model checking is costly (50% or more of the computation time). Furthermore, sometimes reordering even does not decrease the OBDD sizes.

QUALITY: Model checking demands a lot to variable orders. During model checking only a few reorderings take place, but thousands of OBDD operations require small OBDD sizes to work efficiently.

2.1 Block Restricted Sifting

Our goal is to accelerate the variable reordering process while simultaneously retaining reasonable OBDD sizes. To manage this, we adapted a method called *block restricted sifting* (BRS) [5] to the needs of model checking. The idea behind BRS is to move the variables during reordering only within fixed blocks instead of moving them through the complete order. From theory it is known, that changing the variable order of a block does not affect the size of the other blocks.

The determination of the block boundaries follows from a communication complexity argument. A small information flow between two parts of an OBDD indicates a good candidate for a block boundary. If there is only little information flow between two blocks the distribution of variables to these blocks is well chosen. Improving the variable order inside the blocks might lead to a significant reduction of the OBDD size. The information flow is best indicated by the number of subfunctions that cross one level. The *subfunction profile* of an OBDD counts not only the number of nodes per level, it also adds the edges that cross a level without having a node on it to the profile. With the aid of this profile we get easily computable structure information of the represented function.

For a successful application of BRS to symbolic model checking, we have to find solutions for the following problems:

1. By restricting the search space, one should not foreclose finding good variable orders.

2. Use all the acceleration power that BRS provides.

3. Find appropriate settings of BRS-parameters for model checking.

1. Restricting the search space. Sifting only within fixed blocks significantly accelerates the reordering process, but it may keep one away from good orders for the following two reasons:

– If starting with a very bad order, variables that are placed totally wrong cannot move to arbitrary positions due to the block boundaries.

– Since placing the block boundaries is a heuristic decision, it may artificially separate variables, that should be placed in one single block.

In conventional applications like combinational verification the larger number of reordering with changing block boundaries compensates these effects. For symbolic model checking this is not true, because of the small number of reorderings. Therefore, we have changed the concept of block boundaries. We now allow a small overlapping of the blocks, i.e. a few levels beside the boundaries of the block are also incorporated in the reordering. These additional levels are used for choosing variables to be reordered as well as for placing variables. The reason for this is to allow variables to cross a block boundary. This concept partially remedies the problems stated above. Preliminary experiments during the design phases of our algorithm have shown, that these *weak boundaries* highly increase the quality of the computed orders.

2. Acceleration power. To take full advantage of the BRS approach one should restrict the size of blocks. The native BRS searches for local minima in the subfunction profile. This may lead to unnecessary large blocks in the lower part of the OBDD, where the number of nodes and represented subfunctions naturally decreases. We have changed this strategy to a *first-fit* strategy, i.e the first level that fulfills the given properties is chosen. This strategy usually places at least one more boundary in the lower part of the OBDD, what improves the reordering time, but does not lower the quality of the variable order, because of the smaller influence of the lower part of the OBDD to the overall size.

3. Settings. The parameter, that is mostly responsible for the trade-off between reordering time and the quality of the computed order is the minimal fraction of variables that a block must contain. This fraction is denoted MINBLOCK. The smaller the blocks are, the faster the reordering works, but the computed orders are getting less optimal due to the strongly restricted search space. If the Blocks are of the same size, the search space is reduced from $O(\#vars^2)$ to $O(\#vars^2 \cdot MINBLOCK)$. In contrast to combinatorial verification, where MINBLOCK = 10% is an average setting for model checking larger blocks are appropriate. Blocksizes smaller than 10% will lead to extremely poor orders.

2.2 Sample Sifting

Sampling is a common heuristic technique applied to optimization problems with huge search spaces. The idea behind the sampling strategy is to choose a relevant sample from the given problem instance to solve the optimization problem for the chosen subset and to generalize the solution to the complete instance.

Applied to the problem of reordering OBDD variables the sampling strategy can be described as follows [7; 3]: (1) Choose some OBDDs or subOBDDs from the common shared OBDD. (2) Copy these OBDDs to a different location. (3) Reorder only the Sample. (4) Shuffle the variables of the original BDD to the newly computed order of the sample.

Step 1 is the most critical during this sample sifting process. As mentioned before, one should choose a relevant sample. If there is some knowledge about the represented functions, it can be used for the choice of samples. The chosen OBDDs should not be too small, so that as many variables of the representation as possible are contained in the sample. If there is no knowledge about the represented functions the sample may be chosen randomly from the single roots of the shared OBDDs. The reordering (Step 3) can be done with any common reordering technique (we used the standard sifting algorithm). If the OBDD size increases after Step 4 the OBDD is reshuffled to the original order, but one may repeat the complete process to obtain better results.

A successful application of the sampling method to model checking is challenging, because the two main problems of variable reordering for model checking instantiate as follows:

Time:
-T1 A smaller sample will accelerate the reordering process.
-T2 One may accelerate the sample reordering process itself.
-T3 Determine a useful number of trials for the sampling per reordering.
Quality:
-Q1 Choice of a sample without given semantical information.
-Q2 Choice of a sample if some semantical information is available.
-Q3 Appropriate methods for copying fractions of OBDDs.

T1. Small samples. The size of the sample is the most important parameter of sample sifting. Choosing a smaller sample will reduce the computational overhead for copying the sample. But even more important: The accelerating effect of sample sifting results from the fact that only a small OBDD is reordered, also resulting in smaller intermediate OBDD sizes during the reordering. The smaller the sample is, the faster the reordering performs. But, the sample cannot be chosen arbitrarily small, because in this case it does not represent the original OBDD's properties sufficiently. The result of the reordering usually will be a poor ordering for the original OBDD. Thus, the size of the sample directly influences the quality of the computed order. To fulfill the quality requirements of model checking the sample has to be chosen larger than for combinatorial applications.

T2. Accelerating sample reordering. As stated above the time saved be sample sifting results from sifting a smaller OBDD. One may try to accelerate even this reordering, but this will usually result in variable orders of less quality. Instead, we suggest to reorder the sample even more by enlarging the search space, e.g. by allowing a larger growth of the OBDD during reordering. This may compensate the quality losses resulting from reordering only a fraction of the OBDD.

T3. Number of Trials. More than one trial per sample reordering might be a good idea for combinatorial application but not for model checking for the following reasons:
– Due to the small number of reorderings, several trials will compensate all the time savings, especially if larger samples are used.
– In some situations OBDD sizes grow despite of good variable orders. Here any reordering will fail.

Q1. Sample without semantical information. If no external semantical information is available one may at least use some structural information about the represented functions. We used a pseudo-random strategy proposed by [7]: Starting from the top level of the OBDD nodes that are not representing projection functions (i.e. $f = x_i$) are chosen randomly as roots of subOBDDs for the sample. This process is repeated level by level until the size requirements for the sample are fulfilled.
Another strategy is to chose the sample from the roots with the largest subOBDDs. Unfortunately, this strategy does not work well. Obviously, optimizing the order of only a few OBDDs does not meet the requirements of all represented functions.

Q2. Sample with semantical information. One should make use of the semantical information about represented functions provided by the model checker. In [7] it is proposed to use *recently-used-roots*, i.e. roots involved in operations in the last steps of the computation. Again, this strategy is not suitable for model checking, since the huge number of operations will result in a random choice of roots. Instead, we use *recently-used-important-roots*, i.e. roots involved in elementary model checking operations like Exist-Abstract, Universal-Abstract and And-Abstract (see [4]). If we cannot fulfill the size requirements for the sample by using important roots we fall back to the method of choosing random roots. Using this strategy we obtain the best results for sampling.

Q3. Methods for copying. In [7] copying a fraction of an OBDD is done in the following way (postorder): The OBDD is traversed in DFS order and the nodes are copied to the sample whenever a node is backtracked. This is done until the required size of the sample is reached. This method copies at first the lower part of the OBDD. The resulting sample is a subfunction of the original OBDD. If only a small sample is chosen it will leave some variables of the upper part of the OBDD
To avoid this, we propose the following method (preorder): The OBDD is also traversed in DFS order. But, the nodes are copied to the sample when the node is visited the first time. This results in samples that include usually all variables and the outline of the sample is related to the outline of the original OBDD, i.e. from a level with many nodes a larger number of nodes is chosen for the sample. Applying this method results in unvisited edges that are set to the 1-sink. Thus, the resulting sample is modified but closely related to the original function. Our experience has shown that the preorder method works more stable and produces better results than the postorder method.

3. EXPERIMENTAL RESULTS

For our experiments we used the publicly available SMV-traces of Yang [9]. Traces are recorded calls of OBDD operations during the computation of SMV-models. The underlying models come from different sources and represent a range from communication protocols to industrial controllers. We used those traces, that require less than 250MB of memory and less than 4 hours CPU-time. During reordering grouping of present- and next-state variables was enabled. The maximum allowed growth of the OBDD-size while sifting one variable was set to 20%.
The computation using the standard sifting method showed some evident differences of model checking in comparison to other OBDD applications like combinatorial verification: The number of variables (244 avg.) is comparable to other applications. The computation time is quite high (2044s avg.). The fraction of computation time, that is spent on reordering is extremely large (61% avg. of each reordering fraction), but only a few reorderings occur (4.7 avg.). The average size reduction over all reorderings is not very high. This results from the fact, that some reordering attempts do not result in smaller OBDDs. E. g. four reorderings during the computation of `furnace17` do no lead to smaller OBDDs, but one reordering drastically reduces the OBDD size (85%). Finally, the models are quite large (2.8 Mio. peaknodes avg.). Thus, most of them will not finish computation without reordering.
We implemented our sifting strategies in the CUDD Package [8] (version 2.3.0). All experiments were performed on Intel PentiumPro 200MHz Linux Workstations with 250-MByte datasize and CPU-time limited to 4 hours. For all computations we used the common technique of grouping present- and next-state variables i.e. a present-/next-state pair is always kept in adjacent levels. This on the one hand

	Sift	Block Restricted Sifting Time								Sift	Block Restricted Sifting Nodes ×1000							
Blocksize		10%		15%		20%		25%			10%		15%		20%		25%	
	s	%	s	%	s	%	s	%	s	n.	%	n.	%	n.	%	n.	%	n.
dartes	504	+55	229	+60	201	+47	265	+37	318	583	- 4	605	- 1	587	- 3	604	- 0	583
dme2-16	3757	+59	1538	+54	1713	+47	2001	+33	2511	5151	+12	4554	+ 2	5034	+12	4550	+ 8	4758
dpd75	4574	+47	2440	+40	2723	+31	3162	+ 8	4207	3296	- 2	3348	- 1	3322	- 2	3361	+ 0	3284
ftp3	1119	+17	926	+26	825	+21	889	+19	904	3126	+ 8	2879	+ 8	2879	+ 8	2879	+10	2825
furnace17	3938	+15	3361	+27	2863	+14	3380	+ 2	3843	2373	-34	3587	- 1	2400	-18	2881	-24	3134
key10	846	+56	376	+61	330	+61	332	+57	364	1099	-12	1253	-11	1238	+ 5	1039	+ 5	1039
mmgt20	1610	+18	1315	+32	1095	+28	1159	+22	1250	2904	-10	3222	- 8	3169	- 8	3170	- 1	2935
motors-stuck	265	+11	236	+32	181	+18	218	+20	212	670	- 9	735	- 5	703	- 4	700	- 8	729
over12	3002	+44	1692	+44	1668	+31	2065	+ 6	2825	4725	- 6	5025	- 0	4737	- 0	4737	+ 3	4600
phone-async	2604	- 9	2890	+11	2307	+17	2156	+ 0	2592	6118	- 7	6579	-10	6766	+10	5530	-10	6766
valves-gates	268	+34	177	+38	167	+25	201	+19	218	542	-16	647	-29	768	-25	719	- 8	588
sum	22487	+31	15179	+37	14074	+30	15827	+14	19245	30593	- 6	32439	- 3	31609	+ 1	30175	- 2	31247
avg		+31		+39		+31		+20			- 7		- 5		- 2		- 2	

Table 1: Comparison of CPU-Time and Peaknodes for Standard Sifting and Block Restricted Sifting

	Sifting	Sample Sifting Time				Sifting	Sample Sifting Nodes ×1000			
Sample Size		30%		40%			30%		40%	
	s	%	s	%	s	n.	%	n.	%	n.
dartes	504	+70	149	+62	194	583	-17	707	-17	707
dme2-16	3757	+45	2073	+53	1765	5151	-12	5824	-13	5945
dpd75	4574	+28	3304	+ 9	4144	3296	- 9	3633	- 8	3566
ftp3	1119	+43	635	+34	742	3126	+ 4	2986	+10	2806
furnace17	3938	+41	2341	+35	2545	2373	-16	2841	- 3	2439
key10	846	+33	568	+28	610	1099	-51	2236	-51	2236
mmgt20	1610	- 9	1770	-17	1961	2904	- 1	2945	- 1	2944
motors-stuck	265	+44	147	+38	164	670	-38	1073	-37	1058
over12	3002	+51	1475	+39	1831	4725	+ 4	4550	+ 4	4543
phone-async	2604	+13	2268	+13	2273	6118	- 7	6603	-24	8080
valves-gates	268	+24	202	+14	220	542	-43	950	-42	941
sum	22487	+34	14934	+27	16458	30593	-11	34353	+13	35270
avg		+35		+28			-17		+16	

Table 2: Comparison of CPU-Time and Peaknodes for Standard Sifting and Sample Sifting

accelerates reordering and on the other usually results in better orders. Due to the fact that the original strategies performs very unstable we compare our results to the standard sifting method.

The choice of traces as benchmarks enables us to show that our strategies are not restricted to a special model checker.

3.1 Block Restricted Sifting

For the experiments we used minimal blocksizes of 10%, 15%, 20% and 25%. For experimental results see Figure 1 (e.g.: +30 means 30% faster/smaller). We were able to decrease the average computation time up to 39% and overall computation time up to 37%. The maximum improvement is 61%. There is only a minor increase in peak sizes compared to normal sifting. For a minimum blocksize of 20% there is even a small memory gain. This memory gain surely could be extended, if the reordering is called more frequently. The experiments have shown, that BRS is a good choice for accelerating symbolic model checking, if no information about the represented functions is available.

3.2 Sample Sifting

Due to the random choice when copying a sample, for all experiments 10 single runs were performed.

For experiments we used the method *Important Roots* (IR) with sample size of 30% and 40%. For experimental results see Figure 2. All samples are chosen by using the *preorder* method. We were able to decrease the average computation time up to 35% and the overall computation time up to 34%. The maximum improvement is 70%.

Since we obtained our results with a very loose coupling of the model checker to the OBDD-package, a tighter coupling to the model checker e.g. having exact knowledge about the represented functions would lead to even better results.

4. REFERENCES

[1] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. In *IEEE Transactions on Computers, C-35*, pages 677–691, 1986.

[2] L. Fix and G. Kamhi. Adaptive Variable Reordering for Symbolic Model Checking. In *Proc. IEEE Int. Conf. on Computer-Aided Design*, pages 359–365, 1998.

[3] J. Jain, W. Adams, and M. Fujita. Sampling Schemes for Computing OBDD Variable Orderings. In *Proc. IEEE Int. Conf. on Computer-Aided Design*, pages 331–638, 1998.

[4] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[5] C. Meinel and A. Slobodová. Speeding up Variable Reordering of OBDDs. In *Proc. Int. Conf. on Computer Design*, pages 338–343, 1997.

[6] R. Rudell. Dynamic Variable Ordering for Ordered Binary Decision Diagrams. In *Proc. IEEE Int. Conf. on Computer-Aided Design*, pages 42–47, 1993.

[7] A. Slobodová and C. Meinel. Sample Method for Minimization of OBDDs. In *Proc. Int. Workshop on Logic Synthesis*, pages 311–316, 1998.

[8] F. Somenzi. CUDD-Package. ftp://vlsi.colorado.edu.

[9] B. Yang et al. A performance study of bdd-based model checking. In *Proc. of FMCAD*, pages 255–289, 1998.