BDD Minimization Using Symmetries*

Christoph Scholl, *Member, IEEE*, Dirk Möller,
Paul Molitor, *Member, IEEE*, Rolf Drechsler *Member, IEEE*

Abstract— **In this paper we study the effect of using information about (partially) symmetries for the minimization of Reduced Ordered Binary Decision Diagrams (ROBDDs). The influence of symmetries for the integration in dynamic variable ordering is studied for both completely and incompletely specified Boolean functions.**

The problems above are studied from a theoretical and practical point of view. Statistical results and benchmark results are reported to underline the efficiency of the approach. They prove that our techniques lead to improvements of the ROBDD sizes by up to 70%.

Keywords— **BDD, symmetry, sifting, incompletely specified functions, symmetry detection**

I. Introduction

BINARY Decision Diagrams (BDDs) as a data structure for representation of Boolean functions were first introduced by Lee [30] and further popularized by Akers [1] and Moret [38]. In the restricted form of ROBDDs they gained widespread application, because ROBDDs are a canonical representation and allow efficient manipulations [5]. Some fields of application are logic design verification, test generation, fault simulation, and logic synthesis [33], [6]. Most of the algorithms using ROBDDs have run time polynomial in the size of the ROBDDs. The sizes themselves depend on the variable order used. Thus, there is a need to find a variable order that minimizes the number of nodes in an ROBDD.

As an example of application of ROBDDs consider the use of Field Programmable Gate Arrays (FPGA) in the construction of Combinational Logic Circuits (CLC) from a BDD. The BDD has a direct correspondence to a CLC when each node of the BDD is substituted by a multiplexer. Since it is straightforward to map these multiplexer circuits on an FPGA, where the logic blocks are based on multiplexers, BDDs have become a good framework for logic synthesis. Because of this direct correspondence, saving only a few nodes in a BDD by using good variable orders already pays. The importance of BDD minimization is also obvious for recently proposed methods to synthesize Pass Transistor Logic (PTL) networks directly from BDDs [7], [18]. Also for other FPGA synthesis techniques like functional decomposition (see e.g. [28], [50], [45]) it is a good heuristic to start with minimized BDDs.

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 [33], [20], [21], [19] and dynamic reordering heuristics which try to improve on a given order [26], [43], [17], [2], [14]. Sifting introduced by Rudell [43] has emerged so far as the most successful algorithm for dynamic reordering of variables. This algorithm is based on finding the local 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 in [40], 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.

In this paper we consider partially symmetric Boolean functions, i.e., Boolean functions that are invariant under the permutation of some input variables. Knowing a Boolean function to be symmetric allows application of special logic synthesis tools that can improve the results of the design [16], [27], [13]. Furthermore, knowing the variables of a Boolean function which are symmetric often restricts the search space of a logic design problem which may yield in a remarkable decrease of run time for that problem. Such problems are, e.g., permutation independent Boolean comparison [29], [9], [34], [35] and technology mapping [32].

We show that symmetry properties can be used to efficiently construct good variable orders for ROBDDs using modified gradual improvement heuristics [37], [41][1].

The crucial point is to locate symmetric variables side by side and to treat them as fixed block. This technique is motivated by the following three facts:

1. The exchange of two symmetric variables does not change the size of the ROBDD, because the function remains the same.
2. The size of the ROBDD of any totally symmetric function $f : \{0,1\}^n \to \{0,1\}$ is $O(n^2)$.
3. The value of a function which is symmetric in some variables $\{x_{i_1}, \ldots, x_{i_q}\}$ does not depend on the exact assignment of these variables but only on their weight $\sum_{j=1}^{q} x_{i_j}$.

Using the first fact, the heuristics can skip over the exchange of symmetric variables and so the run time decreases. However, the resulting ROBDD sizes will be the

Christoph Scholl is with the Institute of Computer Science, Albert-Ludwigs-University, 79110 Freiburg, Germany, E-mail: scholl@informatik.uni-freiburg.de

Dirk Möller is with DResearch GmbH, 12681 Berlin, Germany, E-mail: moeller@dresearch.de

Paul Molitor is with the Institute of Computer Science, University of Halle, 06099 Halle, Germany, E-mail: molitor@informatik.uni-halle.de

Rolf Drechsler is with the Institute of Computer Science, Albert-Ludwigs-University, 79110 Freiburg, Germany, E-mail: drechsle@informatik.uni-freiburg.de

*Parts of the article have been presented at *Int'l Conf. on CAD 1993* [36], *IFIP Workshop on Logic and Architecture Synthesis 1994* [37], and *European Design and Test Conf. 1997* [44].

[1]The methods of paper [41] are similar to ours and have been independently developed.

same. The second and third fact leads to the special class of variable orders of our technique, i.e, variable orders where the symmetric variables are located side by side.

If we locate the symmetric variables side by side and treat them as a fixed block, we receive a modification of sifting: the *symmetric sifting* algorithm, 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 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 [40].

To give an impressive example for the fact that it helps to locate the symmetric variables side by side, consider the function $x_1x_{n+1} + x_2x_{n+2} + \ldots + x_nx_{2n}$ of $2n$ variables [5]. The size of the corresponding ROBDD with variable ordering $x_1, x_2, x_3, \ldots, x_{2n}$ is exponential in n whereas the size of any ROBDD with an order where the symmetries are side by side is linear in n.

We present statistical facts for Boolean functions with up to 5 input variables and experimental results for functions taken from the LGSYNTH91 benchmark set proving the new class of orders to be very efficient with respect to ROBDD size. The benchmark results show that the modified reordering heuristic, which does not reorder single variables but whole symmetric blocks, outperforms the original one.

Although, in general, it is reasonable to locate the symmetric variables side by side, it does not lead to optimal results in all cases. A counterexample has been given in [37]. By iterating one of these 'bad' Boolean functions a family of parameterized Boolean functions can be constructed such that there is a linear gap between optimal orders and best symmetric orders [40].

The second part of this paper handles the problem of detecting partial symmetries for both completely and incompletely specified Boolean functions. Of course before exploiting symmetries we need to detect them.

First we concentrate on completely specified Boolean functions. So far, logic synthesis tools that work with ROBDDs use the well–known naive symmetry check which compares certain cofactor functions.[2] The problem of this naive approach is that it needs to construct the ROBDD of the considered cofactor functions first. Especially for functions with a large number of inputs and a large ROBDD size that may be impractical regarding to ROBDD construction time and storage place.

We present an improved method that tries to detect as many asymmetries of the function as possible without time consuming manipulations of the ROBDD data structure itself before using the naive symmetry check. For these asymmetry checks, we use structural properties of ROBDD as well as simple function properties.

Experimental results on a large suit of benchmarks show that this approach is promising. In many cases, the CPU time decreases dramatically using our sophisticated symmetry check instead of the naive one.

In many applications (e.g. checking the equivalence of two Finite State Machines (FSMs) [11], minimizing the transition relation of an FSM or logic synthesis for FPGA realizations [28], [50], [45]) incompletely specified Boolean functions play an important role. As determining the symmetric groups and applying symmetric sifting results in good variable orders for completely specified functions, it also seems to be a good idea in the case of incompletely specified functions to first determine symmetric groups and then to apply symmetric sifting. However, the symmetric groups of incompletely specified functions are not uniquely defined as will be demonstrated by some counterexamples. Therefore we have to ask for good partitions of the Boolean variables into symmetric groups with respect to ROBDD minimization and their computation.

To the best of our knowledge, no variable ordering algorithm exploiting don't cares has been presented in literature. First approaches [8], [47], [15] investigate the ROBDD minimization problem for incompletely specified Boolean functions, but there it is assumed that the variable ordering is fixed. However, the resulting ROBDD sizes heavily depend on the initial variable order. Thus, there is a strong need to determine good variable orders in the case of incompletely specified functions, too.

In [27] an algorithm has been presented 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 difficulties: 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 V-A 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 efficiency of our approach is underlined by experimental results.

The paper is structured as follows: In Section II we introduce basic notations and review the main definitions. Symmetries for completely specified and incompletely specified functions are defined. The effect of symmetries for ROBDDs representing completely specified Boolean functions is described in Section III. In Section IV we present our asymmetry test. Algorithms for incompletely specified functions are given in Section V. For all methods we give experimen-

[2] In this paper, we do not handle approaches of symmetry detection which do not use ROBDDs as, e.g., the approach proposed in [42] where maximal sets of symmetric inputs of completely specified Boolean functions are computed using test generation procedures for single stuck-at faults.

tal results in Section VI. Finally, the results are summarized.

II. Preliminaries

In this section we review some basics that are needed for the understanding of the paper. First, BDDs are defined and the effect of the variable ordering is discussed. After introducing some notations that are needed for the descriptions of the asymmetry detection algorithm, the definitions of symmetry for (in-)completely specified functions are given.

A. Binary Decision Diagrams

We start with a brief review of the essential definitions and properties of *Binary Decision Diagrams* as introduced in [5].

Definition 1: A *Binary Decision Diagram* (BDD) is a rooted directed acyclic graph $G = (V, E)$ with vertex set V containing two types of vertices, *non-terminal* and *terminal* vertices. A non-terminal vertex v has as label a variable $index(v) \in \{x_1, \ldots, x_n\}$ and two children $low(v), high(v) \in V$. A terminal vertex v is labeled with a value $value(v) \in \{0, 1\}$ and has no outgoing edge.

A BDD can be used to compute a Boolean function $f(x_1, \ldots, x_n)$ in the following way: Each input $a = (a_1, \ldots, a_n) \in \{0, 1\}^n$ defines a computation path through the BDD that starts at the root. If the path reaches a non-terminal node v that is labeled by x_i it follows the path $low(v)$ iff $a_i = 0$ and it follows the path $high(v)$ iff $a_i = 1$. On all paths a terminal vertex is reached since a BDD is directed and acyclic. The label of the terminal vertex determines the return value of the BDD on input a.

More formally, we can define the Boolean function corresponding to a BDD recursively.

Definition 2: A BDD having root vertex v *denotes* a Boolean function f_v defined as follows:

1. If v is a terminal vertex and $value(v) = 1$ ($value(v) = 0$), then $f_v = 1$ ($f_v = 0$).
2. If v is a non-terminal vertex and $index(v) = x_i$, then f_v is the function

$$f_v(x_1, \ldots, x_n) = \overline{x}_i \cdot f_{low(v)}(x_1, \ldots, x_n) + x_i \cdot f_{high(v)}(x_1, \ldots, x_n).$$

The variable x_i is called the *decision variable* for v.

It is well-known that for each Boolean function f there exists a BDD denoting f. BDDs are often used as a data structure in design automation and logic synthesis. Thus there is a need of efficient manipulation of BDDs. Unfortunately, this property is not fulfilled by the general BDDs defined above (see [23]). Therefore we need further restrictions on the structure of the BDDs.

Definition 3: A *Reduced Ordered* BDD (ROBDD) is a BDD with the following two properties:

1. The BDD is ordered, i.e., there is a fixed order $\pi : \{1, \ldots, n\} \to \{x_1, \ldots, x_n\}$ such that for any non-terminal vertex v $index(low(v)) = \pi(k)$ with $k > \pi^{-1}(index(v))$ ($index(high(v)) = \pi(q)$ with $q > \pi^{-1}(index(v))$) holds if $low(v)$ ($high(v)$) is also a non-terminal vertex.

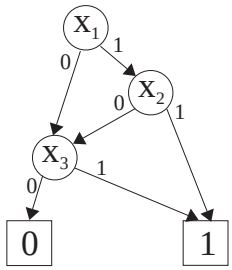


Fig. 1. BDD for $f = x_1x_2 + x_3$

2. The BDD is reduced, i.e., there exists no $v \in V$ with $low(v) = high(v)$ and there are no two vertices v and v' such that the sub-BDDs rooted by v and v' are isomorphic.

Example 1: In Figure 1 the reduced ordered BDD for function $f = x_1x_2 + x_3$ is given. The left (right) outgoing edge of each node v is $low(v)$ ($high(v)$).

Functions denoted by ROBDDs can be manipulated efficiently [5]. For our practical experiments we use a ROBDD package with complemented edges as described in [3].

Since we work only with ROBDDs in the following we briefly call them BDDs.

B. Variable Ordering

The size of a BDD is largely influenced by the choice of the variable ordering. This is illustrated by the following example from [5]:

Example 2: Let $f = x_1x_2 + \ldots + x_{2n-1}x_{2n}$.
If the variable ordering is given by $(x_1, x_2, \ldots, x_{2n})$, i.e., $\pi(i) = x_i \ \forall i$, the size of the resulting BDD is $2n$. On the other hand if the variable ordering is chosen as $(x_1, x_3, \ldots, x_{2n-1}, x_2, x_4, \ldots, x_{2n})$ the size of the BDD is $\Theta(2^n)$. Thus the number of nodes in the graph varies from linear to exponential depending on the variable ordering. In Figure 2 the BDDs of the function $f = x_1x_2 + x_3x_4 + x_5x_6$ with variable orderings $(x_1, x_2, x_3, x_4, x_5, x_6)$ and $(x_1, x_3, x_5, x_2, x_4, x_6)$ are illustrated.

C. Notations

For a constant $b \in \{0, 1\}$ and a variable $x_i \in X$ $f|_{x_i=b}(x_1, \cdots, x_n) = f(x_1, \ldots, x_{i-1}, b, x_{i+1}, \ldots, x_n)$ denotes the *Shannon cofactor* or *restriction* of f with respect to $x_i = b$. Instead of $f|_{x_i=0}$ and $f|_{x_i=1}$ we also write $f_{\overline{x}_i}$ and f_{x_i}, respectively.

The *restriction* of f with respect to a set of variables and constants is defined inductively:

$$f|_{x_{i_1}=b_1, \cdots, x_{i_r}=b_r} = (f|_{x_{i_1}=b_1, \cdots, x_{i_{r-1}}=b_{r-1}})|_{x_{i_r}=b_r}.$$

The *satisfy set* of f is the set of all inputs for which the function value is 1. The *satisfy count* of f, denoted by $|f|$, is the cardinality of this set.

It is easy to see that each node of a BDD is itself the root of a BDD which represents one or more restrictions of f. Two sets of those restrictions will be introduced in the

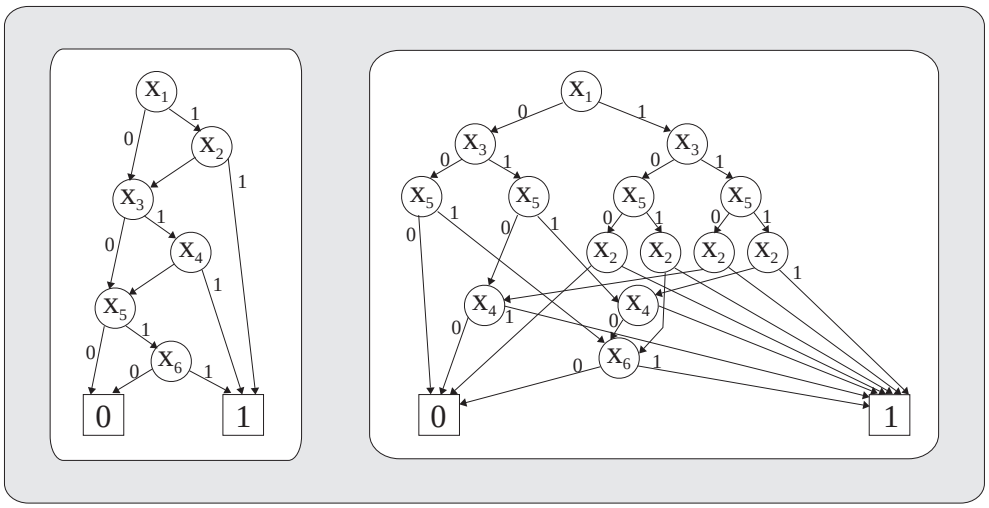


Fig. 2. BDDs of the function $f = x_1x_2 + x_3x_4 + x_5x_6$

following. When using these restrictions for our purposes in this paper, we can assume w.l.o.g. that the variable order is $(x_1, x_2, \ldots, x_n)$. For a variable x_i we define

$$\mathcal{F}^f_{x_i} = \{f|_{x_1=b_1,\cdots,x_{i-1}=b_{i-1}} : b \in \{0,1\}^{i-1}\}$$

as the set of *all* restrictions of f with respect to *all* variables that precede x_i. For x_1 we set $\mathcal{F}^f_{x_1} = \{f\}$. The node which represents a restriction in $\mathcal{F}^f_{x_i}$ can be found if we follow the path from the root using the appropriate vector of constants b. We stop at the first node with label greater than x_{i-1}.

The second set is a little bit more complicated. For two variables x_i, x_j (x_i precedes x_j) and a restriction $g \in \mathcal{F}^f_{x_i}$ we define

$$\mathcal{R}^g_{x_i\overline{x}_j} = \{g|_{x_i=1,\cdots,x_{i+l}=b_l,\cdots,x_j=0} : b \in \{0,1\}^{j-i-1}\}$$

as the set of all restrictions of g with respect to the variables $x_i, x_{i+1}, \ldots, x_{j-1}, x_j$ with x_i set to 1 and x_j set to 0. The set $\mathcal{R}^g_{\overline{x}_i x_j}$ is defined in the same way except that x_i is set to 0 and x_j is set to 1. The node which represents a restriction in $\mathcal{R}^g_{x_i\overline{x}_j}$ can be found as described for $\mathcal{F}^f_{x_i}$ starting at the node that represents g and branching to the right(left) son for nodes with label $x_i(x_j)$. E.g. for the variables x_1 and x_3, and $g = f \in \mathcal{F}^f_{x_1}$ we have

$$\mathcal{R}^g_{\overline{x}_1x_3} = \{g_{\overline{x}_1x_2x_3}, g_{\overline{x}_1\overline{x}_2x_3}\}.$$

We will use these sets to formulate necessary conditions for symmetry and to develop preprocessing algorithms that check these conditions. Note that in the next sections we use the terms $\mathcal{F}^f_{x_i}$ and $\mathcal{R}^g_{x_i\overline{x}_j}$ to denote a set of functions as well as to denote the set of the nodes that represent these functions.

We also use the fact, that a function f which is represented by a BDD G depends essentially on x_i if and only if at least one node in G is labeled with x_i.

D. Symmetry for (In-)Completely Specified Functions

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

First, 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 4: 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 $\forall 1 \leq i \leq k$.

If f is symmetric in a subset λ of the set of input variables, then we say that '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.

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 5: 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 $f'(\epsilon) = f(\epsilon)$ $\forall \epsilon \in D$.

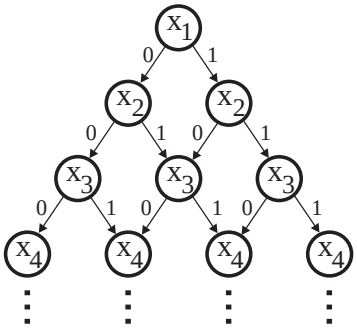
Definition 6: An incompletely specified Boolean function $f : D \to \{0,1\}$ is symmetric in a pair of input variables (x_i, x_j) (in a subset λ of X / in a partition $P = \{\lambda_1, \ldots, \lambda_k\}$ of X) iff there is a completely specified extension f' of f, which is symmetric in (x_i, x_j) (in λ / in P).

III. BDDs for Completely Specified Functions

In this section we focus on completely specified Boolean functions. A polynomial upper bound for the sizes of BDDs of totally symmetric functions is given. Motivated by this symmetric variable orders are defined.

A. Totally Symmetric Functions

For totally symmetric functions it is well known that the size of the BDD is bounded by $O(n^2)$. This is due to the observation that for functions symmetric in (x_i, x_j) the equation $f_{x_i \overline{x}_j} = f_{\overline{x}_i x_j}$ holds. For BDDs this implies that for $\{\pi(1), \pi(2)\} = \{x_i, x_j\}$ the left son of the right son of the root is the right son of the left son of the root. Thus, BDDs representing totally symmetric functions grow in each level at most by one node. This is demonstrated by the following diagram:



A more detailed analysis shows that the least upper bound on the sizes is given by $\Theta(n^2)$ [49], [24], [31].

B. Symmetric Variable Orderings

We now introduce the class of symmetry variable orders that we will use to improve the existing reordering heuristics.

Definition 7: Let f be a partially symmetric function with the set of symmetry sets $S = \{\lambda_1, \ldots, \lambda_k\}$. A variable order π is called a *symmetry variable order* if for each symmetry set $\lambda_i \in S$ there exists j so that $\{\pi[j], \pi[j+1], \ldots, \pi[j+|\lambda_i|-1]\} = \lambda_i$.

By this definition, the class of symmetry variable orders consists of all variable orders where the variables of each symmetry set are located side by side. The BDDs that correspond to symmetry orders are called *symmetry ordered* BDDs. In the remainder of this section the efficiency of symmetry orders will be motivated.

As discussed above the BDD size of any totally symmetric function f is $O(n^2)$. In a symmetry ordered BDD there exist a lot of sub-BDDs where all variables in the upper part form a symmetry set. If k is the size of such a symmetry set, the upper parts of these sub-BDDs consisting of all nodes labeled by variables from the symmetry set have $O(k^2)$ nodes.

Furthermore, the value of a function that is symmetric in some variables $\{x_{i_1}, \ldots, x_{i_q}\}$ does not depend on the exact assignment of these variables but only on their weight $\sum_{j=1}^{q} x_{i_j}$. If one uses symmetry ordered BDDs, this weight is computed in neighboring levels and no information about partial weights has to be kept over several non-symmetric levels – and keeping information may cause large BDD sizes. Symmetry variable orders often avoid this drawback[3].

It is also worth to mention that the restriction to symmetric variable orderings is justified not only by experimental results but also from a theoretical point of view [48].

IV. Detection of Symmetries of Completely Specified Functions

In this section we present an efficient method for determining symmetries of completely specified Boolean functions represented by BDDs. We first give some conditions for symmetry and then present a fast algorithm that can efficiently identify asymmetric structures in BDDs. This algorithm is based on several ideas that have a direct correspondence to efficient algorithms, i.e. algorithms that can be carried out in polynomial time and space on the BDD representation. We assume w.l.o.g. that the variable order is given by $(x_1, x_2, \ldots, x_n)$.

A. Conditions for Symmetries

We give some theorems that we will use to develop methods detecting symmetry and asymmetry of a function.

Lemma 1: Let x, x_i, x_j be three distinct variables. f is symmetric with respect to $\{x_i, x_j\}$ if and only if both cofactors f_x and $f_{\overline{x}}$ are symmetric with respect to $\{x_i, x_j\}$.
Applying Lemma 1 recursively to f_x and $f_{\overline{x}}$ we get:[4]

Corollary 1: f is symmetric with respect to $\{x_i, x_j\}$ if and only if each function $g \in \mathcal{F}_{x_i}^f$ is symmetric with respect to $\{x_i, x_j\}$.
For this, we can restrict ourselves to the functions in $\mathcal{F}_{x_i}^f$ in order to detect symmetries in f.

Lemma 2: If $g \in \mathcal{F}_{x_i}^f$ is symmetric with respect to $\{x_i, x_j\}$ then g either depends on both x_i and x_j or depends neither on x_i nor on x_j.
This is clear, because of the following fact: If g depends on x_i but does not depend on x_j, then these variables cannot be permuted without changing g. With Corollary 1 we get:

Theorem 1: If f is symmetric with respect to $\{x_i, x_j\}$ then each $g \in \mathcal{F}_{x_i}^f$ depends on both x_i and x_j or depends neither on x_i nor on x_j.

Lemma 3: A function $g \in \mathcal{F}_{x_i}^f$ is symmetric with respect to $\{x_i, x_j\}$ if and only if for all $w \in \{0,1\}^{j-i-1}$

$$g|_{x_i=1,\cdots,x_{i+l}=w_l,\cdots,x_j=0} = g|_{x_i=0,\cdots,x_{i+l}=w_l,\cdots,x_j=1}.$$

[3] This approach has been extended to 'nearly' symmetric functions in [40]. In the following we restrict our approach to 'pure' symmetry.
[4] In the following we always assume $i < j$, such that x_i precedes x_j in the variable order.

In Lemma 3, a sufficient and necessary condition for pairwise symmetry is given. Unfortunately, it requires to verify 2^{j-i-1} equations. However, if we consider the two sets of functions that are on the left and on the right side of these equations (that are $\mathcal{R}^g_{x_i\overline{x}_j}$ and $\mathcal{R}^g_{\overline{x}_i x_j}$, respectively), we see that they are equal in the case of symmetry. Note, that the cardinality of these sets is restricted by the number of nodes in the BDD of f. Of course, the equality $\mathcal{R}^g_{x_i\overline{x}_j} = \mathcal{R}^g_{\overline{x}_i x_j}$ does not necessarily imply that all the 2^{j-i-1} equations given above must hold. So, we obtain more efficiency losing sufficiency:

Theorem 2: If f is symmetric with respect to $\{x_i, x_j\}$ then for all $g \in \mathcal{F}^f_{x_i}$ we have

$$\mathcal{R}^g_{x_i\overline{x}_j} = \mathcal{R}^g_{\overline{x}_i x_j}.$$

That means, if there exists at least one function g in $\mathcal{F}^f_{x_i}$ which is not symmetric in $\{x_i, x_j\}$ because of the inequality of the sets $\mathcal{R}^g_{x_i\overline{x}_j}$ and $\mathcal{R}^g_{\overline{x}_i x_j}$, then f is not symmetric in $\{x_i, x_j\}$.

Now let us consider a special case of Theorem 2, namely $x_j = x_{i+1}$, i.e., symmetric variables which are neighbors with respect to the variable order. For $g \in \mathcal{F}^f_{x_i}$ we have

$$\mathcal{R}^g_{x_i\overline{x}_{i+1}} = \{g|_{x_i=1, x_{i+1}=0}\}$$

and analogous for $\mathcal{R}^g_{\overline{x}_i x_{i+1}}$. Both sets contain only one function and they are equal if and only if the equation in Lemma 3 holds. Note that we have to test only one equation. By this we get a necessary and *sufficient* condition for symmetry for pairs $\{x_i, x_{i+1}\}$:

Theorem 3: f is symmetric with respect to $\{x_i, x_{i+1}\}$ if and only if for all $g \in \mathcal{F}^f_{x_i}$

$$\mathcal{R}^g_{x_i\overline{x}_{i+1}} = \mathcal{R}^g_{\overline{x}_i x_{i+1}}.$$

B. Symmetries and Asymmetries

First the basic underlying ideas of our method to find the symmetries of a function f are explained. For that we use structural properties of BDDs as well as function properties.

First, we know that f is symmetric with respect to $\{x_i, x_j\}$ if and only if $f_{x_i\overline{x}_j} = f_{\overline{x}_i x_j}$. That can easily be checked by testing if the BDDs of $f_{x_i\overline{x}_j}$ and $f_{\overline{x}_i x_j}$ are isomorphic. We call it the *naive method.*

Although this method is very popular, a handicap of it is that creating the necessary BDDs may be very time consuming. That is why we have tried to find methods to accelerate symmetry detection by detecting as many asymmetric pairs of variables as possible to be able to avoid the naive symmetry check for those pairs. Of course, these tests have to be done with as little effort as possible and without creating new BDDs.

According to these constraints, we have developed four ideas to detect asymmetric pairs. The first idea is based on a simple function property. The other three ideas make use of Theorem 1, Theorem 2, Theorem 3, and of certain properties of BDDs.

B.1 Idea 1

Our first method uses the fact that the satisfy count $|f_{x_i}|$ is a characteristic of x_i which is independent of the permutation of the input variables of f [34]. Thus, if two variables x_i and x_j are symmetric, then the restrictions f_{x_i} and f_{x_j} have the same satisfy count, such that the following lemma holds:

Lemma 4: $f : \{0,1\}^n \to \{0,1\}$ is asymmetric in $\{x_i, x_j\}$ if $|f_{x_i}| \neq |f_{x_j}|$.

These satisfy counts can be computed by a bottom–up traversal of the BDD of f, i.e., without constructing the BDDs of the restrictions [34]. This can be done in time $O(n \cdot |G|)$, where $|G|$ indicates the number of nodes in the BDD of f. (Note that such an asymmetry test can be done by using any time efficient signature.)

B.2 Idea 2

The background of this idea is Theorem 1. Because of this theorem two variables x_i and x_j $(i < j)$ do not form a symmetric pair if at least one of the restrictions in $\mathcal{F}^f_{x_i}$ depends essentially on x_i but does not depend on x_j, or vice versa.

Consider a restriction $g \in \mathcal{F}^f_{x_i}$. The node v which represents g may be labeled either with x_i or with a variable greater than x_i. In the first case g depends on x_i. However, if the BDD, rooted by the node v, does not contain any node with label x_j, then g does not depend on x_j and thus x_i and x_j do not form a symmetric pair. Thus, we have:

Lemma 5: $f : \{0,1\}^n \to \{0,1\}$ is asymmetric in $\{x_i, x_j\}$ if a node in the BDD of f with label x_i does not have any successor with label x_j.

In the second case g does not depend on x_i. If the BDD, rooted by the node v, contains a node with label x_j, then g depends on x_j and thus x_i and x_j do not form a symmetric pair. With other words, there exists a path from the root (of the BDD of f) to a node with label x_j, which does not contain any node with label x_i, and we have:

Lemma 6: $f : \{0,1\}^n \to \{0,1\}$ is asymmetric in $\{x_i, x_j\}$ if in the BDD of f a node with label x_j can be reached from the root via a path which does not contain any node with label x_i.

To realize this idea, we first establish, for each node v in the BDD of f, the set of all variables which are labels of any successor of v and the set of all variables which are labels of nodes on *each* path from the root to the node v. The sets can be determined for all nodes simultaneously in one bottom–up and one top–down traversal of the BDD of f in time $O(n \cdot |G|)$. Finally, to detect the asymmetries we have to look for missing variables in these sets. This can be done in time $O(n \cdot |G|)$.

B.3 Idea 3

Now, we want to make use of Theorem 2. Here, our task is to construct the two sets of nodes $\mathcal{R}^g_{x_i\overline{x}_j}$ and $\mathcal{R}^g_{\overline{x}_i x_j}$ for each function $g \in \mathcal{F}^f_{x_i}$ and to check their equivalence.

Suppose, we have already checked that $\{x_i, x_j\}$ is not asymmetric according to Theorem 1. Then each restriction g in $\mathcal{F}^f_{x_i}$ depends on neither x_i nor x_j or depends on both, x_i and x_j. If g depends on neither x_i nor x_j, then $\mathcal{R}^g_{x_i\overline{x}_j}$ and $\mathcal{R}^g_{\overline{x}_i x_j}$ are equal, because the variables x_i and x_j will not be tested. So, we do not have to construct these sets. For the case that g depends on both, x_i and x_j, let us establish the sets $\mathcal{R}^g_{\overline{x}_i x_j}$ and $\mathcal{R}^g_{x_i\overline{x}_j}$. The root node v of the BDD of g is labeled with x_i. Constructing $\mathcal{R}^g_{\overline{x}_i x_j}$ means to collect the right sons of nodes with label x_j in the left subgraph of v. In the following, we call these sons *right x_j-sons*. Analogous, $\mathcal{R}^g_{x_i\overline{x}_j}$ contains the left sons of nodes with label x_j in the right subgraph of v. These sons are called *left x_j-sons* in the following. If the left x_j-sons in the right subgraph of v are different from the right x_j-sons in the left subgraph of v, then the set $\mathcal{R}^g_{x_i\overline{x}_j}$ is different from $\mathcal{R}^g_{\overline{x}_i x_j}$. Together with the fact that v is labeled with x_i we get our sufficient condition for asymmetry:

Lemma 7: $f : \{0,1\}^n \to \{0,1\}$ is asymmetric in $\{x_i, x_j\}$ if in the BDD of f one node v with label x_i exists such that the set of the left x_j-sons in the right subgraph of v is different from the set of the right x_j-sons in the left subgraph of v.

To realize this idea, we start for each node v with label x_i a depth–first–search (*dfs*) procedure on the left subgraph and on the right subgraph of v to collect the right x_j-sons and the left x_j-sons, respectively, and compare these sets. Unfortunately, we have to run (at most) $2 \cdot n$ of the *dfs* procedures for each node in the BDD of f. Each *dfs* procedure visits (at most) $|G|$ nodes. This implies an overall run time of $O(n \cdot |G|^2)$. However, the number of nodes that are visited in one search is not very large if the distance between the two variables x_i and x_j is small. Furthermore, there is a hope that an existing asymmetry is detected early and so the number of searches keeps small. Thus, it seems possible that the run time on average is not quadratic in the size of the BDD. This presumption is clearly underlined by our experimental results.

For pairs of *neighboring* variables we get a special case of *idea* 3 which we will call *idea* 3_n in the following:

B.4 Idea 3_n

Using Theorem 3 we can formulate:

Lemma 8: $f : \{0,1\}^n \to \{0,1\}$ is symmetric in $\{x_i, x_{i+1}\}$ if and only if in the BDD of f for all nodes v with label x_i the left x_{i+1}-son in the right subgraph of v is the same node as the right x_{i+1}-son in the left subgraph of v.

The procedure to test this condition works similar to the one for *idea* 3. For each node we have to start only two *dfs* calls that visit at most four nodes. So, the complete procedure requires time $O(|G|)$. Note, that, similarly to *idea* 3, we need to filter out asymmetries with *idea* 1 in order to guarantee the correctness of this procedure.

Although the introduced ideas work very well in practice, it cannot be guaranteed that all asymmetries of a function can be detected using them. For all other pairs, for that no symmetry or asymmetry could be established so far, we use the naive method to test if they are symmetric or not.

V. Symmetries of Incompletely Specified Functions

In this section we discuss the problem of detecting symmetries of incompletely specified Boolean functions represented by BDDs. First, we outline the occurring difficulties. This leads to the definition of 'strong symmetries'. Then, we discuss an algorithm to solve the minimum sized partitioning of the variables of an incompletely specified function into symmetry groups. And finally, we investigate the relationship of our don't care assignment to maximize the number of symmetries and the BDD minimization procedure presented by Chang [8] and Shiple [47].

A. Difficulties with Symmetry of Incompletely Specified Functions

In order to minimize the BDD 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 [12] or [27]):

Example 3: The following function shows that symmetry in two variables does not lead to an equivalence relation on the variable set in the case of incompletely specified Boolean functions:

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

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 does not form an equivalence relation, it will be much more difficult to deduce symmetries in larger variable sets from symmetries in pairs of variables than in the case of completely specified Boolean functions.

In the rest of the paper we use symmetry graphs to illustrate symmetries of Boolean functions. 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). For completely specified Boolean functions f G^f_{sym} has a special structure: The connected components of the graph form cliques as symmetry in two variables forms an equivalence relation. For incompletely specified functions there is not any structural property. On the contrary one can prove (see proof of Theorem 4), that for every graph G with n nodes, there is an (incompletely specified) Boolean function $f : D \to \{0,1\}$ such that the symmetry graph of f coincides with G. All possible graphs can occur as symmetry graphs of an incompletely specified function.

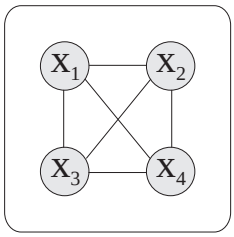


Fig. 3. Symmetry graph of the function of Example 2

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

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

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 3. It is 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'' 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 $\{x_1,x_2,x_3,x_4\}$.

Example 4 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 $P = \{\{x_1,x_3\},\{x_2,x_4\}\}$).

B. 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 does not form an equivalence relation on the variable set X. If we change the definition of symmetry of incompletely specified functions as given in the following, symmetry in pairs of variables provides an equivalence relation as in the case of completely specified functions:

Definition 8 (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 *strong* symmetry of incompletely specified functions the symmetry defined so far is called *weak* symmetry. (Notice that for completely specified Boolean functions *strong* symmetry and *weak* symmetry are identical.)

The following lemma holds for strong symmetry:

Lemma 9: 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 9 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 $\forall 1 \le i \le k$ 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 6 that there is an extension of f which is strongly symmetric in P.

Before we deal with the computation of extensions of incompletely specified Boolean functions which are strongly symmetric in minimum sized variable partitions, we will characterize weak and strong symmetry in variable partitions in more detail. To do this, we need the term of the 'weight class' of a given partition.

Definition 9 (Weight class of a partition P)
Let $P = \{\lambda_1,\ldots,\lambda_k\}$ be a partition of $\{x_1,\ldots,x_n\}$. We call $w^1(\epsilon_1,\ldots,\epsilon_n) = \sum_{i=1}^n \epsilon_i$ the 1–weight of $(\epsilon_1,\ldots,\epsilon_n)$ and $w^0(\epsilon_1,\ldots,\epsilon_n) = n - w^1(\epsilon_1,\ldots,\epsilon_n)$ the 0–weight of $(\epsilon_1,\ldots,\epsilon_n) \in \{0,1\}^n$. For $\lambda_i = \{x_{i_1},\ldots,x_{i_l}\}$, $w^1_{\lambda_i}(\epsilon_1,\ldots,\epsilon_n) = \sum_{j\in\{i_1,\ldots,i_k\}} \epsilon_j$ is the 1–weight of the 'λ_i–part' of $(\epsilon_1,\ldots,\epsilon_n)$.
$C^P_{w_1,\ldots,w_k} = \{(\epsilon_1,\ldots,\epsilon_n) \in \{0,1\}^n : w^1_{\lambda_i}(\epsilon_1,\ldots,\epsilon_n) = w_i,\ 1 \le i \le k\}$ is called *weight class* of the partition P with weights $(w_1,\ldots,w_k)$.

Example 5: Let $P = \{\{x_1,x_2\},\{x_3,x_4,x_5\}\}$. $C^P_{1,2}$ is the subset of all vertices of $\{0,1\}^n$ with a 1–weight 1 of the $\{x_1,x_2\}$–part and a 1–weight 2 of the $\{x_3,x_4,x_5\}$–part, i.e., the subset of all vertices with exactly one 1 in the first two components and exactly two 1's in the remaining components:

$$C^P_{1,2} = \{(0,1,0,1,1),(0,1,1,0,1),(0,1,1,1,0),$$
$$(1,0,0,1,1),(1,0,1,0,1),(1,0,1,1,0)\}.$$

By means of 'weight classes' there is an easy characterization of weak and strong symmetry:

Lemma 10: Let $P = \{\lambda_1,\ldots,\lambda_k\}$ be a partition of $\{x_1,\ldots,x_n\}$. $f : D \to \{0,1\}$ is

(1) strongly symmetric in P iff

$$\forall 0 \le w_i \le |\lambda_i| \quad (1 \le i \le k)$$

$$f(C^P_{w_1,\ldots,w_k}) = \begin{cases} \{0\} & \text{or} \\ \{1\} & \text{or} \\ \{dc\} & \end{cases}$$

(2) (weakly) symmetric in P iff

$$\forall 0 \le w_i \le |\lambda_i| \quad (1 \le i \le k) \quad \{0,1\} \not\subseteq f(C^P_{w_1,\ldots,w_k}).$$

Proof: See Appendix A. ■

C. Minimum Sized Partition

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

Given: Incompletely specified function $f : D \rightarrow \{0,1\}$, represented by BDDs for f_{on} and f_{dc}. (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.)

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 the following theorem by a polynomial–time transformation from the NP–complete problem 'Partition into Cliques' (**PC**) (see [22]) to MSP:

Theorem 4: MSP is NP-hard.

Proof: See Appendix B. ■

To solve the problem heuristically, we use a heuristic for 'Partition into Cliques' for the symmetry graph G^f_{sym} of f. However, the examples in Section V-A showed that f is *not* symmetric in *all* partitions 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 11: A graph $G = (V, E)$ can be partitioned into k disjoint cliques iff $\overline{G} = (V, \overline{E})$ can be colored with k colors. ($\overline{G}$ 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., $\overline{E} = \{\{v, w\} : \{v, w\} \notin E\}$.)

Thus, heuristics for node coloring can be directly used for the solution of partition into cliques. Nodes with the same color in $\overline{G}$ form an 'independent set' and thus a clique in G. Our implementation is based on Brélaz algorithm for node coloring [4] which has a run time of $O(N)$ in an implementation of Morgenstern [39], where N denotes the number of nodes of the graph which has to be colored. It is a greedy algorithm, which colors node by node and does not 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 [4], [39]).

Figure 4 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 $[x_j]$ ($[x_j]$ denotes λ_q, if $x_j \in \lambda_q$ and $P = \{\lambda_1, \ldots, \lambda_k\}$). 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 12), and we can assign don't cares of f such that f is strongly symmetric in P (line 18).

The fact that the conditions given in the algorithm imply that f is symmetric in the new partition P is shown in Lemma 12. In addition we have to point out how f can be made 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:

Lemma 12: Let $f : D \rightarrow \{0,1\}$ 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 $P' = P \setminus \{[x_j], \{x_i\}\} \bigcup \{[x_j] \cup \{x_i\}\}$.

Proof: Let $P = \{\lambda_1, \ldots, \lambda_k\}$ and w.l.o.g. $\lambda_1 = \{x_i\}$, $\lambda_2 = [x_j]$. Then we have $P' = \{\lambda_1 \cup \lambda_2, \lambda_3, \ldots, \lambda_k\}$. Because of Lemma 10, we have to show that there is no weight class $C^{P'}_{w_2,\ldots,w_k}$ of P' with $\{0,1\} \subseteq f(C^{P'}_{w_2,\ldots,w_k})$.

Case 1: $w_2 \geq 1$

$C^{P'}_{w_2,\ldots,w_k}$ can be written as a disjoint union of two weight classes of P:

$$C^{P'}_{w_2,\ldots,w_k} = C^{P}_{0,w_2,\ldots,w_k} \cup C^{P}_{1,w_2-1,w_3,\ldots,w_k}.$$

Since f is strongly symmetric in P, $|f(C^P_{0,w_2,\ldots,w_k})| = |f(C^P_{1,w_2-1,w_3,\ldots,w_k})| = 1$ holds according to Lemma 10. Suppose $\{0,1\} \subseteq f(C^{P'}_{w_2,\ldots,w_k})$, then we have $f(C^P_{0,w_2,\ldots,w_k}) = c$ and $f(C^P_{1,w_2-1,\ldots,w_k}) = \overline{c}$ for $c \in \{0,1\}$.

This leads to a contradiction to the condition that f is symmetric in x_i and x_j, since there are $\epsilon \in C^P_{0,w_2,\ldots,w_k}$ and $\delta \in C^P_{1,w_2-1,\ldots,w_k}$ such that ϵ results from δ only by

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:

```
1   Compute symmetry graph G^f_sym = (V, E) of f (or complement(G^f_sym) = (V, Ē)).
2   ∀1 ≤ k ≤ n : color(x_k) := undef.
3   P = {{x_1}, {x_2}, ..., {x_n}}
4   node_candidate_set := {x_1, ..., x_n}
5   while (node_candidate_set ≠ ∅) do
6       /* f is strongly symmetric in P */
7       Choose x_i ∈ node_candidate_set according to Brélaz/Morgenstern criterion
8       color_candidate_set := {c : 1 ≤ c ≤ n, ∃x_j with {x_i, x_j} ∈ Ē and color(x_j) = c}
9       while (color(x_i) = undef.) do
10          curr_color := min(color_candidate_set)
11          color(x_i) := curr_color
12          if (∃ colored node x_j with color(x_j) = color(x_i))
13            then
14                if (f symmetric in (x_i, x_j))
15                  then
16                      P := P \ {[x_j], {x_i}} ⋃ {[x_j] ∪ {x_i}}
17                      /* f is symmetric in P */                          (*)
18                      Make f strongly symmetric in P.                    (**)
19                  else
20                      color_candidate_set := color_candidate_set \ {curr_color}
21                      color(x_i) := undef.
22                fi
23          fi
24      od
25      node_candidate_set := node_candidate_set \ {x_i}
26  od
```

Fig. 4. Algorithm to solve MSP

exchange of the ith and jth component, but $f(\epsilon) = c$ and $f(\delta) = \overline{c}$.

Case 2: $w_2 = 0$
$C^{P'}_{w_2,\ldots,w_k} = C^{P}_{0,w_2,\ldots,w_k}$ and $\{0,1\} \not\subseteq f(C^{P'}_{w_2,\ldots,w_k})$ follows from the strong symmetry of f in P.

■

Remark 1: The statement of Lemma 12 is *not* correct, if we replace 'f strongly symmetric in P' by 'f (weakly) symmetric in P' or if we don't assume $|[x_i]| = 1$. But note that the given conditions coincide exactly with the conditions existing in the algorithm.

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 BDD representations of f_{on}, f_{dc} and f_{off} by the procedure *make_strongly_symm* in Figure 5.

We can use a sequence of calls of the procedure *make_strongly_symm* to make f strongly symmetric in the partition P in line 18 of the algorithm. For this purpose we can prove the following theorem:

Theorem 5: Let $f : D \to \{0,1\}$ strongly symmetric in P, $\{x_i\}, [x_{j_1}] \in P$, $[x_{j_1}] = \{x_{j_1}, \ldots, x_{j_k}\}$, $f =: f^{(0)}$ symmetric in (x_i, x_{j_1}).

$$\begin{aligned} f^{(1)} &= make\_strongly\_symm(f^{(0)}, x_i, x_{j_1}) \\ f^{(2)} &= make\_strongly\_symm(f^{(1)}, x_i, x_{j_2}) \\ &\vdots \\ f^{(k)} &= make\_strongly\_symm(f^{(k-1)}, x_i, x_{j_k}). \end{aligned}$$

Then $f^{(k)}$ is strongly symmetric in

$$P' = P \setminus \{[x_{j_1}], \{x_i\}\} \bigcup \{[x_{j_1}] \cup \{x_i\}\}.$$

Proof: See Appendix C. ■

There are examples where we need the complete sequence of calls given in the theorem. However, in many cases there is a $p < k$ such that $f^{(p)}$ does not differ from $f^{(p-1)}$. We can prove that the sequence of calls can be stopped in such cases with the result $f^{(k)} = f^{(p-1)}$.

D. Compatibility with other BDD Minimization Techniques

In the last section we presented an algorithm to compute a minimum sized partition P of the input variables in which an incompletely specified function f is symmetric. In addition we assigned values to don't cares to make f *strongly* symmetric in P. Usually the result will still contain don't cares after this assignment.

We try to make use of these remaining don't cares by applying the technique of Chang [8] and Shiple [47] to further minimize BDD sizes. Since this method removes don't cares, we have to ask the question, if the method can destroy symmetries which were found earlier.

Procedure *make_strongly_symm*

Input: $f : D \rightarrow \{0,1\}$, 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. $f'_{dc} = \overline{f'_{on} + f'_{off}}$

Fig. 5. Procedure *make_strongly_symm*

The answer to this question given in this section is that we can preserve these symmetries using a slightly modified version of Chang's technique.

The algorithm proposed by Chang [8] minimizes the number of nodes at every level of the BDD 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 two adjacent variables is minimized. After the minimization of nodes at a certain level of the BDD 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 BDD. We can prove that under certain conditions, this method does preserve strong symmetry:

Lemma 13: Let f be an incompletely specified Boolean function which is strongly symmetric in $P = \{\lambda_1, \ldots, \lambda_k\}$ and assume that the variable order of the BDD 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 in [8] to cut lines between two symmetric groups λ_i and λ_{i+1}, then it preserves strong symmetry in P.

Proof: See Appendix D. ■

Since we will use such 'symmetric orders' to minimize BDD sizes (see Section VI), we only have to restrict *remove_z* to cut lines between symmetric groups to guarantee that we will not lose any symmetries.

VI. Experimental Results

A. Completely Specified Functions

In this section we present experimental results for completely specified functions, in the next section results for incompletely specified functions.

A.1 Asymmetry Test

We compare the performance of our sophisticated symmetry check with the naive one. For that we have implemented the ideas described in the last section. We have used the CMU-BDD package contained in SIS-1.2 [46]. This package is based on the ideas of [3]. The algorithms were tested for the multi-level circuits[5] from the LGSYNTH91 benchmark set. In Table I we give only results for benchmarks where run times for the naive symmetry check (or our procedure) were larger than 10 CPU seconds measured on a SPARC station 20.

In Table I the first four columns provide information about the name of the circuit, the number of primary inputs, the number of primary outputs, and the number of nodes in the BDDs. Columns 5–9 show CPU times in seconds for the *naive* method and implementations of our ideas of Section IV, respectively. Column 6 shows CPU times for the symmetry detection using only *idea* 1, column 7 CPU times using *idea* 1 followed by *idea* 2. In column 8 CPU times for the sequence of running *idea* 1, *idea* 2, and *idea* 3_n are given and in column 9 CPU times for *idea* 1, *idea* 2, *idea* 3_n, and *idea* 3. The CPU times include the run times of the naive tests applied to those variable pairs for which asymmetry has not been detected. Note, that the realization of *idea* 3 starts with a realization of the special case for neighboring variables in order to filter out symmetries of those pairs of variables. Column 10 (*symsets*) gives information on symmetries of the benchmark circuits: 2(5) means that there are two symmetry sets of five variables.

For the circuits in Table I the run time for our method decreases drastically compared with the naive method. The experimental results show that already the application of *idea* 1 leads to a large reduction of run times. For larger examples (e.g. C2670, C7552) application of ideas 2, 3_n and 3 leads to further reductions.

The reason for this is the obviously large ratio of asymmetric pairs detected by the asymmetry preprocessing, as shown by Table II. Table II gives the number of computations of cofactors to check symmetry for the different methods. The number of cofactor computations needed to check symmetry is decreased step by step by the sequence of running *idea* 1, *idea* 2, *idea* 3_n and *idea* 3. In many cases ideas 1, 2, 3_n and 3 to check asymmetry (and symmetry in case of idea 3_n) are sufficient in the sense that no cofactor computation is necessary at the end, i.e., a 0 in column *Idea* $1, 2, 3_n, 3$ denotes that all pairs of asymmetric variables have been found by ideas 1, 2, 3 or 3_n and that all pairs of symmetric variables have been found by idea 3_n. The last column gives the number of cofactors which have to be computed after application of ideas 1, 2, 3_n and 3 for pairs of variables in which the function is asymmetric. It shows that almost all pairs of asymmetric variables could be detected by the sequence of idea 1, 2, 3_n and 3.

[5] except C6288.blif and i10.blif

TABLE I
CPU TIMES IN SECONDS.

circuit				*time*					*symsets*
name	*in*	*out*	*nodes*	*naive*	*Idea* 1	*Idea* 1, 2	*Idea* $1, 2, 3_n$	*Idea* $1, 2, 3_n, 3$	
C1355	41	32	29609	109.9	2.9	2.9	2.9	2.9	41(1)
C1908	33	25	7764	19.3	0.6	0.6	0.6	0.7	33(1)
C2670	233	140	7469	140.4	17.5	11.4	8.9	7.2	1(8) 2(2) 221(1)
C3540	50	22	27666	40.4	3.0	3.0	3.0	3.0	50(1)
C499	41	32	34113	128.5	3.4	4.0	4.0	4.0	41(1)
C5315	178	123	2433	231.9	1.9	2.1	2.1	2.1	2(2) 174(1)
C7552	207	108	9808	186.5	14.5	13.8	9.3	9.3	2(5) 4(4) 1(3) 6(2) 166(1)
apex6	135	99	1621	90.9	0.8	1.1	1.0	0.9	1(2) 133(1)
dalu	75	16	2235	15.8	0.4	0.6	0.6	0.6	1(2) 73(1)
des	256	245	7255	1100.1	5.6	5.6	5.7	5.6	256(1)
example2	85	66	757	13.9	0.2	0.3	0.3	0.3	1(2) 83(1)
frg2	143	139	3748	142.6	1.8	2.7	2.7	2.7	1(2) 141(1)
i2	201	1	1585	62.6	2.4	2.5	0.6	0.6	2(64) 3(16) 3(4)
i4	192	6	348	41.6	0.4	0.5	0.2	0.2	16(3) 50(2) 44(1)
i5	133	66	961	97.5	0.4	0.4	0.4	0.4	133(1)
i6	138	67	415	47.9	0.2	0.2	0.2	0.2	138(1)
i7	199	67	503	113.7	0.4	0.4	0.4	0.4	199(1)
i8	133	81	2637	93.8	1.1	1.2	1.1	1.1	133(1)
i9	88	63	2391	66.8	0.7	0.7	0.7	0.7	88(1)
pair	173	137	4918	132.4	2.6	3.9	3.8	3.8	2(2) 169(1)
rot	135	107	10223	556.8	4.9	6.9	5.9	5.7	2(3) 2(2) 125(1)
too_large	38	3	4402	31.5	0.8	0.9	0.5	0.5	1(3) 3(2) 29(1)
x3	135	99	996	52.0	0.5	0.7	0.7	0.7	1(2) 133(1)
x4	94	71	756	24.3	0.2	0.4	0.4	0.4	1(2) 92(1)

TABLE II
NUMBER OF COFACTORS WHICH HAVE TO BE COMPUTED

circuit	*No. of cofactors*					*No. of cofactors for asymmetric pairs*
	naive	*Idea* 1	*Idea* 1, 2	*Idea* $1, 2, 3_n$	*Idea* $1, 2, 3_n, 3$	
C1355	1640	0	0	0	0	0
C1908	1056	0	0	0	0	0
C2670	3639353	118546	13012	10660	1560	12
C3540	9925	0	0	0	0	0
C499	1640	2	2	0	0	0
C5315	1336816	10719	563	123	123	0
C7552	1734673	14203	4369	1669	1525	17
apex6	282351	621	321	74	0	0
dalu	11246	17	17	16	16	0
des	3173371	0	0	0	0	0
example2	114807	66	66	0	0	0
frg2	789287	139	139	0	0	0
i2	4794	180	180	2	2	0
i4	80308	492	492	72	72	0
i5	239690	0	0	0	0	0
i6	421264	0	0	0	0	0
i7	859421	0	0	0	0	0
i8	150515	0	0	0	0	0
i9	84488	0	0	0	0	0
pair	152263	1232	274	0	0	0
rot	233762	659	643	107	107	0
too_large	1318	15	15	0	0	0
x3	283519	336	336	83	0	0
x4	169498	71	71	0	0	0

A.2 Sifting Using Symmetries

Here we consider statistical and benchmark results with respect to completely specified Boolean functions.

A.2.a Statistical Results. Due to the remarks in Section III-B and the theoretical results proven in [48], it seems to be reasonable to consider only symmetric variable orders for BDD minimization. To check this assumption, we investigated all partially symmetric functions with three, four

and five inputs. For each function we determined the number of general orders and the number of symmetry orders that create a BDD with $x\%$ more nodes than the minimum BDD. Using these data, we computed the probability to get a BDD with more than $x\%$ more nodes than the minimum for an arbitrary function and an arbitrary order. Figure 6 shows the result obtained for the four and five input functions. The dashed line shows the probability that the BDD for an arbitrary partially symmetric function with an arbitrary *symmetry* order has more than $x\%$ additional nodes with respect to the minimum. The solid line shows the same for *general* orders. It turns out that the probability to get an $x\%$ oversized BDD with a symmetry order is always smaller than it is for general orders. This shows from a statistical point of view that symmetry orders constitute an efficient subclass of variable orders.

Furthermore, this statistical study gives a negative answer to the question, whether for each BDD a symmetry order exists that gives the minimal number of nodes. Consider the $\{x_0, x_1\}$–symmetric function shown in Figure 7. For this function each symmetry ordered BDD has 4 internal nodes while the minimum BDD has only 3 internal nodes.

For completeness, we computed the number of partially symmetric functions for that each symmetry order results in a non-minimal BDD. For the 120 partially symmetric functions with three inputs there are 24 (20%) such functions.

For the 20.548 partially symmetric functions with four inputs there are 960 (4.7%) and for the 162.535.140 partially symmetric functions with five inputs there are only 972.280 (0.6%) such functions. The distance of the best symmetry order to the minimum was at most two nodes. We confirmed our results by performing experiments with some functions with more than five inputs. Thus, it seems to be a good heuristic to confine ourselves to the subclass of symmetry variable orders.

In [41] it was conjectured that for BDDs *without* complemented edges for each function one of its symmetry orders results in a BDD of minimal size. A negative answer to this conjecture was given by our experiments with this kind of BDDs. For the four input function

$$f = \overline{x}_1 x_2 x_3 + x_1 \overline{x}_2 x_3 + x_1 x_2 \overline{x}_3 + \overline{x}_1 \overline{x}_2 \overline{x}_3 x_4$$

which is symmetric in $\{x_1, x_2, x_3\}$ the BDD with best symmetry order has size 9 and the minimum size is 8 (with variable orders x_1, x_2, x_3, x_4 and x_1, x_2, x_4, x_3, respectively). For BDDs without complemented edges there are 80 (0.4%) partially symmetric functions with four inputs and 1.262.800 (0.8%) functions with five inputs without a minimum symmetry order.

A.2.b Benchmark Results. Now we will show the efficiency of the symmetry variable orders in practical application. We processed 109 combinational two-level and multi-level circuits from the LGSYNTH91 benchmark set. We also processed each primary output of each circuit separately, since the single primary outputs of a multiple output function sometimes have more symmetry. Symmetry detection was executed on the BDDs using the algorithm proposed above. We slightly modified this algorithm to detect equivalence symmetry as well. The notion of equivalence symmetry was introduced by Hurst [25] and describes the situation that not $\{x_i, x_j\}$ but $\{x_i, \overline{x_j}\}$ is a symmetric pair. The additional consideration of equivalence symmetry results in about 10% more symmetry.

If a BDD is to be created from a circuit description, a heuristic [33] generates a good initial order which is not necessarily a symmetry order. As discussed above, the size of the BDD may be reduced, if the initial non-symmetry order is transformed into a symmetry one. We have applied three algorithms to get a symmetry order. They differ only in the way they select the new position for a symmetry set. Heuristic first selects as position for a symmetry set the position of the first variable of the symmetry set, median selects the position of the middle variable and last selects the position of the last symmetric variable.

Heuristic best calls all three methods and then selects the best order. The suffix _so denotes the methods that handle each primary output separately. The results obtained by initial reordering are shown in Table III. The first column gives the name of the reordering heuristic. The second, third and fourth column shows the total number of benchmark functions where the size of the symmetry ordered BDD is smaller, equal-sized, or larger than the initial one when it was reordered with the corresponding heuristic. The last column shows the total number of nodes of all BDDs and the average improvement over all benchmarks.

For the 109 multiple output functions we detected 56 to be partially symmetric. The initial ordering heuristic already generates a symmetry order for 39 of these functions. For more than half of the remaining non-symmetry ordered BDDs the order has been improved by each of the three symmetry reordering methods and the overall number of nodes decreases. The best heuristic seems to be last and we select it for our next experiments. However, row best shows that the heuristics work well on different functions. There are only three of the single output functions for which all three heuristics generate a symmetry ordered BDD that is larger than the initial one. This shows that symmetry orders are also good in practice.

To reduce the size of a BDD several reordering heuristics have been developed. Two of them, win3 and sift [43] are implemented in the CMU-BDD package. To work with symmetry orders we make use of the variable blocking feature of the CMU-BDD package. Before starting reordering, we block the symmetric variables which were made adjacent by last. The modified heuristics are called Swin3 and Ssift, respectively.

For all symmetric functions from the benchmark set the original heuristics win3 and sift and the modified heuristics Swin3 and Ssift were applied to the initial BDDs. Results are presented in Table IV. The first column denotes the reordering heuristic. The second, third and fourth column shows the total number of benchmark functions for that the modified heuristics generate a smaller, equal-sized, or larger BDD than the original heuristic. Column nodes shows

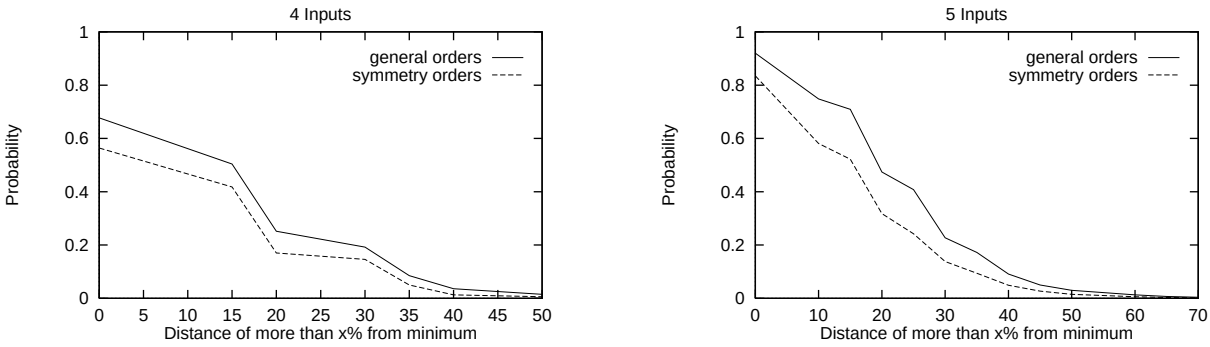


Fig. 6. Distribution of general orders and symmetry orders

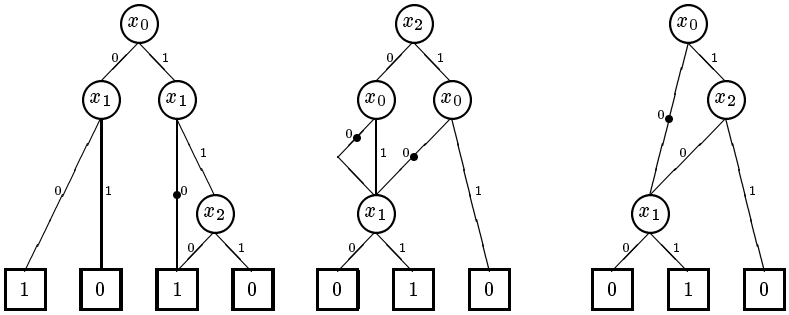


Fig. 7. BDDs of $f = \overline{x}_0\overline{x}_1 + x_0x_1\overline{x}_2$

TABLE III
INITIAL ORDERING WITH SYMMETRY ORDERS

Heuristic	BDD size			nodes		Heuristic	BDD size			nodes	
	<	=	>				<	=	>		
initial				58688		initial_so				49199	
first	9	5	3	58432	1.3%	first_so	312	236	101	47275	1,2%
median	14	3	0	58020	1.5%	median_so	398	174	77	46353	1.5%
last	14	3	0	58007	1.8%	last_so	409	161	79	46362	1.6%
best	15	2	0	57888	2.5%	best_so	534	112	3	45252	2.4%

TABLE IV
REORDERING WITH SYMMETRY ORDERS

Heuristic	BDD size			nodes		time (sec)
	<	=	>			
win3				66350		14
Swin3	25	29	2	64200	5.7%	16
sift				33878		92
Ssift	26	26	4	33149	7.1%	93
win3_so				67961		36
Swin3_so	693	1443	42	63668	3.4%	41
sift_so				58177		116
Ssift_so	452	1695	4	54970	2.6%	99

the number of nodes of all the optimized BDDs and the average improvement over all benchmarks. Column time shows the run time[6] of the heuristics. The additional overall run time for symmetry detection for multiple-output and single-output functions is about 88 seconds.

It is shown that the heuristics that use symmetry orders generate better or same results in most cases. Swin3 saves 5.7% nodes and Ssift3 saves 7.1% nodes on the average. The run time for symmetric reordering remains nearly the same. Unfortunately, there is the extra run time for symmetry detection. This increases the run time of sift in general by factor 2 and of win3 up to factor 7. One can overcome this difficulty if the symmetry detection is integrated in the reordering method following *idea* 3_n above (see also [41]).

Table V shows the effect of symmetry based reordering for some individual benchmarks. In column symsets information on symmetry is given (like in the previous section 2(3) means that there are two symmetry sets of three input variables). The following columns show the BDD size achieved by the mentioned heuristics. The leading S denotes the symmetric version. (Column init gives the initial BDD sizes (sifting was used as a dynamic reordering method to compute these BDDs) and column Sinit gives the result of making symmetric variables adjacent by heuristic last as described above.) If the symmetric reordering results in the same size as the original the results are omitted.

It is shown that the symmetry modified algorithms in general outperform the original ones. Furthermore, even

[6] All run times are seconds on SPARCstation 10/64 MB.

TABLE V
BENCHMARK RESULTS OF REORDERING WITH SYMMETRY ORDERS

circuit	symsets				init	Sinit	win3	Swin3	sift	Ssift
C2670			1(8)	2(2)			7306	7300		
C5315				2(2)			2407	2406	2379	2378
C7552	2(5)	4(4)	1(3)	6(2)			9747	9727	9415	8838
C880				3(2)			7134	7132	5164	4879
apex2			1(3)	3(2)	2947	2846	910	634	700	654
cps				1(4)	1455	1445	1301	1294	1035	991
ex4				14(2)	895	822	692	691	537	539
seq				2(2)	5638	5532	3737	2586		
t481				8(2)	63	33	33	21	33	31
vg2				2(2)	390	385			132	146
comp				16(2)			146	128	146	107
count				1(2)			232	201	201	82
dalu				1(2)	4575	4346			1322	1323
frg2				1(2)					2297	2171
i2		2(64)	3(16)	3(4)	1586	1582	795	298		
i4			16(3)	50(2)	349	333	333	245	308	233
lal				5(2)	122	110	97	95	75	72
my_adder			1(3)	15(2)			457	452	457	411
pcler8_cl				1(2)			138	122	130	86
rot			2(3)	2(2)	10224	10222	8212	8204	4574	4568
too_large			1(3)	3(2)			667	676	500	439
x1				1(2)	1211	1190	784	799	544	518
z4ml			1(3)	2(2)	37	30	21	17	24	17

a small number of symmetry sets and variables can cause a large improvement. For example, for seq with only two symmetry sets of size two Swin3 saves about 30% of all nodes and for count with only one symmetry pair Ssift saves about 60%. Thus, symmetry based ordering is not only suitable for functions with a very large number of symmetries.

B. Incompletely Specified Functions

We have carried out experiments to test the effect of the algorithms for symmetry detection in the case of incompletely specified Boolean functions.

To generate incompletely specified functions from completely specified functions, we used a method proposed in [8]: After collapsing each benchmark circuit to two level form, we randomly selected cubes in the on-set with a probability of 40% to be included into the don't care set[7]. The last three Boolean functions in Table VI are partial multipliers $partmult_n$[8].

We performed three experiments: First of all, we applied symmetric sifting to the BDDs representing the on-set of each function. The results are shown in column 6 (sym_sift) of Table VI. The entries are BDD 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 *sym_group* of Table VI 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. On the average, we can improve the BDD 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 [8] and Shiple [47] according to Lemma 13. Lemma 13 leads to a modification of the technique of Chang which does not destroy strong symmetry supplied by *sym_group*: We restrict the *remove_z* operation [8] only to cut lines between groups of symmetric variables. Since our technique to restrict *remove_z* to cut lines between symmetric groups does not destroy the symmetric groups, we can perform symmetric sifting after the node minimization with the same symmetric groups as before. Figure 8 illustrates our modification of Chang's technique. Column *sym_cover* of Table VI shows the resulting BDD sizes. On the average, the new technique leads to an improvement of the BDD sizes by 70%.

A comparison to the results of the *restrict* operator [10] (applied to BDDs whose variable order was optimized by regular *sifting*) in column *restrict* of Table VI shows that our BDD sizes are on the average 44% smaller. Even if sifting is called again after the *restrict* operator has been applied, the improvement is still more than 40% on average (see column $restrict\_sift$).

Finally, we carried out the same experiment once more, but this time the probability for a cube to be included in the don't care set was reduced to 10% (instead of 40%)[9].

[7] Because of this method to generate incompletely specified functions we had to confine ourselves to benchmark circuits which could be collapsed to two level form.

[8] 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.

[9] Note that the sizes of the don't care sets for the partial multipliers $partmult_n$ are fixed, since these don't care sets arise in a 'natural

TABLE VI

EXPERIMENTAL RESULTS. THE TABLE SHOWS THE NUMBER OF NODES IN THE BDDS OF EACH FUNCTION. NUMBERS IN PARENTHESIS SHOW THE CPU TIMES (MEASURED ON A SPARCSTATION 20 (96 MBYTE RAM)).

circuit	in	out	*restrict*	*restrict_sift*	*sym_sift*	*sym_group*		*sym_cover*	
5xp1	7	10	63	63	67	66	(0.2 s)	53	(0.5 s)
9symml	9	1	67	65	108	25	(0.3 s)	25	(0.4 s)
alu2	10	6	192	182	201	201	(0.7 s)	152	(2.6 s)
apex6	135	99	993	940	1033	983	(267.6 s)	612	(459.7 s)
apex7	49	37	730	716	814	728	(27.7 s)	340	(52.2 s)
b9	41	21	213	211	256	185	(8.6 s)	122	(11.5 s)
c8	28	18	110	98	156	95	(1.7 s)	70	(3.2 s)
example2	85	66	497	496	491	484	(69.2 s)	416	(119.4 s)
mux	21	1	32	32	34	29	(0.6 s)	29	(0.7 s)
pcler8	27	17	111	111	78	73	(1.9 s)	72	(3.3 s)
rd73	7	3	75	74	76	34	(0.3 s)	27	(0.4 s)
rd84	8	4	135	132	144	42	(0.7 s)	42	(0.7 s)
sao2	10	4	89	89	104	104	(0.4 s)	70	(0.8 s)
x4	94	71	814	812	829	633	(121.9 s)	485	(203.4 s)
z4ml	7	4	47	46	51	32	(0.2 s)	17	(0.3 s)
partmult3	9	6	70	65	152	35	(1.0 s)	29	(1.2 s)
partmult4	16	8	307	294	971	222	(49.5 s)	114	(50.6 s)
partmult5	25	10	857	843	4574	998	(1540.4 s)	365	(1548.4 s)
total			5402	5269	10139	4969		3040	

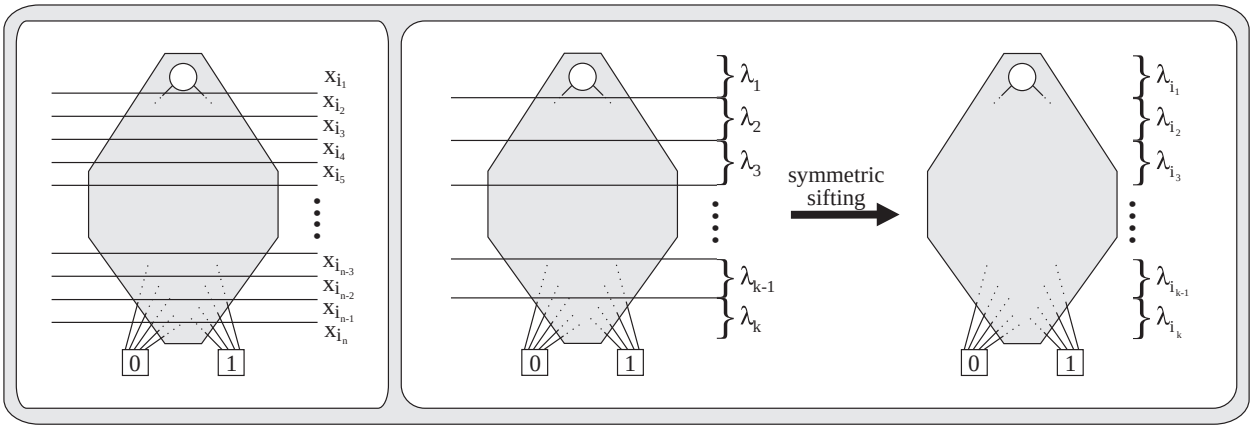


Fig. 8. On the left hand side the method presented by Chang is illustrated (cut lines between all levels). On the right hand side our method is illustrated.

The numbers for *sym_sift*, *sym_group* and *sym_cover* are given in Table VII in columns 4, 5 and 6, respectively. It can easily be seen that the reduction ratio decreases, when only a smaller number of don't cares is available, but with only 10% don't cares still more than 30% of the nodes can be saved on average.

VII. CONCLUSIONS

We presented methods for symmetry detection for completely specified functions represented by BDDs. The main idea of our symmetry detection algorithm is to use fast preprocessing algorithms to detect asymmetric variable pairs. These methods were applied to improve the quality of BDD reordering heuristics for the class of partially symmetric functions by using symmetry variable orders. The concept of symmetry variable orders was successfully extended to incompletely functions, where there are two means to minimize BDD sizes: the assignment of values to don't cares and the optimization of the variable order. Experimental results prove our approach to be very effective.

APPENDIX

A. Proof of Lemma 10

Proof:

1. "'$\Longleftarrow$"': Suppose f is not strongly symmetric in P. Then there must be $\lambda_i \in P$, such that f is not strongly symmetric in λ_i and there must be a pair of variables $(x_i, x_j) \in \lambda_i$, such that f is not strongly symmetric in (x_i, x_j). By definition there must be $e_1 = (\epsilon_1, \ldots, \epsilon_i, \ldots, \epsilon_j, \ldots, \epsilon_n)$ and $e_2 = (\epsilon_1, \ldots, \epsilon_j, \ldots, \epsilon_i, \ldots, \epsilon_n)$, such that $e_1 \in D$ and $e_2 \notin D$ or $e_1, e_2 \in D$ and $f(e_1) \neq f(e_2)$. But e_1 and e_2 belong to the same weight class C of P. Both cases lead to a contradiction: In the first case we have $\{dc, 1\}$ or $\{dc, 0\} \subseteq f(C)$, in the second case

way' as described above.

TABLE VII

EXPERIMENTAL RESULTS. THE TABLE SHOWS THE NUMBER OF NODES IN THE BDDS OF EACH FUNCTION WITH 10% DON'T CARES.

circuit	in	out	*sym_sift*	*sym_group*	*sym_cover*
5xp1	7	10	75	73	68
9symml	9	1	75	25	25
alu2	10	6	199	199	166
apex6	135	99	961	911	585
apex7	49	37	807	753	428
b9	41	21	203	195	141
c8	28	18	180	161	83
example2	85	66	547	540	464
mux	21	1	40	35	33
pcler8	27	17	83	83	81
rd73	7	3	65	35	31
rd84	8	4	126	42	42
sao2	10	4	106	106	79
x4	94	71	677	670	499
z4ml	7	4	50	30	17
total			4194	3858	2742

$\{0,1\} \subseteq f(C)$.

"$\Longrightarrow$": If f is strongly symmetric in P, then the following holds for all $\sigma \in \Sigma = \{\sigma_{i,l} : \exists \lambda_j \in P \text{with} x_i, x_l \in \lambda_j\}$[1]: $\forall e = (\epsilon_1, \ldots, \epsilon_n) \in \{0,1\}^n$ $f(e) = f(\sigma(e))$ (including the extended interpretation $f(e) = f(\sigma(e)) = dc$). Let e_1 and e_2 be members of a arbitrary weight class C of P. Then there is a sequence of permutations $\sigma_1 \ldots \sigma_l \in \Sigma$ with $e_2 = (\sigma_1 \circ \ldots \circ \sigma_l)(e_1)$. Thus $f(e_1) = f(e_2)$ holds, such that $f(C) = \{0\}$ or $f(C) = \{1\}$ or $f(C) = \{dc\}$.

2. "$\Longleftarrow$": Let $\forall 0 \leq w_i \leq |\lambda_i| (1 \leq i \leq k)$ $\{0,1\} \not\subseteq f(C^P_{w_1,\ldots,w_k})$.
We have to prove that there is a completely specified extension f' of f, which is symmetric in P.
Define f' as follows:
If $f(C) = \{\epsilon\}$ $(\epsilon \in \{0,1\})$ for a weight class C, then $f'(C) = f(C)$.
If $f(C) = \{dc\}$ for a weight class C, then $f'(C) = 0$.
If $f(C) = \{\epsilon, dc\}$ $(\epsilon \in \{0,1\})$ for a weight class C, then $f'(C) = \epsilon$.
Then f' is a completely specified function and because of part 1 of the theorem f' is strongly symmetric in P and thus symmetric in P according to the symmetry definition for completely specified functions.

"$\Longrightarrow$": Let f be (weakly) symmetric in P. Thus there is a completely specified extension f' of f, which is symmetric in P. If there would be a weight class C with $\{0,1\} \subseteq f(C)$, then $\{0,1\} \subseteq f'(C)$, since f' is an extension of f. Since f' is completely specified, we have for all weight classes C of P according to part 1 of the theorem: $f'(C) = \{0\}$ or $f'(C) = \{1\}$ which contradicts our assumption.

■

[1] $\sigma_{i,l} : \{0,1\}^n \to \{0,1\}^n$, $\sigma_{i,l}(\epsilon_1, \ldots, \epsilon_i, \ldots, \epsilon_l, \ldots, \epsilon_n) = (\epsilon_1, \ldots, \epsilon_l, \ldots, \epsilon_i, \ldots, \epsilon n)$ $\forall \epsilon_1, \ldots, \epsilon_n \in \{0,1\}^n$

B. Proof of Theorem 4

Proof: Let an instance of PC be given by a graph $G = (V, E)$ and a number $K \leq |V|$. We can determine in polynomial time BDDs f_{Gon} and f_{Gdc} of an incompletely specified function f_G with the property that there is a partition of G into K cliques iff there is a partition P of the variable set X of f, such that f is symmetric in P and $|P| = K$.

W.l.o.g. $V = \{x_1, \ldots, x_n\} = X$.
$f_G \in S(D)$ $(D \subseteq \{0,1\}^n)$ is defined by

$$f_G(\epsilon_1, \ldots, \epsilon_n) = \begin{cases} 1 \quad \text{if} & \epsilon_1 = \ldots = \epsilon_i = 1, \\ & \epsilon_{i+1} = \ldots = \epsilon_n = 0, \\ & 1 \leq i \leq n-1 \\ 0 \quad \text{if} & \epsilon_1 = \ldots = \epsilon_{i-1} = 1, \\ & \epsilon_i = \ldots = \epsilon_{j-1} = 0, \\ & \epsilon_j = 1, \\ & \epsilon_{j+1} = \ldots = \epsilon_n = 0, \\ & 1 \leq i \leq n-1, j > i \\ & \text{and } \{x_i, x_j\} \notin E \\ dc \quad \text{otherwise} \end{cases}$$

From the definition of f_G it is easy to see that if f_G is symmetric in a partition $P = \{\lambda_1, \ldots, \lambda_K\}$ of the input variables, then the variables (nodes) from λ_i form a clique in G.

Let $P = \{\lambda_1, \ldots, \lambda_K\}$ be a partition of the input variables, such that the nodes from λ_i form cliques in G. Suppose f is *not* symmetric in P.
According to lemma 10 there is a weight class $C^P_{w_1,\ldots,w_K}$ for P with $\{0,1\} \subseteq f_G(C^P_{w_1,\ldots,w_K})$. From the definition of f_G the only vertex of $C^P_{w_1,\ldots,w_K}$ with function value 1 is $\epsilon^{(1)} := (\underbrace{1, \ldots, 1}_{w \text{ times}}, 0, \ldots, 0)$ $(w = \sum_{i=1}^{k} w_i)$. There has to be $j > w$ with $\epsilon^{(0)} := (\underbrace{1, \ldots, 1}_{w-1 \text{ times}}, 0, 0, \ldots, 0, \underbrace{1}_{\epsilon^{(0)}_j}, 0, \ldots, 0)$ with $f(\epsilon^{(0)}) = 0$, i.e. $(x_w, x_j) \notin E$. Let $\lambda' \in P$ with $x_w \in \lambda'$. If $x_j \notin \lambda'$, then $w^1_{\lambda'}(\epsilon^{(0)}) = w^1_{\lambda'}(\epsilon^{(1)}) - 1$. This contradicts the fact that $\epsilon^{(0)}$ and $\epsilon^{(1)}$ are in the same weight class. If $x_j \in \lambda'$, then we obtain a contradiction to the fact that $(x_w, x_j) \notin E$.

Since $ON(f_G)$ and $OFF(f_G)$ are of polynomial size, the BDDs for f_{Gon} and f_{Gdc} can be computed in polynomial time. ■

C. Proof of Theorem 5

Proof: Let $P = \{\lambda_1, \lambda_2, \lambda_3, \ldots, \lambda_l\}$ and w.l.o.g. $\lambda_1 = \{x_i\}$ and $\lambda_2 = \{x_{j_1}, \ldots, x_{j_k}\}$.
f is strongly symmetric in P and we have to show that $f^{(k)}$ is strongly symmetric in $P' = \{\lambda_1 \cup \lambda_2, \lambda_3, \ldots, \lambda_l\}$.
Because of lemma 10 we have to show that for all weight classes $C^{P'}_{w_{1,2}, w_3, \ldots, w_l}$ of P' holds:

$$f^{(k)}(C^{P'}_{w_{1,2}, w_3, \ldots, w_{n+1-k}}) = \begin{cases} \{0\} & \text{or} \\ \{1\} & \text{or} \\ \{dc\} \end{cases}$$

Case 1: $w_{1,2} = 0$ or $w_{1,2} = k+1$
Then the following holds:

$$C^{P'}_{w_{1,2},w_3,\ldots,w_l} = C^{P}_{0,0,w_3,\ldots,w_l} \quad \text{or}$$

$$C^{P'}_{w_{1,2},w_3,\ldots,w_l} = C^{P}_{1,k,w_3,\ldots,w_l}$$

and thus $|f(C^{P'}_{w_{1,2},w_3,\ldots,w_l})| = 1$ because of the strong symmetry of f in P.
If $f(C^{P'}_{w_{1,2},w_3,\ldots,w_l}) = c$, $c \in \{0,1\}$,
then $f^{(p)}(C^{P'}_{w_{1,2},w_3,\ldots,w_{n+1-k}}) = \{c\}$ for all $1 \le p \le k$, since $f^{(p)}$ is an extension of f.
If $f(C^{P'}_{w_{1,2},w_3,\ldots,w_l}) = \{dc\}$,
then $f^{(p)}(C^{P'}_{w_{1,2},w_3,\ldots,w_l}) = \{dc\}$ for all $1 \le p \le k$, since *make_strongly_symm*$(f^{(p-1)}, x_i, x_{j_p})$ provides a *minimal* extension, which is strongly symmetric in (x_i, x_{j_p}) and $w^1_{\lambda_1}(\epsilon) = w^1_{\lambda_2}(\epsilon) = 0$ or $w^0_{\lambda_1}(\epsilon) = w^0_{\lambda_2}(\epsilon) = 0$ for all $\epsilon \in C^{P'}_{w_{1,2},w_3,\ldots,w_l}$.
Case 2: $1 \le w_{1,2} \le k$
In this case we have the following disjoint union $C^{P'}_{w_{1,2},w_3,\ldots,w_l} = C^{P}_{0,w_{1,2},w_3,\ldots,w_l} \cup C^{P}_{1,w_{1,2}-1,w_3,\ldots,w_l}$.
It follows from our precondition

$$f(C^{P}_{0,w_{1,2},w_3,\ldots,w_l}) = \begin{cases} \{0\} \text{ or} \\ \{1\} \text{ or} \\ \{dc\} \end{cases}$$

and

$$f(C^{Q}_{1,w_{1,2}-1,w_3,\ldots,w_l}) = \begin{cases} \{0\} \text{ or} \\ \{1\} \text{ or} \\ \{dc\} \end{cases}$$

Case 2.1: $f(C^{P}_{0,w_{1,2},w_3,\ldots,w_l}) = f(C^{P}_{1,w_{1,2}-1,w_3,\ldots,w_l})$
Since the calls of *make_strongly_symm*$(f^{(p-1)}, x_i, x_{j_p})$ give minimal extensions, which are strongly symmetric in (x_i, x_{j_p}), the assignment for $C^{P}_{0,w_{1,2},w_3,\ldots,w_l}$ and $C^{P}_{1,w_{1,2}-1,w_3,\ldots,w_l}$ is not changed. $f^{(p)}(C^{P}_{0,w_{1,2},w_3,\ldots,w_l}) = f^{(p)}(C^{P}_{1,w_{1,2}-1,w_3,\ldots,w_l})$ holds and thus

$$f^{(p)}(C^{P'}_{w_{1,2},w_3,\ldots,w_l}) = \begin{cases} \{0\} & \text{or} \\ \{1\} & \text{or} \\ \{dc\} \end{cases}$$

Case 2.2: $f(C^{P}_{0,w_{1,2},w_3,\ldots,w_l}) \neq f(C^{P}_{1,w_{1,2}-1,w_3,\ldots,w_l})$
Since f is symmetric in (x_i, x_{j_1}), there are $c \in \{0,1\}$ and $u \in \{0,1\}$, such that
$f(C^{P}_{u,w_{1,2}-u,w_3,\ldots,w_l}) = \{dc\}$ and
$f(C^{P}_{\overline{u},w_{1,2}-\overline{u},w_3,\ldots,w_l}) = \{c\}$.
In the following we assume $u = 0$ (case $u = 1$ is analogous).

From the definition of *make_strongly_symm* follows that for all $1 \le p \le k$ $f^{(p)}(\epsilon) \in \{c, dc\}$ $\forall \epsilon \in C^{P}_{0,w_{1,2},w_3,\ldots,w_l} \cup C^{P}_{1,w_{1,2}-1,w_3,\ldots,w_l}$.
A call of *make_strongly_symm*$(f^{(p)}, x_i, x_{j_{p+1}})$ assigns the function values to vectors $\epsilon \in C^{P}_{0,w_{1,2},w_3,\ldots,w_l}$ with $\epsilon_i = 0$ and $\epsilon_{j_{p+1}} = 1$, namely to the value $f^{(p)}(\sigma_{ij_{p+1}}(\epsilon)) = c$ $(\sigma_{ij_{p+1}}(\epsilon) \in C^{P}_{1,w_{1,2}-1,w_3,\ldots,w_l})$.[2]
It remains to be shown that $f^{(k)}(\epsilon) = c$ $\forall \epsilon \in C^{P}_{0,w_{1,2},w_3,\ldots,w_l}$, i.e. that the sequence of k calls is enough to assign function value c to *all* elements of $C^{P}_{0,w_{1,2},w_3,\ldots,w_l}$.
The following statement is proven by induction:
$f^{(p)}(\epsilon) = c$ $\forall \epsilon \in C^{P}_{0,w_{1,2},w_3,\ldots,w_l}$ with $\epsilon_{j_1} = 1$ or $\epsilon_{j_2} = 1$ or $\ldots$ or $\epsilon_{j_p} = 1$.
$p = 0$: Trivial.
$p \to p+1$:
Because of the inductive assumption and since $f^{(p+1)}$ is an extension of $f^{(p)}$, we have:
$f^{(p+1)}(\epsilon) = c$ $\forall \epsilon \in C^{P}_{0,w_{1,2},w_3,\ldots,w_l}$ with $\epsilon_{j_1} = 1$ or $\epsilon_{j_2} = 1$ or $\ldots$ or $\epsilon_{j_p} = 1$.
We have to show that $f^{(p+1)}(\epsilon) = c$ $\forall \epsilon \in C^{P}_{0,w_{1,2},w_3,\ldots,w_l}$ with $\epsilon_{j_{p+1}} = 1$.
Let $\delta \in C^{P}_{1,w_{1,2}-1,w_3,\ldots,w_l}$ with

$$\delta_i = \overline{\epsilon_i} = 1^{\ddagger}, \delta_{j_{p+1}} = \overline{\epsilon_{j_{p+1}}} = 0$$

$$\text{and } \delta_l = \epsilon_l \text{ for } l \neq i, j_{p+1},$$

thus $\delta = \sigma_{i,j_{p+1}}(\epsilon)$. (There is such a $\delta \in C^{P}_{1,w_{1,2}-1,w_3,\ldots,w_l}$ because of $1 \le w_{1,2}$).
We have

$$f^{(p)}(\delta) = f(\delta) = c$$

and thus

$$f^{(p+1)}(\epsilon) = f^{(p)}(\sigma_{i,j_{p+1}}(\epsilon)) = f^{(p)}(\delta) = c.$$

It follows from the statement shown by induction:

$$f^{(k)}(\epsilon) = c \, \forall \epsilon \in C^{P}_{0,w_{1,2},w_3,\ldots,w_l}$$

$$\text{with } \epsilon_{j_1} = 1 \text{ or } \ldots \text{ or } \epsilon_{j_k} = 1$$

or

$$f^{(k)}(\epsilon) = c \, \forall \epsilon \in C^{P}_{0,w_{1,2},w_3,\ldots,w_l} \text{ with } w^1_{\lambda_2}(\epsilon) \ge 1.$$

But $w^1_{\lambda_2}(\epsilon) = w_{1,2} \ge 1$ holds for *all* $\epsilon \in C^{P}_{0,w_{1,2},w_3,\ldots,w_l}$ (assumption in Case 2).

■

D. Proof of Lemma 13 (sketch)

Proof: W.l.o.g. let $\cup_{j=1}^{i} \lambda_j = \{x_1, \ldots, x_p\}$ and $\cup_{j=i+1}^{n} \lambda_j = \{x_{p+1}, \ldots, x_n\}$.
Suppose we apply the *remove_z* operation defined in [8] to a cut line between two symmetric groups λ_i and λ_{i+1}. The *remove_z* operation works as follows:

- The BDD nodes below the cut line correspond to all different cofactors of f with respect to the first p variables. Let the set of these cofactors be $COF = \{cof_1, \ldots, cof_l\}$. Note that these cofactors are incompletely specified functions.

[2] $\sigma_{ij} : \{0,1\}^n \to \{0,1\}^n, \sigma_{ij}(\epsilon_1, \ldots, \epsilon_i, \ldots, \epsilon_j, \ldots, \epsilon_n) = (\epsilon_1, \ldots, \epsilon_j, \ldots, \epsilon_i, \ldots, \epsilon_n)$

[‡] For all elements ϵ of the weight class $C^{P}_{0,w_{1,2},w_3,\ldots,w_l}$ is $\epsilon_i = 0$.

- Two cofactors cof_i and cof_j are compatible iff there is no $(\epsilon_{p+1}, \ldots, \epsilon_n) \in \{0,1\}^{n-p}$ such that

$$cof_i(\epsilon_{p+1}, \ldots, \epsilon_n) = c, cof_j(\epsilon_{p+1}, \ldots, \epsilon_n) = \overline{c}$$

for $c \in \{0,1\}$.
A partition $P_{COF} = \{COF_1, \ldots, COF_m\}$ of COF is computed such that all pairs cof_i, $cof_j \in COF_q$ $(1 \leq q \leq m)$ are compatible.
- For a set COF_q of compatible cofactors an extension $extension_q$ is computed as follows:
$extension_q(\epsilon_{p+1}, \ldots, \epsilon_n) = c$ $(c \in \{0,1\})$ iff $\exists cof_j \in COF_q$ with $cof_j(\epsilon_{p+1}, \ldots, \epsilon_n) = c$ and $extension_q(\epsilon_{p+1}, \ldots, \epsilon_n) = dc$ iff $\forall cof_j \in COF_q$ $cof_j(\epsilon_{p+1}, \ldots, \epsilon_n) = dc$.
- The cofactors $cof_j \in COF_q$ are all replaced by their (common) extension $extension_q$. This leads to an extension f' of f, the result of the *remove_z* operation. The number of BDD nodes of the representation for f' which are located immediately below the cut line between λ_i and λ_{i+1} equals the size m of the partition P_{COF}.

We have to prove that f' is strongly symmetric in all sets $\lambda_j \in P = \{\lambda_1, \ldots, \lambda_k\}$.

Case 1: $j \leq i$

Let x_{j_1} and $x_{j_2} \in \lambda_j$.
Choose $\epsilon^{(1)}, \epsilon^{(2)} \in \{0,1\}^n$ arbitrarily with

$$\epsilon^{(1)} = (\epsilon_1, \ldots, \epsilon_{j_1}, \ldots, \epsilon_{j_2}, \ldots, \epsilon_n)$$

$$\text{and } \epsilon^{(2)} = (\epsilon_1, \ldots, \epsilon_{j_2}, \ldots, \epsilon_{j_1}, \ldots, \epsilon_n).$$

Since f is strongly symmetric in x_{j_1}, x_{j_2} the cofactors $f|_{x_1=\epsilon_1,\ldots,x_{j_1}=\epsilon_{j_1},\ldots,x_{j_2}=\epsilon_{j_2},\ldots,x_p=\epsilon_p}$
and $f|_{x_1=\epsilon_1,\ldots,x_{j_1}=\epsilon_{j_2},\ldots,x_{j_2}=\epsilon_{j_1},\ldots,x_p=\epsilon_p}$ are equal. By *remove_z* this cofactor is replaced by some extension $extension_q$ and of course the corresponding cofactors $f'|_{x_1=\epsilon_1,\ldots,x_{j_1}=\epsilon_{j_1},\ldots,x_{j_2}=\epsilon_{j_2},\ldots,x_p=\epsilon_p}$ and $f'|_{x_1=\epsilon_1,\ldots,x_{j_1}=\epsilon_{j_2},\ldots,x_{j_2}=\epsilon_{j_1},\ldots,x_p=\epsilon_p}$ of the result f' of this replacement are still equal. Thus, $f'(\epsilon^{(1)}) = f'(\epsilon^{(2)})$ and f' is strongly symmetric in x_{j_1} and x_{j_2}.

Case 2: $j \geq i+1$

Let x_{j_1} and $x_{j_2} \in \lambda_j$.
Choose $\epsilon^{(1)}, \epsilon^{(2)} \in \{0,1\}^n$ arbitrarily with

$$\epsilon^{(1)} = (\epsilon_1, \ldots, \epsilon_p, \ldots, \epsilon_{j_1}, \ldots, \epsilon_{j_2} \ldots, \epsilon_n)$$

$$\text{and } \epsilon^{(2)} = (\epsilon_1, \ldots, \epsilon_p, \ldots, \epsilon_{j_2}, \ldots, \epsilon_{j_1} \ldots, \epsilon_n).$$

Suppose the cofactor $f|_{x_1=\epsilon_1,\ldots,x_p=\epsilon_p}$ is in the set COF_q. *All cofactors* $\in COF_q$ are strongly symmetric in x_{j_1} and x_{j_2}.
If for all cofactors

$$cof_j \in COF_q \quad cof_j(\epsilon_{p+1}, \ldots, \epsilon_{j_1}, \ldots, \epsilon_{j_2}, \ldots, \epsilon_n) = dc$$

then also for all cofactors

$$cof_j \in COF_q \quad cof_j(\epsilon_{p+1}, \ldots, \epsilon_{j_2}, \ldots, \epsilon_{j_1}, \ldots, \epsilon_n) = dc$$

because of strong symmetry and according to the definition of $extension_q$ given above

$$extension_q(\epsilon_{p+1}, \ldots, \epsilon_{j_1}, \ldots, \epsilon_{j_2}, \ldots, \epsilon_n) =$$

$$extension_q(\epsilon_{p+1}, \ldots, \epsilon_{j_2}, \ldots, \epsilon_{j_1}, \ldots, \epsilon_n) = dc.$$

If $cof_j \in COF_q$ exists with

$$cof_j(\epsilon_{p+1}, \ldots, \epsilon_{j_1}, \ldots, \epsilon_{j_2}, \ldots, \epsilon_n) = c (c \in \{0,1\})$$

then

$$cof_j(\epsilon_{p+1}, \ldots, \epsilon_{j_2}, \ldots, \epsilon_{j_1}, \ldots, \epsilon_n) = c$$

because of strong symmetry and according to the definition of $extension_q$

$$extension_q(\epsilon_{p+1}, \ldots, \epsilon_{j_1}, \ldots, \epsilon_{j_2}, \ldots, \epsilon_n) =$$

$$extension_q(\epsilon_{p+1}, \ldots, \epsilon_{j_2}, \ldots, \epsilon_{j_1}, \ldots, \epsilon_n) = c.$$

$f|_{x_1=\epsilon_1,\ldots,x_p=\epsilon_p}$ is replaced by $extension_q$ and for the result f' we have

$$f'(\epsilon_1, \ldots, \epsilon_p, \ldots, \epsilon_{j_1}, \ldots, \epsilon_{j_2}, \ldots \epsilon_n) =$$

$$f'(\epsilon_1, \ldots, \epsilon_p, \ldots, \epsilon_{j_2}, \ldots, \epsilon_{j_1}, \ldots \epsilon_n).$$

■

References

[1] S.B. Akers, "Binary decision diagrams," *IEEE Trans. on Comp.*, vol. 27, pp. 509–516, 1978.

[2] B. Bollig, M. Löbbing, and I. Wegener, "Simulated annealing to improve variable orderings for OBDDs," In *Int'l Workshop on Logic Synth.*, pp. 5b:5.1–5.10, May 1995.

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

[4] D. Brélaz, "New methods to color vertices of a graph," *Comm. of the ACM*, vol. 22, pp. 251–256, 1979.

[5] R.E. Bryant, "Graph - based algorithms for Boolean function manipulation," *IEEE Trans. on Comp.*, vol. 35, no. 8, pp. 677–691, Aug. 1986.

[6] R.E. Bryant, "Symbolic Boolean manipulation with ordered binary decision diagrams," *ACM, Comp. Surveys*, vol. 24, pp. 293–318, 1992.

[7] P. Buch, A. Narayan, A.R. Newton, and A.L. Sangiovanni-Vincentelli, "Logic Synthesis for Large Pass Transistor Circuits," In *Proc. Int'l Conf. on CAD*, pp. 663–670, Nov. 1997.

[8] S. Chang, D. Cheng, and M. Marek-Sadowska, "Minimizing ROBDD size of incompletely specified multiple output functions," In *Proc. European Design & Test Conf.*, pp. 620–624, Mar. 1994.

[9] D.I. Cheng and M. Marek-Sadowska, "Verifying equivalence of functions with unknown input correspondence," In *Proc. European Conf. on Design Automation*, pp. 81–85, Feb. 1993.

[10] O. Coudert, C. Berthet, and J.C. Madre, "Verification of sequential machines based on symbolic execution," In *Proc. Automatic Verification Methods for Finite State Systems, LNCS 407*, pp. 365–373, 1989.

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

[12] D.L. Dietmeyer and P.R. Schneider, "Identification of symmetry, redundancy and equivalence of Boolean functions," *IEEE Trans. on Electronic Comp.*, vol. 16, pp. 804–817, 1967.

[13] R. Drechsler and B. Becker, "Sympathy: Fast exact minimization of fixed polarity Reed-Muller expressions for symmetric functions," In *Proc. European Design & Test Conf.*, pp. 91–97, Mar. 1995.

[14] R. Drechsler, B. Becker, and N. Göckel, "A genetic algorithm for variable ordering of OBDDs," In *Int'l Workshop on Logic Synth.*, pp. 5c:5.55–5.64, May 1995.

[15] R. Drechsler and N. Göckel, "Minimization of BDDs by evolutionary algorithms," In *Int'l Workshop on Logic Synth.*, May 1997.

[16] C.R. Edwards and S.L. Hurst, "A digital synthesis procedure under function symmetries and mapping methods," *IEEE Trans. on Comp.*, vol. 27, pp. 985–997, 1978.

[17] E. Felt, G York, R. Brayton, and A. Sangiovanni-Vincentelli, "Dynamic variable reordering for BDD minimization," In *Proc. European Design Automation Conf.*, pp. 130–135, Sept. 1993.

[18] F. Ferrandi, A. Macii, E. Macii, M. Poncino, R. Scarsi, and F. Somenzi, "Symbolic Algorithms for Layout-Oriented Synthesis of Pass Transistor Logic Circuits," In *Proc. Int'l Conf. on CAD*, Nov. 1998.

[19] H. Fujii, G. Ootomo, and C. Hori, "Interleaving based variable ordering methods for ordered binary decision diagrams," In *Proc. Int'l Conf. on CAD*, pp. 38–41, Nov. 1993.

[20] M. Fujita, H. Fujisawa, and N. Kawato, "Evaluation and improvements of Boolean comparison method based on binary decision diagrams," In *Proc. Int'l Conf. on CAD*, pp. 2–5, Nov. 1988.

[21] M. Fujita, Y. Matsunaga, and T. Kakuda, "On variable ordering of binary decision diagrams for the application of multi-level synthesis," In *Proc. European Conf. on Design Automation*, pp. 50–54, Feb. 1991.

[22] M.R. Garey and D.S. Johnson, *Computers and Intractability - A Guide to NP-Completeness.* Freemann, San Francisco, 1979.

[23] J. Gergov and C. Meinel, "Analysis and manipulation of Boolean functions in terms of decision graphs," In *WG'92, LNCS*, pp. 310–320, 1992.

[24] M. Heap, "On the exact ordered binary decision diagram size of totally symmetric functions," *Jour. of Electronic Testing: Theory and Applications*, vol. 4, pp. 191–195, 1993.

[25] S.L. Hurst, "Detection of symmetries in combinatorial functions by spectral means," *IEE Electronic Circuits and Systems*, vol. 5, pp. 173–180, 1977.

[26] N. Ishiura, H. Sawada, and S. Yajima, "Minimization of binary decision diagrams based on exchange of variables," In *Proc. Int'l Conf. on CAD*, pp. 472–475, Nov. 1991.

[27] B.-G. Kim and D.L. Dietmeyer, "Multilevel logic synthesis of symmetric switching functions," *IEEE Trans. on CAD*, vol. 10, no. 4, 1991.

[28] Y.-T. Lai, M. Pedram, and S.B.K. Vrudhula, "EVBDD-based algorithms for integer linear programming, spectral transformation, and function decomposition," *IEEE Trans. on CAD*, vol. 13, no. 8, pp. 959–975, 1994.

[29] Y.-T. Lai, S. Sastry, and M. Pedram, "Boolean matching using binary decision diagrams with applications to logic synthesis and verification," In *Proc. Int'l Conf. on CAD*, pp. 452–458, Nov. 1992.

[30] C.Y. Lee, "Representation of switching circuits by binary decision diagrams," *Bell System Technical Jour.*, vol. 38, pp. 985–999, 1959.

[31] L. Litan, P. Molitor, and D. Möller, "Least upper bounds on the sizes of symmetric variable order based OBDDs," In *Proc. Great Lakes Symp. VLSI*, pp. 126–129, 1996.

[32] F. Mailhot and G. De Micheli, "Technology mapping using Boolean matching and don't care sets," In *Proc. European Conf. on Design Automation*, pp. 212–216, Feb. 1990.

[33] S. Malik, A.R. Wang, R.K. Brayton, and A.L. Sangiovanni-Vincentelli, "Logic verification using binary decision diagrams in a logic synthesis environment," In *Proc. Int'l Conf. on CAD*, pp. 6–9, Nov. 1988.

[34] J. Mohnke and S. Malik, "Permutation and phase independent Boolean comparison," In *Proc. European Conf. on Design Automation*, pp. 86–92, Feb. 1993.

[35] J. Mohnke, P. Molitor, and S. Malik, "Limits of using signatures for permutation independent Boolean comparison," In *Proc. ASP Design Automation Conf.*, pp. 459–464, Aug. 1995.

[36] D. Möller, J. Mohnke, and M. Weber, "Detection of symmetry of Boolean functions represented as ROBDDs," In *Proc. Int'l Conf. on CAD*, pp. 680–684, Nov. 1993.

[37] D. Möller, P. Molitor, and R. Drechsler, "Symmetry based variable ordering for ROBDDs," In *Proc. IFIP Workshop on Logic and Architecture Synthesis*, pp. 47–53, Dec. 1994.

[38] B.M.E. Moret, "Decision trees and diagrams," In *Computing Surveys*, vol. 14, pp. 593–623, 1982.

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

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

[41] S. Panda, F. Somenzi, and B.F. Plessier, "Symmetry detection and dynamic variable ordering of decision diagrams," In *Proc. Int'l Conf. on CAD*, pp. 628–631, Nov. 1994.

[42] I. Pomeranz and S.M. Reddy, "On determining symmetries in inputs of logic circuits," In *VLSI Design Conf.*, pp. 255–260, Jan. 1994.

[43] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," In *Proc. Int'l Conf. on CAD*, pp. 42–47, Nov. 1993.

[44] C. Scholl, S. Melchior, G. Hotz, and P. Molitor, "Minimizing ROBDD sizes of incompletely specified functions by exploiting strong symmetries," In *Proc. European Design & Test Conf.*, pp. 229–234, Mar. 1997.

[45] C. Scholl and P. Molitor, "Communication based FPGA synthesis for multi-output Boolean functions," In *Proc. ASP Design Automation Conf.*, pp. 279–287, Aug. 1995.

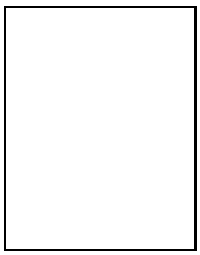
[46] E. Sentovich, K. Singh, L. Lavagno, Ch. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A system for sequential circuit synthesis," Technical report, University of Berkeley, 1992.

[47] T.R. Shiple, R. Hojati, A.L. Sangiovanni-Vincentelli, and R.K. Brayton, "Heuristic minimization of BDDs using don't cares," In *Proc. Design Automation Conf.*, pp. 225–231, June 1994.

[48] D. Sieling, "Variable orderings and the size of OBDDs for partially symmetric Boolean functions," In *SASIMI*, pp. 189–196, Nov. 1996.

[49] I. Wegener, "Optimal decision trees and one-time-only branching programs for symmetric Boolean functions," *Information and Control*, vol. 62, pp. 129–143, 1984.

[50] B. Wurth, K. Eckl, and K. Antreich. "Functional multiple-output decomposition: Theory and implicit algorithm," In *Proc. Design Automation Conf.*, pp. 54–59, June 1995.



Christoph Scholl studied computer science and electrical engineering at University of Saarland, Germany, from 1988 to 1993. He received the Dipl.–Inform. and Dr.–Ing. degrees in 1993, and 1997, respectively, from University of Saarland.

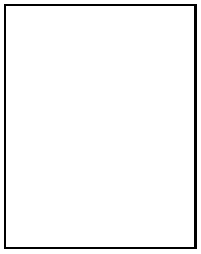
In 1993 he was with the Sonderforschungsbereich "VLSI Design Methods and Parallelism", and from 1993 to 1996 with the Graduiertenkolleg "Efficiency and Complexity of Algorithms and Computers" at the University of Saarland. Since 1996 he is working at the Institute of Computer Science of Albert-Ludwigs-University, Freiburg im Breisgau, Germany.

His research interests include logic synthesis, verification and test of VLSI circuits.

Dirk Möller received the diploma degree in computer science from the Humboldt-University Berlin, Germany in 1992.

Between 1992 to 1994, he was with the Sonderforschungsbereich "VLSI Design Methods and Parallelism" at the Humboldt-University. From 1995 to 1996 he was with the Computer Science Department at Martin-Luther University Halle-Wittenberg. In 1996 he joined Dresearch Digital Media Systems GmbH, Germany.

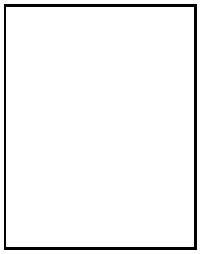
His research interests are logic and physical synthesis.



Paul Molitor studied computer science at University of Saarland, Germany, from 1978 to 1986. He received the Dipl.-Inform., the Dr.rer.nat., and the Dr.habil. degrees in 1982, 1986, and 1992, respectively.

Between 1982 to 1992, he was with the research group on VLSI design methods and parallelism at the University of Saarland. After being visiting professor at the University of Halle (1992/93) and the University of Freiburg i.Br. (1993), he was an associate professor in the Computer Science Department of the Humboldt-University Berlin in 1993/94. Since 1994, he is a full professor at the Martin-Luther University Halle-Wittenberg, Germany. He is author and coauthor of two books and about 35 international papers in the field of physical design, logic synthesis, verification of digital circuits, and genetic algorithms.

Dr. Molitor is a member of IEEE, the ACM, and the GI.



Rolf Drechsler received his diploma and Ph.D. degree in computer science from the J.W. Goethe-University in Frankfurt am Main, Germany, in 1992 and 1995, respectively.

He is currently working at the Institute of Computer Science at the Albert-Ludwigs-University of Freiburg im Breisgau, Germany. He is the Symposium's Chair of the IEEE International Symposium on Multiple-Valued Logic 1999 in Freiburg.

He recently published two books with Kluwer Academic Publishers, one on BDD techniques co-authored by Bernd Becker and one on using evolutionary algorithms for VLSI CAD.

His research interests include verification, logic synthesis, and evolutionary algorithms.