Minimizing ROBDD Sizes of Incompletely Specified Boolean Functions by Exploiting Strong Symmetries[1]

Christoph Scholl, *Universität Freiburg*
Stefan Melchior, *Universität des Saarlandes*
Günter Hotz, *Universität des Saarlandes*
Paul Molitor, *Universität Halle*[2]

Abstract

We present a method computing a minimum sized partition of the variables of an incompletely specified Boolean function into symmetric groups. The method can be used during minimization of ROBDD*s of incompletely specified Boolean functions. We apply it as a preprocessing step of symmetric sifting presented by Panda (1994) and Möller (1994) and of techniques for* ROBDD *minimization of incompletely specified Boolean functions as presented by Chang (1994) and Shiple (1994). The technique is shown to be very effective: it improves* ROBDD *sizes of symmetric sifting by a factor of 51% and by a factor of 70% in combination with a slightly modified version of the technique of Chang and Shiple.*

[1]This work was supported in part by DFG grant Mo645/2-1, SFB 124, and the Graduiertenkolleg of the Universität des Saarlandes

[2]Contact address: Institut für Informatik, Universität Halle, D-06099 Halle, Germany

1 Introduction

Binary Decision Diagrams (BDDs) as a data structure for representation of Boolean functions were first introduced by Lee (1959) and further popularized by Akers (1978) and Moret (1982). In the restricted form of reduced ordered BDDs (ROBDDs) they gained widespread application because ROBDDs are a canonical representation and allow efficient manipulations (Bryant 1986). Some fields of application are logic verification, test generation, fault simulation, and logic synthesis (Malik 1988, Bryant 1992). Most of the algorithms using ROBDDs have running time polynomial in the size of the ROBDDs. The sizes depend on the variable order used.

The existing heuristic methods for finding good variable orders can be classified into two categories: initial heuristics which derive an order by inspection of a logic circuit (Malik 1988, Fujita 1988, Fujita 1991) and dynamic reordering heuristics which try to improve on a given order (Ishiura 1991, Rudell 1993, Felt 1993, Bollig 1995, Drechsler 1995). Sifting introduced by Rudell (1993) has emerged so far as the most successful algorithm for dynamic reordering of variables. This algorithm is based on finding the optimum position of a variable, assuming all other variables remain fixed. The position of a variable in the order is determined by moving the variable to all possible positions while keeping the other variables fixed. As already observed by Panda (1995), one limitation of sifting, however, is that it uses the absolute position of a variable as the primary objective, and only considers the relative positions of groups of variables indirectly.

Recently, it has been shown by Möller (1994) and Panda (1994) that symmetry properties can be used to efficiently construct good variable orders for ROBDDs using modified gradual improvement heuristics. The crucial point is to locate the symmetric variables side by side and to treat them as fixed block. This results in *symmetric sifting* which sifts symmetric groups simultaneously[‡]. Regular sifting usually puts symmetric variables together in the order, but the symmetric groups tend to be in sub-optimal positions. The sub-optimal solutions result from the fact that regular sifting is unable to recognize that the variables of a symmetric group have a strong attraction to each other and should be sifted together. When

[‡]Symmetric sifting is very efficient but does not result in optimal orders in any case as proven by Möller (1994) and Sieling (1995)

a variable of a symmetric group is sifted by regular sifting, it is likely to return to its initial position due to the attraction of the other variables of the group (cf. Panda 1995).

The papers mentioned above only handle completely specified functions. But in many applications (e.g. checking the equivalence of two finite state machines (FSMs) (Coudert 1989), minimizing the transition relation of an FSM or logic synthesis for FPGA realizations (Lai 1994, Wurth 1995, Scholl 1995)) incompletely specified Boolean functions play an important role. In applications where ROBDD sizes have a large influence on the quality of the results (such as logic synthesis for FPGA realizations) there is a strong need for ROBDD minimization techniques for incompletely specified functions. [§]

To the best of our knowledge, no variable ordering algorithm exploiting don't cares has been presented in literature. A couple of papers, e.g., Chang (1994) and Shiple (1994) investigate the ROBDD minimization problem for incompletely specified Boolean functions. They start with a fixed variable order obtained by any ordering heuristics and greedily minimize the number of nodes at every level by assigning as few don't cares as possible to either the on-set or the off-set. The variable order remains fixed during this process. However, the resulting ROBDD sizes heavily depend on the variable order. Thus, there is a need to determine good variable orders in the case of incompletely specified functions, too.

As determining the symmetric groups before applying sifting has been proven to result in good variable orders for completely specified functions, it seems to be a good idea in the case of incompletely specified functions to first determine symmetric groups, then to apply symmetric sifting and techniques as those from Chang (1994) and Shiple (1994). However, the symmetric groups of incompletely specified functions are not uniquely defined. We will give some counterexamples. Therefore we have to ask for good partitions into symmetric groups with respect to ROBDD minimization and their computation.

Kim and Dietmeyer (1991) presented an algorithm, which decides for an incompletely specified Boolean function (represented by a cube array), whether a given set λ of input variables forms a symmetric group or not. However, for our problem to partition the input variables into symmetric groups there remain two difficul-

[§]The effect of our ROBDD minimization techniques to the quality of the results of an FPGA synthesis algorithm will be subject of a seperate paper.

ties: first the question, how to find large candidate sets λ (of course, we cannot test for each subset of the variables whether it is a symmetric group) and secondly the question, how to combine symmetric groups to a partition of the input variables, such that the incompletely specified function is symmetric in each set of the partition *at the same time* (in Section 3 we will show that this cannot be done in a straightforward manner). To the best of our knowledge, no technique has been developed so far that targets on computing minimal partitions into symmetric groups for incompletely specified functions.

The paper is structured as follows. In Section 2 we briefly review the definitions of symmetric groups of completely and incompletely specified Boolean functions. Section 3 studies the difficulties with symmetry of incompletely specified functions. To overcome these difficulties we introduce *strong symmetry of incompletely specified functions* in Section 4. We then concentrate on computing a minimum sized partition of the variables of incompletely specified functions into symmetric groups exploiting strong symmetries in Section 5. We adjust a greedy algorithm for node coloring to heuristically solve our problem. The paper closes with experimental results proving our method to be very effective. It improves ROBDD sizes of symmetric sifting by a factor of 51% and by a factor of 70% in combination with a slightly modified version of Chang's technique.

2 Symmetric groups

In the following, let X be the set of variables $\{x_1, \ldots, x_n\}$ of a Boolean function f and D some subset of $\{0,1\}^n$.

2.1 Completely specified functions

In this section we will briefly review definitions and basic properties of symmetries of completely specified Boolean functions. We start with the definition of symmetry in two variables, in a set of variables, and in a partition of the set of input variables of a completely specified Boolean function.

Definition 2.1 (Symmetry of completely specified Boolean functions) *A completely specified Boolean function $f : \{0,1\}^n \to \{0,1\}$ is symmetric in a pair* *of input variables* (x_i, x_j) *if and only if*

$$f(\epsilon_1, \ldots, \epsilon_i, \ldots, \epsilon_j, \ldots, \epsilon_n) = f(\epsilon_1, \ldots, \epsilon_j, \ldots, \epsilon_i, \ldots, \epsilon_n)$$

holds $\forall \epsilon \in \{0,1\}^n$. f *is symmetric in a subset* λ *of* X *iff* f *is symmetric in* x_i *and* x_j $\forall x_i, x_j \in \lambda$. f *is symmetric in a partition* $P = \{\lambda_1, \ldots, \lambda_k\}$ *of the set of input variables iff* f *is symmetric in* λ_i $\forall 1 \le i \le k$.

If f is symmetric in a subset λ of the set of input variables, then we say 'the variables in λ form a symmetric group'.
It is well known, that symmetry of a completely specified Boolean function f in pairs of input variables of f leads to an equivalence relation on X. Thus, there is a unique minimal partition P of X (namely the set of the equivalence classes of this relation) such that f is symmetric in P. The computation of a minimal partition of f such that f is symmetric in P can be done by testing for symmetry in all pairs of input variables (see Möller 1993 and Tsai 1996).

2.2 Incompletely specified functions

The definition of symmetry of an incompletely specified Boolean function f is reduced to the definition of symmetry of completely specified extensions of f. An extension of an incompletely specified Boolean function is defined as follows:

Definition 2.2 *Let* $f : D \to \{0,1\}$ $(D \subseteq \{0,1\}^n)$ *be an incompletely specified Boolean function.* $f' : D' \to \{0,1\}$ $(D' \subseteq \{0,1\}^n)$ *is an extension of* f *iff* $D \subseteq D'$ *and* $f'(\epsilon) = f(\epsilon)$ $\forall \epsilon \in D$.

Definition 2.3 (Symmetry of incompletely specified Boolean functions) *An incompletely specified Boolean function* $f : D \to \{0,1\}$ *is symmetric in a pair of input variables* (x_i, x_j) *(in a subset* λ *of* X */ in a partition* $P = \{\lambda_1, \ldots, \lambda_k\}$ *of* X*) iff there is a completely specified extension* f' *of* f*, which is symmetric in* (x_i, x_j) *(in* λ */ in* P*).*

3 Difficulties with symmetry of incompletely specified functions

In order to minimize the ROBDD size for an incompletely specified Boolean function f, we are looking for a minimal partition (or for maximal variable sets) such that f is symmetric in this partition (or these sets). Unfortunately there are some difficulties in the computation of such partitions: First of all, symmetry of f in two variables does not form an equivalence relation on X in the case of *incompletely* specified Boolean functions (see also Dietmeyer/Schneider (1967) or Kim/Dietmeyer (1991)).

Example 3.1 The following function shows that symmetry in two variables doesn't lead to an equivalence relation on the variable set in the case of incompletely specified Boolean functions[¶]:

$$f(1,0,0) = 1,\ f(0,1,0) = \star,\ f(0,0,1) = 0,\ f(\epsilon) = 0 \text{ otherwise} .$$

It is easy to see that f is symmetric in x_1 and x_2 (for the corresponding completely specified extension f' of f $f'(0,1,0) = 1$ holds) and f is symmetric in x_2 and x_3. However f is *not* symmetric in x_1 and x_3.

Since symmetry in pairs of variables doesn't form an equivalence relation, it will be much more difficult to deduce symmetries in larger variable sets from symmetries in pairs of variables in the case of incompletely specified Boolean functions.
Even if f is symmetric in *all* pairs of variables x_i and x_j of a subset λ of the variable set of f, f is *not* necessarily symmetric in λ. This is illustrated by the following example:

Example 3.2 Consider $f : D \to \{0,1\}$, $D \subseteq \{0,1\}^4$.

$$f(\epsilon) = \begin{cases} 1 & \text{for } \epsilon = (0,0,1,1) \\ \star & \text{for } \epsilon = (0,1,0,1), \epsilon = (0,1,1,0), \epsilon = (1,0,0,1), \epsilon = (1,0,1,0) \\ 0 & \text{for } \epsilon = (1,1,0,0) \\ 0 & \text{otherwise} \end{cases}$$

It is easy to see, that f is symmetric in all pairs of variables x_i and x_j, $i,j \in \{1,2,3,4\}$. The symmetry graph[‖] of f is shown in Figure 1. It's the complete graph. For each completely specified extension f' of f, which is symmetric in (x_1, x_3), $f'(0,1,1,0) = 0$ holds and for each completely specified extension f''

[¶]In the following $f(\epsilon) = \star$ means that $\epsilon \notin D$ for the incompletely specified Boolean function $f : D \to \{0,1\}$.

[‖]The symmetry graph $G^f_{sym} = (X, E)$ of a Boolean function $f : D \to \{0,1\}$ is a undirected graph with node set X (the set of input variables of f) and edges $\{x_i, x_j\} \in E$ iff f is symmetric in (x_i, x_j).

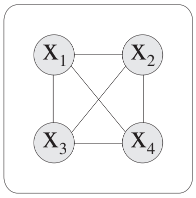

Figure 1: Symmetry graph of the function of Example 2

of f, which is symmetric in (x_2, x_4), $f''(0,1,1,0) = 1$ holds. Hence there is no completely specified extension of f which is symmetric in (x_1, x_3) *and* (x_2, x_4) and therefore no extension which is symmetric in $\{x_1, \ldots, x_4\}$.

Example 3.2 also points out another fact: If an incompletely specified Boolean function f is symmetric in all variable sets λ_i of a partition $P = \{\lambda_1, \ldots, \lambda_k\}$, it is *not* necessarily symmetric in P (choose $P = \{\{x_1, x_3\}, \{x_2, x_4\}\}$ in the example).

4 Strong symmetry

The difficulties with the detection of large symmetry groups of incompletely specified functions result from the fact that symmetry in pairs of variables doesn't form an equivalence relation on the variable set X. If we change the definition of symmetry of incompletely specified functions as given in Definition 4.1, symmetry in pairs of variables provides an equivalence relation as in the case of completely specified functions:

Definition 4.1 (Strong symmetry) *An incompletely specified Boolean function* $f : D \to \{0,1\}$ *is called* strongly symmetric *in a pair of input variables* (x_i, x_j) *iff* $\forall(\epsilon_1, \ldots, \epsilon_n) \in \{0,1\}^n$ *either (a) or (b) holds.*

(a) $(\epsilon_1, \ldots, \epsilon_i, \ldots, \epsilon_j, \ldots, \epsilon_n) \notin D$ and $(\epsilon_1, \ldots, \epsilon_j, \ldots, \epsilon_i, \ldots, \epsilon_n) \notin D$

(b) $(\epsilon_1, \ldots, \epsilon_i, \ldots, \epsilon_j, \ldots, \epsilon_n) \in D$ and $(\epsilon_1, \ldots, \epsilon_j, \ldots, \epsilon_i, \ldots, \epsilon_n) \in D$ *and* $f(\epsilon_1, \ldots, \epsilon_i, \ldots, \epsilon_j, \ldots, \epsilon_n) = f(\epsilon_1, \ldots, \epsilon_j, \ldots, \epsilon_i, \ldots, \epsilon_n)$.

In contrast to the *strong* symmetry of incompletely specified functions the symmetry defined so far is called *weak* symmetry.
The following lemma holds for strong symmetry:

Lemma 4.1 *Strong symmetry in pairs of variables of an incompletely specified Boolean function $f : D \to \{0,1\}$ forms an equivalence relation on the variable set X of f.*

Due to Lemma 4.1 there is a unique minimal partition P of the set X of input variables such that f is strongly symmetric in P. As in the case of completely specified Boolean functions, f is strongly symmetric in a subset λ of X iff $\forall x_i, x_j \in \lambda$ f is strongly symmetric in (x_i, x_j). f is strongly symmetric in a partition $P = \{\lambda_1, \ldots, \lambda_k\}$ of X iff $\forall 1 \leq i \leq k$ f is strongly symmetric in λ_i.
Of course, if a function f is weakly symmetric in a partition P, it needs not to be strongly symmetric in P, but it follows directly from Definition 2.3 that there is an extension of f which is strongly symmetric in P.

5 Minimum sized partition of the variables of an incompletely specified function into symmetric groups

We have to solve the following problem **MSP** (**M**inimal **S**ymmetry **P**artition):

Given: Incompletely specified function $f : D \to \{0,1\}$, represented by ROBDDs for f_{on} and f_{dc}. **
Find: Partition P of the set $X = \{x_1, \ldots, x_n\}$ such that

- f is symmetric in P and
- for any partition P' of X in which f is symmetric, the inequation $|P| \leq |P'|$ holds.

We can prove (see Scholl 1996) the following theorem:

Theorem 5.1 *MSP is NP-hard.*

To solve the problem heuristically, we use a heuristic for the problem 'Partition into Cliques (PC)' (see Garey/Johnson 1979) for the symmetry graph G^f_{sym} of f. However, the examples in Section 3 showed that f is *not* symmetric in *all* partitions

** f_{on} is the completely specified Boolean function with the same on-set as f and f_{dc} is the completely specified function with $\{0,1\}^n \setminus D$ as on-set.

into cliques of G^f_{sym}. The heuristic has to be changed in order to guarantee that f is symmetric in the resulting partition P.

The heuristic to solve the problem PC makes use of the following well known lemma:

Lemma 5.1 *A graph $G = (V, E)$ can be partitioned into k disjoint cliques iff $\bar{G} = (V, \bar{E})$ can be colored with k colors. ($\bar{G}$ is the inverse graph of G, which has the same node set V as G and an edge $\{v, w\}$ between two nodes v and w iff there is no edge $\{v, w\}$ in G, i.e., $\bar{E} = \{\{v, w\} | \{v, w\} \notin E\}$.)*

Thus, heuristics for node coloring can be directly used for the solution of partition into cliques. Nodes with the same color in $\bar{G}$ form an 'independent set' and thus a clique in G. Our implementation is based on Brélaz algorithm for node coloring (Brélaz 1979) which has a running time of $\mathcal{O}(N)$ in an implementation of Morgenstern (Morgenstern 1992) (N is the number of nodes of the graph which has to be colored). It's a greedy algorithm, which colors node by node and doesn't change the color of a node which is already colored. In the algorithm there are certain criteria to choose the next node to color and the color to use for it in a clever way (see Brélaz 1979, Morgenstern 1992). Figure 2 shows our heuristic for the problem MSP, which is derived from the Brélaz/Morgenstern heuristic for node coloring.

First of all the symmetry graph G^f_{sym} of f (or the inverse graph $\overline{G^f_{sym}}$) is computed. The nodes of $\overline{G^f_{sym}}$ are the variables $x_1, \ldots, x_n$. These nodes are colored in the algorithm. Nodes with the same color form a clique in G^f_{sym}. Note that partition P (see line 3) has the property that it contains set $\{x_k\}$ for any uncolored node x_k and that nodes with the same color are in the same set of P, at any moment. The crucial point of the algorithm is that the invariant 'f is strongly symmetric in P' of line 6 is always maintained.

Now let us take a look at the algorithm in more detail. At first glance, the set of all admissible colors for the next node x_i is the set of all colors between 1 and n except the colors of nodes which are adjacent to x_i in $\overline{G^f_{sym}}$. In the original Brélaz/Morgenstern algorithm the minimal color among these colors is chosen for x_i (*curr_color* in lines 10, 11). However, since we have to guarantee that f is symmetric in the partition P which results from coloring, it is possible that we are not allowed to color x_i with *curr_color*. If there is already another node x_j

which is colored by *curr_color*, then f has to be symmetric in the partition P' which results by union of $\{x_i\}$ and $[x_j]$[††]. If there is such a node x_j, we have to test whether f is symmetric in (x_i, x_j) (line 14) (this test can have a negative result, since the don't care set of f is reduced during the algorithm). If f is not symmetric in (x_i, x_j), *curr_color* is removed from the set of color candidates for x_i (line 20) and the minimal color in the remaining set is chosen as the new color candidate (line 10). If the condition of line 14 is true, the new partition P results from the old partition P by union of $\{x_i\}$ and $[x_j]$ (line 16). Now f is symmetric in the new partition P (invariant (*) from line 17, see Lemma 5.2), and we can assign don't cares of f such that f is strongly symmetric in P (line 18).

At the end we receive an extension of the original incompletely specified Boolean function which is strongly symmetric in the resulting partition P.

To prove invariant (*) in line 17, we need the following lemma (see Scholl 1996):

Lemma 5.2 *Let $f : D \to \{0,1\}$ be* strongly symmetric *in P, $[x_i], [x_j] \in P$ two subsets with $|[x_i]| = 1$, and let f be symmetric in (x_i, x_j), then f is symmetric in $P' = P \setminus \{[x_j], \{x_i\}\} \bigcup \{[x_j] \cup \{x_i\}\}$.*

Note that the lemma cannot be proved if we replace 'f *strongly* symmetric in P' by 'f (weakly) symmetric in P' or if we don't assume $|[x_i]| = 1$. However the given conditions coincide exactly with the conditions existing in the algorithm. (Thus it is *necessary* to make f *strongly* symmetric in P in line 18 of the algorithm and to maintain the invariant 'f is strongly symmetric in P' of line 6.)

Next we have to explain how f is made strongly symmetric in the partition P in line 18 of the algorithm. From the definition of symmetry of incompletely specified functions it is clear that it is possible to extend a function f, which is (weakly) symmetric in a partition P, to a function which is strongly symmetric in P. From the set of all extensions of f which are strongly symmetric in P we choose the extension with a maximum number of don't cares. If f is (weakly) symmetric in a pair of variables (x_i, x_j), the extension f' of f, which is strongly symmetric in (x_i, x_j) and which has a maximal don't care set among all extensions of f with that property, can be easily computed from the ROBDD representations of f_{on}, f_{dc} and f_{off} by the procedure *make_strongly_symm* in Figure 3. We can prove that

[††]If $P = \{\lambda_1, \ldots, \lambda_k\}$ is a partition of $\{x_1, \ldots, x_n\}$, then $[x_j]$ denotes λ_q with $x_j \in \lambda_q$.

a sequence of at most $|[x_j]|$ calls of the procedure *make_strongly_symm* is enough to make f strongly symmetric in the partition P in line 18 of the algorithm.

6 Experimental results

We have carried out experiments to test the algorithms described above. To generate incompletely specified functions from completely specified functions, we used a method proposed by Chang (1994): After collapsing each benchmark circuit to two level form, we randomly selected minterms in the on-set with a probability of 40% to be included into the don't care set. The last three Boolean functions in Table 1 are partial multipliers $partmult_n$[‡‡].

We performed three experiments: First of all, we applied symmetric sifting to the ROBDDs representing the on-set of each function. The results are shown in column 4 (*sym_sift*) of Table 1. The entries are ROBDD sizes in terms of internal nodes. In a second experiment, we applied our algorithm to minimize the number of symmetric groups followed by symmetric sifting. Column 5 (*sym_group*) of Table 1 shows the results. *sym_group* provides a partition $P = \{\lambda_1, \ldots, \lambda_k\}$ and an extension f' of the original function f, such that f' is strongly symmetric in P. The variable order of the ROBDD representing f' is a 'symmetric order' (see Panda 1994, Möller 1994) with the variables in λ_i before the variables in λ_{i+1} $(1 \leq i < k)$. On the average, we can improve the ROBDD size by 51%.

In a last experiment we started with the results of *sym_group* and then went on with a slightly modified version of the technique of Chang (1994) and Shiple (1994). This technique minimizes the number of nodes at every level of the ROBDD by an operation *remove_z* assigning as few don't cares as possible to either the on-set or the off-set, i.e., the number of so-called linking nodes immediately below a cut line between 2 adjacent variables is minimized. After the minimization of nodes at a certain level of the ROBDD they use the remaining don't cares to minimize the number of nodes at the next level. The cut line is moved from top to bottom in the ROBDD. Under certain conditions, this method does preserve strong symmetry: Let f be an incompletely specified Boolean function which is

[‡‡]The n^2 inputs are the bits of the n partial products and the $2n$ outputs are the product bits. The don't care set contains all input vectors which cannot occur for the reason that the input bits are not independent from each other, because they are conjunctions $a_i b_j$ of bits of the operands $(a_1, \ldots, a_n)$ and $(b_1, \ldots, b_n)$ of the multiplication.

Circuit	i	o	*sym_sift*	*sym_group*		*sym_cover*	
5xp1	7	10	67	66	(0.2 s)	53	(0.5 s)
9symml	9	1	108	25	(0.3 s)	25	(0.4 s)
alu2	10	6	201	201	(0.7 s)	152	(2.6 s)
apex6	135	99	1033	983	(267.6 s)	612	(459.7 s)
apex7	49	37	814	728	(27.7 s)	340	(52.2 s)
b9	41	21	256	185	(8.6 s)	122	(11.5 s)
c8	28	18	156	95	(1.7 s)	70	(3.2 s)
example2	85	66	491	484	(69.2 s)	416	(119.4 s)
mux	21	1	34	29	(0.6 s)	29	(0.7 s)
pcler8	27	17	78	73	(1.9 s)	72	(3.3 s)
rd73	7	3	76	34	(0.3 s)	27	(0.4 s)
rd84	8	4	144	42	(0.7 s)	42	(0.7 s)
sao2	10	4	104	104	(0.4 s)	70	(0.8 s)
x4	94	71	829	633	(121.9 s)	485	(203.4 s)
z4ml	7	4	51	32	(0.2 s)	17	(0.3 s)
partmult3	9	6	152	35	(1.0 s)	29	(1.2 s)
partmult4	16	8	971	222	(49.5 s)	114	(50.6 s)
partmult5	25	10	4574	998	(1540.4 s)	365	(1548.4 s)
total			10139	4969		3040	

Table 1: Experimental results. The table shows the number of nodes in the ROBDDs of each function. Numbers in parenthesis show the CPU times (measured on a SPARCstation 20 (96 MByte RAM)).

strongly symmetric in $P = \{\lambda_1, \ldots, \lambda_k\}$ and assume that the variable order of the ROBDD representing f is a 'symmetric order' with the variables in λ_i before the variables in λ_{i+1} ($1 \leq i < k$). If we restrict the operation *remove_z* presented by Chang (1994) and Shiple (1994) to cut lines between 2 symmetric groups λ_i and λ_{i+1} then it preserves strong symmetry in P. Since our technique to restrict *remove_z* to cut lines between symmetric groups doesn't destroy the symmetric groups, we can perform symmetric sifting after the node minimization with the same symmetric groups as before. Column 6 (*sym_cover*) of Table 1 shows the resulting ROBDD sizes. On the average, the technique leads to an improvement of the ROBDD sizes by 70%.

7 Conclusions

In many applications of CAD we have to deal with incompletely specified Boolean functions which are represented by ROBDDs. Looking for extensions of such functions with small ROBDD representations can reduce memory requirements and running times by minimizing the size of intermediate results of the computation (e.g. in the equivalence check of two FSMs, Coudert 1989) and it can have a large effect on the quality of the results of such algorithms (e.g. in logic synthesis for FPGA realizations). In this paper we present algorithms to minimize the ROBDD sizes for incompletely specified Boolean functions based on the exploitation of strong symmetries. Experimental results prove our approach to be very effective.

References

[Akers, 1978] Akers, S. (1978). Binary Decision Diagrams. *IEEE Trans. on CAD*, 27(6):509–516.

[Bollig et al., 1995] Bollig, B., Löbbing, M., and Wegener, I. (1995). Simulated annealing to improve variable orderings for OBDDs. In *Int'l Workshop on Logic Synthesis*, Granlibakken (CA).

[Brelaz, 1979] Brelaz, D. (1979). New methods to color vertices of a graph. *Comm. of the ACM*, 22:251–256.

[Bryant, 1986] Bryant, R. (1986). Graph–Based Algorithms for Boolean Function Manipulation. *IEEE Trans. on CAD*, 35(8):677–691.

[Bryant, 1992] Bryant, R. (1992). Symbolic Boolean Manipulation with Ordered Binary-Decision diagrams. *ACM, Comp. Surveys*, 24:293–318.

[Chang et al., 1994] Chang, S., Cheng, D., and Marek-Sadowska, M. (1994). Minimizing ROBDD Size of Incompletely Specified Multiple Output Functions. In *The European Design and Test Conference*, pages 620–624.

[Dietmeyer and Schneider, 1967] Dietmeyer, D. and Schneider, P. (1967). Identification of Symmetry, Redundancy and Equivalence of Boolean Functions. *IEEE Transact. on Electronic Computers*, 16(6):804–817.

[Drechsler et al., 1995] Drechsler, R., Becker, B., and Göckel, N. (1995). A genetic algorithm for variable ordering of obdds. In *Int'l Workshop on Logic Synthesis*, Granlibakken (CA).

[Felt et al., 1993] Felt, E., York, G., Brayton, R., and Sangiovanni-Vincentelli, A. (1993). Dynamic Variable Reordering for BDD Minimization. In *European Conf. on Design Automation*, pages 130–135.

[Fujita et al., 1988] Fujita, M., Fujisawa, H., and N.Kawato (1988). Evaluation and Improvements of Boolean Comparison Methods Based on Binary Decision Diagrams. In *IEEE Int'l Conf. on CAD*, pages 2–5.

[Fujita et al., 1991] Fujita, M., Matsunaga, Y., and Kakuda, T. (1991). On Variable Ordering of Binary Decision Diagrams for the Application of Multi–level Logic Synthesis. In *European Conf. on Design Automation*, pages 50–54.

[Garey and Johnson, 1979] Garey, M. and Johnson, D. (1979). *Computers and Intractability: A Guide to the Theory of NP–Completeness*. W.H.Freeman & Co., San Francisco.

[Ishiura et al., 1991] Ishiura, N., Sawada, H., and Yajima, S. (1991). Minimization of Binary Decision Diagrams Based on Exchanges of Variables. In *IEEE Int'l Conf. on CAD*, pages 472–475.

[Kim and Dietmeyer, 1991] Kim, B.-G. and Dietmeyer, D. (1991). Multilevel Logic Synthesis of Symmetric Switching Functions. *IEEE Trans. on CAD*, 10(4):436–446.

[Lee, 1959] Lee, C. (1959). Representation of switching circuits by binary decision diagrams. *Bell System Technical Journal*, 38:985–999.

[Malik et al., 1988] Malik, S., Wang, A., Brayton, R., and Sangiovanni-Vincentelli, A. (1988). Logic Verification using Binary Decision Diagrams in a Logic Synthesis Environment. In *IEEE Int'l Conf. on CAD*, pages 6–9.

[Möller et al., 1993] Möller, D., Mohnke, J., and Weber, M. (1993). Detection of Symmetry of Boolean Functions Represented by ROBDDs. In *IEEE Int'l Conf. on CAD*, pages 680–684.

[Möller et al., 1995] Möller, D., Molitor, P., and Drechsler, R. (1995). *Novel Approaches in Logic and Architecture Synthesis*, chapter *Symmetry based Variable Ordering for ROBDDs*, pages 70–81. Chapman & Hall.

[Moret, 1982] Moret, B. (1982). Decision Trees and Diagrams. *ACM, Comp. Surveys*, 14(4):593–623.

[Morgenstern, 1992] Morgenstern, C. (1992). A new backtracking heuristic for rapidly four-coloring large planar graphs. Technical Report CoSc-1992-2, Texas Christian University, Fort Worth, Texas.

[Panda and Somenzi, 1995] Panda, S. and Somenzi, F. (1995). Who Are the Variables in Your Neighborhood. In *IEEE Int'l Conf. on CAD*, pages 74–77.

[Panda et al., 1994] Panda, S., Somenzi, F., and Plessier, B. (1994). Symmetry Detection and Dynamic Variable Ordering of Decision Diagrams. In *IEEE Int'l Conf. on CAD*, pages 628–631.

[Rudell, 1993] Rudell, R. (1993). Dynamic Variable Ordering for Ordered Binary Decision Diagrams. In *IEEE Int'l Conf. on CAD*, pages 42–47.

[Scholl, 1996] Scholl, C. (1996). *Logic Synthesis by Exploiting Functional Properties.* PhD thesis, Institut für Informatik, Universität des Saarlandes, Germany. (in german).

[Shiple et al., 1994] Shiple, T., Hojati, R., Sangiovanni-Vincentelli, A., and Brayton, R. (1994). Heuristic Minimization of BDDs Using Don't Cares. In *Design Automation Conf.*

[Sieling, 1996] Sieling, D. (1996). Variable orderings and the size of OBDDs for partially symmetric boolean functions. Talk at the German Workshop on Complexity Theory held in G"ottingen.

[Tsai and Marek-Sadowska, 1996] Tsai, C. and Marek-Sadowska, M. (1996). Generalized Reed-Muller Forms as a Tool to Detect Symmetries. *IEEE Trans. on CAD*, 45(1):33–40.

Input: Incompletely specified function $f : D \to \{0,1\}$, $D \subseteq \{0,1\}^n$, represented by f_{on} and f_{dc}

Output: Partition P of $\{x_1, \ldots, x_n\}$, such that f is symmetric in P

Algorithm:

Compute symmetry graph $G^f_{sym} = (V, E)$ of f (or $\overline{G^f_{sym}} = (V, \bar{E})$).
$\forall 1 \leq k \leq n : \mathit{color}(x_k) := \mathit{undef}.$
$P = \{\{x_1\}, \{x_2\}, \ldots, \{x_n\}\}$
$\mathit{node\_candidate\_set} := \{x_1, \ldots, x_n\}$
while $(\mathit{node\_candidate\_set} \neq \emptyset)$ **do**
  /* f is strongly symmetric in P */
  Choose $x_i \in \mathit{node\_candidate\_set}$ according to Brélaz/Morgenstern criterion
  $\mathit{color\_candidate\_set} := \{c \mid 1 \leq c \leq n, \nexists x_j \text{ with } \{x_i, x_j\} \in \bar{E} \text{ and } \mathit{color}(x_j) = c\}$
  while $(\mathit{color}(x_i) = \mathit{undef}.)$ **do**
    $\mathit{curr\_color} := \min(\mathit{color\_candidate\_set})$
    $\mathit{color}(x_i) := \mathit{curr\_color}$
    if ($\exists$ colored node x_j with $\mathit{color}(x_j) = \mathit{color}(x_i)$)
     then
       if (f symmetric in (x_i, x_j))
        then
          $P := P \setminus \{[x_j], \{x_i\}\} \bigcup \{[x_j] \cup \{x_i\}\}$
          /* f is symmetric in P */ (*)
          Make f *strongly symmetric* in P. (**)
        else
          $\mathit{color\_candidate\_set} := \mathit{color\_candidate\_set} \setminus \{\mathit{curr\_color}\}$
          $\mathit{color}(x_i) := \mathit{undef}.$
       fi
    fi
  od
  $\mathit{node\_candidate\_set} := \mathit{node\_candidate\_set} \setminus \{x_i\}$
od

Figure 2: Algorithm to solve MSP.

Procedure *make_strongly_symm*

Input: $f : D \to \{0,1\}$, represented by f_{on}, f_{off}, f_{dc}. f is (weakly) symmetric in (x_i, x_j).

Output: minimal extension f' of f (represented by f'_{on}, f'_{off}, f'_{dc}), which is strongly symmetric in (x_i, x_j).

Algorithm:

1. $f'_{on} = \overline{x_i}\,\overline{x_j} f_{on_{\overline{x_i}\,\overline{x_j}}} + x_i x_j f_{on_{x_i x_j}} + (x_i \overline{x_j} + \overline{x_i} x_j)(f_{on_{x_i \overline{x_j}}} + f_{on_{\overline{x_i} x_j}})$
2. $f'_{off} = \overline{x_i}\,\overline{x_j} f_{off_{\overline{x_i}\,\overline{x_j}}} + x_i x_j f_{off_{x_i x_j}} + (x_i \overline{x_j} + \overline{x_i} x_j)(f_{off_{x_i \overline{x_j}}} + f_{off_{\overline{x_i} x_j}})$
3. $f'_{dc} = \overline{f'_{on} + f'_{off}}$

Figure 3: Prozedur *make_strongly_symm*