Why is Symmetry So Hard?

Peter M. Maurer
Dept. of Computer Science
Baylor University
Waco, Texas 76798

Abstract – The problem of detecting virtually any type of symmetry is shown to be co-NP-complete. We start with totally symmetric functions, then extend the result to partially symmetric functions, then to more general cofactor relations, and finally to generic permutation-group symmetries. We also show that the number of *types* of symmetry grows substantially with the number of inputs, compounding the complexity of an already difficult problem.

1 Introduction

Even though symmetry is useful in a number of different contexts within the field of Electronic Design Automation [1-7], existing algorithms have worst-case time-bounds that are exponential, or worse. Even if an algorithm were polynomially bounded with respect to the size of the ROBDD, transforming a function from a standard form such as a Boolean expression into a ROBDD can take an exponential amount of time, giving an exponential time-bound for the entire process [8]. Indeed, since the very first design automation paper on symmetry [9] it has been clear that detecting symmetry this is a difficult problem. Work such as that described in [10] suggests that even given a fully tabulated truth-table for a function, it can still be difficult to determine whether a function is symmetric. Other algorithms have hidden traps, like the time required to compute cofactors [1], which can be exponentially bounded if minimization of the cofactors is required.

Despite the fact that many algorithms appear to give acceptable performance for realistic functions, it is unlikely that the number of inputs of these functions could be increased substantially (to tens of thousands of inputs, say) and still give acceptable performance. In this paper we attempt to explain this phenomenon by showing that even the simplest problems in detecting symmetry are co-NP-complete. (A problem is co-NP-complete if it is the complement of an NP-complete problem.) Because the problems are co-NP-complete, as opposed to NP-complete, they are harder to approximate than many NP-complete problems.

Even if the problem of detecting known types of symmetry were easy, the problem of detecting general symmetries would still be difficult because the number of *types* of symmetry increases substantially with an increase in the number of function inputs. It is difficult or impossible to give a uniform characterization of these symmetry types, requiring the use of ad-hoc techniques that are customized for each particular type of symmetry. This adds an additional layer of difficulty to an already difficult problem.

2 Total and Partial Symmetry

Much of the work being done today focuses on cofactor relations. Partial and total symmetries can be detected by comparing the cofactors of a function [11]. A cofactor of a Boolean function f is computed by setting one or more variables to constants. Subscripts are used to indicate which variables have been replaced. For example, consider the function $f = ab + cd$. If we set a to zero we obtain the cofactor $f_{0xxx} = cd$, and if we set a to one we obtain $f_{1xxx} = b + cd$. When there is no danger of ambiguity, we omit the *x*'s from the subscripts.

In symmetry detection, cofactors computed with respect to pairs of variables. The cofactors are compared, and detected symmetric pairs are combined into larger sets based on symmetric-pair transitivities. [1, 12]. If all pairs of variables are symmetric, then the function is totally symmetric. Because of transitivities, only a subset of the variable pairs need be tested. Even though only $n-1$ comparisons need be done to show that a function is totally symmetric, each comparison needs to determine whether two Boolean functions are equivalent, which is itself an NP-complete problem. Furthermore, computing cofactors can be costly if minimization or canonicalization is required.

These difficulties are not due to a defect in the algorithm, but to an inherent difficulty in the problem itself, as Theorem 1 shows. In Theorem 1, we show that the problem of showing that a function has at least one pair of inputs that are *not* symmetric is NP-Complete. This shows that the problem of determining total symmetry is co-NP-complete.

Theorem 1. *The set $\overline{SYM}$ of Boolean expressions that represent non-totally-symmetric functions is NP-complete.*
Proof. A function is totally symmetric if and only if any two input vectors of the same weight produce the same output. The set $\overline{SYM}$ is in NP, because for any Boolean expression e we can non-deterministically select two input vectors, and evaluate the two vectors on e. If the two vectors are of equal weight and if the result of the evaluations is different we accept. To show that $\overline{SYM}$ is NP-Hard, we start with SAT, the set of satisfiable Boolean expressions. Let e be any Boolean expression and x_1 and x_2 be two variables not contained in e. Let $g = x_1(e) + x_2'(e)$. If e is not satisfiable, then neither is g. All non-satisfiable formulas represent the zero function, which is symmetric, so if e is non-satisfiable,

then g is totally symmetric. If e is satisfiable, let v be a satisfying assignment of e. We append 01 and 10 to v obtaining $v01$ and $v10$, inputs to g. Now, $g(v01) = 0 \cdot 1 + 0 \cdot 1 = 0$ and $g(v10) = 1 \cdot 1 + 1 \cdot 1 = 1$ so g is not symmetric in the variables x_1 and x_2.■

Theorem 1 shows that the problem of detecting totally symmetric functions is not just NP-complete, but co-NP-complete. In some respects, a problem that is co-NP-complete is no harder than a problem that is NP-complete, but in many cases the co-NP-complete problems are harder to approximate than the NP-complete ones. For example, suppose we wish to find a function that is not totally symmetric, but don't care precisely which function we find. We can generate functions at random, and evaluate them on a number of pairs of inputs of the form xxx01xxx, xxx10xxx, where the 01 and 10 appear in the same position, and the remainder of the positions are identical. If we obtain a different result for a pair, we know the function is not totally symmetric.

The random approach cannot be used to prove that a function *is* totally-symmetric, however. For a function to be totally symmetric, every pair of input variables must be symmetric, $n-1$ pairs must be tested, and 2^{n-2} input vector pairs must be tested for each pair of variables. If we miss testing even one pair of vectors, the function may not be totally symmetric.

Because the same basic algorithms can be used to detect both partial and total symmetry, it is reasonable to suppose that partial symmetry is just as hard to detect as total symmetry. Theorem 2 confirms this by showing that the problem of testing a particular pair of input variables for symmetry is co-NP-Complete. This is a far stronger result than that of Theorem 1 because instead of dealing with the overall problem, Theorem 2 speaks to the most basic operation used to detect symmetry.

Theorem 2. *Let i and j be integers with $1 \le i < j \le n$. Let $SYM(i,j)$ be the set of n-input Boolean expressions where the variable pair (x_i, x_j) is symmetric. The complement of $SYM(i,j)$, $\overline{SYM(i,j)}$, is NP-complete.*

Proof. If (x_i, x_j) is a symmetric variable pair of function f. then the cofactors f_{01} and f_{10} with respect to (x_i, x_j) must be equal. $\overline{SYM(i,j)}$ is in NP, because we can nondeterministically select an $n-2$ element vector, inserting 01 into positions i and j to create v_1 and 10 into the same positions to create v_2. We evaluate f on v_1 and v_2 and accept if the results are different.

Let $G = x_i x'_j f$. If f is zero then G is zero. If f is one, the f is equal to zero for $(x_i, x_j) = (0,1)$ and equal to one when $(x_i, x_j) = (1,0)$. If f is not satisfiable, then neither is G, and G is symmetric in all variable pairs, including (x_i, x_j). If f is satisfiable, then it has the value one for at least one input, but we don't know the value of (x_i, x_j) for this input.

If we complement the variable x_i in f we obtain the function $C_i(f)$. These two functions have the following relationship. $f(\ldots,0,\ldots) = C_i(f)(\ldots,1,\ldots)$ and $f(\ldots,1,\ldots) = C_i(f)(\ldots,0,\ldots)$, where the 0 and 1 appear in position i. If f is satisfiable, then one of f and $C_i(f)$ has a satisfying assignment with 0 in position i and the other has a satisfying assignment with a 1 in position i, and both of these satisfy the function $f + C_i(f)$. The function $g = f + C_i(f) + C_j(f) + C_i(C_j(f))$ has at least four different satisfying assignments, v_0, v_1, v_2 and v_3 which are identical in all positions except i and j, with 00, 01, 10, and 11 in positions i and j respectively. If we substitute g for f in G, we obtain $h = x_i x_j g$. If f is not satisfiable, then neither is h, and h is symmetric in the variable pair (x_i, x_j). However, if f is satisfiable, then $h(v_1) = 0$ and $h(v_2) = 1$, so (x_i, x_j) is not a symmetric variable pair for h. The function h can be computed in polynomial time.■

Theorem 2 concerns itself with only one type of cofactor relation. There are many others, and one might suspect that Theorem 2 could be expanded to cover these other relations as well. This is indeed the case, and the proofs are all virtually identical to that of Theorem 2. The only difference is the function h used to prove NP-Hardness. Rather than repeat the same argument over and over, we simply provide a table of relations and the respective function h used to prove NP-Hardness. The function g given in the proof of Theorem 2 is an essential part of these functions, because it is necessary to have predictable values in certain positions of the satisfying assignment.

The relations can be broken down into several categories. Figure 1 gives the table for the classical relations which define ordinary symmetry, multi-phase symmetry, and the four types of single-variable symmetry. Detecting any one of these types of symmetry is co-NP-complete.

Symmetry	Relation	Function
Ordinary	$f_{01} = f_{10}$	$x_i x'_j g$
Multi-Phase	$f_{00} = f_{11}$	$x_i x_j g$
Single-Variable	$f_{00} = f_{01}$	$x'_i x_j g$
Single-Variable	$f_{00} = f_{10}$	$x_i x'_j g$
Single-Variable	$f_{01} = f_{11}$	$x_i x_j g$
Single-Variable	$f_{10} = f_{11}$	$x_i x_j g$

Figure 1. The classical Relations.

The simplest extension to the classical relations is the anti-relations. One can recast the classical relations in terms of the XOR function, which we designate as $\oplus$. Ordinary symmetry would be recast as $f_{01} \oplus f_{10} = \overline{0}$, where $\overline{0}$ is the zero function. The other five relations would be recast in a similar manner. To obtain the anti-relations we replace the zero function with the constant-one function, $\overline{1}$. Figure 2 gives the table of functions for the anti-relations.

Relation	Function
$f_{01} \oplus f_{10} = \overline{1}$	$x_i x_j' g'$
$f_{00} \oplus f_{11} = \overline{1}$	$x_i x_j g'$
$f_{00} \oplus f_{01} = \overline{1}$	$x_i' x_j g'$
$f_{00} \oplus f_{10} = \overline{1}$	$x_i x_j' g'$
$f_{01} \oplus f_{11} = \overline{1}$	$x_i' x_j g'$
$f_{10} \oplus f_{11} = \overline{1}$	$x_i x_j' g'$

Figure 2. The Anti Relations.

The Kronecker relations are three and four-cofactor relations. The NP-completeness results for these relations are no more difficult to obtain than for the classical and anti relations. Figure 3 lists the Kronecker relations and the functions used in the corresponding NP-completeness proofs.

Relation	Function
$f_{01} \oplus f_{10} \oplus f_{00} = \overline{0}$	$x_i' x_j' g$
$f_{01} \oplus f_{10} \oplus f_{11} = \overline{0}$	$x_i x_j g$
$f_{11} \oplus f_{10} \oplus f_{00} = \overline{0}$	$x_i x_j g$
$f_{11} \oplus f_{01} \oplus f_{00} = \overline{0}$	$x_i x_j g$
$f_{01} \oplus f_{10} \oplus f_{00} \oplus f_{11} = \overline{0}$	$x_i x_j g$

Figure 3. The Positive Kronecker Relations.

The Kronecker relations can be extended in the same way as the classical relations by replacing the zero function with the constant-one function. This gives us the negative Kronecker relations. These relations are listed in Figure 4 along with the function to be used in the corresponding NP-Completeness proof.

Relation	Function
$f_{01} \oplus f_{10} \oplus f_{00} = \overline{1}$	$x_i' x_j' g'$
$f_{01} \oplus f_{10} \oplus f_{11} = \overline{1}$	$x_i x_j g'$
$f_{11} \oplus f_{10} \oplus f_{00} = \overline{1}$	$x_i x_j g'$
$f_{11} \oplus f_{01} \oplus f_{00} = \overline{1}$	$x_i x_j g'$
$f_{01} \oplus f_{10} \oplus f_{00} \oplus f_{11} = \overline{1}$	$x_i x_j g'$

Figure 4. The Negative Kronecker Relations.

Although the list of relations given in Figures 1-4 is extensive, it is far from comprehensive. One could extend these relations by replacing the constant-one, and constant-zero functions with more general functions, and one could also replace the XOR functions with other types of function such as AND and OR. It is unlikely that these extensions would prove to be any easier to handle than the relations listed here.

3 Generalized Symmetry

The list of relations given in Section 2, though extensive, involves cofactors of only two variables. Recent work has shown that there are useful types of symmetry that cannot be expressed in terms of two-variable cofactor relations. For example, consider the function $x_1x_2 + x_3x_4$. In addition to the partial symmetries in variable pairs (x_1, x_2) and (x_3, x_4), the pair of variables (x_1, x_2) can be exchanged with the pair (x_3, x_4). This type of symmetry is called *hierarchical* symmetry. One way to deal with general symmetries is to define symmetry in terms of input-variable *permutations* rather than independent relations between pairs of variables.

In simplest terms, a permutation on a set of n elements is a rearrangement of the numbers 1, 2, 3, …, n. If we assume that the inputs of a function are indexed by these numbers, then any permutation of these numbers can be translated into a permutation of the input variables.

There are many ways to express permutations, but one of the simplest is to write it as a sequence of cyclic shifts. A permutation that rotates the variables x_1, x_2, x_3 and x_4 so that x_1 replaces x_2, x_2 replaces x_3, x_3 replaces x_4 and x_4 replaces x_1, is written as (1,2,3,4). A permutation that exchanges variables x_5 and x_6 is written as (5,6). Cyclic shifts are known as *cycles*, with the name being further qualified with the length. Thus (1,2,3,4) is a 4-cycle, and (5,6) is a 2-cycle. 2-cycles are often referred to as *transpositions*.

Many permutations can be expressed as a single cycle, but many others must be expressed as a sequence of two or more disjoint cycles, such as (1,2)(3,4). (Two cycles are disjoint if they have no number in common.)

It is important to emphasize that a permutation is actually a one-to-one function from a set to itself. This allows us to combine two permutations using function composition and obtain another permutation. We sometimes call this *multiplying* two permutations. It is easy to multiply two cycles as in the following example: (1,2)(1,3)=(1,2,3). The cycles (1,2) and (1,3) are not disjoint, so their product is another cycle. For disjoint products like (1,2)(3,4), the product-form cannot be further simplified.

The set of all permutations on a set of n elements is called the *symmetric group of degree n*, and is written S_n. S_n is a group under function composition, because the operation is closed, associative, there is an identity element, and every permutation has an inverse.

If it is possible to permute the inputs of a function f using permutation p without changing f, then we say that f is *invariant* with respect to p. The set of all permutations that leave a function f invariant is a group called the symmetry group of f. Since the identity function is a permutation, the symmetry group always exists for any

Boolean function f. For non-symmetric functions, the symmetry group is just $\{I\}$ the identity permutation. For totally symmetric functions, the symmetry group is S_n. For other types of symmetry, the symmetry group is a subgroup of S_n. Non-total symmetries can be broken down into partial symmetry, and other kinds of symmetry which are usually lumped together and called "weak symmetry." There are many different kinds of weak symmetry, and the two primary questions are, "How many kinds of weak symmetry are there?" and "Is weak symmetry any easier to detect than total and partial symmetry?"

As near as we can determine, the number of symmetry types is infinite. A number of different types have been documented in the literature, including hierarchical and rotational symmetry [13, 14]. In some sense, each subgroup of S_n represents a different type of symmetry, but some of these are only trivially distinct from one another. For example, a partial symmetry in the first two variables has a different subgroup than a partial symmetry in the second two variables, but these two groups don't really represent distinct types of symmetry. The two groups in question are $\{I,(1,2)\}$ and $\{I,(2,3)\}$. These two groups are isomorphic to one another, but we must be careful, because the two groups $\{I,(1,2)\}$ and $\{I,(1,2)(3,4)\}$ are also isomorphic, and represent two different types of symmetry.

The important relationship is that of conjugacy. Two permutations, p and q, are conjugate to one another if there is another permutation g with inverse g^{-1} such that $p = g^{-1}qg$. This definition can easily be extended to subgroups of S_n. If two groups are conjugate to one another, they are said to belong to the same conjugacy class. So the real question is not the number of subgroups of S_n, but the number of conjugacy classes of S_n. We don't know of any formula that will give us the number of conjugacy classes of S_n for a given n, but we have empirical data for the symmetric groups S_2 through S_7. Figure 6 gives the number of conjugacy classes for S_2 through S_n. Because S_n contains n isomorphic copies of S_{n-1}, the number of symmetry groups grows at least factorially with n. From the limited data we have obtained, the number of classes appears to grow more or less linearly, but this is still a substantial increase when we consider that different classes normally require different detection algorithms.

Unfortunately, knowing the number of conjugacy classes, still does not give us the number of distinct symmetry types. There are some subgroups of S_n that cannot be the symmetry group of any Boolean function. These are called the *forbidden groups*. To see why forbidden groups exist, it is necessary to understand the concept of Boolean orbits.

Degree	Classes	Subgroups
2	1	1
3	4	6
4	11	30
5	19	156
6	**56**	1455
7	96	Unknown

Figure 6. Conjugacy Classes.

To examine the Boolean orbits of a symmetry group, we have to shift our viewpoint slightly. Instead of considering a *n*-input function f to be a function of n independent variables, we consider f to be a function with a single vector-valued input. Instead of using permutations to rearrange the variables of f, we use permutations to rearrange the input vectors of f. Let G be the symmetry group of f, and let u and v be two input vectors of f. We say that u and v are in the same Boolean orbit of G if there is a permutation $p \in G$ such that applying p to u gives us v.

For example, the three input AND function is totally symmetric and has symmetry group S_n. The Boolean orbits of S_n are given in Figure 7.

Boolean Orbit 0 – 000
Boolean Orbit 1 – 001 010 100
Boolean Orbit 2 – 011 101 110
Boolean Orbit 3 – 111

Figure 7. Boolean Orbits of S_3.

The symmetry class of a function is completely determined by the Boolean orbits of its symmetry group. If two inputs are in the same Boolean orbit, they must yield the same output value. The Boolean orbits are a partition of the truth table of the function based on its symmetry.

If two different subgroups have the same Boolean orbits, then they define the same symmetry class. For example, $S_3 = \{I,(1,2),(1,3),(2,3),(1,2,3),(1,3,2)\}$ and $A_3 = \{I,(1,2,3),(1,3,2)\}$ are two distinct subgroups with $A_3 \subset S_3$. The Boolean orbits of A_3 are given in Figure 8. Note that these Boolean orbits are identical to those of S_3. Therefore, any function that is invariant with respect to A_3 must be totally symmetric, and A_3 is forbidden.

Boolean Orbit 0 – 000
Boolean Orbit 1 – 001 010 100
Boolean Orbit 2 – 011 101 110
Boolean Orbit 3 – 111

Figure 8. Boolean Orbits of A_3.

If two distinct subgroups G and H of S_n have the same Boolean orbits then either $G \subset H$ or $H \subset G$ or there is a third group K with the same Boolean orbits as G and H such that $G \subset K$ and $H \subset K$.

As of yet, generalized symmetry deals only with permutations, not with the extensions offered by multi-phase symmetry, single-variable symmetry, anti-symmetry and Kronecker symmetry.

This brings us back to the problem of testing for various types of symmetry. From Theorems 1 and 2, it seems likely that testing for generalized symmetry would be difficult, and we can show that this is indeed the case.

Theorem 3. *Let p be an arbitrary k-cycle on a set of* n *elements. Let* $\overline{SYM(p)}$ *be the set of Boolean expressions that are* not *left invariant by p. The set* $\overline{SYM(p)}$ *is NP-complete.*

Proof. $\overline{SYM(p)}$ is in NP, because given a Boolean expression *e*, we can nondeterministically guess two input vectors v_1 and v_2.and evaluate *e* on both. If $v_1 = p(v_2)$ and the result of the two evaluations is different, then accept.

To show that $\overline{SYM(p)}$ is NP-hard, assume that *p* is at least a 3-cycle (otherwise Theorem 2 applies) and is of the form (*s*,*q*,*r*,...), where *s*, *q*, and *r* are integers. Let *e* be an arbitrary Boolean expression, and let g_1 be the function defined as follows. $g_1(e) = e + C_q(e) + C_r(e) + C_q(C_r(e))$. If *e* is not satisfiable, then neither is $g_1(e)$, however if *e* is has a satisfying assignment, then $g_1(e)$ has (at least) four satisfying assignments which are identical in all positions except q and r. In particular, $g_1(e)$ has a satisfying assignment, v, with $x_q = 1$, and $x_r = 0$.

If we now apply the permutation p to the variables of g_1 we obtain the function g_2. The variable x_s has replaced x_q and x_q has replaced x_r, so g_2 has a satisfying assignment, u with $x_s = 1$ and $x_q = 0$. Furthermore if we apply the permutation p to u, we obtain $v = p(u)$, a satisfying assignment of g_1. The function $g_1 + g_2$ is satisfied by both u and v, but the function $g = x_q(g_1 + g_2)$ has the value one for v and the value zero for u, so g is not invariant with respect to p. The function g can be computed in polynomial time.

If e is not satisfiable, then g is the constant zero function, which is invariant with respect to all permutations, in particular, p.■

Although we now know that detecting invariance for an arbitrary *k*-cycle is NP-complete, most permutations must be expressed as a series of disjoint cycles, all of which are applied simultaneously. Consider, for example, the permutation (1,2,3,4,5)(6,7,8,9)(10,11,12)(13,14). We could use constructions such as the functions g and g_1 of Theorems 2 and 3, but these constructions quickly become tedious, and must be repeated for each cycle in the permutation. In the proof of Theorem 4, we will side-step the problem by adding new variables to move the "extra" cycles out of the way.

Theorem 4. *Let p be an arbitrary permutation on a set of* n *elements. Let* $\overline{SYM(p)}$ *be the set of Boolean expressions that are* not *left invariant by p. The set* $\overline{SYM(p)}$ *is NP-complete.*

Proof. If p is a single cycle, then Theorems 2 and 3 apply, so let us assume that p has more than one cycle, and that the cycles are disjoint. What we would like to do is start with an arbitrary Boolean expression e and modify it in some way to control the form of the satisfying assignments. This will enable us to construct an expression that is non-symmetric whenever the expression e is satisfiable. This is difficult to do when we have multiple cycles, so we will "move the extra cycles out of the way," so we can concentrate on a single cycle. Let c be the longest cycle of p. If there is more than one longest cycle, then choose the one that contains the smallest number. Let S be the set of numbers that appear in p, S_1 be the set of numbers that appear in c, and $S_2 = S - S_1$. Let $m = |S_2|$ be the size of S_2. Let e be an arbitrary Boolean expression containing the variables $\{x_1, x_2, x_3, ..., x_n\}$, not all of which need to appear in e. Create m new variables $K_1 = \{x_{n+1}, x_{n+2}, ..., x_{n+m}\}$. Let K_2 be the set of variables affected by p, but not by c. In other words, $K_2 = \{x_i \mid i \in S_2\}$. Obviously, K_2 has m elements. In the expression e, replace each variable in K_2 with a distinct member of K_1, giving the new expression e'. The expression e' does not contain any of the variables in K_2. Now create the new expression $g = e' y_1 y_2 ... y_m$, where y_i is the i^{th} element of K_2. The expression g is satisfiable if and only if e is satisfiable, and every satisfying assignment v of g must have the form, $y_1 = 1, y_2 = 1, ..., y_m = 1$. Applying the permutation p to v does not change this, so applying p changes only those inputs indexed by c. The expression g can be computed in polynomial time.

We may now complete the proof using the arguments of Theorems 1 and 2.■

Theorem 4 is indeed bad news. Not only is it necessary to search through a set of $n!$ permutations to determine which leave a function invariant, each search is co-NP-complete. As pointed out in [13], it is actually not necessary to test every element of a group for invariance, since testing the generators of the group is sufficient to show invariance for the entire group. (The set of generators is usually very small compared to the size of the group.) But this still does not get around the co-NP-completeness of the individual permutation tests.

4 Conclusion

We have shown that detecting virtually any type of symmetry is a co-NP-complete problem. Testing for any of the well-known relations of two-variable cofactors leads to a co-NP-complete problem. These relations include the classical, anti, and Kronecker relations.

Detecting general symmetries not based on two-variable cofactor relations is also a co-NP-complete problem. In fact, as the number of input variables increases, the number of different types of symmetry becomes extremely large. For eight or more inputs, it is not even clear what symmetries can exist.

What is surprising is that there are many algorithms that give perfectly acceptable performance in practice, even though their worst-case performance (assuming a Boolean expression for input) is exponential in the number of inputs. These algorithms are generally used with relatively small numbers of inputs, and on functional representations (ROBDDs for example) which can be exponentially large compared to the corresponding Boolean expressions. If these algorithms were run on Boolean expressions with many thousands of inputs, their exponential nature would be much more evident.

This paper shows that the worst-case exponential nature of existing symmetry detection algorithms is an inherent property of the problem being solved and not a flaw in the algorithms themselves.

5 References

[1] P. M. Maurer, "Conjugate Symmetry," *Formal Methods Syst. Des.,* Vol.38, *No.3,* pp. 263-288, 2011.

[2] C. R. Edwards and S. L. Hurst, "A digital synthesis procedure under function symmetries and mapping methods," *IEEE Transactions on Computers,* Vol.27, *No.11,* pp. 985-997, 1978.

[3] D. Moller, P. Molitor, R. Drechsler and J. W. G. U. Frankfurt, "Symmetry based variable ordering for ROBDDs," *IFIP Workshop on Logic and Architecture Synthesis,* pp. 47-53, 1994.

[4] C. Scholl, D. Moller, P. Molitor and R. Drechsler, "BDD minimization using symmetries," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* Vol.18, *No.2,* pp. 81-100, 1999.

[5] T. Sasao, "A new expansion of symmetric functions and their application to non-disjoint functional decompositions for LUT type FPGAs," *IEEE International Workshop on Logic Synthesis,* pp. 105-110, 2000.

[6] V. N. Kravets and K. A. Sakallah, "Constructive library-aware synthesis using symmetries," *Design Automation and Test in Europe,* pp. 208-213, 2000.

[7] C. C. Tsai and M. Marek-Sadowska, "Boolean functions classification via fixed polarity Reed-Muller forms," *IEEE Transactions on Computers,* Vol.46, *No.2,* pp. 173-186, 1997.

[8] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Transactions on Computers,* Vol.35, *No.8,* pp. 677-691, 1986.

[9] C. E. Shannon, "The synthesis of two-terminal switching circuits," *Bell System Technical Journal,* Vol.28, *No.1,* pp. 59-98, 1949.

[10] A. Mukhopadhyay, "Detection of total or partial symmetry of a switching function with the use of decomposition charts," *IEEE Transactions on Electronic Computers,* Vol.12, *No.5,* pp. 553-557, 1963.

[11] M. Chrzanowska-Jeske, A. Mishchenko and J. R. Burch, "Linear Cofactor Relationships in Boolean Functions," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* Vol.25, *No.6,* pp. 1011-1023, 2006.

[12] C. C. Tsai and M. Marek-Sadowska, "Generalized Reed-Muller forms as a tool to detect symmetries," *IEEE Transactions on Computers,* Vol.45, *No.1,* pp. 33-40, 1996.

[13] V. N. Kravets and K. A. Sakallah, "Generalized symmetries in boolean functions," *IEEE International Conference on Computer Aided Design,* pp. 526-532, 2000.

[14] J. Mohnke, P. Molitor and S. Malik, "Limits of using signatures for permutation independent Boolean comparison," *Formal Methods Syst. Des.,* Vol.21, *No.2,* pp. 167-191, 2002.