THE COMPLEXITY OF DETECTING SYMMETRIC FUNCTIONS

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

ABSTRACT

The characterization of the symmetries of boolean functions is important both in automatic layout synthesis, and in automatic verification of manually created layouts. It is possible to characterize the symmetries of an *n*-input boolean function as an arbitrary subgroup, G, of S_n, the symmetric group of order *n*. Given an expression *e*, which represents an *n*-input boolean function F, and a subgroup G of S_n, the problem of whether F possesses symmetry G is an **NP**-complete problem. The concept of an orbit can be used to characterize the various types of symmetry for a specified number of inputs. This classification can then be used, along with a few partitioning rules to completely determine the symmetries of a boolean function. This technique requires that the truth-table of a function be completely enumerated, and thus has a running time proportional to 2^n, where *n* is the number of inputs of the function. Some of the mathematical concepts presented to support the **NP**-completeness result have intriguing possibilities for circuit minimization.

THE COMPLEXITY OF DETECTING SYMMETRIC FUNCTIONS

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

1. Introduction.

The study of symmetric boolean functions in *n* variables has a long history. The original motivation for studying such functions was to simplify the analysis and design of relay-based switching networks[1,2]. (Although curiously enough, more recent research in circuit complexity shows that symmetric functions do not necessarily have simple implementations[3].) Over the years attempts have been made to find efficient algorithms for identifying symmetric functions[4] and for creating symmetric functions from general boolean formulas [5,6,7]. The techniques for identifying symmetric functions require that the truth-table of the function (or some variation thereof) be enumerated. When applied to boolean expressions, these techniques are exponentially bounded with respect to the length of the expression. Indeed, Section 4 shows that the set of boolean formulas that represent non-totally-symmetric functions is **NP**-complete[8,9,10].

Today there are still strong motivations for studying symmetric functions. First, the symmetry of certain functions can be used to simplify routing problems encountered in the automatic synthesis of integrated-circuit layouts. (This is generally known as the *pin assignment* problem[11].) For a pin-assignment algorithm to work effectively, it must possess an accurate knowledge of the symmetries of the gates used in the layout. Second, when manually-created integrated circuit layouts are automatically verified against logic specifications[12], an accurate knowledge of gate-symmetries is essential to avoid generating spurious error messages. In neither case is it necessary for the algorithms to dynamically analyze the symmetry of a gate. It is sufficient for the analysis of a gate's symmetry to be determined "off-line" and either recorded in a cell-library or hard-coded into the algorithms.

The primary purpose of this paper is to show that the problem of determining whether the function represented by a boolean expression has a certain type of symmetry is **NP**-complete. This paper uses a somewhat more general definition of symmetry than is found in most discussions of symmetric functions. In particular, Section 2 of this paper introduces the concept of G-symmetry for a general permutation group G. This more general definition is necessary because there exist functions that possess these types of symmetries, and these functions must be correctly handled by pin-assignment and layout verification programs. Section 2 introduces the mathematical background of the work presented in the paper. For the most part, this background is taken from elementary group theory. Readers requiring a more comprehensive introduction should consult an introductory text such as [13]. Section 3 presents the primary **NP**-Completeness results. Section 4 suggests a method for determining the symmetries of a specific function. Section 5 presents an analysis of the symmetries of 2, 3, and 4-input gates. Section 6 discusses opportunities for future research, and elaborates on some of the mathematical concepts introduced in Section 2. Section 7 draws conclusions.

2. Mathematical Background.

Much of the existing work on symmetric boolean functions relies on an intuitive definition of symmetry. This approach works quite well, and could be used to obtain the results presented in this paper. However, there are some interesting twists to the formal mathematical definition of symmetry that will be discussed in section 6. Furthermore, the mathematical approach tends to clarify some of the concepts that are intuitively difficult to grasp. For this reason, this paper will lean more toward mathematical rigor than toward intuition, although formalism for the sake of formalism will be avoided.

Intuitively, a symmetric function is on in which the inputs may be permuted, or rearranged, without changing the value of the function. When a function is written as a logical expression, this is taken to mean rearranging the variables of the expression. The primary problem with such a definition is that it may be possible to represent a 6-input function (say) with an expression containing only four variables. Some concepts discussed in this paper require us to describe the effect of rearranging *all* inputs of a function, not just those that appear in an expression representing the function.

We begin with the concept of a *permutation*. A permutation on X_n, a set of *n* objects, is a one-to-one function from X_n to itself. For simplicity, we will assume that $X_n = \{1,2,3, \ldots , n\}$. The set of all permutations on a set of *n* objects forms a group called the *symmetric group* of order *n*, written S_n. Now, suppose V is an *n*-dimensional vector space, and N is the set of all non-singular linear transformations from V to itself. A *representation* of S_n is a homomorphism **K** from S_n to N. If **K** is an isomorphism, the representation is said to be *faithful*.

Any *n*-input boolean function, F, can be considered to be a function on $GF(2)^n$, the vector space of dimension *n*, where GF(2) is the integers modulo-2. Let $p \in Sn$, and $(a_1,a_2, \ldots ,a_n) \in GF(2)^n$. Let $\mathbf{M}(p)=\mathbf{T}_p$ be the transformation defined as follows: $\mathbf{T}_p(a_1,a_2, \ldots , a_n) = (a_{p(1)},a_{p(2)}, \ldots , a_{p(n)})$. For all $p \in S_n$, $\mathbf{T}_p$ is a non-singular linear transformation from $GF(2)^n$ to itself. **M** is a faithful representation of S_n called the *canonical* modulo-2 representation of S_n. Although rearrangements of *n*-element vectors can be easily understood without this formalism, there are modulo-2 representations of S_n other than the canonical representation. These representations will be discussed briefly in Section 6. It is impossible to discuss these other representations without recourse to the formal definition of a representation. In most cases, it will not be necessary to formally distinguish between a permutation p and the linear transformation $\mathbf{T}_p$ representing it. Therefore, we will speak of permutations whenever there is no possibility of confusion.

Let F be an *n*-input boolean function, and let p be a permutation, and let Fp be the composition of the two. The function Fp is called a *rearrangement* of F. The permutation p is *compatible* with F if and only if Fp=F. For any *n*-input function F, the set of all permutations compatible with F is a subgroup, G, of S_n., called the *symmetry rule of the function*. If G is equal to S_n, then F is said to be *totally symmetric*, if G contains only the identity element, then F is said to be *non-symmetric*, if G is isomorphic to S_m, where $m<n$, then F is said to be *partially symmetric*, otherwise F is said to be *G-symmetric*. Of course, the concepts of rearranging the inputs of a boolean function, and rearranging the variables of a boolean expression representing the function are equivalent.

The proofs presented in the next section require that certain operations be performed on expressions rather than on the functions they represent. Furthermore, it is necessary to be able to characterize the performance of these operations to permit a complete analysis of the computational complexity of various algorithms. To this end, consider the grammar for boolean expressions pictured in Figure 1.

```
<exp> -> <exp> + <term>
<exp> -> <term>
<term> -> <term> <factor>
<term> -> <factor>
<factor> -> <var>
<factor> -> <var>'
<factor> -> ( <exp> )
```

Figure 1. A Grammar for Boolean Expressions.

For simplicity, the logical-complement operation has been omitted from the grammar illustrated in Figure x, however, since complemented inputs *are* allowed, the grammar is sufficiently powerful to describe any boolean function. Furthermore, it is obvious that adding the complement operation will not significantly change the grammar Note that this grammar is known to be both unambiguous and LR(1).

Let *e* be an expression representing an *n*-input function. The value of *e* on an *n*-element vector *v*, is given by the function eval(*e*,*v*), which is defined in Figure 2.. It is obvious from Figure x that "eval" is polynomially bounded in the length of *e*.

eval(exp+term,v) = OR(eval(exp,v),eval(term, v))
eval(term factor,v) = AND(eval(term,v),eval(factor,v))
eval((exp),v) = eval(exp,v)
eval(a_i ,v) = v_i
eval(a_i ',v) = NOT(v_i)

Figure 2. An Expression Evaluator.

The function $\mathbf{C}_i$ is used to complement the *i*th variable of an expression. The dual of this function in the domain of boolean vectors is $\mathbf{c}_i$, which complements the *i*th element of a boolean vector. The definition of $\mathbf{C}_i$ is given in Figure 3. Again, it is obvious from Figure 3, that $\mathbf{C}_i$ is polynomially bounded in the length of *e*. Using the definition of $\mathbf{C}_i$ and the definition of "eval" it is possible to prove that $\mathbf{C}_i$ and $\mathbf{c}_i$ are equivalent.

$\mathbf{C}_i$(exp + term) = $\mathbf{C}_i$(exp) + $\mathbf{C}_i$(term)
$\mathbf{C}_i$(term factor) = $\mathbf{C}_i$(term) $\mathbf{C}_i$(factor)
$\mathbf{C}_i$((exp)) = ($\mathbf{C}_i$(exp))
$\mathbf{C}_i$(var) = var' if var = a_i, and var otherwise.
$\mathbf{C}_i$(var') = var if var = a_i, and var' otherwise.

Figure 3. A Variable Complementer.

Finally, function $\mathbf{V}_p$ is the dual of $\mathbf{T}_p$ in the domain of boolean expressions.. The function $\mathbf{V}_p$ is defined in Figure 4. The expression to which $\mathbf{V}_p$ is applied is assumed to represent an *n*-input function, and the variable-names used in the expression are assumed to be taken from the set $\{a_1, a_2, ... , a_n\}$. (If the expression is not already in this form, it can be placed in this form in polynomial time.)

Let $q = p^{-1}$.
$\mathbf{V}_p$(exp+term) = $\mathbf{V}_p$(exp)+$\mathbf{V}_p$(term)
$\mathbf{V}_p$(term factor) = $\mathbf{V}_p$(term) $\mathbf{V}_p$(factor)
$\mathbf{V}$p((exp)) = ($\mathbf{V}_p$(exp))
$\mathbf{V}$p(a_i) = $a_{q(i)}$
$\mathbf{V}$p(a_i') = $a'_{q(i)}$

Figure 4. A Variable Permuter.

If *e* represents a function F with "don't care" inputs,it is possible for $\mathbf{V}_p(e)$ to contain variable names that do not occur in *e*. Like the other functions presented in this section, $\mathbf{V}_p$ is polynomially bounded.

The *weight* of an *n*-element boolean vector *v* is the number of "ones" in the vector. The function w(*v*) represents the weight of *v*. As noted by Shannon[2], the value of a totally symmetric function depends only on the weight of its input vector, not on the positions of the ones and zeros. This fact will be used in the next section to demonstrate that certain functions are not totally symmetric.

3. Complexity Results

The first result is to show that the set of expressions that represent non-totally-symmetric functions is **NP**-complete. Let **EXP** be the set of all boolean expressions, and let **SYM** be the set of boolean expressions that represent totally symmetric functions. Let **NSYM** be the complement of **SYM** with respect to **EXP**. Membership in **NSYM** is complicated by the fact that an expression *e* can represent several different functions. It is conceivable that *e* represents a non-totally symmetric function on $GF(2)^n$, but is not in **NSYM** because it represents a totally symmetric function on $GF(2)^{n+k}$ for some $k>0$. Happily, this situation does not occur, as the following lemma shows.

LEMMA. If *e* represents a non-totally symmetric function on $GF(2)^n$ then *e* represents a non-totally symmetric function on $GF(2)^{n+k}$ for all $k>0$.

Proof. The proof is by induction on *k* with the basis given by the statement of the lemma. Suppose *e* represents a non-totally symmetric function f on $GF(2)^n$. Then there are two *n*-element vectors of equal weight $v1=(x_1,x_2, \ldots ,x_n)$ and $v2=(y_1,y_2, \ldots ,y_n)$ such that $f(v1)=1$ and $f(v2)=0$. Now $v1'=(x_1,x_2, \ldots ,x_n,0)$, and $v2'=(y_1,y_2, \ldots ,y_n,0)$ are of equal weight. And certainly, $eval(e,v1')=eval(e,v1)=1$, while $eval(e,v2')=eval(e,v2)=0$. Therefore *e* represents a non-totally symmetric function on $GF(2)^{n+1}$.$\therefore$

This lemma shows that membership in **NSYM** can be established by showing that *e* represents a non-totally symmetric function on $GF(2)^n$, where *n* is the number of distinct variables in *e*. This fact will be used in the proof of Theorem 1.

THEOREM 1. The set **NSYM** is **NP**-complete.

Proof. To prove **NP**-completeness, we must establish two things. First we must show that any element of **NSYM** can be identified in polynomial time by a non-deterministic Turing Machine. Second, we must exhibit a polynomially bounded function f such that for some known **NP**-complete set X, $f(x)\in$**NSYM** if and only if $x\in X$. We will use **SAT**, the set of all satisfiable boolean expressions as our known **NP**-complete set. To show that **NSYM** is in NP, consider the following algorithm.

```
Input $e$
n<- the number of distinct variables in $e$.
Choose an $n$-element vector $v1$.
Choose an $n$-element vector $v2$.
$a$<- the weight of $v1$
$b$<- the weight of $v2$
$x$<- eval($e$,$v1$)
$y$<- eval($e$,$v2$)
if $a=b$ and $x\neq y$ then ACCEPT
```

Each each step of this algorithm is polynomially bounded. Since the procedure contains no loops, the overall time bound is also given by a polynomial.

Given a boolean expression *e* let *x* and *y* be variables that do not appear in *e*. In general *x* and *y* can be found in no more than $O(n^2)$ time where *n* is the length of *e*. Now let $f(e)=x(e)+y\,'(e)$. Assume that $e\in$SAT. Then there exists a $v\in GF(2)^k$ such that $eval(e,v) = 1$. Suppose $v=(x_1,x_2, \ldots ,x_k)$. Let $v1=(x_1, \ldots ,x_k,1,0)$, and $v2 = (x_1, \ldots ,x_k,0,1)$. Then

eval(f(e),$v1$)=
OR(AND(1,eval(e,$v1$)),AND(NOT(0),eval(e,$v1$)))=
OR(AND(1,eval(e,v)),AND(NOT(0),eval(e,v)))=
OR(AND(1,1),AND(1,1))=
OR(1,1)=1.

But

eval(f(e),$v2$)=
OR(AND(0,eval(e,$v2$)),AND(NOT(1),eval(e,$v2$)))=
OR(AND(0,eval(e,v)),AND(NOT(1),eval(e,v)))=
OR(AND(0,1),AND(0,1))=
OR(0,0)=0.

Since $v1$ and $v2$ obviously have the same weight, f(e)$\in$**NSYM**. Now suppose that e is not satisfiable. Then eval(e,v)=0 for all $v \in GF(2)^n$ $n>=k$. Let $v1=(x_1,x_2,...,x_{k+2})$ be an element of $GF(2)^{k+2}$. Then

eval(f(e),$v1$)=
OR(AND(x_{k+1},eval(e,$v1$)),AND(x_{k+2},eval(e,$v1$)))=
OR(AND(x_{k+1},eval(e,v)),AND(x_{k+2},eval(e,v)))=
OR(AND(x_{k+1},0),AND(x_{k+2},0))=
OR(0,0)=0.

Therefore f(e) is not satisfiable. Since f(e) is not satisfiable, it represents the totally symmetric zero function.$\therefore$

Theorem 1 shows that the set of expressions that represent non-totally-symmetric functions is **NP**-complete. Now, the symmetry group of a totally-symmetric function is as large as it could possibly be. If a smaller symmetry group were chosen, would the problem of identifying functions not compatible with the group become easier? Theorem 2 shows that the answer is no. Even if the "smallest possible" symmetry group were chosen, the problem of identifying functions not compatible with the group remains **NP**-complete. The group chosen for Theorem 2 is one containing the identity element and a single 2-cycle. Recall that a 2-cycle is a permutation that exchanges two elements of X sub n, say i and j, and leaves the other elements of X sub n intact. It is written (i j).

THEOREM 2. Let i and j be integers such that $i<j$. Let **EXP** be the set of boolean expressions, and let $\mathbf{Y}_{i,j}$ be the set of expressions in **EXP** that represent functions compatible with the permutation group {I,(i j)}. The complement of $\mathbf{Y}_{i,j}$ in **EXP**, denoted $\mathbf{NY}_{i,j}$, is **NP**-complete.

Proof. Note that if an n-input boolean function f is compatible with (i , j) then for all values of $(x_1,...,x_{i-1},x_i,x_{i+1},...,x_{j-1},x_j,x_{j+1},...,x_n)$,
$f(x_1,...,x_{i-1},1,x_{i+1},...,x_{j-1},0,x_{j+1},...,x_n)=$
$f(x_1,...,x_{i-1},0,x_{i+1},...,x_{j-1},1,x_{j+1},...,x_n)$.
The following algorithm shows that $\mathbf{NY}_{i,j}$ is in **NP**.

```
Choose $n$-2 boolean values $x_1,...,x_{i-1},x_{i+1},...,x_{j-1},x_{j+1},...,x_n$ ;
$a$<- f($x_1,...,x_{i-1},1,x_{i+1},...,x_{j-1},0,x_{j+1},...,x_n$);
$b$ <- f($x_1,...,x_{i-1},0,x_{i+1},...,x_{j-1},1,x_{j+1},...,x_n$);
If a != b ACCEPT ;
```

To show that $\mathbf{NY}_{i,j}$ is **NP**-hard, we will exhibit a polynomial-time mapping g from the set of expressions to **EXP** such that g(e)$\in\mathbf{Y}_{i,j}$ if and only if $e\in$**SAT**. The function g is defined as follows. If e is a boolean expression, then g(e) = $x_i(e+\mathbf{C}_i(e)+\mathbf{C}_j(e)+\mathbf{C}_j(\mathbf{C}_i(e)))$. If e is not satisfiable, then neither is $\mathbf{C}_i(e)$. This, in turn implies that if e is not satisfiable then neither is g(e), by the following calculation.

eval(g(e),v)=
eval($x_i(e+\mathbf{C}_i(e)+\mathbf{C}_j(e)+\mathbf{C}_j(\mathbf{C}_i(e)))$,$v$)=
AND(eval(x_i,v),
    OR(OR(OR(eval(e,v),eval($\mathbf{C}_i(e)$,v)),eval($\mathbf{C}_j(e)$,v)),eval($\mathbf{C}_j(\mathbf{C}_i(e))$,$v$)))=
AND(eval(x_i,v),OR(OR(OR(0,0),0),0))=
AND(eval(x_i,v),0)=0.

Any non-satisfiable expression represents the zero function which is totally symmetric and compatible with any permutation. So if e is not satisfiable, then g(e)$\in\mathbf{Y}_{i,j}$. Now suppose e has a satisfying assignment $(x_1,...,x_{i-1},x_i,...,x_{j-1},x_j,...,x_n)$. Then $\mathbf{C}_i(e)$ has a satisfying assignment $(x_1,...,x_{i-1},$NOT$(x_i),...,x_{j-1},x_j,...,x_n)$, and $e+\mathbf{C}_i(e)$ has (at least) two satisfying assignments,
$(x_1,...,x_{i-1},0,...,x_{j-1},x_j,...,x_n)$ and
$(x_1,...,x_{i-1},1,...,x_{j-1},x_j,...,x_n)$.
By a similar argument, $e+\mathbf{C}_i(e)+\mathbf{C}_j(e)+\mathbf{C}_j(\mathbf{C}_i(e))$ has (at least) four satisfying assignments:
$(x_1,...,x_{i-1},0,...,x_{j-1},0,...,x_n)$,
$(x_1,...,x_{i-1},1,...,x_{j-1},0,...,x_n)$,
$(x_1,...,x_{i-1},0,...,x_{j-1},1,...,x_n)$, and
$(x_1,...,x_{i-1},1,...,x_{j-1},1,...,x_n)$.

Therefore:

eval(g(e),$(x_1,...,x_{i-1},0,...,x_{j-1},1,...,x_n)$)=
AND(eval(x_ii,$(x_1,...,x_{i-1},0,...,x_{j-1},1,...,x_n)$),1)=
AND(0,1)=0.

But

eval(g(e),$(x_1,...,x_{i-1},1,...,x_{j-1},0,...,x_n)$)=
AND(eval(x_ii,$(x_1,...,x_{i-1},1,...,x_{j-1},0,...,x_n)$),1)
AND(1,1)=1.

Thus, if e is satisfiable, g(e)$\in\mathbf{NY}_{i,j}$. $\therefore$

Since the set of expressions that represent functions not compatible with a permutation group is **NP**-complete when the group is as large as possible, and is **NP**-complete when the group is as small as possible, it is reasonable to ask whether it is *always* **NP**-complete. Theorem 7 shows that the set of expressions that represent functions not compatible with an arbitrary permutation is **NP**-complete, thus extending the **NP**-completeness result to all cyclic subgroups of S_n. A corollary to Theorem 7 establishes **NP**-completeness for all permutation groups.

THEOREM 3. Let p be an arbitrary permutation on a set of k elements. Let $\mathbf{Y}_\mathrm{p}$ be the set of expressions in **EXP** that represent functions that are compatible with p, and let $\mathbf{NY}_\mathrm{p}$ be the complement of $\mathbf{Y}_\mathrm{p}$ with respect to **EXP**. The set $\mathbf{NY}_\mathrm{p}$ is **NP**-complete.

Proof. The following algorithm shows that $\mathbf{NY}_\mathrm{p}$ is in **NP**.

```
Input expression $e$.;
Choose a boolean vector $a$;
$b$<- p($a$);
$x$<- eval($e$,$a$);
$y$<- eval($e$,$b$);
if $x \neq y$ then ACCEPT;
```

To show that $\mathbf{NY}_\mathrm{p}$ is **NP**-hard, first assume that p is represented as a product of disjoint cycles. If p contains a 2-cycle, then the proof of Theorem 2 applies. Therefore assume that the shortest cycle of p is a j-cycle, where j>2. Assume that the j-cycle 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. $g_1(e)=e+\mathbf{C}_q(e)+\mathbf{C}_r(e)+\mathbf{C}_q(\mathbf{C}_r(e))$. If e is not satisfiable, then neither is $g_1(e)$, however if e is has a satisfying assignment v, then g1(e) has (at least) four satisfying assignments, v, $\mathbf{c}_q(v)$, ... Now let g_2 be the function $g_1(\mathbf{V}_\mathrm{p}(e))$. If e is not satisfiable, then neither is $g_2(e)$, however if e has a satisfying assignment v, then $g_2(e)$ has (at least) four satisfying assignments of the form $\mathbf{T}_\mathrm{p}(v)$, $\mathbf{T}_\mathrm{p}(\mathbf{c}_q(v))$, $\mathbf{T}_\mathrm{p}(\mathbf{c}_r(v))$, $\mathbf{T}_\mathrm{p}(\mathbf{c}_q(\mathbf{c}_r(v)))$. Finally, let g be the function $g(e) = x_q(g_1(e)+g_2(e))$. From the above it is clear that $g_1(e)$ has a satisfying assignment v with x_q=1 and x_r=0 such that $\mathbf{T}_\mathrm{p}(v)$ is a satisfying assignment of $g_2(e)$. Therefore both v and $\mathbf{T}_\mathrm{p}(v)$ satisfy $g_1(e)+g_2(e)$. Note, however, that in bold $\mathbf{T}_\mathrm{p}(v)$, x_q=0. Therefore eval(g(e),v)= AND(eval(x_q,v),1)= 1 , and eval(g(e),$\mathbf{T}_\mathrm{p}(v)$)= AND(eval(x_q,$\mathbf{T}_\mathrm{p}(v)$),1)= 0. So if e is satisfiable, then g(e) is not compatible with p. On the other hand, if e is not satisfiable, then neither is g(e). A non-satisfiable expression represents the totally symmetric zero function, which is compatible with any permutation $\therefore$.

COROLLARY. Given an arbitrary permutation group G $\subseteq$ S_n, let $\mathbf{Y}_\mathrm{G}$ be the subset of expressions in **EXP** that represent functions compatible with G, and $\mathbf{NY}_\mathrm{G}$ be the complement of $\mathbf{Y}_\mathrm{G}$ with respect to **EXP**. The set $\mathbf{NY}_\mathrm{G}$ is **NP**-complete.

Proof. Modify the algorithm used in the proof of Theorem 3 to non-deterministically choose a permutation p. For some element q of G, create a function g_q, in the manner of the proof of Theorem 3. Use this function to show NP-hardness in the manner of Theorem 3.$\therefore$

4. Orbits in GF(2)$^\mathbf{n}$.

One way to determine the symmetry rule of an n-input function F would be to test every element of Sn to determine if it is compatible with F. Unfortunately, this process can be quite time consuming. The concept of an *orbit* of a subgroup can simplify the process of obtaining the symmetry rule of a gate. To begin with, the orbit of a particular element $v \in$GF(2)^n under a subgroup H$\subseteq$S$_n$ is the set of all elements of GF(2)^n onto which v is mapped by elements of H, and is written O(v,H). That is, O(v,H)={$\mathbf{T}_\mathrm{p}(v)$|p$\in$H}. Belonging to the same orbit is an equivalence relation, so every subgroup H partitions GF(2)^n into disjoint subsets. It is possible for several different subgroups to have the same partitioning, but for each partitioning, there will be one largest subgroup with that partitioning, in the sense that this subgroup will contain all others with the same partitioning.

Now, if H is the symmetry rule of a function F, and $a \in O(b,H)$, then by the definition of compatibility, $F(a)=F(b)$. Therefore, the function F can be viewed as a function defined on the orbits of H. Not surprisingly, $a \in O(b,S_n)$ if and only if a and b have the same weight. This implies that if a and b have different weights then $a \notin O(b,H)$ for all subgroups $H \subseteq S_n$.

Now, let $\mathbf{c}(v)$ be the function that complements every element of the vector v, and if $X \subseteq GF(2)^n$, let $\mathbf{c}(X)$ be the set obtained by applying the function $\mathbf{c}$ to every element of X. It is easy to show that if $p \in S_n$ and $a \in GF(2)^n$, then $\mathbf{c}(p(a))=p(\mathbf{c}(a))$. This implies that $\mathbf{c}$ maps orbits to orbits, in other words, $\mathbf{c}(O(a,H))=O(\mathbf{c}(a),H)$.

Given an n-input function F whose truth table is known, $GF(2)^n$ can be partitioned as follows. First, the elements of $GF(2)^n$ are partitioned according to weight. Next each set X from the first partitioning is split into two sets $X_1=\{x|F(x)=1 \ \& \ x \in X\}$ and $X_0=\{x|F(x)=0 \ \& \ x \in X\}$. (One of X_1 or X_0 may be empty.) Finally, $\mathbf{c}(Y)$ is computed for each subset Y in the second partition. If $\mathbf{c}(Y)$ is a proper subset of some set Z of the second partition, then Z is split into two sets $\mathbf{c}(Y)$ and $Z-\mathbf{c}(Y)$. This last process continues until no further partitioning is possible. The symmetry rule of F.is the largest subgroup of S_n that preserves the third partition. With respect to a boolean expression e, this process is exponentially bounded, because in the worst case the length of the truth table of a function represented by an expression e, is an exponential function of the length of e. However, the **NP**-Completeness results of Section x, suggest that this is the best we can do for an arbitrary expression e.

5. An Analysis of 2-, 3-, and 4-Input Functions.

When computing the symmetries of different n-input gates, it quickly becomes obvious that some symmetry rules are fundamentally different from one another, and some are "the same, but applied to different inputs." This distinction is more than intuitive and can be used to simplify the categorization of symmetry rules. Mathematically, the symmetry rules that are "the same but applied to different inputs," are conjugates of one another. Formally, a conjugate of a permutation $p \in S_n$ is the permutation $q^{-1}pq$ where q is any element of S_n. If H is a subgroup of S_n, and p is any element of S_n, then the conjugate of H by p, written H^p, is the set $(p^{-1}qp \mid q \in H\}$. It is easy to show that H^p is also a subgroup of S_n. In fact if H is the symmetry rule of an n-input function F, then H^p is the symmetry rule of Fp. (Recall that the function Fp is the definition of a rearrangement of a function F.*)*

The relationship of being conjugate to one another is an equivalence relation on the subgroups of S_n. Because of this, it is possible to enumerate symmetry rules for a particular n by listing one example from each conjugacy class. To clarify the concept of a conjugacy class, consider the example of a 4-input function which is partially symmetric with respect to its first two inputs. The symmetry-rule of this function would be $H=\{I, (1\ 2)\}$. The conjugates of H are all of the form $K=\{I, (i\ j)\}$, where $1 \le i < j \le 4$, that is the symmetry rules that specify partial symmetry with respect to inputs i and j. Conceptually, the permutation of the form $p^{-1}qp$ is constructed so that p moves inputs i and j to positions 1 and 2, then p, an element of H, is applied to inputs 1 and 2, and then p^{-1} is used to move inputs 1 and 2 back to positions i and j.

In the remainder of this section we enumerate the various symmetry rules for 2, 3, and 4-input boolean functions. First, 2-input functions cannot be partially symmetric or G-symmetric, but they can be totally symmetric. The following theorem characterizes all symmetric 2-input functions.

THEOREM 4. A 2-input gate G is totally symmetric if and only if $G(0,1)=G(1,0)$.

There are 16 2-input functions, and 8 totally symmetric 2-input functions. Thus half of all 2-input functions are totally symmetric. An example of a totally symmetric 2-input function is ab while an example of a non-symmetric 2-input function is *ab'*. Gates with three inputs are also simple to categorize. There are only three forms of symmetry.

THEOREM 5. A 3-input gate is either non-symmetric, totally symmetric, or partially symmetric with a symmetry rule conjugate to the subgroup {I, (1 2)}.

There are 256 3-input functions, 16 of which are totally symmetric. Since {I, (1 2)} has six distinct orbits in $GF(2)^3$, there are 64 functions compatible with {I, (1 2)}. However, 16 of these are totally symmetric, so there are 48 functions with symmetry rule {I, (1 2)}. There are two conjugates of the subgroup {I, (1 2)}, and for each there are 48 functions that possess it as a symmetry-rule. Therefore, there are 144 partially symmetric and 96 non-symmetric 3-input gates. Examples of the three types of functions are abc, ab+c, and a'b+c.

The symmetries of 4-input gates are considerably more interesting than those for 2- and 3-input gates, as the following theorem shows.

THEOREM 6. A 4- input gate is either non-symmetric, totally symmetric, partially symmetric with a symmetry rule conjugate to {I, (1 2)} or {I, (1 2), (1 3), (2 3), (1 2 3), (1 3 2)}, or G-symmetric with a symmetry rule conjugate to one of the following three subgroups
{I, (1 2)(3 4)},
{I, (1 2), (3,4), (1 2)(3 4)}, or
{I, (1 2), (3 4), (1 2)(3 4), (1 3)(2 4), (1 4)(2 3), (1 4 2 3), (1 3 2 4)}.

The function abcd is an example of a totally-symmetric function, while the function ab'+c+d' is an example of a non-symmetric function. The functions ab+cd' and abc+d are examples of the two types of partial symmetry. The function ac'+bd' is an example of the first type of G-symmetry, while ab+c'd' is an example of the second. The third type of G-symmetry is exemplified by the function ab+cd.

6. Opportunities for Future Research.

One opportunity for future research is the complexity of the problem of determining whether a function is partially symmetric or G-symmetric, without necessarily determining the symmetry rule. In Theorem 3, it is assumed that the permutation or the symmetry rule has already been given. The question of whether a function is partially symmetric with respect to *any* inputs seems more difficult to answer than the question of partial symmetry with respect to a specific set of inputs, however the complexity of this problem is currently unknown.

Another avenue for future research is the investigation of non-canonical modulo-2 representations of S_n. As mentioned in section 2, the mapping from S_n to the set of linear transformations on the vector space $GF(2)^n$ defined by $\mathbf{M}(p)=\mathbf{T}_p$ is an isomorphism, called the canonical modulo-2 representation of S_n. It is possible to consider a general modulo-2 representation of S_n, which is a homomorphism **K** from S_n to the set of linear transformations on $GF(2)^n$. Recall that if **K** is an isomorphism, then **K** is said to be a *faithful* representation of S_n. The faithful representations of S_n are quite easy to obtain. If **L** is a non-singular linear transformation from $GF(2)^n$ to itself, let $\mathbf{M_L}(p)=\mathbf{L}^{-1}\mathbf{M}(p)\mathbf{L}$. Any faithful representation of S_n must be of the form $\mathbf{M_L}$ for some non-singular linear transformation **L**.

Let **K** be a representation of S_n. If $p \in S_n$, $q=\mathbf{K}(p)$, and F is an *n*-input boolean function such that Fq=F then p is **K**-compatible with F. Using this relationship, it is possible to define symmetry with respect to a general representation **K**, in the same way as symmetry was defined for the canonical representation **M**. The terms "totally **K**-symmetric," "partially **K**-symmetric," "**K**-G-Symmetric," and "non-**K**-symmetric" are defined in the obvious way.

For a representation $\mathbf{M_L}$ the transformation **L** can be used for logic simplification by observing that for any *n*-input boolean function F, $F\mathbf{L}^{-1}\mathbf{L}=F$, and that in many cases, $F\mathbf{L}^{-1}$ is simpler to implement than F. Now, if it is necessary to implement several functions F_1, ..., F_k on the same set of inputs, it may be advantageous to first transform the input vector using **L** and then implement $F_1\mathbf{L}^{-1}$, ... , $F_k\mathbf{L}^{-1}$. As an example, let F be the 4-input even-parity function. The 2-level boolean equation for this function is a'b'c'd'+abc'd'+ab'cd'+ab'c'd+a'bcd'+a'bc'd+a'b'cd+abcd. Let **L** be the linear transformation defined by the following matrix.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

The boolean equation for $F\mathbf{L}^{-1}$ is a'c'+bd. Of course to implement $F\mathbf{L}^{-1}$, one must first implement **L**. The implementation of **L** requires 3 2-input exclusive OR gates. Although the reduction in the complexity of the implementation of F will be negated by the necessity of implementing **L**, if several functions can be simplified using a single linear transformation **L**, the savings may be considerable. Furthermore if $\mathbf{K}=\mathbf{L}^{-1}\mathbf{ML}$, and F is totally **K**-symmetric (partially **K**-symmetric, **K**-G-symmetric) then $F\mathbf{L}^{-1}$ is totally symmetric (partially symmetric, G-symmetric).

The concept of non-canonical modulo-2 representations of S_n is an intriguing idea that merits more research.

7. Conclusion.

It has been shown that the symmetry of an *n*-input boolean function can be expressed as a subgroup of S_n, the symmetric group of order *n*. Given an arbitrary subgroup G of S_n, the problem of determining whether the function represented by an expression *e* possesses G as a symmetry rule is **NP**-complete, regardless of the structure of G. The paper presents a method for determining the symmetry rule of a boolean function. This technique requires that one enumerate the truth table of the function, so it is exponentially bounded in the number of function inputs. Furthermore, the method requires a knowledge of the orbits in $GF(2)^n$ of every subgroup of S_n. These restrictions imply that the technique is limited to a small number of inputs, (probably less than 10,) but this is precisely the type of function for which determination of the symmetry rule is essential.

An analysis of 2, 3, and 4 input gates is presented, along with examples of functions that exhibit each type of symmetry. Finally opportunities for future research are presented. For the most part, these are based on the mathematical concepts introduced in Section 2. In particular, non-canonical modulo-2 representations of S_n appear to merit more investigation.

REFERENCES

1. C. E. Shannon, "A Symbolic Analysis of Relay and Switching Circuits," *AIEE Transactions*, Vol. 57, pp. 713-723, 1938.

2. C. E. Shannon, "The Synthesis of Two-Terminal Switching Circuits," *Bell Systems Technical Journal*, Vol. 28, pp. 59-98, Jan., 1948.

3. A, Salomaa, *Formal Languages*,Academic Press, New York, 1973.

4. A. Mukhopadhyay, "Detection of total or Partial Symmetry of a Switching Function with the Use of Decomposition Charts," *IEEE Transactions on Electronic Computers,* Vol. EC-12, pp. 553-557, Oct., 1983.

5. W. H. Kautz, "The Realization of Symmetric Switching Functions With Linear-Input Logical Elements," *IRE Transactions on Electronic Computers*, Vol. EC-10, pp. 371-378, Sept., 1961.

6. S. S. Yau and Y. S. Tang, "Transformation of an Arbitrary Switching Function to a Totally Symmetric Function," *IEEE Transactions on Computers*, Vol. C-20, pp. 1606-1609, Dec., 1971.

7. B. Dahlberg, "On Symmetric Functions With Redundant Variables - Weighted Functions," *IEEE Transactions on Computers*, Vol. C-22, pp. 450-458, May, 1983.

8. S. A. Cook, "The Complexity of Theorem Proving Procedures," in *Proc. Third Annual ACM Symp. on Theory of Computing*, 1971, pp. 151-158.

9. R. M. Karp, "Reducibility Among Combinatorial Problems," in *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, Eds.,m New York: Plenum, 1972, pp. 85-103.

10. M. R. Garey, D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, New York: W. H. Freeman and Co., 1979.

11. Brady, H. "An Approach to Topological Pin Assignment," *IEEE Transactions on Computer Aided Design*, Vol. CAD-3, pp. 250-255, Jul. 1984.

12. P. M. Maurer, A. D. Schapira "A Logic-to-Logic Comparator for VLSI Layout Verification", *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, Vol. 7, No. 8, pp.897-907, Aug 1988.

13. D. J. S. Robinson, *A Course in the Theory of Groups*, New York: Springer-Verlag, 1982.