

A Universal Symmetry Detection Algorithm

Peter M. Maurer
Dept. of Computer Science
Baylor University
Waco, Texas 76798-7356
Waco, Texas 76798

Abstract – Research on symmetry detection focuses on identifying and detecting new types of symmetry. We present an algorithm that is capable of detecting any type of permutation-based symmetry, including many types for which there are no existing algorithms. General symmetry detection is library-based, but symmetries that can be parameterized, (i.e. total, partial, rotational, and dihedral symmetry), can be detected without using libraries. In many cases it is faster than existing techniques. Furthermore, it is simpler than most existing techniques, and can easily be incorporated into existing software.

1 Introduction

A symmetric Boolean function is a function whose inputs can be rearranged in some fashion without changing the output of the function. The importance such functions was first recognized by Shannon in [1], who characterized function symmetries using permutations of the input variables. Virtually all existing symmetry detection algorithms are based on Shannon's Theorem which detects symmetry by comparison of two-variable cofactors. This approach can detect all total and partial symmetries, but there are many types of symmetries that cannot be detected in this manner. As the number of input variables grows, these types of symmetry become more common than partial and total symmetry. Some progress has been made in detecting symmetries beyond partial and total symmetry [2-4], but the problem of universal symmetry detection has remained open since 1949.

Our experiments with standard benchmarks [5] show that such symmetries are common. Ignoring such symmetries can cause major failures in layout verification and regression [6]. For such algorithms, incorrect handling of symmetry can cause many false errors, making it easy to miss the real errors. Correct handling of symmetry is also important when attempting to match design specifications to an existing library of functions [7]. If symmetry handling fails, functions may have to be created by hand even though there is an acceptable library function to implement it.

In this paper, we use an entirely new approach which, effectively, considers all inputs simultaneously instead of in pairs. This approach allows us to detect virtually any type of symmetry, including some types that go beyond permutations. For small numbers of inputs (<8) our approach is faster than using cofactors. In addition, the coding is simpler. We provide pseudo code in Section 4, which can easily be adapted for use in existing EDA algorithms. Our algorithm also is somewhat easier to parallelize than the conventional algorithm, because it does not require the accumulation of results to completely characterize a gate.

2 Basic Principles

Symmetries can be categorized into total symmetry, partial symmetry, and strong symmetry. Total symmetry permits the inputs of a function to be rearranged arbitrarily without changing the output of the function. Partial symmetry is similar to total symmetry in that it permits one or more subsets of inputs to be rearranged arbitrarily.

Strong symmetry is a catch-all term that includes every type of symmetry that is neither total nor partial. The function $a+b+c+d$ is totally symmetric and the function $abc+d$ is partially symmetric. The functions $a'b+c'd$ and $ab+cd$ are strongly symmetric. In $a'b+c'd$ no single variable can be exchanged with any other single variable, but the set $\{a,b\}$ can be exchanged with the set $\{c,d\}$. Most existing algorithms will detect two partial symmetries in $ab+cd$ but ignore the fact that the set $\{a,b\}$ can be exchanged with the set $\{c,d\}$. Strong symmetry is not detectable using two-variable cofactors.

There are many more kinds of strong symmetry than partial and total symmetry [8]. Various sub-categories of strong of symmetry have been discovered, and algorithms have been created to detect and exploit some of these symmetries [7]. Examples of such symmetries are hierarchical symmetry, rotational symmetry and dihedral symmetry [9].

The primary tool for categorizing symmetry is the permutation group [10]. Let X be a finite set of objects. A permutation is a one-to-one function from X to itself. The set of all permutations of a set X is called the *symmetric group* on X and is written S_X . The only thing that affects the structure of S_X is the size of X . If X and Y are two sets of the same size, then S_X and S_Y are identical. If $p \in S_X$, and the size of X is n then we say that n is the *degree* of p . If $X = \{1,2,\dots,n\}$ we write S_X as S_n .

Let p be a permutation of degree n and let f be an n -input Boolean function. We say that p and f are *compatible* if using p to rearrange the variables of f leaves the output of f unchanged. We also say that f is *invariant with respect to p* . We extend this terminology in the obvious way to subgroups of S_n , and define the *symmetry group* G_f to be the set of all permutations that leave f invariant. Because the identity element leaves every function invariant, G_f is never empty. Most recognized types of symmetric functions can be characterized using symmetry groups. For example, an n -input function f is totally symmetric, if and only if $G_f = S_n$. A function is *non-symmetric* if $G_f = \{I\}$.

Symmetric variable pairs are detected by comparing the cofactors of a function [3]. A cofactor of f is found by setting one or more input variables to constant values. For example, let $f = ab+cd$. Two cofactors of f are $f_{x=1,x} = ab+d$ and $f_{0,xx} = cd$. When the unaffected variables are obvious, it is common to omit the x 's. The pair (a,b) is a symmetric variable pair if and only if $f_{01} = f_{10}$. Because symmetric variable pairs are transitive, they can be used to detect all partial and total symmetries.

3 Orbits and Boolean Orbits

Orbits have been used for many years to analyze and categorize permutation groups [7, 10]. Two variables $a, b \in X$ are said to be in the same *orbit of G* if there is a permutation $p \in G$, such that $p(a) = b$. Intuitively, an orbit contains all the variables that can be exchanged with one another, so the function $abc + d$ has two orbits $\{a, b, c\}$ and $\{d\}$. Belonging to the same orbit is an equivalence relation, so it breaks the set of input variables into a collection of disjoint subsets.

Orbits can be used to distinguish total and partial symmetries, but are not particularly effective with strong symmetries. Consider the function $ab + cd$, which possesses dihedral symmetry. This function has only one orbit, $\{a, b, c, d\}$. The totally symmetric function $a + b + c + d$ also has a single orbit, $\{a, b, c, d\}$. Thus the functions $a + b + c + d$ and $ab + cd$ have the same orbits even though their symmetries are quite different.

We have discovered a new type of orbits, called *Boolean Orbits*, that permit us to deal with strong symmetries as well as partial and total symmetries. Boolean orbits are computed with respect to the Boolean input vectors of a function rather than with respect to the variables. Permutations of degree n can operate on n -element vectors by permuting the indices of the elements. This has the effect of permuting the elements of the vector. For example, we can apply the permutation (1,2,3) to the vector (1,1,0) to obtain the vector (0,1,1).

Two n -input vectors v and w are in the same *Boolean orbit of G* if there is a permutation $p \in G$ such that $p(v) = w$. Like ordinary orbits, belonging to the same Boolean Orbit is an equivalence relation, so this relation breaks the set of n -input Boolean vectors into a collection of disjoint sets. If G_f is the symmetry group of a Boolean function f , then the Boolean orbits of G_f will partition the truth-table of f into disjoint sets. In fact, the symmetry of a Boolean function is *completely determined* by the Boolean orbits of its permutation group. Figure 1 shows the symmetry groups and the Boolean orbits of the two functions $a + b + c + d$ and $ab + cd$. Note that the Boolean orbits of the two functions are quite different, even though the conventional orbits are the same.

$a + b + c + d$	$ab + cd$
$\{a, b, c, d\}$	$\{a, b, c, d\}$
0000	0000
0001 0010 0100 1000	0001 0010 0100 1000
0011 0101 1001 0110 1010 1100	0011 1100
0111 1011 1101 1110	0101 0110 1001 1010
1111	0111 1011 1101 1110
	1111

Figure 1. Orbits, Symmetry Groups, and Boolean Orbits.

We can summarize the important properties of Boolean orbits in the following theorems. Theorem 1 states that a symmetric Boolean function must map the elements of each of its Boolean orbits to a unique value. (Due to space limitations, we omit the proofs of our theorems.)

Theorem 1. Let f be a Boolean function, and G_f be the symmetry group of f . If K is a Boolean orbit of G_f and $u, v \in K$, then $f(u) = f(v)$.

Theorem 2 is the converse of Theorem 1. It states that if the Boolean function, f , maps the orbits of a symmetry group, G , to unique values, then f is compatible with G .

Theorem 2. Let f be an n -input Boolean function and let $G \subseteq S_n$ be a group such that for every Boolean orbit, K , of G , and for every pair of elements $u, v \in K$, $f(u) = f(v)$ then f is invariant with respect to G , and $G \subseteq G_f$.

Boolean orbits have two important properties that can be useful in some applications. The first concerns the weight, $w(v)$, of a vector v , which is the number of ones in the vector.

Theorem 3. Let $K \subseteq B^n$ be a Boolean orbit of some group $G \subseteq S_n$, and let $v_1, v_2 \in K$. Then $w(v_1) = w(v_2)$.

Theorem 3 implies that the minimum number of orbits for any n -input function is $n + 1$. The next property involves the complement v' of a vector v . Let $v = (v_1, v_2, \dots, v_n)$ and $v' = (v'_1, v'_2, \dots, v'_n)$. If $v_i = 1$, then $v'_i = 0$ and vice-versa. We start with the following lemma.

Lemma 1. Suppose v is an n -element Boolean vector, and that p is a permutation of order n . If $p(v) = u$, then $p(v') = u'$.

Let S be a set of n -element Boolean vectors. Then $S' = \{v' \mid v \in S\}$. We can apply Lemma 1 to Boolean orbits to obtain the following result.

Theorem 4. If K is a Boolean orbit of a group $G \subseteq S_n$, then K' is also a Boolean orbit of G .

Theorem 5 deals with the problem of functions that have more than one type of symmetry. As this theorem shows, if a Boolean function f has two different types of symmetry A, and B, then f also possesses an overarching symmetry that includes both A and B. Thus if we are able to identify the largest symmetry group that is compatible with f , then we are guaranteed to have discovered all of the symmetries possessed by f .

Theorem 5. Let f be a Boolean function and let H be two permutation groups that are compatible with f . Then there is a permutation group, K compatible with f such that G and H are both subgroups of K .

Let S be any set of n -element Boolean vectors. The characteristic function of S , C_s is an n -input Boolean function which is equal to 1 on every element of S , and zero elsewhere. Figure 2 gives a set of orbits along with their characteristic functions in truth-table form.

000	00000001
001 010 100	00010110
011 101 110	01101000
111	10000000

Figure 2. Boolean Orbits and Characteristic Functions.

For an n -input Boolean function, f , we divide the set n -element Boolean vectors into two sets U_f and Z_f , which are the sets of points where f takes the value one and the value zero respectively. The characteristic functions of these sets are just f and \bar{f} .

Let f and g be two n -input Boolean functions. The function f is said to *imply* g if $g(v)=1$ whenever $f(v)=1$. Our detection algorithm is based on implication, as Theorem 6 shows.

Theorem 6. Let $G \subseteq S_n$ be a permutation group, and let f be an n -input Boolean function. Let $K = \{K_1, K_2, \dots, K_n\}$ be the collection of Boolean orbits with characteristic functions $\{C_1, C_2, \dots, C_k\}$. The group G leaves f invariant if and only if C_i implies either f or \bar{f} for every i , $1 \leq i \leq k$.

Theorem 6 shows how to detect symmetry with respect to *any* permutation group. Given a permutation group G it is straightforward to compute the characteristic functions of its Boolean orbits. Given f it is a straightforward task to compute \bar{f} . Once these functions have been computed, we only need to check the characteristic function of each orbit to determine whether f is symmetric with respect to G .

Although most symmetry detection will be done on single-output functions, the algorithm can be applied just as easily to multiple-output functions.

4 The Symmetry Detection Algorithm

Figure 3 gives the pseudo code for the Universal Symmetry Detection (USD) algorithm. It is assumed that the algorithm is being applied to a collection of functions, and that a library of symmetries is being used. Each library entry contains a set of characteristic functions that correspond to the Boolean orbits of a symmetry group. The library may be all subgroups of S_n for some integer n , or some other more specialized library. We currently have complete libraries for S_2 through S_8 . When used with a complete library for S_n , symmetry detection begins with the largest group so the algorithm may stop as soon as a compatible group is found.

The algorithm reads each function, and compares each function to each library entry until a compatible entry is found. If no compatible subgroup can be found in the library, the function is marked as non-symmetric.

Libraries are not necessary for symmetries that can be parameterized for an arbitrary number of inputs. As yet, only a few symmetries have been so categorized, the most well known of which are total, symmetry, partial symmetry, rotational symmetry, dihedral symmetry, and various types of hierarchical symmetry. The USD algorithm has special generators for total, partial, dihedral and rotational symmetry, which permits these types of symmetries to be detected without having a precomputed library.

Most of our libraries contain one entry per symmetry group. For large numbers of inputs it is not feasible to store libraries in this fashion. We use the *conjugacy* relation to reduce the size of the library for large numbers of inputs. Certain types of symmetry are fundamentally the same, but applied to different inputs, and certain types of symmetry are fundamentally different. For example, a 3-input partial symmetry on the first three inputs of a function is not fundamentally different from a three-input partial symmetry on the last three variables. But a partial symmetry in the first three variables *is* fundamentally different from a partial symmetry in the first *two* variables. The conjugacy relation is used to distinguish symmetries that are essentially the same from symmetries that are fundamentally different.

Two permutations p and q are *conjugate* to one another if there is another permutation s such that $p = s^{-1}qs$. (Conjugacy can be best understood by visualizing it in this way: to permute the last three variables of a function, we move them to the first three variables

using s , then apply q to the first three variables, and then use s^{-1} to move the variables back where they were.) This relationship can be extended to permutation groups in the following way: $s^{-1}Gs = \{s^{-1}ps \mid p \in G\}$. If two symmetries are fundamentally the same then their permutation groups will be conjugate to one another.

Conjugacy is an equivalence relation, so the subgroups of a group can be partitioned into a set of conjugacy classes. In the full libraries, we store each subgroup of the symmetric group. In reduced libraries we store only one member of each conjugacy class.

Load Library

Sort Library into descending order by subgroup size.

For each function f

For each subgroup G in Library

 GroupCompatible = True;

For each orbit K of G **While** GroupCompatible

 OrbitCompatible = False;

For each orbit P of f

If C_K implies C_P

 OrbitCompatible = True;

Break;

EndIf

EndFor

If Not OrbitCompatible

 GroupCompatible = False;

Break;

EndIf

EndFor

If GroupCompatible

 Assign G as the symmetry group of f ;

Break;

EndIf;

EndFor;

If Not GroupCompatible

 Mark f as nonsymmetric

EndIf;

EndFor;

Figure 3. The Universal Symmetry Detection Algorithm.

To regenerate a conjugacy class, it is necessary to compute the conjugates of each library entry. However for each subgroup G of S_n there are many pairs of permutations (p, q) such that $p \neq q$, but $p^{-1}Gp = q^{-1}Gq$. To avoid duplicated work we store a set of permutations with the library entry.

The permutations are computed when creating the library. A group theoretic result states that “the number of conjugates of a group is equal to the index of its normalizer [11].” The normalizer of a group is the set of permutations that leave G unchanged with respect to conjugacy. That is, the set $\{p \mid p^{-1}Gp = G\}$. If G is a subgroup of S_n , then its index is equal to $|S_n|/|G|$, which is the number of right cosets of G in S_n . (A right coset of G is obtained by multiplying every element of G by some element p of S_n . It is written Gp .) Let $N(G)$ be the normalizer of G . If p and q are members of the same right coset of $N(G)$, then $p^{-1}Gp = q^{-1}Gq$. If the permutations come from two different right cosets, then the conjugates will be different. A set of *coset representatives*, which includes one permutation from each right coset of $N(G)$, can be used to generate the entire conjugacy class of G . Figure 4 gives the library entry that is used to detect 2-variable partial symmetry in 3-input functions. The

permutations are coded in the form of a list of numbers from the set $\{0,1,2\}$. The first permutation is the identity, I . The first line of the entry is the name of the symmetry, the second is the number of inputs, and the remaining lines give the orbits, one orbit per line.

For the larger symmetric groups, reconstructing all conjugacy classes is a physical impossibility. So instead we use the set of stored permutations to alter the function under test. Let's suppose that g is invariant with respect to $p^{-1}Gp$. Then, there is an f which is invariant with respect to G such that $p^{-1}f = g$. If g is invariant with respect to $p^{-1}Gp$, then pg is invariant with respect to G .

5 Sub-Symmetries

For functions with many inputs, it may be more useful to detect smaller, more manageable symmetries on a subset of inputs. We call such symmetries *Sub-Symmetries*. The procedure alters the functions under test, and is based on the following theorem.

Theorem 7. Let R be a symmetry rule of degree k , and let f be a function of $n > k$ inputs. Let S be a subset of k inputs taken from the n inputs of f . If f possesses R symmetry in the set of k variables, then every cofactor obtained by fixing the $n - k$ variables to constant values must possess R symmetry.

There are 2^{n-k} such cofactors for each set of k inputs. When testing an n -input function using a symmetry rule of degree $k < n$, the USD algorithm begins by generating all combinations of n inputs taken k at a time. For each combination, the USD algorithm generates all 2^{n-k} cofactors, and tests each one for R symmetry. It generates an ascending sequence of k numbers taken from the set $\{0,1,\dots,n-1\}$.

The algorithm then tests for a sub-symmetry in the k variables selected by the combination. Each such test requires computing 2^{n-k} cofactors. These cofactors are computed by setting the variables not selected by the permutation to every possible combination of zeros and ones. The procedure continues until a sub-symmetry is found, or until all combinations have been exhausted

```
S3.S2
3
CSR: (0,1,2), (0,2,1), (2,1,0)
000
001
010 100
011 101
110
111
```

Figure 4. A Reduced Library Entry.

6 Experimental Data

The USD algorithm was designed to be used with a variety of Boolean function implementations. The only requirement for the algorithm to function correctly is the ability to compute the implication relation. For compressed libraries and sub-symmetries it is also necessary to compute cofactors and the product of a permutation with a function.

We ran all experiments on modest hardware: a Dell laptop containing an Intel P9500 Core 2 Duo 2.53Ghz CPU with 3.48 Gigabytes of RAM and Windows XP Professional with Service Pack 3. We measured the amount of real time required to determine the symmetry of 1,000,000 functions. The results are reported in Figure 5. For three and four input functions, it was necessary to test the same

functions repeatedly. For five, six and seven inputs, the first 1,000,000 functions were tested, treating the truth tables of the functions as 32-bit and 64-bit integers respectively.

We ran tests on a number of standard benchmark circuits: ISCAS85, LGSynth89, and LGSynth91. For the ISCAS85 benchmarks, we clustered the circuit into fanout-free networks, partitioned each network into 6 input (or less) functions. The other benchmarks were used without modification. We found many functions with symmetries that are undetectable by other methods, including some whose symmetries are almost too complicated to describe. The run-time for the benchmark circuits was too small to be measured.

Number of Inputs	USD	Conventional
3	08.268	124.736
4	12.464	258.660
5	30.420	433.378
6	246.751	640.029
7	3189.840	893.361

Figure 5. Seconds per 1,000,000 Functions.

7 Conclusion

The USD algorithm is a simple, yet powerful and efficient algorithm for detecting virtually any type of symmetry. It is our belief that many types of symmetry *could* be exploited if there were methods to detect them. Because the USD algorithm makes these types of symmetry accessible, we expect to see significantly more exploitation of symmetry in the future. The USD is a powerful tool that can be used in many different contexts.

8 References

- [1] C. E. Shannon, "The synthesis of two-terminal switching circuits," *Bell System Technical Journal*, vol. 28, pp. 59-98, 1949.
- [2] C. C. Tsai and M. Marek-Sadowska, "Boolean matching using generalized reed-muller forms," in *Proceedings of the 31st Annual Design Automation Conference*, 1994, pp. 339-344.
- [3] M. Chrzanowska-Jeske, "Generalized symmetric variables," in *The 8th IEEE International Conference on Electronics, Circuits and Systems*, 2001, pp. 1147-1150.
- [4] V. N. Kravets and K. A. Sakallah, "Generalized symmetries in boolean functions," in *IEEE International Conference on Computer Aided Design*, 2000, pp. 526-532.
- [5] F. Brglez, P. Pownall and R. Hum. Accelerated ATPG and fault grading via testability analysis. Presented at Proceedings of IEEE Int. Symposium on Circuits and Systems. 1985, .
- [6] P. M. Maurer and A. D. Schapira, "A Logic-to-Logic Comparator for VLSI Layout Verification," *IEEE Transactions on Computer-Aided Design*, vol. 7, pp. 897-907, 1988.
- [7] J. Mohnke, P. Molitor and S. Malik, "Limits of using signatures for permutation independent Boolean comparison," *Formal Methods Syst. Des.*, vol. 21, pp. 167-191, 2002.
- [8] D. F. Holt. Enumerating subgroups of the symmetric group. *Computational Group Theory and the Theory of Groups, II* pp. 33-37. 2010.
- [9] V. N. Kravets and K. A. Sakallah, "Generalized symmetries in boolean functions," Advanced Computer Architecture Laboratory Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109, 2002.
- [10] D. S. Passman, *Permutation Groups*. New York: W. A. Benjamin, 1968.
- [11] D. Robinson, *A Course in the Theory of Groups*. New York: Springer, 1995.