IEEE International Symposium on Circuits and Systems (ISCAS'2000)
Geneva, May 2000

OPTIMIZATION OF SEQUENTIAL VERIFICATION BY HISTORY-BASED DYNAMIC MINIMIZATION OF BDDS

Rolf Drechsler *Wolfgang Günther*

Institute of Computer Science
Albert-Ludwigs-University
79110 Freiburg im Breisgau, Germany
email: {drechsle,guenther}@informatik.uni-freiburg.de

ABSTRACT

Binary Decision Diagrams (BDDs) are the state-of-the-art data structure in VLSI CAD, especially in sequential verification tasks, like state-space exploration and image computation. Since their size largely depends on the chosen variable ordering, dynamic variable reordering methods, like sifting, often have to be applied while the BDD is constructed. Usually sifting is called each time a given node limit is reached and it is therefore called frequently during the construction of large BDDs. Often most of the runtime is spent for sifting while the BDD is built.

Recently an approach to reduce runtime during BDD construction for combinational circuits by using history-based decision procedures has been proposed. In this paper we show that for sequential circuits different criteria should be used to select the type of sifting that involve knowledge from the sequential domain. The algorithm has been included in VIS. Experimental results show that the overall runtime can be reduced significantly and is also clearly superior to the combinational approach.

1. INTRODUCTION

Based on *ordered Binary Decision Diagrams* (BDDs) [3], *Finite State Machines* (FSMs) of more than 10^{20} states could be symbolically traversed [5, 6, 4]. In the meantime, these techniques have been successfully applied in practice and have also been integrated in industrial verification tools.

However, as well known BDDs are very sensitive to the variable ordering, i.e. the size of a BDD (measured in the number of nodes) may vary from linear to exponential. Finding the optimal variable ordering is an NP-hard problem [1, 16].

This is the reason why many authors proposed heuristics for finding good variable orderings from circuit descriptions in the last few years. The most promising methods for BDD minimization are based on *Dynamic Variable Ordering* (DVO) [9], i.e. improving graph size using exchanges of neighboring variables. The best results measured in the number of nodes of the resulting BDD were obtained using *sifting* [15, 14], but unfortunately sifting is very time consuming for large functions. Furthermore, during BDD construction it is often necessary to start sifting several times.

For this, recently several techniques for speeding up sifting have been proposed. In [13] an algorithm has been used that partitions the search space to improve sifting runtimes. But this algorithm is largely dependent on the initial variable ordering. Another approach based on "sampling" has been suggested in [17, 12], but dependent on the chosen candidates the quality of the result varies widely, i.e. the results can be up to a factor of two worse than "classical" sifting. Lower bound sifting [8] is an approach to speed up sifting without loss of quality by computing lower bounds during variable reordering. Also a relaxed version has been proposed where a parameter has to be set by the user allowing to trade off runtime versus quality.

As has been observed for model checking applications, the heuristics that work very well in the combinational case might fail for sequential applications [19]. This is the reason why recent work also considers reordering techniques for FSM traversal [11].

The focus of this paper is to show that it is not clever to use the same sifting technique all the time during FSM traversal. Usually, dynamic minimization is called if a node limit is reached. Dependent on the previous results of dynamic minimization or the time elapsed since the last call of sifting, respectively, an history-driven minimization approach automatically chooses one minimization strategy [7]. In this paper we present techniques to choose a reordering strategy during FSM traversal. The algorithm has been integrated in the VIS package [10]. Experiments show that speed-ups of more than 40% can be observed on average.

2. BINARY DECISION DIAGRAMS

As is well-known, each Boolean function $f : \mathbf{B}^n \rightarrow \mathbf{B}$ can be represented by a *Binary Decision Diagram* (BDD) [3], i.e. a directed acyclic graph where a Shannon decomposition is carried out in each node.

A BDD is called *ordered* if each variable is encountered at most once on each path from the root to a terminal node and if the variables are encountered in the same order on all such paths. A BDD is called *reduced* if it does neither contain vertices with isomorphic sub-graphs nor with both edges pointing to the same node. Reduced and ordered BDDs are canonical, i.e. for each Boolean function the BDD can be uniquely determined. Furthermore, for functions

```
unique_find_or_add(node) {
    if (node exists) return node;
    /* a new node has to be created */
    if (next_reorder node limit is reached) {
        call sifting;
        increase next_reorder node limit;
    }
    /* some other test may follow */
    create and return new node;
}
```

Figure 1: Sketch of unique_find_or_add

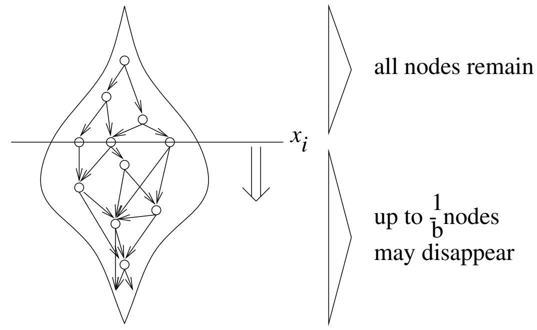

Figure 2: Relaxed lower bound when sifting x_i down

represented by reduced ordered BDDs efficient manipulations are possible [3, 2]. In the following, only reduced, ordered BDDs are considered and for briefness these graphs are called BDDs.

3. SIFTING

The basic operation of dynamic variable ordering is the exchange of adjacent variables [9, 15]. The exchange is performed very quickly since only edges must be redirected within these levels. Thus, the size is optimized without a complete reconstruction of the BDD, i.e. only local transformations for the two levels are performed, since BDDs are a canonical form.

The sifting algorithm [15] successively considers all variables of a given BDD. When a variable is chosen, the goal is to find the best position of the variable, assuming that the relative order of all other variables remains the same. In a first step, the order in which the variables are considered is determined. This is done by sorting the levels according to their size with largest level first. To find the best position, the variable is moved across the whole BDD. In [15], this is done in three steps:

1. The variable is exchanged with its successor variable until it is the last variable in the ordering.

2. The variable is exchanged with its predecessor until it is the topmost variable.

3. The variable is moved back to the closest position which has led to the minimal size of the BDD.

In CUDD [18], a state-of-the-art BDD package, dynamic reordering is applied if a given node limit is reached[1]. This is tested each time a new node is created. If this happens while e.g. ITE is computed [2], ITE is started again after sifting has been carried out. A sketch of the look-up technique used in ITE is shown in Figure 1.

[1] Other packages, like CAL from Berkeley, use different criteria. All algorithms presented in the following are also applicable in this case.

4. SPEEDING UP SIFTING

Some improvements to the original sifting algorithm have already been proposed:

Upper limit: As the size of the BDD can grow much during the movement of one variable, it is possible to set an upper limit to the growth of the BDD. If this limit is exceeded, moving into this direction is aborted. This avoids large intermediate BDDs. Using the notation from [18] in the following this "growing factor" is denoted as $maxgrowth$, e.g. $maxgrowth = 1.2$ means that the BDD may grow by at most 20% of its original size.

Closest end: The considered variable is not always moved downwards first. Instead it is moved to the closest end and then to the opposite one.

Several other techniques have been proposed like the use of symmetries [14] and of interaction matrix[2] [18].

In [8] lower bounds have been used to speed up sifting (so-called *lb-sifting*) by preventing that variables are always moved to both ends of the variable ordering: if a lower bound for the BDD size is larger than the best size found so far, then moving a variable can be canceled. Furthermore, a relaxed lower bound has been considered which is an overestimation of the resulting size, i.e. it is assumed that only $\frac{1}{b}$ ($b \geq 2$) of the nodes affected by the level exchanges may vanish (see Figure 2). This extension allows to trade off runtime versus quality, i.e. the larger b, the faster the algorithm, but the resulting sizes get larger on average.

5. STARTING POINT FOR SIFTING

All these methods try to accelerate *one* call of sifting. To *select* such a method dependent on the history, some basic observations have to be made. In [7] it was shown how a typical BDD construction works. Similar properties hold in the sequential domain. Figure 3 shows data of an example of a BDD construction processduring the computation

[2] For each two variables the interaction matrix provides the information whether there is an output in the BDD which essentially depends on both variables. If two levels do not interact, the level exchange can be computed in constant time.

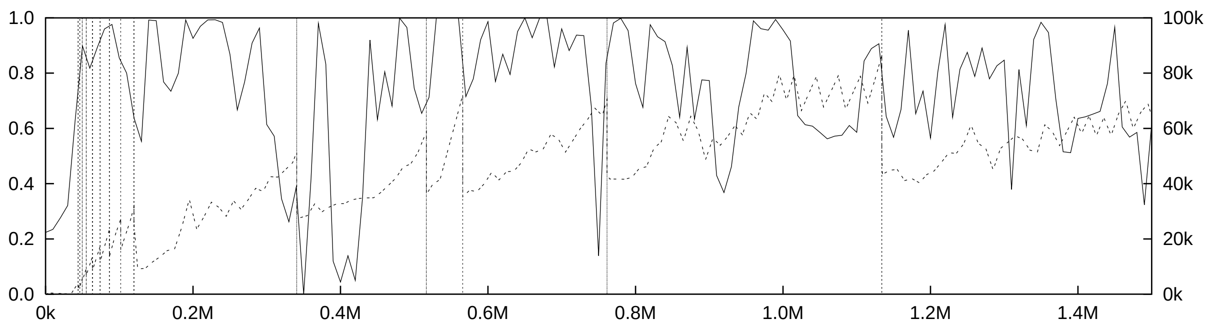

Figure 3: Increase of BDD nodes and total node count for s9234.1 measured in terms of calls to **unique_find_or_add()**

of the set of reachable states. The solid line gives the relative number of nodes which are not found in the hash table during BDD operations, i.e. the percentage of newly created nodes (left y-axis). The x-axis shows the number of calls of **unique_find_or_add()**. The vertical lines show where sifting is called. The total memory consumption counted in number of BDD nodes is given by the dashed line (right y-axis). The following can be observed:

1. The percentage of nodes found in the unique table varies a lot and it is (more or less) randomly distributed and independent of the starting points of sifting.

2. Nevertheless, the behavior is very robust, i.e. there are phases where nodes are found all the time and phases where nodes are found very rarely. Oscillation cannot be observed.

3. At the beginning of the construction sifting gives very good reductions, i.e. often more than a factor of two, while at the end (where the BDDs are large and sifting becomes even more time consuming) the profit is marginal in many cases.

For this, calling the original sifting algorithm each time is not clever, since the algorithm is very slow, especially for large BDDs. In [7] an approach, called *Last Reduction* (LR) has been proposed. Using LR time consuming sifting is always applied, if the last run gave significant reductions. Otherwise a "light" version has been used. This technique was tuned towards symbolic simulation of combinational circuits. In the following we present criteria to choose faster variants of sifting dependent on the *history* of the minimization process in the sequential domain.

6. HISTORY-BASED DYNAMIC MINIMIZATION FOR FSM TRAVERSAL

In this section we present a method to select a reordering heuristic that is called if an upper node limit is reached during the computation of the set of reachable states.

In the beginning of the computation, sifting is called very frequently and large reductions of the size can be observed. On the other hand, improvements are smaller in the long run, and after a few iterations a relatively good order has been found.

This observation leads to the following algorithm: the parameter b of relaxed lb-sifting is chosen dependent on the number of elapsed iterations:

$$b = (\text{number of the current iteration}) \times const,$$

i.e. after many iterations relaxed lb-sifting is used with a large value of b, otherwise it is assumed that the computation has just started, and exact lb-sifting is used. This method directly uses knowledge of a higher level of abstraction, i.e. of the traversal process.

In our experiments it turned out that $const = 8$ is a good choice in order to trade off quality and runtime. We refer to this technique as *EI* (elapsed iterations) in the following.

7. EXPERIMENTAL RESULTS

In this section we describe experimental results that have been carried out on a *SUN Ultra 4*. The algorithms have been integrated in the VIS package [10] using the CUDD package [18] as underlying BDD package. For all our experiments, we used a runtime limit of 2 hours and a memory limit of 256 MB (which was never reached). All runtimes are given in CPU seconds. Node counts are given in 1000 nodes. We used several benchmarks from LGSynth93 to compute the reachable states and measured the BDD peak node count and runtime.

We compared our approach (column *EI*) to the best results obtained using lb-sifting each time (column *sifting*) and to the approach developed for combinational circuits [7] (column *LR*). The results are given in Table 1. For each benchmark the number of iterations computed is given. In columns *states*, *nodes* and *time* the number of reached states, the peak number of BDD nodes, and the time needed in CPU seconds is given, respectively, for each of the three methods. As can be seen both algorithms clearly outperform sifting. Furthermore, the approach based on the number of iterations is superior to *LR* with respect to runtime and space requirement. In some cases *EI* is even the only approach to complete the computation.

All in all, history-based selection of the reordering technique significantly speeds up the computation and often also reduces the memory needed.

Table 1: History-based dynamic minimization

circuit	iterations	sifting			*LR*			*EI*		
		states	nodes	time	states	nodes	time	states	nodes	time
bigkey	3	1.2e+67	35k	181	1.2e+67	49k	242	1.2e+67	42k	211
dsip	16	2.7e+67	45k	231	2.7e+67	50k	334	2.7e+67	75k	230
ecc	80	4.5e+14	113k	249	4.5e+14	145k	425	4.5e+14	196k	209
gcd	264	—	—	—	1.4e+10	2110k	5821	1.4e+10	2174k	3570
mm30a	5	2.1e+26	238k	107	2.1e+26	238k	133	2.1e+26	238k	129
parker1986	5	4.4e+12	779k	2964	4.4e+12	679k	4167	4.4e+12	679k	2803
parker1986	6	—	—	—	—	—	—	1.1e+15	679k	5004
s1423	9	4.9e+08	874k	2494	4.9e+08	630k	787	4.9e+08	661k	492
s1423	10	1.7e+09	1346k	5288	1.7e+09	1192k	2744	1.7e+09	1556k	1241
s1423	11	—	—	—	8e+09	2925k	5565	8e+09	3453k	3653
s5378	3	1.7e+12	533k	2098	1.7e+12	163k	617	1.7e+12	139k	311
s5378	4	—	—	—	2.6e+15	413k	2898	2.6e+15	643k	2583
s5378	5	—	—	—	1.2e+16	694k	5123	1.2e+16	715k	5072
s9234.1	9	6.4e+10	282k	887	6.4e+10	326k	966	6.4e+10	232k	309
s9234.1	10	1.6e+11	470k	1607	1.6e+11	487k	1487	1.6e+11	553k	663
s9234.1	11	1.3e+12	629k	2960	1.3e+12	782k	2429	1.3e+12	827k	1091
s9234.1	12	2.5e+13	1284k	6427	2.5e+13	1609k	6170	2.5e+13	1513k	2011

8. CONCLUSIONS

In this paper a history-based dynamic minimization technique for FSM traversal has been proposed. Instead of applying the same reordering algorithm all the time, dependent on the history a parameter for the reordering algorithm is determined and a faster heuristic may be used in some cases. Experimental results carried out in VIS are given that clearly show that for sequential circuits different methods have to be applied. The heuristic proposed takes the number of iterations considered so far into account and by this computes states for which other methods fail.

9. REFERENCES

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

[2] K.S. Brace, R.L. Rudell, and R.E. Bryant. Efficient implementation of a BDD package. In *Design Automation Conf.*, pages 40–45, 1990.

[3] R.E. Bryant. Graph - based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[4] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[5] O. Coudert, C. Berthet, and J.C. Madre. Verification of sequential machines using Boolean functional vectors. In *Proceedings IFIP International Workshop on Applied Formal Methods for Correct VLSI Design*, pages 111–128, 1989.

[6] O. Coudert and J.C. Madre. A unified framework for the formal verification of sequential circuits. In *Int'l Conf. on CAD*, pages 126–129, 1990.

[7] R. Drechsler and W. Günther. History-based dynamic minimization during BDD construction. In *X IFIP Int'l Conf. on VLSI*, pages 334–345, 1999.

[8] R. Drechsler and W. Günther. Using lower bounds during dynamic BDD minimization. In *Design Automation Conf.*, pages 29–32, 1999.

[9] M. Fujita, Y. Matsunaga, and T. Kakuda. On variable ordering of binary decision diagrams for the application of multilevel synthesis. In *European Conf. on Design Automation*, pages 50–54, 1991.

[10] The VIS Group. VIS: A system for verification and synthesis. In *8th Int'l Conf. on Computer Aided Verification*, LNCS 1102, pages 428–432, 1996.

[11] H. Higuchi and F. Somenzi. Lazy group sifting for efficient symbolic state traversal of FSMs. In *Int'l Conf. on CAD*, pages 45–49, 1999.

[12] J. Jain, W. Adams, and M. Fujita. Sampling schemes for computing OBDD variable orderings. In *Int'l Conf. on CAD*, pages 631–638, 1998.

[13] C. Meinel and A. Slobodová. Speeding up variable reordering of OBDD. In *Int'l Conf. on Comp. Design*, pages 338–343, 1997.

[14] S. Panda and F. Somenzi. Who are the variables in your neighborhood. In *Int'l Conf. on CAD*, pages 74–77, 1995.

[15] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Int'l Conf. on CAD*, pages 42–47, 1993.

[16] D. Sieling. The nonapproximability of OBDD minimization. Technical report, Forschungsbericht No. 663, Universität Dortmund, 1998.

[17] A. Slobodová and C. Meinel. Sample method for minimization of OBDD. In *Int'l Workshop on Logic Synth.*, pages 311–316, 1998.

[18] F. Somenzi. *CUDD: CU Decision Diagram Package Release 2.3.0.* University of Colorado at Boulder, 1998.

[19] B. Yang, R.E. Bryant, D.R. O'Hallaron, A. Biere, O. Coudert, G. Janssen, R.K. Ranjan, and F. Somenzi. A performance study of BDD-based model checking. In *Proceedings of Formal Methods in Computer-Aided Design, LNCS 1522*, pages 255–289, 1998.