Generalized Symmetries in Boolean Functions*

Victor N. Kravets  Karem A. Sakallah
Department of Electrical Engineering and Computer Science
University of Michigan, Ann Arbor, MI 48109
{vkravets, karem}@eecs.umich.edu

Abstract
In this paper we take a fresh look at the notion of symmetries in Boolean functions. Our studies are motivated by the fact that the classical characterization of symmetries based on invariance under variable swaps is a special case of a more general invariance based on unrestricted variable permutations. We propose a generalization of classical symmetry that allows for the simultaneous swap of ordered and unordered groups of variables, and show that it captures more of a function’s invariant permutations without undue computational requirements. We apply the new symmetry definition to analyze a large set of benchmark circuits and provide extensive data showing the existence of substantial symmetries in those circuits. Specific case studies of several of these benchmarks reveal additional insights about their functional structure and how it might be related to their circuit structure.

I. Introduction

Symmetries usually refer to permutations of an object’s parameters that leave it unchanged. They provide insights into the structure of the object that can be used to facilitate computations on it. They can also serve as a guide for preserving that structure when the object is transformed in some way. The object we study in this work is an n-variable Boolean function and the symmetries we explore are variable permutations, with possible complementation, that leave the function unchanged (see Fig. 1). The context for this work is logic synthesis which we view as a process that transforms an initial representation of the function (e.g. as a list of cubes or a BDD [2]) into a final implementation as a multi-level network of primitive cells from a given technology library. We contend, based on ample empirical evidence, that when guided by knowledge of functional symmetries, such a process yields more “natural” implementations of the function [11]. In this paper, though, we focus exclusively on the study of functional symmetries with only occasional reference to their utility in logic synthesis.

The study of symmetries in Boolean functions dates back to Shannon [19] who recognized that symmetric functions have particularly efficient switch network implementations. Since then several attempts were made to devise synthesis procedures for symmetric functions [4, 10]. These efforts, however, failed to yield practical synthesis tools and are generally viewed as inapplicable to the types or sizes of functions typically encountered in today’s design automation environments. In recent years, the increasing use of BDDs for the manipulation of Boolean functions sparked renewed interest in the study of function symmetries. In [9], for example, the authors showed that the size of a BDD can be reduced by using a variable order that places symmetric variables contiguously. This observation led to the development of sifting procedures for dynamic BDD variable ordering based on function symmetries [15, 18]. Symmetries were also utilized to improve the efficiency of functional equivalence checking for functions with unknown input correspondence [3, 13], and in the context of Boolean matching [8, 12, 20].

Much of the existing literature on symmetry is based on function invariance under swaps of variable pairs in the function’s support; we’ll refer to this type of symmetry as classical symmetry to distinguish it from the more general symmetry we study in this paper. For completely-specified functions, classical symmetry can be represented as a partition on the set of variables: variables that belong to a given block of that partition are equivalent, i.e. symmetric, whereas variables that belong to different blocks are non-equivalent, i.e. non-symmetric. The blocks of such a partition are referred to as the function’s symmetry groups [13], and variables within a symmetry group are equivalent in the sense that they can be permuted arbitrarily without changing the value of the function. The advent of BDDs led to the development of efficient symbolic methods for the identification of a function’s symmetry groups. The computational core in such algorithms is the check that determines the equivalence of a pair of variables; larger symmetry groups are then built incrementally using transitivity. Many of the recently-proposed techniques for symmetry identification achieve their efficiency through careful analysis of the structure of the BDD that represents the function [14, 15, 18]. In [21], the authors approach this problem by using the generalized Reed-Muller transform to speed-up computation of symmetries. A notable exception to the commonly-used definition of symmetry was proposed in [13, 17]. Rather than invariance under swaps of variables, symmetry is extended to swaps between symmetry groups. In [16] the authors also describe symmetry under swaps of ordered groups of variables.

In this work we focus on the efficient computation and representation of symmetry defined as an invariance under arbitrary variable permutation rather than invariance under swaps of variable pairs. Under this broader definition, partitions on the set of variables fail to capture all the invariant input permutations. We explore the relation between symmetry groups and variable permutations in Section II and highlight the inherent limitation of variable partitions as a means of representing arbitrary variable permutations. As an alternative to the explicit, and computationally infeasible, listing of all invariant permutations, we propose an efficient approach...
hierarchical extension to the notion of symmetry groups—a hierarchical partition—that allows us to represent a larger (but not necessarily the complete) set of invariant variable permutations. These hierarchical partitions represent higher-order symmetries that arise from simultaneously swapping groups of, rather than single, variables. They are more general than the partitions obtained from group swaps, proposed in [13], since they allow swaps between ordered as well as unordered groups. In Section III we formally state the conditions under which the classical first-order symmetries exist and provide computational procedures for the construction of the corresponding flat partition. In Section IV we generalize these conditions to define the higher-order symmetry and show how the corresponding hierarchical partition can be computed efficiently. In Section V we expand the notion of invariance to include the assignment of inversion phases to the function inputs. In Section VI we report on the results of applying hierarchical partitioning to a large set of benchmarks; we also provide detailed analyses of a few benchmarks to show that additional symmetries, missed by hierarchical partitioning or hidden through netlist flattening, can still be found. We conclude in Section VII by recapping the main contributions and suggesting several possible extensions.

II. Motivation

Consider the six-variable function \( f = abxy + cdxy \). It is relatively straightforward to show that its classical symmetry groups are \( \{a, b\}, \{c, d\} \), and \( \{x, y\} \) (the exact procedure for computing these groups is described in Section III). To appreciate the need for a broader notion of symmetry it is useful to view these symmetry groups as an implicit representation of the variable permutations that leave the function unchanged. Specifically, a group \( G_j \) consisting of \( n_j \) variables corresponds to \( n_j! \) permutations; the total number of permutations represented by all groups is the product of the number of permutations for each of the individual groups. Thus, the three-group partition on the variables of this function corresponds to the eight \((2! \times 2! \times 2! ) \) permutations:

\[
\{ \langle abcdxy \rangle, \langle abcdyx \rangle, \langle abdcxy \rangle, \langle abdcdxy \rangle, \langle badcxy \rangle, \langle badcdxy \rangle, \langle badcdyx \rangle, \langle badcxy \rangle \} \tag{1}
\]

Direct substitution of each of these permutations in the expression for the function confirms that they do indeed leave it unchanged. We will refer to such permutations as the function’s invariant permutations. In addition, we will encode the “flat” partition that induced them by an ordered list of unordered groups:

\[
\{ \{a, b\}, \{c, d\}, \{x, y\} \} \tag{2}
\]

This encoding emphasizes the fact that the permutations in (1) are generated from variable swaps that are strictly within, and not across, groups.

Further examination of this function, however, reveals that it remains invariant under the following additional set of eight permutations

\[
\{ \langle cdabxy \rangle, \langle cdabyx \rangle, \langle dcabxy \rangle, \langle dcabyx \rangle, \langle dcbaxy \rangle, \langle dcabxy \rangle, \langle dcbaxy \rangle, \langle dcbaxy \rangle \} \tag{3}
\]

which are not captured by the flat partition in (2). Note that each of these permutations can be derived from a corresponding permutation in (1) by swapping the groups \( \{a, b\} \) and \( \{c, d\} \). Thus, a suitable encoding that acts as an implicit representation for all sixteen permutations in (1) and (3) is the following hierarchical partition on the set of variables:

\[
\{ \{ \{a, b\}, \{c, d\} \}, \{x, y\} \} \tag{4}
\]

In both (2) and (4) we use angle brackets to indicate a fixed order (single permutation) and curly brackets to indicate all possible orders of the enclosed elements. A pictorial representation of the

![Diagram](https://via.placeholder.com/150)

Fig. 2. Symmetry-induced hierarchical partition of variables for function \( f(a, b, c, d, x, y) = abxy + cdxy \).

This small example serves to illustrate several important points that motivate our desire for a fresh exploration of functional symmetries:

- Function invariance under unrestricted variable permutations expands the classical notion of symmetry by identifying more structure in functions than can be inferred from simple variable swaps.
- Functional structure may be specified by an explicit listing of all invariant permutations. However, such a listing may be infeasible due to the exponentially large number of such permutations. Compact implicit representations of this structure include flat and hierarchical partitions on the variable set that act as “stylized permutation generators.” The quality of an implicit representation of invariant permutations can be measured in terms of the number of permutations it generates; representation \( R_i \) is deemed superior to representation \( R_j \) if it corresponds to a larger number of invariant permutations; in some sense, \( R_i \) identifies more of a function’s structure than \( R_j \). An ideal representation would identify the complete set of invariant permutations.
- In addition to compactness, an implicit representation should be efficiently computable. Compact representations whose construction procedures require exponential run times are as infeasible as explicit listings of invariant permutations.

Fig. 3 depicts a Venn diagram that establishes the relation between the classical and new definitions of function symmetry. The universe is taken to be the entire set of variable permutations and \( e \) is used to denote the identity permutation, i.e. the normal variable order. The permutations induced by flat and hierarchical partitions of the variables are thus seen to be subsets of all invariant permutations. In addition, the permutations induced by hierarchical partitioning are clearly seen as a superset of those induced by flat partitioning. The shaded subset is meant to represent an alternative

![Diagram](https://via.placeholder.com/150)

Fig. 3. Limitation of symmetry groups to represent all invariant permutations of a function.)
implicit representation of invariant permutations that is distinct from those based on partitions on the variable set.

In the remainder of this paper we develop the concept of hierarchical partitions on the set of variables as a means of characterizing functional structure which extends the classical notion of symmetry. It is important, however, to keep in mind that, while provably superior to flat partitioning, hierarchical partitioning may still be too limited in its ability to capture a sizeable subset of invariant permutations. It does, however, serve as a catalyst for exploring other implicit representations of a function’s invariant permutations, a point that we will allude to later when we analyze some of the benchmark circuits.

III. Classical First-Order Symmetries

First-order symmetries correspond to function invariance under swaps of variable pairs. Specifically, if variables \( a \) and \( b \) in the support of function \( f \) satisfy the condition:

\[
f(\ldots, a, \ldots, b, \ldots) = f(\ldots, b, \ldots, a, \ldots)
\]

(5)

then we say that \( f \) has a first-order symmetry between variables \( a \) and \( b \). These two variables are also said to form a symmetry group \( \langle a, b \rangle \). It is well known, as can be readily shown using Boole’s expansion theorem [6], that condition (5) is equivalent to the following equality constraint on the function’s cofactors [1, 4]:

\[
f_{a'b'} = f_{ab'}
\]

(6)

Equation (6) serves as the computational check for first-order symmetry between variables \( a \) and \( b \) in function \( f \). It also defines an equivalence relation on the set of variables that can be used to partition the set into its equivalence classes, i.e. symmetry groups, in quadratic time.

The extension of these classical results to higher-order symmetries is facilitated by adopting a matrix formulation of the symmetry check in (6). Let \( m_i(x_1 x_2 \ldots x_I) \) denote the \( i \)th minterm function on the specified ordered set of variables; for example, \( m_2(ab) = ab' \) and \( m_9(cbd\bar{a}) = cb'd'a = ab'cd' \). Cofactors of a function \( f \) with respect to variables \( a \) and \( b \) can now be expressed as the \( 2 \times 2 \) matrix:

\[
F_{(a), (b)} = \begin{bmatrix} f_{m_0(a), m_0(b)} & f_{m_0(a), m_1(b)} \\ f_{m_1(a), m_0(b)} & f_{m_1(a), m_1(b)} \end{bmatrix} = \begin{bmatrix} f_{a'b'} & f_{ab'} \\ f_{ab'} & f_{a'b} \end{bmatrix}
\]

(7)

When clear from context, we will also adopt the following shorthand notation for this matrix:

\[
F_{(a), (b)} = \begin{bmatrix} f_{0,0} & f_{0,1} \\ f_{1,0} & f_{1,1} \end{bmatrix}
\]

(8)

where \( f_{i,j} \) is implicitly understood to stand for \( f_{m_i(a), m_j(b)} \).

Comparison of (7) or (8) with (6) immediately suggests that (6) is equivalent to requiring the cofactor matrix \( F_{(a), (b)} \) to be symmetric, i.e., that:

\[
F_{(a), (b)}^T = F_{(a), (b)}
\]

(9)

where the \( T \) superscript denotes matrix transpose. This result should not be too surprising since condition (6) expresses function invariance under the swap of two variables, which in the matrix formulation corresponds to interchanging rows and columns.

IV. Higher-Order Symmetries

Swaps of variable pairs can be extended in a straightforward man-

<table>
<thead>
<tr>
<th>Function</th>
<th>( f = x(ab(c + d) + cd(a + b)) + x'(a' + b')(c' + d') + y(abc'd' + a'b'cd) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hierarchical Partition</td>
<td>Graphical</td>
</tr>
<tr>
<td>Invariant Permutations</td>
<td>( \langle {a, b}, {c, d} \rangle, {x}, {y} )</td>
</tr>
</tbody>
</table>

Fig. 4. Symmetry structures for two example functions.
itself and forms the symmetry structure \( S_i = \{ x_i \} \).

2. **(Recursion)**
   a. If \( S_1, \ldots, S_m \) are \( m \geq 2 \) symmetry structures that are pairwise symmetric, then \( S = \{ S_1, \ldots, S_m \} \) is a symmetry structure.
   b. If \( \{ S_1, \ldots, S_k \}, \ldots, \{ S_{m-1}, \ldots, S_m \} \) are \( m \geq 2 \) ordered lists of \( k \geq 2 \) symmetry structures that are pairwise symmetric, then \( S = \{ \{ S_1, \ldots, S_k \}, \ldots, \{ S_{m-1}, \ldots, S_m \} \} \) is a symmetry structure.

3. **(Termination)** If \( S_1, \ldots, S_m \) is a collection of \( m \geq 2 \) symmetry structures then \( S = \{ S_1, \ldots, S_m \} \) is a symmetry structure.

When applied to equal-sized variable groups that have disjoint support, the above construction induces a hierarchical partition that can be represented by a tree with two types of nodes:

1. **Nodes**, depicted as circles, that represent unordered sets \( \{ \} \).
2. **Nodes**, depicted as rectangles, that represent ordered sets \( \langle \rangle \).

Let \( v \) be a node in such a tree, and let \( |v| \) be the cardinality of \( v \). The size of the set it represents; note that \( |v| \) is equal to the number of \( v \)'s immediate predecessors in the tree. The number of variable permutations corresponding to \( v \), denoted by \( \pi(v) \), can be computed according to the formula:

\[
\pi(v) = \begin{cases} 
|v|! \times \prod_{u \in \text{Pred}(v)} \pi(u) & \text{if } v \text{ is un-ordered} \\
\prod_{u \in \text{Pred}(v)} \pi(u) & \text{if } v \text{ is ordered}
\end{cases}
\]

The symmetry check in the recursive step 2 of the above construction can be performed by invoking a condition similar to (11) on representative variable permutations from each of the two symmetric structures being compared. Specifically, to check structures on representative variable permutations from each of the two symmetry structures can be performed by invoking a condition similar to (11)

\[
\prod_{u \in \text{Pred}(v)} \pi(u)
\]

Condition (13) can be verified by checking the equality of \( 2^{p-1}(2^p-1) \) pairs of cofactors on \( \{ x_1, \ldots, x_p \} \) and \( \{ y_1, \ldots, y_p \} \). Clearly, such a check becomes quite expensive as the structures grow in size. Fortunately, the complexity of the check is reduced to \((1/2)p(p+1)\) if the structures checked consist of pairwise symmetric sub-structures. This can be illustrated for \( p = 2 \) by noting that, when \( \{a, b\} \) and \( \{c, d\} \) are assumed to be first-order symmetry groups, the two middle columns (resp. rows) of the cofactor matrix in (12) become identical. This makes it possible to reduce the size of the matrix to \( 3 \times 3 \) by merging rows and columns of equal minor weight:

\[
F_{\langle ab \rangle, \langle cd \rangle} = \begin{bmatrix}
\{f_{0,0}\} & \{f_{0,1}, f_{0,2}\} & \{f_{0,3}\} \\
\{f_{1,0}, f_{2,0}\} & \{f_{1,1}, f_{1,2}\} & \{f_{1,3}\} \\
\{f_{3,0}\} & \{f_{3,1}, f_{3,2}\} & \{f_{3,3}\}
\end{bmatrix}
\]

For groups of \( p \) symmetric variables, the reduction yields a \( (p+1) \times (p+1) \) matrix.

To further reduce the computational cost of constructing a function's hierarchical symmetry partition, we have developed the following necessary condition for two ordered groups of variables to be exchangeable:

**Theorem 4.1** If two ordered disjoint variable groups, \( G_1 = \{ x_1, \ldots, x_p \} \) and \( G_2 = \{ y_1, \ldots, y_q \} \), are symmetric in function \( f \), then variables \( x_1 \) and \( y_1 \) must be symmetric in function \( f^* = \exists G_1 x_1, G_2 y_1(f) \) (15)

The proof to this theorem is given in the Appendix, and shows how the notion of the cofactor matrix can be used to study symmetric properties of a function. To illustrate how one can use this theorem, consider the function \( f = abc + xyz \). According to the theorem, symmetry between \( \langle a, b, c \rangle \) and \( \langle x, y, z \rangle \) requires that \( a \) and \( x \) be symmetric in \( f^* = \exists b, c, y, z(abc + xyz) = a + x \) which trivially, they are. To determine if these two groups are indeed symmetric, we can check the corresponding cofactor matrix for symmetry, which it is.

To emphasize the fact that the condition in Theorem 4.1 is necessary but not sufficient, consider the function \( f = abd' + ab'd + abcd + bcd + b'cd \) which has first-order symmetry groups \( \{a, c\} \) and \( \{b, d\} \). The cofactor matrix on groups \( \langle a, c \rangle \) and \( \langle b, d \rangle \) is

\[
F_{\langle ac \rangle, \langle bd \rangle} = \begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 \\
0 & 1 & 1 & 0 \\
0 & 1 & 1 & 0
\end{bmatrix}
\]

and is clearly asymmetric. This fact, however, is not detected by the theorem's condition since \( a \) and \( b \) are trivially symmetric in \( f^* = \exists c, d(abd' + ab'd + bcd + b'cd) \) = 1.

A final example illustrates the utility of the condition in the above theorem as a way to prune unnecessary symmetry checks on large sets of variables. It is easy to show that the function \( f = (a \oplus c)x + acy)(b \oplus d) \) has first-order symmetry groups \( \{a, c\} \) and \( \{b, d\} \). It is also evident, on the other hand, that \( a \) and \( b \) are not symmetric in the function

\[
f^* = \exists c, d((a \oplus c)x + acy)(b \oplus d)] = x + ay
\]

This immediately implies that there are no second-order symmetries between \( \{a, c\} \) and \( \{b, d\} \), and obviates the need for the more expensive symmetry check implied by (11).

It is interesting to note that a variation on Theorem 4.1, in which the existential quantifier \( \exists \) is replaced by the universal quantifier \( \forall \), is possible. Stronger variations that abstract smaller subsets of variables are also possible and may provide useful trade-offs between runtime efficiency and the accuracy of estimating the function's symmetry. In general, abstracting variables from \( f \) gives a function \( f^* \) with more invariances in the remaining variables, while preserving their \( f \) invariances. This observation provides us with a necessary condition to derive powerful hints for the identification of symmetry substructures.

For a given function a hierarchical partitioning for a set of its variables may not be unique, allowing multiple symmetry structures. Such non-uniqueness of the symmetry structures contrasts to the flat partitioning induced by the equivalence relation of classical first-order symmetries, which is unique. As an example consider function \( f = ab + bc + cd + de + ae \). The function has 5 symmetry structures derived from the hierarchical partition of its variables: \( \{\{ac\}, \{be\}\}, \{d\} \), \( \{\{ab\}, \{ed\}\}, \{c\} \), \( \{\{ad\}, \{ce\}\}, \{b\} \), \( \{\{bc\}, \{ed\}\}, \{a\} \), and \( \{\{ab\}, \{dc\}\}, \{e\} \). Observe that permutations from distinct symmetry structures can be composed to derive a new invariance of \( f \) that are not contained in any of the five listed structures. Indeed, composing non-trivial permutations of the
first four symmetry structures in the listed order we obtain new
permutation \( \langle e, a, b, c, d \rangle \) of a rotational type \([5, 13]\). Such composi-
tional property of the symmetry structures allows us to list implicitly
invariances which cannot be described by a single hierarchical par-
titioning of a variable set.

V. Symmetries Under Phase Assignment

The symmetry condition in (13) can be relaxed to admit more
invariant permutations by allowing the variables being swapped
to have selective inversions. Specifically, let \( \varphi = \langle \varphi_1, \ldots, \varphi_p \rangle \) be a vec-
tor of binary phase assignment variables, and replace (13) with
\[
3 \exists \varphi^T (F_{x \oplus \varphi, y} = F_{x \oplus \varphi, y} )
\]
where \( x = \langle x_1, \ldots, x_p \rangle \), \( y = \langle y_1, \ldots, y_p \rangle \), and the exclusive OR is per-
formed bit-wise.

For example, condition (13) applied to \( f(a, b) = a + b' \)
requires that
\[
(F_{x \oplus \varphi, y}^T) (\varphi, b) = F_{x \oplus \varphi, y} (\varphi, b) \iff \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}
\]
which obviously does not hold. On the other hand, applying (16)
relaxes this requirement, replacing it instead with
\[
(F_{x \oplus \varphi, y}^T) (\varphi, b) = F_{x \oplus \varphi, y} (\varphi, b) + (F_{x \oplus \varphi, y}^T) (\varphi, b) = F_{x \oplus \varphi, y} (\varphi, b)
\]
\[
\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} + \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}
\]
whose second term is true implying the truth of the entire con-
tdition. Thus, the function can be said to have a first-order symmetry
between \( a' \) and \( b \), and that \( \{ a', b \} \) is a symmetry group.

The effect of introducing the phase assignment variables in the
symmetry check can be seen as a “normalization” of the function
that makes it insensitive to the inversion polarity of its inputs. In the
matrix formulation of the symmetry check, different assignments
to the phase variables correspond to different row orderings; if one
or more such orderings yields a symmetric cofactor matrix, the
function can be said to have symmetry under phase assignment.
An alternative formulation of (16) in which the phase assignment
variables are associated with the \( y \) instead of the \( x \) variables is possible
and leads to an equivalent requirement on the function cofactors.
In this case, however, different phase assignments correspond to dif-
ferent column orderings. For our simple example above, we would
deduce that the function is symmetric in \( a \) and \( b' \) implying that
\( \{ a, b' \} \) is a symmetry group. Fig. 5 illustrates the equivalence of
this symmetry group with the one we identified earlier and pictori-
ally shows the corresponding inversions on the function’s inputs,
the resulting hierarchical partitions, and invariant permutations.

Care must be taken when applying (16) to check for higher-
order symmetries. Specifically, the assignments available for the
phase variables at any level of the partition hierarchy must neces-
sarily be constrained by their assignments at earlier levels. The
only flexibility in choosing phase assignments at higher levels of
the hierarchy is to reverse the polarity of the support of a symme-
try structure; this amounts to choosing one of the two alternative phase
assignments propagated from earlier levels of the tree. Symboli-
cally, let \( \hat{\varphi} = \langle \hat{\varphi}_1, \ldots, \hat{\varphi}_p \rangle \) be a phase assignment for which (16) held
at some level of the hierarchy tree. The symmetry check at subse-
quent levels in the tree can then be simplified to:
\[
(F_{x \oplus \varphi, y}^T) (\varphi, y) = F_{x \oplus \varphi, y} + (F_{x \oplus \varphi, y}^T) (\varphi, y) = F_{x \oplus \varphi, y}
\]
where complementation of the phase assignment is bit-wise.

As an example of high-order symmetry under phase assign-
ment consider the function \( f(a, b, c, d) = a'b' + c'd' \). It has first-order
symmetries represented by the symmetry groups \( \{ a', b \} \) and
\( \{ c, d' \} \). A check of second-order symmetry between these two
groups entails the construction of the following two cofactor matrices:
\[
F_{\langle a'b \rangle, \langle c'd \rangle} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}, \quad F_{\langle a'b \rangle, \langle c'd \rangle} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}
\]
Since the second matrix is symmetric, we can infer that
\( \{ a', b \}, \{ c', d \} \) is a symmetry structure for this function.

VI. Characterization of Function Symmetry in Bench-
mark Circuits

We performed an extensive study of available benchmark circuits
to determine their symmetry partitions based on the generalized
symmetry model presented in this paper. Specifically, we analyzed
the 2812 output functions of the 101 logic synthesis and optimiza-
tion benchmarks available from MCNC [23]. These circuits come
from three suites: the multi-level MCNC benchmarks, the multi-
level ISCAS-85 benchmarks, and the two-level MCNC bench-
marks.

The multi-level circuits were flattened before the symmetry par-
titions of their outputs were computed; thus, the reported symmetry
partitions reflect the intrinsic functional structure of these outputs
rather than any structural regularity in their circuit implementa-
tions. A summary of these results is shown in Table 1.

Several observations can be made from these data. The most
striking is the relatively small number of output functions that do
not exhibit any symmetries. Considering that fact that some of
these functions were generated synthetically to stress synthesis
algorithms, this suggests that the majority of functions one is likely
to encounter in practical design situations will possess some degree
of symmetry. The data also show that a small number of functions
have higher order symmetries. In the majority of those cases, the
order of symmetry was 2; several functions exhibited symmetries
of order 3 and 4. As a measure of the additional symmetries found
by hierarchical partitioning, we tabulate the ratio of the number of
invariant permutations induced by the hierarchical partition to
those induced by the first-order partition. This ratio ranged from
a minimum of 2 to a maximum of 10^91. Finally, symmetry groups
ranget in size from a minimum of 2 to a maximum of 64.

We should point out that the symmetry structures were computed under a restriction on the size of ordered groups as well as the variable order within those groups. Specifically, ordered groups chosen for symmetry checks were selected by partitioning the variables into equal-sized subsets using their netlist order. This was done for subset sizes from 2 to 10. This restriction was motivated by the desire to keep the computational effort reasonable, but is otherwise arbitrary.

In the remainder of this section we provide a closer examination of the symmetries discovered in four of the benchmark circuits: t481, C432, C499, and C6288.

**t481.** This benchmark is interesting because its only output has a 4-level hierarchical symmetry partition. Its symmetry structure, annotated with the number of permutations induced at each node, is given below:

The symmetry involves both ordered and un-ordered groups of variables and requires phase assignment to normalize the function. The number of invariant permutations induced by this partition is 8192 which is 16 times the number of invariant permutations induced by the flat partition of first-order symmetries. The multi-level netlist for this benchmark is quite irregular and large (over one thousand gates). This is at odds with the highly regular symmetry structure shown above and suggests that other implementations that are more compact might be possible.

**C432.** Of the seven output functions for this benchmark, only one (223GAT(89)) has first-order symmetry:

The second-order symmetries exhibited by the other outputs are between ordered groups of variables that range in size from 3 to 6. These symmetries correspond to a substantial number of invariant permutations (up to $10^{3.6}$ for two of the outputs) that would have been overlooked by classical symmetry. The detailed hierarchical

<table>
<thead>
<tr>
<th>Benchmark suite</th>
<th>Total # of output functions</th>
<th># of output functions with no symmetries</th>
<th>High-order symmetry statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td># of output functions with high-order symmetries</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Max order of symmetry</td>
</tr>
<tr>
<td>Multi-level MCNC</td>
<td>1560</td>
<td>76</td>
<td></td>
</tr>
<tr>
<td>Multi-level ISCAS85</td>
<td>703</td>
<td>85</td>
<td></td>
</tr>
<tr>
<td>Two-level MCNC</td>
<td>549</td>
<td>68</td>
<td></td>
</tr>
</tbody>
</table>

**C499.** This circuit has 32 outputs, none of which exhibits any symmetry in terms of its 41 inputs. Based on this fact, one may be led to believe, erroneously, that these functions lack any regularity. Closer examination, however, reveals that a significant amount of symmetry exists in this circuit when its high-level structure is recognized. This structure is depicted below:

The circuit performs single-error-correction [7] and consists of two main modules $M1$ (syndrome generator) and $M2$ (error correction) that are quite regular. The circuit illustrates that completely asymmetric functions may result from the composition of highly symmetric functions. It also suggests that a suitable high-level decomposition might help uncover such latent symmetries. A characterization of the symmetry inherent in C499 can be given in terms of some of the circuit's internal signals in addition to its primary outputs. For the module $M1$ we first give symmetries of some of its internal signals $D_i$ and $H_i$:

$H_i: \{R, IC_i\}, D_i: \{ID_{i_1}, \ldots, ID_{i_2}\}$

$(0 \leq i \leq 7, i_j \in \{0, \ldots, 31\})$

These signals then combine to form symmetries of the $M1$ syndrome outputs:

$S_i: \{D_i, H_i\}$

$(0 \leq i \leq 7)$

We first describe the $M2$ equations in terms of the module’s internal signals $E_j$:

$E_j: \{\bar{S}_0, \ldots, \bar{S}_7\}$

$(0 \leq i \leq 31)$

where dots above $S_j$’s indicate either complemented or non-complemented phase. Together with $ID_j$’s we can use the $E_j$ signals to describe the circuit outputs:

$OD_j: \{E_j, ID_j\}$

$(0 \leq i \leq 31)$

**C6288.** Another example that lacks first-order symmetry is the C6288 16-bit multiplier. In fact, this circuit is not included in data counts of Table 1 because it cannot be flattened due to the exponential memory requirements for its BDD. Smaller multipliers that could be flattened showed little first-order symmetry. However,
under a remapping of the multiplier’s 32 inputs into the domain of its 16^2 partial products, a three-level structure that is rich with symmetries emerges. In fact, this structure can be readily obtained by a partial flattening of the circuit that stops at the first level of 256 AND gates whose output signals correspond to the partial products. Expressing the output functions of the circuit in terms of these signals we obtain functions which are highly symmetric. The first-order symmetry profiles of the 32 remapped output functions \( p_k \) are given in terms of following formula:

\[
\begin{align*}
1(2) & \quad \text{if } k = 0 \\
1(2) \ldots 1(k) & \quad \text{if } 1 \leq k \leq 15 \\
1(2) \ldots 1(30 - k) & \quad 2(30 - k + 1) \ldots 2(15) \quad 1(16) & \quad \text{if } 16 \leq k \leq 29 \\
1(1) & \quad 2(2) \ldots 2(15) \quad 1(16) & \quad \text{if } k = 30, 31
\end{align*}
\]

With each output \( p_k \) of the partial multiplier this formula identifies a list of groups using \( n(s) \) notation, where \( n \) is the number of groups of size \( s \).

VII. Conclusions

In this work we studied functional symmetry relying on invariance under unrestricted variable permutations. This definition encompasses the classical notion of symmetry, based on variable swaps, as a special case. Our studies are based on the new hierarchical partitioning scheme that generalizes the flat partitioning implied by classical symmetry and yields more invariant permutations. The hierarchical partitioning algorithm is based on the symbolic detection of symmetry in specially-constructed cofactor matrices. The runtime efficiency of hierarchical partitioning was shown to be quite reasonable, aided in part by the application of a necessary condition that quickly detects asymmetry. Application of hierarchical partitioning to a large number of benchmark circuits revealed the existence of significant symmetries.

Acknowledgments

We would like to thank the reviewers, Igor L. Markov, and Juan Antonio Carballo for their valuable feedback in preparation of the paper.

References


Appendix

(Theorem 4.1 proof) The proof is done by contradiction. Suppose groups \( G_1 \) and \( G_2 \) are symmetric in \( f \), and variables \( x_1 \) and \( y_1 \) are not symmetric in \( f^* \). Symmetry between \( G_1 \) and \( G_2 \) implies that the cofactor matrix

\[
F(G_1) = \begin{bmatrix}
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\end{bmatrix}
\]

must be symmetric, where \( n = 2^n - 1 \). Since in the symmetric matrix we have \( \forall i, j, f_{i,j} = f_{j,i} \) it must also be true that

\[
\sum_{i<j} f_{i,j} = \sum_{i<j} f_{j,i}
\]

Here the left and right hand sides correspond to the summation of cofactors in the boxed upper-right and lower-left corners of matrix \( F(G_1) G_2 \). Since \( x_1 \) and \( y_1 \) occupy the most significant bit positions in their respective groups, it is should be clear that these sums define the cofactors of \( f^* \):

\[
\begin{align*}
\sum_{i<j} f_{i,j} &= f^*_{x_1 y_1} = \sum_{i<j} f_{i,j} \\
\sum_{i<j} f_{j,i} &= f^*_{x_2 y_2} = \sum_{i<j} f_{j,i}
\end{align*}
\]

But this implies that \( f^*_{x_1 y_1} \neq f^*_{x_2 y_2} \), contradicting the assumption that \( x_1 \) and \( y_1 \) are not symmetric in \( f^* \). Therefore \( x_1 \) and \( y_1 \) must be symmetric in \( f^* \). It is easy to show that the same holds true for any corresponding pair of variables \( x_i \) and \( y_i \).