Chapter 4

Limits of Using Signatures

The method to handle the permutation equivalence problem introduced in the previous chapter is not complete. There is *no* signature function which can uniquely identify *all* the variables of the investigated benchmark sets. Comparing the most successful signature functions, we observed that those benchmarks that have variables that could not be uniquely identified are always the same over the different signatures. In other words, there is a nearly constant set of benchmarks for which signatures could not help to solve the permutation problem. Unfortunately, there are not just 2 or 3 variables of those benchmarks that are not uniquely identified, but about 15 and more, such that the number of possible correspondences is still large. Furthermore, this seems to be independent of the method used to solve that problem: in [29, 31], a totally different method has been used, based on another data structure – circuits described on the gate level. However, even here the same group of benchmarks causes problems [31]. Another observation is that those benchmarks with non-uniquely identified variables are not the benchmarks with the most number of inputs. From a statistical point of view, we can conjecture that the quality of the used signatures is not the problem for practical applications: we did not find a relationship between the number of input variables of a function and the ability of the signatures to distinguish between all these inputs. Hence, it seems a likely supposition that the variables that cannot be distinguished by the signatures have special properties that make it *impossible* to distinguish between them for the permutation independent comparison of two Boolean functions.

In this chapter, we discuss a property of input variables that make it impossible to distinguish between variables with aliasing with the help of signatures. This property is $\mathcal{G}$–symmetry. In **Section 4.1**, we introduce $\mathcal{G}$–symmetry and explain why $\mathcal{G}$–symmetry avoids a unique identification of input variables with the help of signatures. In the **Sections 4.2–4.4** , we discuss some special kinds of $\mathcal{G}$–symmetry that often appear in practice. In **Section 4.5** we discuss our experimental results on the 8% of benchmarks with aliasing. Finally, in **Section 4.6** the variety of $\mathcal{G}$–symmetry in general is discussed. Parts of these results are presented in [26].

4.1 The Property of $\mathcal{G}$–Symmetry

$\mathcal{G}$–symmetry can be defined as follows:

Definition 4.1 *Consider a group $\mathcal{G} \subseteq \mathcal{P}_n$ of permutations. A Boolean function $f \in \mathcal{B}_{n,1}$ is* **$\mathcal{G}$–symmetric** *if f keeps invariant under all permutations π in $\mathcal{G}$.*

$\mathcal{G}$–symmetry was defined similar by Hotz in 1974 [17]. The simplest example for $\mathcal{G}$–symmetry is a Boolean function f that is symmetric in all input variables. Here, $\mathcal{G}$ is equal to the permutation group $\mathcal{P}_n$, and we say, f is $\mathcal{P}_n$ – symmetric.

By definition, the group of permutations $\mathcal{G}$ may also be the group which contains the identity $\mathbf{1}$ only. So we can consider those Boolean functions with no $\mathcal{G}$–symmetry as functions that are $\mathcal{G}$–symmetric with respect to the set $\mathcal{G} = \{\mathbf{1}\}$. In other words, we can formulate the following property:

Property 4.1 *For each Boolean function $f \in \mathcal{B}_{n,1}$ the set $\mathcal{G} \subseteq \mathcal{P}_n$ of permutations such that f is symmetric with respect to all permutations in $\mathcal{G}$ forms a group.*

Proof: Let $\pi_1 \in \mathcal{G}$ and $\pi_2 \in \mathcal{G}$ be two permutations of $\mathcal{G}$. We need to prove that the permutation $\pi_1 \circ \pi_2 \in \mathcal{G}$:

$$f \circ (\pi_1 \circ \pi_2) = (f \circ \pi_1) \circ \pi_2 = f \circ \pi_2 = f$$

In other words, the permutation $\pi_1 \circ \pi_2$ keeps the function f invariant, thus $\pi_1 \circ \pi_2 \in \mathcal{G}$. □

In order to understand the significance of $\mathcal{G}$–symmetry for the permutation equivalence problem, let us consider the relation $R_{\mathcal{G}}$ on the set of input variables, $X = [x_1, x_2, \ldots, x_n]$, which is defined by a group of permutations $\mathcal{G} \subseteq \mathcal{P}_n$. Let $x_i \in X$ and $x_j \in X$ be two input variables:

$$x_i R_{\mathcal{G}} x_j \Longleftrightarrow \exists \pi \in \mathcal{G} : \pi(x_i) = x_j.$$

Property 4.2 *The relation $R_{\mathcal{G}}$ is an equivalence relation.*

Proof: This follows immediately from the property that $\mathcal{G}$ is a group.□

Let us denote with $\mathcal{A} = \{A_1, A_2, \ldots, A_k\}$ the partition of the input variables in X which corresponds to a relation $R_{\mathcal{G}}$. This partition is well–defined with respect to a $\mathcal{G}$–symmetry, and it follows from Property 4.1 and Property 4.2:

Fact 4.1 *For each Boolean function $f \in \mathcal{B}_{n,1}$ and its set of input variables X there is a well–defined partition $\mathcal{A}$ of the input variables of f.*

Often it is enough to consider this partition $\mathcal{A}$ as an *unordered* set of subsets of the inputs. However, sometimes it is necessary to look at it as an *ordered* set.

Property 4.3 *Given a Boolean function $f \in \mathcal{B}_{n,1}$ and the well–defined partition of its input variables, $\mathcal{A} = \{A_1, A_2, \ldots, A_k\}$, there is a well–defined and* ordered *partition of the input variables with respect to f,*

$$\mathcal{A}^f = \{A_1^f, A_2^f, \ldots, A_k^f\},$$

which is constructed as follows:

1. *$\mathcal{A}^f$ considered as* unordered *set of subsets is equal to partition $\mathcal{A}$.*
2. *The ordering step works as follows:*
   (a) *At first, we order the subsets by their size, such that $|A_i^f| \leq |A_{i+1}^f|$ for all $i = 1, 2, \ldots, k$.*
   (b) *If some subsets have the same size, then we use the elements of them to establish an ordering, such that the result looks as follows: if $|A_i^f| = |A_{i+1}^f|$ for any $i \in \{1, 2, \ldots, k\}$, then $j < l$ for $x_j \in A_i^f$ with $j = \min\{h : x_h \in A_i^f\}$ and $x_l \in A_{i+1}^f$ with $l = \min\{h : x_h \in A_{i+1}^f\}$.*

Proof: The so constructed partition $\mathcal{A}^f$ of the input variables of a Boolean function f contains the same elements as the original partition $\mathcal{A}$. So it is well–defined as an *unordered* set of subsets. Furthermore, the ordering process is well–defined: Step 1.a orders those subsets uniquely that have different sizes. Step 1.b orders the subsets with the same size uniquely. Note, that this second step depends on the order of the input variables of the actual function f. In other words, the order of the subsets in partition $\mathcal{A}^f$ is just unique with respect to the Boolean function f which has been considered. □

Now we consider signatures again. What can we say about the relationship between signatures and $\mathcal{G}$–symmetry? There are two properties that complete our picture about signatures. The first property is very important and can be formulated without further considerations:

Property 4.4 *Let $\mathcal{G} \subseteq P_n$ be the group of permutations of X which constructs partition $\mathcal{A}$. Consider any element A_l of $\mathcal{A}$, any $\mathcal{G}$–symmetric Boolean function f, and any signature function $s \in \mathcal{S}_n$. For any $x_i, x_j \in A_l : s(f, x_i) = s(f, x_j)$.*

Proof: For all $x_i, x_j \in A_l$ there is a permutation $\pi \in \mathcal{G}$ such that $\pi(x_i) = x_j$ (see definition of partition $\mathcal{A}$). Consider any $x_i, x_j \in A_l$, a permutation $\pi \in \mathcal{G}$ such that $\pi(x_i) = x_j$, and any signature function s. As defined, $s(f, x_i) = s(f \circ \pi, \pi(x_i))$. This is equal to $s(f, \pi(x_i))$ since f is $\mathcal{G}$–symmetric. Thus, $s(f, x_i) = s(f, x_j)$ for any $x_i, x_j \in A_l$. □

This result now gives us a possible explanation for the trouble several benchmarks have with the signature approach: it is possible that the benchmarks with aliasing include $\mathcal{G}$–symmetric functions.

Then *there is no unique description by signatures* for the input variables of these functions. Thus, it is futile to try and distinguish all the variables with additional signatures. The immediate follow-up question is then: what do we do in this case? Our response to this is that we do not really need to uniquely identify the variables in most cases. Perhaps we can achieve our end goal of establishing permutation independent function equivalence by identifying the variables involved in the $\mathcal{G}$–symmetry and exploring the exact nature of the $\mathcal{G}$–symmetry. This is further explored in the next sections.

Before we discuss it, let us develop the second property of signatures with respect to $\mathcal{G}$–symmetry. For this, we need to think about a universal signature function.

Definition 4.2 *A* **universal signature function** $u : \mathcal{B}_{n,1} \times X \longrightarrow U$ *is a signature function which has the following property. Given a Boolean function* $f \in \mathcal{B}_{n,1}$ *and the partition of its input variables,* $\mathcal{A} = \{A_1, A_2, \ldots, A_k\}$*:*

$$\forall x_i, x_j \in X : \text{ if } x_i \in A_l, x_j \in A_h \text{ with } l \neq h \Longrightarrow u(f, x_i) \neq u(f, x_j).$$

If two input variables x_i and x_j are not in the same subset of partition $\mathcal{A}$ of a Boolean function $f \in \mathcal{B}_{n,1}$, then the universal signature function must provide different signatures for these two variables.

Such a universal signature function has the property to dominate any other signature function $s \in \mathcal{S}_n$ in the following sense:

$$\forall f \in \mathcal{B}_{n,1} \forall x_i, x_j \in X : \quad \text{if } \exists s \in \mathcal{S}_n : s(f, x_i) \neq s(f, x_j) \Rightarrow u(f, x_i) \neq u(f, x_j).$$

In other words, if there is any signature function which can distinguish between two input variables of a Boolean function, then the universal signatures of these two variables must be able to distinguish between them as well. This property follows immediately from Property 4.4 and the definition of a universal signature function. If two input variables x_i and x_j of a Boolean function $f \in \mathcal{B}_{n,1}$ have different signatures for any signature function $s \in \mathcal{S}_n$, then they have to be in different subsets of the partition $\mathcal{A}$ of the input variables of f (see Property 4.4). That is why the universal signatures of these two variables must be different as well (see Definition 4.2). In this sense, we can say that a universal signature function is the strongest signature function which can be constructed.

Theorem 4.1 *There is a universal signature function* $\mathcal{U} : \mathcal{B}_{n,1} \times X \longrightarrow U$.

Proof: At first we construct a candidate for a universal signature function. Then we show, that this candidate is a signature function and that it is universal.

1. Construction:

   Let us consider a Boolean function $f \in \mathcal{B}_{n,1}$. From Fact 4.1, it follows that there is a well–defined partition of the input variables of function f. Let us construct this partition:

   $$\mathcal{A} = \{A_1, A_2, \ldots, A_k\} = \{\{x_1^1, \ldots, x_{l_1}^1\}, \{x_1^2, \ldots, x_{l_2}^2\}, \ldots, \{x_1^k, \ldots, x_{l_k}^k\}\}.$$

   Now, we select the lexicographical smallest of all functions that can be constructed by permuting the input variables of f:

   $$f_{\min} = \min\{g \in \mathcal{B}_{n,1} : g = f \circ \pi \text{ with } \pi \in \mathcal{P}_n\}.$$

   This function, $f_{\min}$ has a unique partition $\mathcal{A}'$ of its input variables as well.

   What do we know about the relationship between function f and function $f_{\min}$ and the partition of their input variables? We know, that $f_{\min} = f \circ \hat{\pi}$ for a permutation $\hat{\pi} \in \mathcal{P}_n$. In other words, these two functions are permutation equivalent. From this fact, it follows that the partition $\mathcal{A}'$ has the same structure as partition $\mathcal{A}$:

   $$\mathcal{A}' = \{A_1', A_2', \ldots, A_k'\} = \{\{y_1^1, \ldots, y_{l_1}^1\}, \{y_1^2, \ldots, y_{l_2}^2\}, \ldots, \{y_1^k, \ldots, y_{l_k}^k\}\}.$$

   Furthermore, there is a 1–1–mapping $\Phi : \mathcal{A} \longrightarrow \mathcal{A}'$ which maps the subsets of partition $\mathcal{A}$ to the subsets of partition $\mathcal{A}'$. Φ depends on the permutation $\hat{\pi}$ which we use to construct $f_{\min}$ and is defined as follows:

   $$\Phi(A_i) = \hat{\pi}(A_i) = A_j'$$

   for all $i = 1, \ldots, k$ and the corresponding $j \in \{1, 2, \ldots, k\}$.

   Furthermore, there is a well–defined and ordered partition of the input variables of $f_{\min}$:

   $$\mathcal{A}^{f_{\min}} = \{A_1^{f_{\min}}, A_2^{f_{\min}}, \ldots, A_k^{f_{\min}}\},$$

   which is constructed by ordering the subsets of partition A' (see Property 4.3). With the help of this partition we are able to construct our candidate for a universal signature function. Let $x_i \in X$ be an input variable of f. Then,

   $$\mathcal{U}(f, x_i) = j \text{ with } \hat{\pi}(x_i) \in A_j^{f_{\min}},$$

   where $\hat{\pi}$ is a permutation of $\mathcal{P}_n$ which constructs the lexicographical smallest function, $f_{\min} = f \circ \hat{\pi}$.

2. Proof of correctness:

   We need to prove that $\mathcal{U}$ is a universal signature function. This is done in three parts:

   (a) $\mathcal{U}$ is a well–defined mapping from $\mathcal{B}_{n,1} \times X$ into an ordered set $(U, \leq)$.

   Proof: $\mathcal{U}$ maps the set $\mathcal{B}_{n,1}$ of all Boolean functions and their input variables X into the set of indices of the subsets of the ordered partition $\mathcal{A}^{f_{\min}}$, which is equal to the set of integers from $\{1, 2, \ldots, k\}$ (k is the number of subsets in partition $\mathcal{A}$). In other words, $\mathcal{U}$ maps into the set of non–negative integers. This is an ordered set.

   In order to prove that $\mathcal{U}$ is a *well–defined* mapping, we need to consider the case that there are more than one permutations $\hat{\pi} \in \mathcal{P}_n$ that construct the lexicographical smallest function $f_{\min} = f \circ \hat{\pi}$.

   Suppose, there are two of those permutations, π_1 and π_2. What can we say about these two permutations? We know that

   $$f_{\min} = f \circ \pi_1 = f \circ \pi_2.$$

   So it follows that $f = f \circ \pi_1 \circ \pi_2^{-1}$, i.e., the permutation $\pi_1 \circ \pi_2^{-1}$ keeps the Boolean function f invariant. Now consider any input variable x_i of function f. By definition of partition $\mathcal{A}$, it follows that the two variables x_i and $\pi_2^{-1}(\pi_1(x_i))$ are in the same subset of $\mathcal{A}$.

   Let us see what happens on applying the permutation π_2 to these two variables. Remember that this permutation can be used to construct partition $\mathcal{A}'$ from partition $\mathcal{A}$. We immediately deduce that the variables $\pi_2(x_i)$ and $\pi_1(x_i)$ are in the same subset of partition $\mathcal{A}'$. However, these are the two variables that x_i is mapped on using the permutation π_1 and π_2, respectively. So, we finally know that the input variable x_i is mapped into the same subset of partition $\mathcal{A}'$, independent of the permutation which we use to construct this partition. □

   (b) $\mathcal{U}$ is a signature function.

   Proof: Let us consider the function $f_{\min}$. This function is information about the Boolean function f which is independent of any permutation of the input variables of f since we use *all* $\pi \in \mathcal{P}_n$ for its construction. Also the partition $\mathcal{A}^{f_{\min}}$ is permutation independent information for f. From this, it follows that the information we get for an input variable x_i using $\mathcal{U}$ is independent of any permutation of the inputs. □

   (c) $\mathcal{U}$ is universal.

   Proof: Let us consider the mapping $\Phi : \mathcal{A} \longrightarrow \mathcal{A}'$ again, which maps the subsets of the partition of the inputs of function f to those subsets of the partition of the inputs of function $f_{\min}$. From the construction of this mapping, it follows that if two input variables x_i and x_j of function f are in different subsets of partition $\mathcal{A}$, then $\hat{\pi}(x_i)$ and $\hat{\pi}(x_j)$ are in different subsets of the partition $\mathcal{A}'$ of the lexicographical smallest function $f_{\min}$ too. Remember, the permutation $\hat{\pi}$ is a permutation of the input variables of function f, which constructs the lexicographical smallest of all those functions: $f_{\min} = f \circ \hat{\pi}$.

Now, if we order $\mathcal{A}'$ as proposed in Property 4.3, then we get the well–defined and ordered partition $\mathcal{A}^{f_{\min}}$ for the function $f_{\min}$, and it follows for any Boolean function $f \in \mathcal{B}_{n,1}$ and for all input variables $x_i, x_j \in X$:

$$x_i \in A_l \text{ and } x_j \in A_h \text{ with } l \neq h \text{ iff } \hat{\pi}(x_i) \in A_p^{f_{\min}} \text{ and } \hat{\pi}(x_j) \in A_q^{f_{\min}} \text{ with } p \neq q.$$

This is the same as: for all $x_i, x_j \in X$:

$$x_i \in A_l \text{ and } x_j \in A_h \text{ with } l \neq h \text{ iff } \mathcal{U}(f, x_i) \neq \mathcal{U}(f, x_j).$$

In other words, the signature function $\mathcal{U}$ is universal. □ □

The signature function $\mathcal{U}$ demonstrates the property of a universal signature function to dominate all other signature functions very well. The partition $\mathcal{A}$, which is the basic component for the construction of $\mathcal{U}$, is nothing else but the partition of the input variables which we try to construct with the help of our practical signatures. Moreover, this partition is sorted independent of permutation by those signatures. Now, if the universal signature for an input variable is the index of such a set of the partition, then one thing is obvious: if there is a practical signature which is different for two variables, then these two variables will be in two different subsets of the partition. Also the universal signatures of these two variables will be different.

Why does it help to know that there is a universal signature function? It cannot be used in practice since its construction would be computationally too intensive. However, it is useful to have such a universal signature function for theoretical investigations.

In Section 3.2.3 we introduced a method for analytically comparing the effectiveness of a signature function [21]. Remember, given a signature function $s : \mathcal{B}_n \times X \longrightarrow U$, a measure of the effectiveness of a signature function is the cardinality of the co–domain of s, which is $|U|$. However, as we can see here, a small co–domain does not automatically imply that the signature function is not efficient. It depends on all properties of a signature function. In the case of the universal signature function, it is: $|U| \leq n$, where n is the number of input variables of the actual function. However, as we have proven, the universal signature function is the most powerful signature function at all.

At the moment, we are interested in signatures in relationship to $\mathcal{G}$–symmetry. With the help of the universal signature function U we can prove, that if any two input variables of a Boolean function have the same signature $\mathcal{U}$, then there is a permutation $\pi \in \mathcal{P}_n \setminus \mathbf{1}$ of the input variables of this function, such that $f = f \circ \pi$. This permutation π of the input variables keeps the function f invariant, i.e., the function f is $\mathcal{G}$–symmetric with respect to a group of permutations $\mathcal{G} \neq \{\mathbf{1}\}$. This property can be formulated as follows.

Property 4.5 *Let f be any Boolean function of $\mathcal{B}_{n,1}$.*
Let $\mathcal{A} = \{A_1, A_2, \ldots, A_k\}$ be the partition of the input variables of f.

$$\forall x_i, x_j \in X : \mathcal{U}(f, x_i) = \mathcal{U}(f, x_j) \Longrightarrow x_i, x_j \in A_l \quad for \quad some \quad l \in 1, 2, \ldots, k.$$

Proof: This follows immediately from the definition of $\mathcal{U}$. □

The property ensures that we do not need to think about other problems with signatures identifying input variables for permutation independent function equivalence. It demonstrates that $\mathcal{G}$–symmetry is indeed the only handicap for signatures to be able to uniquely identify an input variable of a Boolean function independent of permutation. Thus, we now need to focus on $\mathcal{G}$–symmetries.

Specifically, we now discuss some special kinds of $\mathcal{G}$–symmetry which often appear in practice. Of course, this cannot be a complete enumeration of possible $\mathcal{G}$–symmetries. We discovered these cases in our quest to understand why signatures were proving to be inadequate for permutation independent Boolean comparison in some cases.

4.2 Partial Symmetries

At first, let us consider the most common case. A Boolean function $f \in \mathcal{B}_{n,1}$ is symmetric with respect to a subset of input variables $\hat{X} \subseteq X$ if f is invariant under all permutations $\pi \in \mathcal{P}_k$ of the input variables in $\hat{X} \subseteq X$, where k is the number of input variables in $\hat{X}$. We say, that f is *partial symmetric* with respect to $\hat{X}$. Furthermore, the set $\hat{X}$ is a *maximal symmetry group* (a maximal set of symmetric variables) of f if f is symmetric with respect to $\hat{X}$, and there is no variable $x_i \notin \hat{X}$ such that f is symmetric with respect to $\hat{X} \cup x_i$ as well. This is the simplest kind of $\mathcal{G}$–symmetry, and it is well–known.

In the partition $\mathcal{A}$ of the inputs of a partial symmetric Boolean function f all input variables of the subset $\hat{X}$ are in one subset.

However, these symmetries are easy to detect and to handle.

Fact 4.2 *A Boolean function f is partial symmetric with respect to the variables x_i and x_j iff $f_{x_i \bar{x}_j} = f_{\bar{x}_i x_j}$ [28].*

With the help of this fact we are able to test partial symmetry, since it is an equivalence relation [28]. Furthermore, different methods to improve this basic symmetry test have been developed. We use the methods introduced in [28], while one of the methods, introduced there, is to use simple signatures as proposed in Section 3.2.3.1.

After the detection of partial symmetries we are done, since each correspondence between symmetric variables of two Boolean functions is fine for the purpose to test permutation equivalence.

Finally, let us explain an algorithm to identify the input variables by signatures that considers these symmtries. With our knowledge about symmetric variables, we see that one of the first steps of this procedure is to determine all maximal groups of pairwise symmetric variables. As we know, this can be done fast, and the advantage is that the signature computations can be restricted to one representative of each maximal symmetry group. In order to combine the used methods of symmetry detection [28] with the different signature functions as introduced in Section 3.2.3, we determine the maximal symmetry groups after applying the cofactor satisfy count signature function (see Section 3.2.3 and Section 3.3).

4.3 Hierarchical Symmetries

Investigations on our benchmark set have shown that for several examples the reason for the existence of aliasing groups after computation of all signatures introduced in Section 3.2.3 is the following kind of symmetry.

Definition 4.3 *Let $f \in \mathcal{B}_{n,1}$ be a Boolean function with the input variables $X = [x_1, x_2, \ldots, x_n]$. Let $X_1, X_2 \subset X$ be two subsets of X.*

X_1 and X_2 are **hierarchical symmetric (h–symmetric)** *iff*

1. $|X_1| = |X_2| > 1$
2. *X_1 and X_2 are maximal symmetry groups of f. (see Section 4.2)*
3. *f is $H(X_1, X_2)$–symmetric, where $H(X_1, X_2)$ is the subgroup of the permutation group $\mathcal{P}_n$ generated by the following set of permutations:*

$$\{\pi \in \mathcal{P}_n | \pi(X_1) = X_2 \text{ and } \pi(X_2) = X_1\}.$$

   I.e., f keeps invariant under any exchanging of the variables of X_1 with those of X_2.

A group of subsets of X, $\{X_1, X_2, \ldots, X_k\}$ is **h-symmetric** *iff:*

$$\forall i, j \in \{1, 2, \ldots, k\} : X_i \text{ is h–symmetric to } X_j.$$

Let us consider the following example:

Example 4.1 $f = \overline{(x_1 + x_2)} + \overline{(x_3 + x_4)} + \overline{(x_5 + x_6)}$

Here, $\{x_1, x_2\}$, $\{x_3, x_4\}$ and $\{x_5, x_6\}$ are pairs of partial symmetric variables, but there is no partial symmetry between two variables of different pairs. However, it is easy to see, that exchanging any two of these three pairs keeps the function f invariant. This simple example illustrates h–symmetry.

Property 4.6 *H–symmetry is an equivalence relation on the partition of the set of input variables of a Boolean function f in its maximal symmetry groups.*

Proof: The symmetry and reflexivity is obviously to see. For transitivity we have to show the following. Let X_1, X_2, and X_3 be three disjoint sets of symmetric variables. If X_1 is h–symmetric to X_2 and X_2 is h–symmetric to X_3, then X_1 and X_3 are h–symmetric as well. Therefore, let us go through all points of the definition of h–symmetry:

1. $|X_1| = |X_2| = |X_3| > 1$
2. true by assumption
3. We know, that exchanging X_1 with X_2 as well as exchanging X_2 and X_3 does not change the function f. So, let us do the following. First, exchange the variables of X_1 with those of X_2. After that, exchange the variables of X_2 with those of X_3. The resulting function is f, and what we have done is to exchange the variables of X_1 with those of X_3 using the variables of X_2. This implies that X_1 with X_3 does not change function f. □

The definition of h–symmetry indicates that this is a special kind of $\mathcal{G}$–symmetry. Let us examine the partition $\mathcal{A}$ of the input variables constructed by h–symmetry.

Property 4.7 *If two subsets of input variables, X_1 and X_2 are h–symmetric, then all input variables of X_1 and X_2 are in one element A_i of partition $\mathcal{A}$.*

Proof: This follows from the definition of partition $\mathcal{A}$. □

Thus, from Property 4.4, we know that all of these variables have to have the same signatures, i.e., they form an aliasing group. In other words, there is no way to distinguish between them via signatures.

Luckily, there is a solution for this which is based upon our handling of partial symmetric variables. To understand this, let us consider the algorithm to identify the input variables by signatures. Here, we first determine all maximal groups of pairwise symmetric variables (see previous section). In this way, pairwise symmetric variables are kept together in aliasing groups. Now, let us consider two h–symmetric subsets, X_1 and X_2, of input variables of function f again. As we know, they form an aliasing group: $\{X_1 \cup X_2\}$. A correspondence between these variables and the variables of an aliasing group of any other function g is possible if the variables of the other group have the same signatures, and this aliasing group has the same structure, i.e., $\{Y_1 \cup Y_2\}$ with Y_1 and Y_2 are maximal symmetry groups and $|X_1| = |Y_1|$. Then there are two possible correspondences between these groups: $(X_1 \leftrightarrow Y_1, X_2 \leftrightarrow Y_2)$ as well as $(X_1 \leftrightarrow Y_2, X_2 \leftrightarrow Y_1)$. Because of h–symmetry both of these correspondences are acceptable for our purpose. In other words, our remaining task in terms of h–symmetry is to *detect* this kind of symmetry. That is sufficient to decide that no further work needs to be done with these aliasing groups in order to solve the permutation problem, P_π.

So let us try and see what we have to do. First, we want to answer the following question. Let f be a Boolean function with n input variables, $X = [x_1, x_2, \ldots, x_n]$, and X_1 and X_2 be two disjoint subsets of the set of inputs, $X_1, X_2 \subset X$. When is it possible to exchange X_1 and X_2 in the function f without changing f itself?

Of course, a necessary condition is that the number of variables in X_1 and X_2 must be the same. Otherwise you could not completely exchange one subset with the other. However, this is not sufficient. Supposing that $|X_1| = |X_2| = k$ let us see what *sufficient* condition exists for the exchangeability of X_1 and X_2 in f.

Let $f_{a_1(X_1)a_2(X_2)}$ be the cofactor of f where the variables of X_1 are set to $a_1 \in \{0,1\}^k$ and the variables of X_2 are set to $a_2 \in \{0,1\}^k$. Now we are able to formulate our condition.

Fact 4.3 *Exchanging two different, ordered subsets of variables,* $X_1 = [x_1^1, \ldots, x_k^1]$ *and* $X_2 = [x_1^2, \ldots, x_k^2]$*, i.e., exchanging* x_i^1 *with* x_i^2 *for all* $i = 1, 2, \ldots, k$*, does not change the function* f *iff for all assignments* $a_1, a_2 \in \{0,1\}^k$*:* $f_{a_1(X_1)a_2(X_2)} = f_{a_2(X_1)a_1(X_2)}$.

Note that a well–known special case of this property is $k = 1$, i.e., the question if two variables, x_i and x_j are exchangeable in a Boolean function f without changing f. In this case, we call x_i and x_j a pair of *symmetric variables*, and we know that this symmetry can be shown using Fact 4.2.

In other words, Fact 4.3 is only a generalization of the well–known symmetry test for symmetric variables to a test for the exchangeability of groups of variables.

Let us consider an example:

Example 4.2 $f = \bar{x}_1\bar{x}_2\bar{x}_3x_4 + \bar{x}_1x_2\bar{x}_3\bar{x}_4 + \bar{x}_1\bar{x}_2x_3x_4 + x_1x_2\bar{x}_3\bar{x}_4$

This function f is a Boolean function over the set of input variables $X = [x_1, x_2, x_3, x_4]$. There is no pair of symmetric variables in it. We can prove it using Fact 4.2. However, two subsets of X, $X_1 = \{x_1, x_2\}$ and $X_2 = \{x_3, x_4\}$ can be exchanged without changing f. We can prove that fact having a look at the equation of Example 4.2. For a formal proof using Fact 4.3 it is necessary to check the six different cofactor equations that can be constructed by the different assignments a_1 and a_2 to X_1 and X_2:

a_1	a_2	cofactor equation
00	01	$f_{\bar{x}_1\bar{x}_2\bar{x}_3x_4} = f_{\bar{x}_1x_2\bar{x}_3\bar{x}_4} = 1$
00	10	$f_{\bar{x}_1\bar{x}_2x_3\bar{x}_4} = f_{x_1\bar{x}_2\bar{x}_3\bar{x}_4} = 0$
00	11	$f_{\bar{x}_1\bar{x}_2x_3x_4} = f_{x_1x_2\bar{x}_3\bar{x}_4} = 1$
01	10	$f_{\bar{x}_1x_2x_3\bar{x}_4} = f_{x_1\bar{x}_2\bar{x}_3x_4} = 0$
01	11	$f_{\bar{x}_1x_2x_3x_4} = f_{x_1x_2\bar{x}_3x_4} = 0$
10	11	$f_{x_1\bar{x}_2x_3x_4} = f_{x_1x_2x_3\bar{x}_4} = 1$

Note that we do not consider assignments with $a_1 = a_2$.

However, having a closer look at f we see that while exchanging x_1 with x_3 and x_2 with x_4 keeps f invariant, exchanging x_1 with x_4 and x_2 with x_3 generates another function: $\hat{f} = x_1\bar{x}_2\bar{x}_3\bar{x}_4 + \bar{x}_1\bar{x}_2x_3\bar{x}_4 + x_1x_2\bar{x}_3\bar{x}_4 + \bar{x}_1\bar{x}_2x_3x_4$. This function is not equal to f, although we also have exchanged the variables of X_1 with those of X_2.

Example 4.2 clarifies the following. In general there are two major problems that prevent us to use Fact 4.3 in practice. The first one is its complexity. For two variable groups of size k there are $(2^{2k-1} - 2^{k-1})$ tests necessary to check their exchangeability in the prescribed manner. There are 2^k different vectors of constant values in $\{0,1\}^k$. In the first step, the selected vector a_1, say $00\ldots0$, has to be checked with all possible vectors a_2 except the same as a_1. That gives $2^k - 1$ different possibilities. Once that has been done, we can select another vector a_1, say $00\ldots01$, and start again with selecting the vector a_2. However, we do not have to use $a_2 = 00\ldots0$ again, because the equation

$$f_{X_1 \leftarrow 00\ldots0 X_2 \leftarrow 0\ldots01} = f_{X_1 \leftarrow 0\ldots01 X_2 \leftarrow 00\ldots0}$$

has been tested in Step 1 already. Similarly, there are only $2^k - 2$ equations to test in the second step, $2^k - 3$ in the third step, and so on. All in all, there are $\sum_{i=1}^{2^k-1} i = [2^k \cdot (2^k - 1)]/2$ different cofactor equations that have to be tested for our purpose.

The second problem is that Fact 4.3 only holds for exactly one exchange of the variables of X_1 with those of X_2, namely x_i^1 with x_i^2 for all $i = 1, 2, \ldots, k$. However, as shown in Example 4.2, there are other possibilities to exchange the variables, as well. Those possibilities are not covered by the cofactor tests of Fact 4.3. That implies that if we want to know if none of the possible exchanges of the variables of X_1 with those of X_2 changes f, then it is necessary to apply a series of cofactor tests similar to that formulated in Fact 4.3 to *each* of the $k!$ exchanging possibilities.

Considering these two problems we know that it is not possible to use Fact 4.3 for a practical solution of the general problem: there would be $\Theta(k! \cdot 2^k)$ cofactor tests necessary to check the exchangability of two sets of variables with size k.

However, let us consider our special case – the test, whether X_1 and X_2 are *h–symmetric*.

Fortunately, there is one more property given on X_1 and X_2 that makes it comfortable for us to use Fact 4.3 in this case. *This property is the partial symmetry of the variables of X_1 and X_2, respectively.*

Let us see how we can use this in the detection of h–symmetries. A Boolean function $f \in \mathcal{B}_{n,1}$ is symmetric with respect to a subset of input variables $X_i \subseteq X$ iff f keeps invariant under all permutations of the variables in X_i. Furthermore, since each vector $a \in \{0,1\}^k$ with exactly l one's in it is a permutation of any other vector with l one's in it, it is equivalent to say: f is symmetric with respect to X_i if f only depends on the number of one's in the input assignment to X_i [38]. We will call this number of one's in an input assignment to a set of variables, X_i the *weight* of that input assignment with respect to X_i.

Now we can directly conclude the following two facts about the cofactor of f with respect to a set of symmetric variables.

Fact 4.4 *Let V_l be the set of all assignments of constant values to the set of symmetric variables, X_i with the weight l. Let $a_1, a_2 \in V_l$. Then $f_{a_1} = f_{a_2}$.*

This gives us the number of different cofactors with respect to a set X_i of partial symmetric variables that can be constructed.

Fact 4.5 *Consider the set of all cofactors of f that can be constructed with respect to the set X_i of partial symmetric variables. The maximal cardinality of this set is $|X_i| + 1 = k + 1$. (One cofactor for every weight.)*

With Fact 4.4 and 4.5 we can go back to our two problems checking the exchangeability of two sets of variables in a function. We will see that these two problems are solved for h–symmetry.

First let us consider the complexity of the cofactor test again. Because of Fact 4.4 and 4.5 it can be modified as follows: instead of all 2^k possible assignments for the variables of X_1 and X_2, respectively, we just have to consider $k+1$ assignments of X_1 as well as of X_2, one of every possible weight. In other words, we do not have to test all combinations of the 2^k *assignments* but only all combinations of the $k + 1$ possible *weights*.

Fact 4.6 *Exchanging two different subsets of partial symmetric variables, X_1 and X_2 does not change the function f iff for each set of assignments V_{l_1} and V_{l_2} of $\{0,1\}^k$ to the variables of X_1 and X_2 with weight l_1 and l_2 and $l_1 \neq l_2$:*

$$f_{V_{l_1}(X_1)V_{l_2}(X_2)} = f_{V_{l_2}(X_1)V_{l_1}(X_2)}.$$

Note, that we have to know that X_1 and X_2 are sets of partial symmetric variables before using Fact 4.6 for our purpose.

What about the weight combinations $0/0$, $1/1, \ldots, k/k$? Obviously, we do not have to test the combinations $0/0$ and k/k, i.e., all variables of X_1 and X_2 set to 0 and 1, respectively, since an exchange of the two variable groups with these assignments does not change the function value of f. However, the same holds for all other combinations with the same weight. The reason of this is the partial symmetry of the variable groups. In order to test the combination i/i with $i \in \{1, 2, \ldots, k-1\}$, we have to select an assignment with weight i for the variables of X_1 as well as for the variables of X_2. Since we can select *any* assignment with weight i, let us take the *same* assignment for X_2 as for X_1. Now we have the same situation for the weight combination i/i as for the combinations $0/0$ and k/k – an exchange of the variable groups with these weights cannot change the function value of f.

Suppose the symmetry of the variables of our two groups X_1 and X_2 is tested before, we get the moderate number of $(k^2 + k)/2$ tests necessary to check the h–symmetry of X_1 and X_2: k choices

for the assignment to X_2 in the first step, $k-1$ in the second, and so on. Dealing with ROBDDs we know that these tests need only constant time on the given cofactors. The cofactor construction itself needs time linear in the number of ROBDD nodes of f. So, in all, we need time $O(k^2|V|)$ where $|V|$ denotes the number of nodes in the ROBDD of f and k the size of the two variable groups we want to test.

Let us now discuss the second problem with the cofactor test. As we know, if X_1 and X_2 are different groups of partial symmetric variables, then f only depends on the weight of the inputs of X_1 and X_2, but does not depend on the permutation of those variables in f. That is why it is enough to do the cofactor tests of Fact 4.6 with respect to one possible exchange of X_1 with X_2 in order to get the information about the exchangeability of X_1 and X_2 regarding to all other exchanges as well. In other words, in the case of partial symmetry groups it is possible to exchange the variables of the groups in either all or no combinations without changing the function f.

Now we can start with a description of the complete algorithm to determine the h-symmetry groups of a Boolean function $f \in \mathcal{B}_{n,1}$.

First, let us review on Example 4.1: $f = \overline{(x_1 + x_2)} + \overline{(x_3 + x_4)} + \overline{(x_5 + x_6)}$. As we know, there is h–symmetry between certain variable groups in this example. This can be proved as follows:

1. Consider the groups of symmetric variables. In the case of this example there are three groups with size 2: $X_1 = \{x_1, x_2\}$, $X_2 = \{x_3, x_4\}$, and $X_3 = \{x_5, x_6\}$.

2. To prove the h–symmetry of these groups it is enough to check X_1 with X_2, and X_2 with X_3 (see Property 4.6). Three cofactor test are necessary in both cases: weight combination 0/1, 0/2, and 1/2. For X_1 and X_2 these are:

$$\begin{aligned} f_{\bar{x}_1\bar{x}_2\bar{x}_3x_4} &= f_{\bar{x}_1x_2\bar{x}_3\bar{x}_4} = 1 \\ f_{\bar{x}_1\bar{x}_2x_3x_4} &= f_{x_1x_2\bar{x}_3\bar{x}_4} = 1 \\ f_{\bar{x}_1x_2x_3x_4} &= f_{x_1x_2\bar{x}_3x_4} = \overline{(x_5 + x_6)}. \end{aligned}$$

   Similarly this has to be done to test the exchangeability of X_2 and X_3.

Now, let us come to our algorithm. Given a Boolean function $f \in \mathcal{B}_{n,1}$ we can determine the h–symmetries of these variables in two steps:

Step 1: Create the list of all candidates for h–symmetry.

A *candidate for h–symmetry* is a set of same–sized partial symmetry groups of variables of f, except those of size one. Note, that we construct only maximal candidates. So, there is no h–symmetry possible between different candidates.

Recall Example 4.1, here we have exactly one candidate, *viz.* $\{\{x_1, x_2\}, \{x_3, x_4\}, \{x_5, x_6\}\}$.

Step 2: For each candidate do: use Fact 4.6 and Property 4.6 to establish the h–symmetry groups.

The second step of our algorithm, Step 2 has been described already on Example 4.1. The generalization is straightforward. Note that a candidate for h–symmetry could include sets of input variables that belong to different h–symmetries. If this case has to be taken into account, then we need to test the exchangeability of *each* pair of variable sets in the candidate. In other words, we cannot use the property of *one* h–symmetry to be an equivalence relation.

Let us discuss Step 1. What is the best way to determine the list of candidates? One way is the following: first compute all groups of partial symmetries of X with respect to function f [28], then sort these groups by their size. A candidate is the set of all symmetry groups with the same size, except the one including all groups of size one.

However, this kind of selection of candidates is not the best – any cofactor test (and so any cofactor construction) that can be avoided is a bonus point with respect to CPU time and storage requirements. Nevertheless, we already *know* a better way for this. Property 4.4 tells us that the variables of any two partial symmetry groups must have the same signature to be a candidate for h–symmetry. That is why the signature computation is an efficient pre–processing for the selection of candidates for h–symmetry: only aliasing groups are candidates, furthermore only those groups that consist of groups of partial symmetric variables. In this way we are likely to get a smaller set of candidates, such that we can avoid cofactor constructions. Furthermore, the probability that there are variable groups in one candidate that belong to different h–symmetries is less, because the partition of the input variables by signatures is more qualified than just determining same–sized groups of partial symmetric variable groups.

Thus to create the list of candidates for h–symmetry, we use all the signatures introduced in Section 3.2.3 first. If there are aliasing groups, then we select those groups that consist of partial symmetric variable groups, as candidates for h–symmetry.

4.4 Group Symmetries

Now we change our focus to the following kind of symmetry.

Definition 4.4 *Let $f \in \mathcal{B}_{n,1}$ be a Boolean function with n input variables $X = [x_1, x_2, \ldots, x_n]$.*
Let $X_1, X_2, \ldots, X_k \subset X$ be $k > 1$ non-empty and pairwise disjoint subsets of X.
Let $\pi_i \in \mathcal{P}_{|X_i|} \setminus \mathbf{1}$ be a permutation on the input variables of X_i, for all $i = 1, 2, \ldots, k$.
The k groups of input variables are **group symmetric (g–symmetric)** *iff*

1. *$|X_i| > 1$ for all $i \in \{1, 2, \ldots, k\}$*
2. *There is a set $[\pi_1, \pi_2, \ldots, \pi_k]$ of permutations of the variables in $X_1, X_2, \ldots, X_k$, such that applying the permutations π_i to X_i simultaneously for all $i = 1, 2, \ldots, k$ does not change the function f.*

Note, that manipulating just one or a couple of variable subsets X_i may change the function f. The following examples will help to clarify the definition.

Example 4.3 $f = a_0\overline{(x_0 + x_1)} + b_0\overline{(x_2 + x_3)}$

Here, $\{x_0, x_1\}$ and $\{x_2, x_3\}$ are pairs of partial symmetric variables, but there is no h-symmetry between them because of the existence of the input variables a_0 and b_0. However, exchanging $\{x_0, x_1\}$ and $\{x_2, x_3\}$ AND a_0 and b_0 keeps the function invariant. So, there is what we call g–symmetry between the two subsets of input variables, $\{x_0, x_1, x_2, x_3\}$ and $\{a_0, b_0\}$.

Group symmetries are a special kind of $\mathcal{G}$–symmetry, too. So let us consider the partition $\mathcal{A}$ of the n input variables constructed by this $\mathcal{G}$–symmetry.

Property 4.8 *If k subsets of input variables are g–symmetric, then each of these subsets form a subset of the partition $\mathcal{A}$ of the inputs given by this special $\mathcal{G}$–symmetry.*

Proof: This follows from the definition of partition $\mathcal{A}$. □

In our example, $\mathcal{A}$ is as follows:

$$\mathcal{A} = \{\{x_0, x_1, x_2, x_3\}, \{a_0, b_0\}\}.$$

Again, we have a case where signatures will be unable to distinguish between the variables and thus results in the formation of aliasing groups (see Property 4.4).

Our practical experiences on the benchmark set show that this kind of $\mathcal{G}$–symmetry appears relatively often. One well–known example is an n–bit multiplier:

$$x_n x_{n-1} \ldots x_{\frac{n}{2}+1} * x_{\frac{n}{2}} \ldots x_2 x_1.$$

Exchanging $x_{\frac{n}{2}+i}$ and x_i for each $i = 1, \ldots, \frac{n}{2}$ simultaneously keeps the function invariant. The partition $\mathcal{A}$ of these variables is:

$$\mathcal{A} = \{i = 1, 2, \ldots, \frac{n}{2} : \{x_i, x_{\frac{n}{2}+i}\}\}.$$

So there are $2^{\frac{n}{2}}$ possible variable correspondences for an n-bit multiplier after signature computation. Taking the g–symmetry of the n–bit multiplier function into account, this number would decrease by the half, i.e., instead of $2^{\frac{n}{2}}$ possible variable correspondences there would be $2^{\frac{n}{2}-1}$, which is still too large in general.

Moreover, it seems to be very complicated to detect a g–symmetry in general. Here, one major problem is that not *all* possible permutations of the input variables in $X_1, X_2, \ldots, X_k$ have to change function f (see the definition of group symmetry). Thus, to be able to handle general g–symmetries we need a way to detect g–symmetric groups of input variables *and* a way to select those permutations of these variables that keep the g–symmetric function invariant. Considering the very general property of g–symmetry, this looks like a complicated task.

In looking for possible ways to handle g–symmetry we made the following interesting observation. The subsets of input variables that result in a g–symmetry are often connected with each other in the following sense: if we have identified the variables of one of these subsets, then it is possible to identify the variables of the other subsets as well. With this knowledge we have developed a heuristic to distinguish between variables of g–symmetric subsets.

This heuristic works as follows. Given a Boolean function $f \in \mathcal{B}_{n,1}$ let us consider the following situation. There is a partial, permutation independent order of the n input variables of f constructed by using the signatures introduced in Section 3.2.3. Furthermore, the groups of partial symmetric input variables (see Section 4.2) are identified as well as the aliasing groups with h-symmetric groups of input variables (see Section 4.3). Nevertheless, there are still unidentified groups of aliasing variables. For the sake of simplicity, say these groups are the first k in the permutation independent order of variables:

$$\mathcal{A} = \{A_1, A_2, \ldots, A_k, \ldots\}$$

In this situation, we assume that all k groups of aliasing variables are connected by one g–symmetry and use the observation that the groups of input variables that result in a g–symmetry are connected with each other in the special sense just mentioned.

Now, we just *assume* that the input variables of one of these aliasing groups, say A_1, are uniquely identified. Under this assumption, we apply a couple of function signatures (see Section 3.2.3) to each variable of the other $k-1$ aliasing groups, i.e., we construct function signatures that depend not only on those input variables that can be uniquely identified but also on those of aliasing group A_1.

The basic idea of this heuristic is that we may be able to find a unique function signature for each of the input variables in A_2 to A_k in the case of g–symmetry, because of the connection among all k aliasing groups of such a g-symmetry. If this is the case, we can uniquely identify each input variable in the aliasing groups A_2 to A_k. Our practical experiences have shown that often this is indeed the case.

Let us consider an example.

Example 4.4 $f(a_0, a_1, a_2, x_0, x_1, x_2) = a_0\overline{(x_0 + x_1)} + a_1\overline{(x_1 + x_2)} + a_2\overline{(x_2 + x_0)}$

This function f is g–symmetric with respect to the variable groups $\{a_0, a_1, a_2\}$ and $\{x_0, x_1, x_2\}$. Let us see how the heuristics works. We pick up one of these two aliasing groups, say $\{a_0, a_1, a_2\}$, and assume that we can uniquely identify these three variables. Under this assumption we can apply a function signature to the other three variables which may depend on a_0, a_1, and a_2:

$$f_{x_0\bar{x_1}\bar{x_2}} = a_1$$
$$f_{\bar{x_0}x_1\bar{x_2}} = a_2$$
$$f_{\bar{x_0}\bar{x_1}x_2} = a_0$$

Order of $\{a_0, a_1, a_2\}$	Implicit Order of $\{x_0, x_1, x_2\}$
(a_0, a_1, a_2)	(x_2, x_0, x_1)
(a_0, a_2, a_1)	(x_2, x_1, x_0)
(a_1, a_0, a_2)	(x_0, x_2, x_1)
(a_1, a_2, a_0)	(x_0, x_1, x_2)
(a_2, a_0, a_1)	(x_1, x_2, x_0)
(a_2, a_1, a_0)	(x_1, x_0, x_2)

Table 4.1: Group Symmetry

And indeed, the variables x_0, x_1, and x_2 can be uniquely identified with these signatures. Furthermore, we can use the order relation for function signatures to get a unique order of the variables x_0, x_1, and x_2. Applying the lexicographically order relation for Boolean functions the unique order of the three variables is (x_2, x_0, x_1).

However, this unique description is not permutation independent. It depends on the permutation of the input variables in A_1, which is in the case of Example 4.4 (a_0, a_1, a_2). So we need a way to make the information permutation independent. Therefore, we consider the permutation π of the input variables of f which results into the actual unique order of these variables given by the signatures (assuming that there is a unique ordering). In our example, this permutation π is equal to

$$\pi(a_0, a_1, a_2, x_0, x_1, x_2) = (a_0, a_1, a_2, x_2, x_0, x_1).$$

With the help of π, we construct the Boolean function $\hat{f} = f \circ \pi$. This function is stored in a set $\hat{F}$. Now, the same procedure is carried out for each possible order of the input variables in A_1. In the left column of Table 4.1 these orders are listed for the set of variables $A_1 = \{a_0, a_1, a_2\}$ of Example 4.4. Each of these orders implies an order for the variables $\{x_0, x_1, x_2\}$, which are listed in the right column of the table. As described for the first variable order in Table 4.1 the function $\hat{f}$ is computed and stored in $\hat{F}$.

When this has been done for all possible orders of the variables in aliasing set A_1, then the lexicographical smallest of all Boolean functions in $\hat{F}$ is selected. The variable order which belongs to this function and the function itself are the permutation independent information for function f.

Note, that we can also use any of the other sets of aliasing variables, $A_2, \ldots, A_k$. We decided to use one of those sets with minimal size, since this reduces the number of steps necessary to construct the set $\hat{F}$ of Boolean functions.

This kind of $\mathcal{G}$–symmetry has been independently investigated in [30] as well. In this paper, the authors introduce methods to determine maximal sets of partial symmetric input variables of a circuit without ROBDDs, mention that there are other than partial symmetries among input variables and develop an idea to find candidates for this kind of symmetric input variables. This idea is similar to that what we have used for our group–symmetry heuristics. In a later work,

	circuit			number of correspondences			cpu (in sec.) for	
name	#i	#o	#n	*sig*	*+hsym*	*+grsym*	*sig*	*all*
CM150	21	1	64	$\approx 10^7$	$\approx 10^7$	1	0.8	4.9
CM151	12	2	32	216	216	1	0.2	0.5
act2	8	1	12	4	4	1	0.0	0.0
addm4	9	8	225	16	16	1	0.6	1.0
cordic	23	2	86	4	1	1	0.4	0.4
dist	8	5	135	16	16	1	0.3	0.7
ex4	128	28	896	4	4	1	27.4	40.8
i3	132	6	134	$\approx 10^{23}$	1	1	48.2	55.0
lal	26	19	123	24	1	1	0.2	0.2
misg	56	23	109	5184	216	1	1.6	36.4
mlp4	8	8	141	16	16	1	0.3	0.8
mult3	6	6	44	8	8	1	0.1	0.2
mux	21	1	88	$\approx 10^7$	$\approx 10^7$	1	0.9	4.9
mux_cl	11	1	18	216	216	1	0.1	1.4
ryy6	16	1	27	4	1	1	0.1	0.1
sao2	10	4	123	16	16	1	0.2	0.6
t481	16	1	80	331776	331776	331776	0.6	0.6
ts10	22	16	271	720	720	720	2.1	2.2
term1	34	10	616	$\approx 10^8$	$\approx 10^8$	1	12.7	36.0

Table 4.2: Benchmarks with $\mathcal{G}$–Symmetries

the authors consider the permutation equivalence problem. We briefly described their method to handle the permutation problem in Section 3.3. In this paper, they mention that it is advantageous to detect as many of that what we call group–symmetric variables as possible before attempting to solve the permutation equivalence problem. However, they do not apply the idea introduced in [30] to do this.

We have implemented *our* approach under the assumption that there is only one $\mathcal{G}$–symmetry between the aliasing groups and tested it for the examples of our benchmark set.

4.5 Experimental Results

Similar to our experiments with signatures, we implemented the ideas presented in this chapter in the Berkeley SIS–system, release 1.3 [34]. All experiments were done on a SUN Sparcstation 10 with 64 MB RAM.

Table 4.2 lists the 8% of benchmarks with aliasing after signature computation which are the subject of our research regarding to $\mathcal{G}$–symmetries. In this table, a description of each circuit (name, number of inputs, outputs, and ROBDD nodes) is followed by the number of possible correspondences after

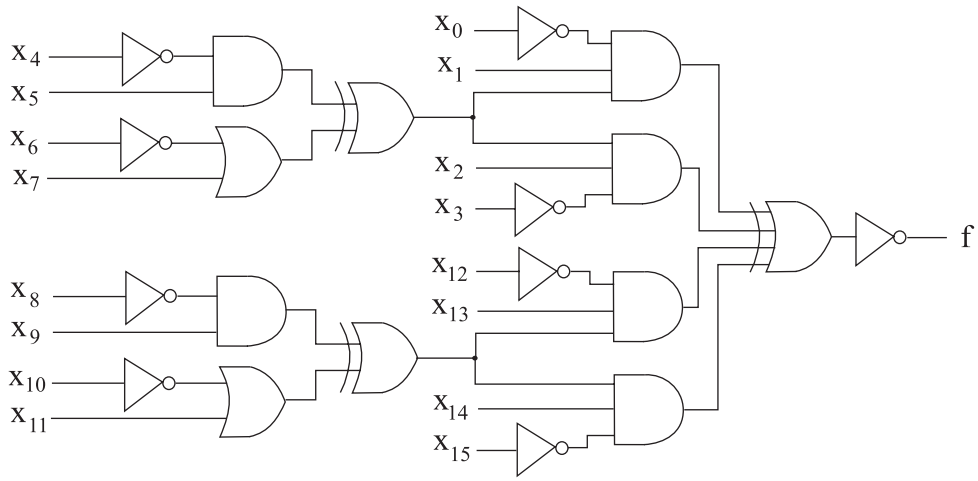

Figure 4.1: Description of Benchmark Circuit *t481*

using signatures only (*sig*), signatures and the algorithm to determine h–symmetry (+*hsym*), and finally, signatures, the h–symmetry algorithm and the heuristic for g–symmetries (+*grsym*). The last two columns include the CPU–time in seconds needed to apply the signatures and to apply the signatures as well as all three heuristics on a SUN Sparcstation 10.

As can be observed, the results are promising. Approx. 20% of all benchmarks with aliasing groups include h–symmetry. Furthermore, in all but one of these cases (namely *misg*) this is the only reason for the existence of aliasing groups. The CPU–time necessary for the detection of h–symmetry after signature computation is very promising. In many cases where h–symmetry could not be detected there is no measurable CPU–time (i.e., 0.0 sec. in the table). Here, the reason is that there are no classes of partial symmetries in the aliasing groups, so testing this necessary condition is enough to establish the fact that there is no h–symmetry. Thus, no cofactor computation is necessary in these cases. However, even in our worst case with respect to the CPU–time, namely *i3*, it only takes 6.8 seconds to establish all h-symmetries. The reason for this fact is that the variable groups that are tested within h–symmetry are rather small (i.e., size 2, 3, or 4), so that the quadratic factor of the complexity ($O(k^2|V|)$, see Section 4.3) is not too expensive. Note that for *i3* the results regarding to the correspondence possibilities is especially impressive, too. After signature computation there are still approx. 10^{23} correspondence possibilities for the input variables of *i3* left, but all come from a permutation of h–symmetric variable groups.

For all but two of the rest the *grsym*–algorithm was successful. These two examples are *ts10* and *t481*. Our investigations have shown that the aliasing variables of both examples are involved in a special symmetry as well. Unfortunately, we have not been able to automate the detection for these cases.

Benchmark *ts10* includes a group of six aliasing variables with rotational symmetry. We discuss rotational symmetry in the next section.

The other example is the benchmark circuit *t481*. Figure 4.1 shows a description of it. This example

has 16 input variables, $X = [x_0, x_1, \ldots, x_{15}]$, and one output. By using the signatures introduced in this thesis the set of inputs is divided into the partition

$$\mathcal{A} = \{\{x_4, x_7, x_8, x_{11}\}, \{x_5, x_6, x_9, x_{10}\}, \{x_0, x_3, x_{12}, x_{15}\}, \{x_1, x_2, x_{13}, x_{14}\}\}.$$

Considering the structure of *t481* we can observe that there is indeed a group symmetry between the four groups of aliasing variables in partition $\mathcal{A}$: for instance, applying the permutation

$$\pi(x_0, \ldots, x_{15}) = (x_{12}, x_{13}, x_{14}, x_{15}, x_8, x_9, x_{10}, x_{11}, x_4, x_5, x_6, x_7, x_0, x_1, x_2, x_3)$$

does not change the function f. However, non of the four groups of aliasing variables can be used to identify *all* other aliasing variables uniquely as described in Section 4.4. In other words, our heuristic to assume that the variables of one of these aliasing groups can be uniquely identified and then to try to distinghuish between the others does not work for *t481*. The reason for this fact may be that there are permutations of the inputs of *t481* that keep this function invariant but do not involve *all* aliasing groups. For example, exchanging the variable x_0 with x_3 and the variable x_1 with x_2 does not change *t481* although just the last two aliasing groups are involved in this permutation. In other words, the four aliasing groups of *t481* are not *completely* connected with each other, thus the heuristics introduced here cannot help to distinguish between the variables completely. The characteristic of circuit *t481* underlines once more the complexity of general $\mathcal{G}$–symmetry.

4.6 The Variety of $\mathcal{G}$–Symmetry

We introduced three kinds of $\mathcal{G}$–symmetry that cover a wide range of symmetries appearing in practice. For these symmetries we could provide efficiently working solution paradigms for the permutation equivalence problem P_π.

Nevertheless, it seems to be not possible to find a general solution paradigm which allows us to handle $\mathcal{G}$–symmetry efficiently because of the generality of this function property. Each kind of $\mathcal{G}$–symmetry seems to demand a special procedure to handle it. Although we feel that the $\mathcal{G}$–symmetries introduced in this chapter are those mostly appearing in practice, other symmetries may appear as well.

Here is one of these $\mathcal{G}$–symmetries for that we have even an example in our benchmark set (This is *ts10*. See the previous section). Consider the following function:

$$f = x_1x_2 + x_2x_3 + x_3x_4 + x_4x_5 + x_5x_1.$$

It is obvious that rotating the five variables in function f (i.e., applying the permutation $\pi = (x_5, x_1, x_2, x_3, x_4)$ to the inputs of f) does not change this function. So it is a kind of $\mathcal{G}$–symmetry, too. All five input variables of the Boolean function f will form one aliasing group that cannot be refined by using signatures. Furthermore, there are no partial, no hierarchical, as well as no group symmetries between these input variables. We call this kind of symmetry *rotational symmetry* and define it as follows.

Definition 4.5 *Let $f \in \mathcal{B}_{n,1}$ be a Boolean function.*
Let $Y = [y_1, y_2, \ldots, y_k] \subseteq X$ be a subset of the input variables of f.
Let $\pi = (y_k, y_1, y_2, \ldots, y_{k-1})$ be a permutation of $\mathcal{P}_k$.
f is **rotational symmetric (r–symmetric)** *iff*

1. *$k \geq 3$,*
2. *f is not partial symmetric in Y, and*
3. *f does not change on applying the permutations π to the variables of Y.*

Note that also the permutation π^{-1} keeps f invariant as well as applying π more than once, since the set $\mathcal{G} \subset \mathcal{P}_n$ which constructs an r–symmetry is a group. Similar to group symmetry, r–symmetry is neither easy to detect nor easy to handle. Furthermore, the heuristic applied for the group symmetries, (i.e., assume one of the input variables of Y has been uniquely identified by a signature, then try to use function signatures to distinguish between the others as well) does not work very successful for rotational symmetry, as our experiments have shown. Rotational symmetry illustrates the difficulties that may appear in general with $\mathcal{G}$–symmetries.

However, partial symmetries, hierarchical symmetries, and group symmetries seem to be the most common $\mathcal{G}$–symmetries appearing in practice. So, the algorithms presented for handling these symmetries have direct practical impact on the complete solution of the permutation problem.