Lazy Group Sifting for Efficient Symbolic State Traversal of FSMs

Hiroyuki Higuchi*
Fujitsu Laboratories Ltd.
Kawasaki, Japan

Fabio Somenzi
University of Colorado
Boulder, CO

Abstract

This paper proposes lazy group sifting for dynamic variable reordering during state traversal. The proposed method relaxes the idea of pairwise grouping of present state variables and their corresponding next state variables. This is done to produce better variable orderings during image computation without causing BDD size blowup in the substitution of next state variables with present state variables at the end of image computation. Experimental results show that our approach is more robust in state traversal than the approaches that either unconditionally group variable pairs or never group them.

1 Introduction

State traversal of Finite State Machines (FSMs) is an important technique for logic verification, synthesis, and test generation of sequential circuits. As circuits become larger, more efficient state traversal techniques are required.

State traversal of an FSM from a given set of initial states computes the set of all the reachable states. Explicit approaches are not applicable to large FSMs because the number of the reachable state sets may be exponential to the number of flip-flops. Symbolic traversal [1] represents both state transition relations and state sets with Binary Decision Diagrams (BDDs) and implicitly traverses state transition graphs in breadth-first manner by iteratively computing the images until a fixed point is reached. Symbolic traversal has been refined in many successive works (e.g., [2, 3, 4, 5]). In most practical examples, the symbolic approach is able to handle significantly larger FSMs than the explicit ones.

The efficiency of symbolic traversal largely depends on the sizes of the BDDs it handles. Since variable ordering has a significant effect on the BDD size, good variable ordering is indispensable to efficient symbolic traversal. Although several static ordering heuristics for symbolic traversal have been proposed [2, 6], it is difficult to find a good static ordering for the state transition relations and every image computation. Dynamic variable reordering [7] of BDDs is a widely used technique in sequential verification using BDDs. The approach drastically reduces BDD sizes during symbolic traversal, mainly because each image computation handles different state sets in general and requires different variable orderings. However, it is still not possible to handle many examples. This has led to a recent increase in the attention paid to variable reordering for efficient sequential verification [8].

In symbolic traversal based on partitioned transition relations [2, 3], a present state variable and its corresponding next state variable (henceforth referred to as a *state pair*) are usually kept adjacent in the variable order throughout the computation. This means that the size of the BDD representing the image does not change in the substitution of the next state variables with the present state variables at the end of image computation. It also makes it possible to do the substitution in a piecemeal fashion, by combining it with the And-Exists operations. However, the state pair grouping approach restricts the reduction ability of dynamic reordering especially during image computation. On the other hand, not grouping the state pairs may cause BDD size explosion in the substitution at the end of each image computation. This paper proposes a lazy group sifting for dynamic variable reordering during state traversal. The proposed method relaxes state pair grouping criteria to produce better variable orderings during image computation without causing BDD size blowup in the substitution of next state variables with present state variables at the end of image computation.

It is well known that pairwise grouping of present state variables with their corresponding next state variables is generally a good heuristic for dynamic variable reordering in sequential verification [9]. However our experimental results indicate that lazy group sifting can be more robust than the approaches that either unconditionally group variable pairs or never group them.

This paper is organized as follows. Section 2 introduces the terminology and the notation; and it then summarizes the basic symbolic traversal algorithm and the dynamic variable reordering algorithm. Section 3 describes the lazy group sifting approach for efficient symbolic traversal. Section 4 is devoted to the experimental results. Section 5 presents conclusions and summarizes future work.

2 Preliminaries

2.1 Finite State Machine (FSM)

A *finite state machine* M is a 6-tuple $M = (I, O, S, \delta, \lambda, S_0)$, where I, O, and S are finite nonempty sets of inputs, outputs, and states, respectively; $\delta : I \times S \to S$ is the state transition function; $\lambda : I \times S \to O$ is the output function; and $S_0 \subseteq S$ is the initial state set. Without loss of generality, we only consider completely specified FSMs.

In this paper we assume that all the input, output, and state symbols are encoded. Let $w = (w_1, w_2, \ldots, w_m)$, $x = (x_1, x_2, \ldots, x_n)$, and $y = (y_1, y_2, \ldots, y_n)$ be input vector, present state vector, and next state vector, respectively. The state transition function for the i-th next state variable can be written as $y_i = f_i(w, x), 1 \leq i \leq n$. We denote the state transition function vector f by $f(w, x) = (f_1(w, x), f_2(w, x), \ldots, f_n(w, x))$.

An FSM is an abstract model describing the behavior of a sequential circuit. Conversely, a sequential circuit can be viewed as an implementation of an encoded FSM.

2.2 Set Representation using BDDs

BDDs [10] are used to represent and manipulate Boolean functions. To represent a set $A \subseteq \{0, 1\}^n$ by a Boolean function, we define the *characteristic function* χ_A of set A as follows:

$$(x_1, x_2, \ldots, x_n) \in A \Leftrightarrow \chi_A(x_1, x_2, \ldots, x_n) = 1.$$

BDDs are used to represent and manipulate state sets by means of their characteristic functions.

*This work was done when Hiroyuki Higuchi was with the University of Colorado at Boulder.

```
Traverse($S_0(x)$, $f(w,x)$) {
1.  $Reach(x) = From(x) = S_0(x)$;
2.  while (1) {
3.      $To(y) \leftarrow Image(From(x), f(w,x))$;
4.      $To(x) \leftarrow To(y)|_{y \leftarrow x}$;
5.      $New(x) \leftarrow To(x) - Reached(x)$;
6.      if ($New(x) = \emptyset$) return $Reached(x)$;
7.      $Reached(x) \leftarrow Reached(x) + To(x)$;
8.      $From(x) \leftarrow BestBdd(New(x), Reached(x))$;
9.  }
}
```

Figure 1: Basic state traversal algorithm.

2.3 Symbolic Traversal

The traversal of the state space of a sequential circuit is done by repeated computation of an image. The image of state set *From* by state transition function vector $f = (f_1, f_2, \ldots, f_n)$ is the set $Image(From, f) = \{y \in \{0,1\}^n | y = f(w,x), w \in \{0,1\}^m, x \in From\}$. The image can be calculated as follows:

$$Image(From, f) = \exists w \exists x [From \cdot \prod_{i=1}^{n} (y_i \equiv f_i)].$$

$T(w,x,y) = \prod_{i=1}^{n} (y_i \equiv f_i)$ is called the *transition relation.* Since T is often too large, *partitioned transition relations* $T_i (i = 1, \ldots, r)$ are used, where $T = \prod_{i=1}^{r} T_i$. Image computation based on partitioned transition relations is done by iterating the following operation for $i = 1, \ldots, r$:

$$P \leftarrow \exists s (P \cdot T_i). \quad (1)$$

Here s are the variables that can be quantified after the conjunction. P is an intermediate result of image computation and is called a *partial product.*

The basic flow of state traversal is shown in Fig.1. Here $BestBdd(f, g)$ returns a function h with a small BDD such that $f \subseteq h \subseteq g$. Note that the result of $Image(From, f)$ is in terms of y variables. To handle large sets of states, BDDs are used for efficiency. State traversal using BDDs is called symbolic traversal.

2.4 Dynamic Variable Reordering

Since variable ordering has a significant impact on the BDD size, dynamic variable reordering is indispensable to efficient symbolic traversal. It is generally based on the sifting algorithm [7], in which each variable in turn is moved up and down to greedily find the best among all possible positions. The algorithm of basic dynamic variable reordering is as follows:

Repeat (1) and (2) for each variable in some order:

(1) Sift the variable up and down, remembering the total BDD node size for each permutation thus generated.

(2) Select the best permutation such that the total BDD size is minimum. □

Dynamic variable reordering based on the sifting algorithm often produces good orders but tends to be time consuming. To speed up sifting, several techniques have been proposed [11, 12, 8]. These methods attempt to restrict the search space of sifting to reduce computation time. In [11] variables are moved only within fixed blocks which are determined from structural analysis of the BDDs. In [12] a small BDD sample is chosen from the entire BDDs that are considered for minimization. Then sifting is done on the sample and the resulting order is applied to the original BDDs. The approach of [8] is intended for symbolic model checking. The method selects variables that need to be repositioned and reorders only such variables. Our approach is also intended for sequential verification. However, we address another issue in reordering for state traversal. We address how to handle the present state variables and the next state variables. Since the technique we propose in this paper is orthogonal to the above approaches, it is possible to combine our technique with them.

2.5 Present and Next State Variable Grouping

In many model checkers, a present state variable and its corresponding next state variable are grouped and are always kept adjacent in the order during state traversal. We call this approach the *state pair grouping* approach. To handle groups of variables instead of single variables in dynamic variable reordering, group sifting is used. Group sifting consists of moving a group of variables, instead of a single variable. There is another conventional approach, in which none of the state pairs are grouped and each variable is sifted freely in reordering. We call this approach the *state pair ungrouping* approach. The merit of state pair grouping is that the relative order of the present state variables is the same as that of the next state variables. This means that the size of the BDD representing the image does not change in the substitution of next state variables with present state variables at the end of each image computation (Line 4 in Fig.1). However, state pair grouping restricts the reduction ability of dynamic reordering during image computation. There are a lot of functions whose BDD sizes are drastically larger in the interleaved order than the minimum sizes. Therefore state pair grouping may cause a blowup of the partial product BDD during image computation. On the other hand, state pair ungrouping may cause BDD size explosion in the substitution at the end of each image computation. This means that additional reordering may take place in each substitution. The additional reordering may cause further reordering in the next image computation, because the good intermediate order during image computation is discarded. Furthermore, a non-interleaved order requires more work to do the substitution itself.

2.6 Group Sifting

In this paper the group sifting scheme based on [13] is utilized. In the scheme two types of groups, hard groups and soft groups, are considered. A *hard group* is a group passed to the reordering procedure by the caller. In general hard groups can be nested. The state pair grouping approach can be considered as hard grouping of all the state pairs. A *soft group* is a group created by the reordering procedure. A variable is sifted up and down, while it is checked against its adjacent variable for soft grouping. If the test succeeds, a soft group is created. Soft groups are dissolved at the end of the reordering procedure.

Group sifting attempts to find a group of variables that should be sifted together. Several criteria have been proposed for soft grouping: symmetry[14], extended symmetry, and the method of the second difference [13]. These criteria are application-independent. In the next section we introduce new criteria intended for state traversal and sequential verification in general.

3 Lazy Group Sifting

In this section a lazy group sifting approach for dynamic variable reordering is proposed. The lazy group sifting proposed here relaxes the state pair grouping criterion. We will discuss how to relax the grouping criterion in detail.

The state pair grouping approach always keeps the state pair adjacent for the substitution of y variables with x variables, while the state pair ungrouping approach sifts variables freely. Therefore the state pair ungrouping is likely to produce better variable orderings during image computation. In the state pair ungrouping approach, however, the relative order of the present state variables may be totally different from that of the next state variables. There is a tradeoff between BDD sizes in image computation and those in the substitution at the end of image computation. Lazy group sifting is an in-between approach. It consists of two techniques:

a) selecting state pairs to be grouped or not grouped,

b) taking into account the distance between the variables of ungrouped state pairs.

Lazy group sifting attempts to keep state pairs as close as possible if the size of partial product BDD, which is often the largest BDD in state traversal, does not increase.

3.1 Selecting State Pairs To Be Grouped

In this subsection we describe which state pairs should be grouped. Groups consist of hard groups and soft groups. We also select state pairs that will remain ungrouped throughout the state traversal.

3.1.1 Hard Grouping

First we create hard groups. We select state pairs (x_i,y_i) for hard grouping such that:

- The corresponding flip-flop (FF) is a λ-FF [4], that is, no next state function depends on x_i, or
- The next state function f_i depends on x_i, and x_i is the only present state variable on which f_i depends.

In the first case, x_i can be quantified before image computation, and thereby hard grouping of x_i and y_i does not increase total BDD sizes. In the second case, x_i and y_i should be grouped even in image computation. This analysis can be done before state traversal.

3.1.2 Hard Ungrouping

Some next state functions do not depend on the corresponding present state variables. We call them *independent state pairs*. For example, pipelined circuits have many independent state pairs. There is no need to keep independent state pairs adjacent during image computation, because they are not related to each other. Since image computation requires more computational effort than the substitution of y variables with x variables at the end of image computation, it is better not to group independent state pairs. In lazy sifting, independent state pairs are ungrouped before state traversal and never grouped throughout the traversal. Even when state pairs are ungrouped, the distances between the present state variables and their corresponding next state variables are taken into account, as will be explained in Subsection 3.2.

3.1.3 Soft Grouping

A soft group is a group created by the reordering algorithm [13]. Lazy group sifting tries to create soft groups of present state variables and their corresponding next state variables using the following criteria.

Variables that become adjacent during sifting are tested to see whether (a) they are corresponding present and next state variables and (b) they should be grouped. Sifting may then continue with the group instead of the single variable. The groups are dissolved at the end of the sifting procedure.

Now the problem is which variables should be grouped in this framework. Grouping present state variables with their corresponding next state variables may impair the reduction ability. However, some state variables have not been introduced yet in the partial product P in Eq.1, or have been quantified out of it when reordering takes place. Such variables can be grouped, because they do not appear in the support of the partial product, in which many present and next state variables may interact. Specifically, the lazy group sifting algorithm outlined above can be detailed as follows:

Repeat (0) to (2) for each variable in some order:

(0) Remember the current total BDD node size N.

(1') Sift the variable v up and down, remembering the total BDD node size for each permutation *and grouping v with the adjacent variable w if all the following conditions are satisfied:*

- *They are corresponding present and next state variables,*
- *Current total BDD node size is not larger than N,*
- *w is not in the support of the partial product.*
- *w has already been sifted,*

(2) Select the best permutation such that the total BDD size is minimum.

The idea behind this procedure is that a state variable should be soft-grouped with the corresponding variable if the grouping does not increase the partial product BDD size. When a state variable is not in the support of the partial product, the position of the variable does not affect the partial product BDD size. Therefore it should be grouped with the corresponding variable. The second condition for total BDD sizes in Step 1' guarantees that the resulting total BDD size will not increase as a result of reordering.

3.2 Taking into Account the Distance Between State Pairs

It is better to keep a present state variable and its corresponding next state variables as close as possible even when they are not grouped, if the total node size does not increase. This is done by modifying Step 2 in Subsection 3.1.3. The modified step is as follows:

(2') Select the best permutation such that the total BDD size is minimum. *In the case of a tie, select the closest position to the corresponding state variable.* □

Instead of Step 2', one can use the following Step 2" when BDD sizes still blow up during the substitution of next state variables with present state variables:

(2") Among the permutations in which the total node size is not larger than $\phi(\mathit{minimum}, \varepsilon)$, select the best permutation such that the distance of the state variable and its corresponding variable is minimum.

Here ε is a small number to be given as a parameter. ϕ could be $\mathit{minimum} + \varepsilon$, $\mathit{minimum} \times (1 + \varepsilon)$, or N.

3.3 Positioning Variables To Be Introduced

Since next state variables are introduced by a series of And-Exists operations in image computation based on partitioned transition relations, some next state variables may only appear in the support of partitioned transition relations but not in the partial product, when reordering takes place. The reordering cannot find a good place for the variables in terms of the partial product BDD size. Positioning such variables affects the BDD sizes in subsequent steps.

In state pair ungrouping, those next state variables are placed such that the BDD size of partitioned transition relations depending on the variables is minimal. On the other hand, state pair grouping puts them next to their corresponding present state variables. Lazy group sifting puts them next to their corresponding variables only if total BDD size is not larger than that in the initial position.

4 Experimental Results

We implemented lazy sifting in VIS [15] with the CUDD package [16]. Experiments were carried out on a 400MHz Pentium II machine with 1GB of RAM. We used several ISCAS'89, ISCAS'89-addendum benchmark circuits and some other examples. "s5378opt" is a version of s5378 optimized by sequential redundancy removal. "bpb" is a branch prediction buffer that works in two stages. "cps1364" is a flat description of a landing gear controller. "sfeistel" is a cryptography circuit. "soap" is a circuit which implements a distributed mutual exclusion algorithm [17]. In our experiments we used VIS with the default settings. Good variable orders for the partitioned transition relations are used as initial orders. Automatic dynamic variable reordering is enabled after the partitioned transition relations are built.

Table 1 compares reachability analysis with the lazy group sifting method ("lazy") against those with the state pair grouping approach ("group") and the state pair ungrouping approach ("ungroup"). Method "lazy" utilizes the algorithm in Section 3, where $\phi = minimum$. Column 1 shows the name of the circuit. "s3271-8" means that the traversal of s3271 is partial and is done for only 8 steps, i.e. the sequential depth is 8. Column 2 shows the number of flip-flops in the circuit. Columns 3–5 compare the peak memory sizes of the reachability analysis. Columns 6–8 compare the peak numbers of live BDD nodes. Columns 9–11 compare run times in seconds. The figures in bold face show the best results among the three methods. The bottom row shows the arithmetic mean, when each best result is assumed to be 1.

The experimental results show that the proposed approach works well. As for peak BDD nodes, "lazy" obtains the best result for all examples except for "s1269", where no reordering takes place in the substitution of y variables with x variables even in state pair ungrouping. This shows that lazy sifting can reduce peak BDD nodes. As for run times, "lazy" obtains the best result for most examples. Group sifting sometimes produces much worse results than state pair ungrouping and lazy sifting, as one can see in "s4863" and "s5378-4". This means that interleaved orders are not suitable for some circuits. On the other hand, state pair ungrouping requires much more computation cost than state pair grouping and lazy group sifting in "s3271", "cps1364", and "sfeistel." These results indicate that lazy group sifting is more robust than the state pair grouping or the state pair ungrouping approaches. It may be worth noting that the apparently not so great performance of lazy sifting in terms of peak memory is partly due to CUDD allocating memory based not only on need, but also on availability.

Table 2 shows statistics of dynamic variable reordering in reachability analysis. Columns 2–4 compare times in seconds for dynamic reordering in reachability analysis. Data in parentheses show times in seconds for dynamic reordering during the substitution of the next state variables with the present state variables at the end of image computation. Columns 5–7 compare the numbers of reordering invoked. Data in parentheses show the numbers of reordering invoked during the substitution at the end of image computation. Columns 8–10 compare the maximum ratios of BDD sizes of image sets after the substitution over those before the substitution. Only the substitutions in which no reordering took place are considered. "–" shows that there are no such substitutions, that is, in every substitution variable reordering took place.

Table 2 indicates that lazy group sifting reduces times for reordering not only in the substitution at the end of image computation but also in image computation. Compared to "ungroup", the number of reorderings invoked during the substitution are reduced in lazy sifting in many cases. Even when no dynamic reordering took place in the substitution, the maximum ratios of BDD sizes of image sets after the substitution over those before the substitution are reduced in almost all the cases. These results shows that, compared to the state pair ungrouping approach, lazy sifting efficiently reduces the BDD sizes both in the image computation and in the substitution.

Table 3 shows the numbers of hard/soft-grouped state pairs in lazy sifting. Columns 3–4 show the number of the total hard-grouped state pairs. Column 3 shows the number of λ-FFs. Column 4 shows the number of state pairs (x_i, y_i) such that x_i is the only present state variable on which the next state function f_i depends. Columns 5 shows the number of hard-ungrouped state pairs. Column 6 shows the average number of soft-grouped state pairs. These results indicate that the state pair grouping approach is not effective for examples with many hard-ungrouped pairs, e.g., "s4863" and "s5378-4", and that lazy sifting is likely to group many state pairs on the examples in which the state pair grouping approach is more effective than the state pair ungrouping approach, e.g., "s3271-8" and "cps1364" .

5 Conclusion

In this paper the lazy group sifting approach for dynamic variable reordering during state traversal has been proposed. The proposed method relaxes the idea of pairwise grouping of present state variables and their corresponding next state variables. This is done to maximize the flexibility of sifting variables of BDDs during image computation without causing BDD size blowup in the substitution of next state variables with present state variables after each image computation. Experimental results indicate that the proposed approach works well. This paper shows the possibility of combining the state pair grouping approach with the state pair ungrouping one.

Future work is to investigate more sophisticated heuristics for lazy group sifting criteria. One approach is to find a good heuristic for determining the value ε in Step 2" shown in Section 3.2. We are also interested in estimating the "right" position of variables that are not yet introduced in the partial product. Another direction for future work is to show the effectiveness of our method in model checking.

References

[1] O. Coudert, C. Berthet, and J. C. Madre. Verification of sequential machines using boolean functional vectors. In L. Claesen, editor, *Proceedings IFIP International Workshop on Applied Formal Methods for Correct VLSI Design*, pages 111–128, Leuven, Belgium, November 1989.

[2] H. Touati, H. Savoj, B. Lin, R. K. Brayton, and A. Sangiovanni-Vincentelli. Implicit enumeration of finite state machines using BDD's. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 130–133, November 1990.

[3] J. R. Burch, E. M. Clarke, and D. E. Long. Representing circuits more efficiently in symbolic model checking. In *Proceedings of the Design Automation Conference*, pages 403–407, San Francisco, CA, June 1991.

[4] G. Cabodi, P. Camurati, and S. Quer. Improved reachability analysis of large finite state machines. In *Proceedings of the International Conference on Computer-Aided Design*, pages 354–360, Santa Clara, CA, November 1996.

[5] A. Narayan, A. J. Isles, J. Jain, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Reachability analysis using partitioned ROBDDs. In *Proceedings of the International Conference on Computer-Aided Design*, pages 388–393, November 1997.

[6] S.-W. Jeong, B. Plessier, G. D. Hachtel, and F. Somenzi. Variable ordering and selection for FSM traversal. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 476–479, Santa Clara, CA, November 1991.

[7] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proceedings of the International Conference on Computer-Aided Design*, pages 42–47, Santa Clara, CA, November 1993.

[8] G. Kamhi and L. Fix. Adaptive variable reordering for symbolic model checking. In *Proceedings of the International Conference on Computer-Aided Design*, pages 359–365, San Jose, CA, November 1998.

[9] 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 G. Gopalakrishnan and P. Windley, editors, *Formal Methods in Computer Aided Design*, pages 255–289. Springer-Verlag, November 1998. LNCS 1522.

circuit	FF	Peak Mem.(MB)			Peak BDD Live Nodes			time(sec.)		
		lazy	group	ungroup	lazy	group	ungroup	lazy	group	ungroup
s1269	37	287	**262**	363	3,216,445	2,644,876	**2,644,210**	**2,463**	4,432	2,849
s1423-10	74	133	**102**	253	**1,586,112**	1,980,587	2,214,442	**3,904**	5,073	7,023
s1512	57	**64**	68	65	**112,597**	147,632	159,372	**753**	1,142	1,179
s3271-8	116	**160**	169	238	**1,668,808**	1,958,046	2,311,196	2,357	**2,096**	6,325
s3330	132	**250**	278	297	**1,611,639**	1,930,568	1,661,904	**2,056**	2,404	4,075
s4863	104	74	**65**	143	**385,161**	591,278	400,872	**529**	12,000	1,137
s5378opt	121	153	**90**	161	**354,165**	475,777	593,286	**1,413**	3,349	1,556
s5378-4	179	191	350	**135**	**878,318**	2,590,726	926,990	**3,107**	18,711	3,546
bpb	36	**15**	16	28	**36,907**	49,319	215,752	**44**	82	393
cps1364	231	**140**	176	286	**903,844**	1,082,988	1,760,932	4,064	**3,757**	9,812
sfeistel	293	**37**	**37**	**37**	**151,705**	**151,705**	**151,705**	**241**	269	870
soap	140	32	**27**	31	**132,713**	147,364	148,307	175	167	**158**
avg.		1.15	1.18	1.55	1.01	1.37	1.69	1.03	3.57	2.51

Table 1: Experimental results for reachability analysis.

circuit	times for reordering			# of reordering			max $\lvert To(x)\rvert / \lvert To(y)\rvert$		
	lazy	group	ungroup	lazy	grp	ungroup	lazy	grp	ungroup
s1269	2252(0)	4177(0)	2522(0)	12(0)	14(0)	17(0)	1.09	1	3.74
s1423-10	3813(0)	4977(0)	6893(0)	11(0)	11(0)	13(0)	1.96	1	1.21
s1512	93(0)	77(0)	119(0)	9(0)	9(0)	10(0)	1.25	1	1.18
s3271-8	2276(0)	2011(0)	6170(21)	13(0)	13(0)	23(3)	1.37	1	2.44
s3330	1584(97)	1868(469)	3464(1297)	13(3)	11(3)	17(7)	2.07	1	2.96
s4863	498(273)	837(0)	829(420)	10(4)	11(0)	13(5)	3.95	1	13.35
s5378opt	416(19)	780(140)	832(790)	12(2)	10(1)	15(10)	1.50	1	10.92
s5378-4	2916(774)	18122(0)	3382(989)	27(7)	23(0)	25(7)	–	1	–
bpb	17(3)	12(0)	123(0)	5(1)	4(0)	8(0)	1.31	1	2.14
cps1364	3849(0)	3622(0)	9536(0)	5(0)	5(0)	8(0)	1.26	1	1.25
sfeistel	258(168)	282(0)	869(297)	8(5)	8(0)	18(7)	1.86	1	1.61
soap	128(0)	133(0)	88(67)	4(0)	4(0)	4(2)	1.31	1	1.58

Table 2: Statistics of dynamic reordering in reachability analysis.

[10] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[11] C. Meinel and A. Slobodova. Speeding up variable reordering of OBDDs. In *Proceedings of the International Conference on Computer Design*, pages 338–343, Austin, TX, October 1997.

[12] A. Slobodova and C. Meinel. Sample method for minimization of OBDDs. Presented at IWLS98, Lake Tahoe, CA., June 1998.

[13] S. Panda and F. Somenzi. Who are the variables in your neighborhood. In *Proceedings of the International Conference on Computer-Aided Design*, pages 74–77, San Jose, CA, November 1995.

[14] S. Panda, F. Somenzi, and B. F. Plessier. Symmetry detection and dynamic variable ordering of decision diagrams. In *Proceedings of the International Conference on Computer-Aided Design*, pages 628–631, San Jose, CA, November 1994.

[15] R. K. Brayton et al. VIS. In *Formal Methods in Computer Aided Design*, pages 248–256. Springer-Verlag, Berlin, November 1996. LNCS 1166.

[16] F. Somenzi. *CUDD: CU Decision Diagram Package*. ftp://vlsi.colorado.edu/pub/.

[17] J. Desel and E. Kindler. Proving correctness of distributed algorithms using high-level Petri nets: A case study. In *International Conference on Application of Concurrency to System Design*, Aizu, Japan, March 1998.

circuit	FFs	hard pairs		hard ungrouped	soft pairs (avg.)
		λFFs	others		
s1269	37	1	8	8	5.6
s1423-10	74	2	1	1	49.5
s1512	57	0	11	0	27.2
s3271-8	116	1	26	4	111.2
s3330	132	12	0	3	90.0
s4863	104	0	0	104	0
s5378opt	121	37	0	82	1.7
s5378-4	179	17	0	161	1.0
bpb	36	0	16	0	17.2
cps1364	231	0	0	7	85.7
sfeistel	293	0	0	0	253.5
soap	140	0	0	24	55.0

Table 3: Numbers of hard/soft groups in lazy sifting.