Autosymmetry of Incompletely Specified Functions

Anna Bernasconi
Dipartimento di Informatica
Università di Pisa, Italy
anna.bernasconi@unipi.it

Valentina Ciriani
Departimento di Informatica
Università degli Studi di Milano, Italy
valentina.ciriani@unimi.it

***Abstract*—Autosymmetric Boolean functions are "regular functions" that are rather frequent in the set of Boolean functions describing standard circuits. Autosymmetry is typically exploited for improving the synthesis time and the quality of the optimized circuits. This paper studies in not a naive way, for the first time, the autosymmetry of incompletely specified functions, i.e., Boolean functions with don't care conditions. The theory of autosymmetry for completely specified functions is extended to the incompletely specified case and a new heuristic algorithm is provided for the detection of autosymmetry. The experimental results validate the theoretical study and show that the 77% of the considered benchmarks has an improved autosymmetry degree.**

I. Introduction

Autosymmetric functions are Boolean functions exhibiting a special structural regularity that is easily expressed using the XOR operation. They were introduced in [15] and further studied in [2], [7], [8], [9], [12], [13], where it was shown how this regularity can be exploited to speed-up the minimization process and to derive more compact logic representations. Recently, an application of the autosymmetry property to estimate and reduce the multiplicative complexity [16], [17] of functions, for security protocols, has been discussed in [3]. Moreover, the authosimmetry property has been recently exploited in [5], [6] for the logic synthesis in emerging technologies, i.e., Switching Nano-Crossbars [1], [4].

Intuitively, a Boolean function f over n binary variables is *k-autosymmetric* if it can be projected onto a smaller function f_k that depends on $n-k$ variables only, and has a smaller on-set. The XOR operation comes into play as the new $n-k$ variables are XOR combinations of some of the original ones. The function f_k is called a *restriction* of f; indeed it is "equivalent" to, but smaller than, f and has $|f|/2^k$ points only, where $|f|$ denotes the number of points of f. Observe that, autosymmetric functions depend in general on all their n input variables, even if they can be studied in a $n-k$ dimensional space. For example, Figure 1(b) shows the restriction of the 1–autosymmetric function depicted in Figure 1(a).

Even if the set of autosymmetric functions is much smaller than the one containing all the Boolean functions[1], a considerable amount of standard Boolean functions of practical interest falls in this set. Indeed, about 24% of the functions in the classical ESPRESSO benchmark suite [18] have at least one truly (i.e., non degenerate) autosymmetric output, as shown and discussed in [7], [9]. Thus, autosymmetry is a property that is frequent enough within Boolean functions to be worth studying.

[1]The number N_B of Boolean functions of n variables is $N_B = 2^{2^n}$, while the number of autosymmetric ones is $N_A = (2^n-1)2^{2^{n-1}}$ [9].

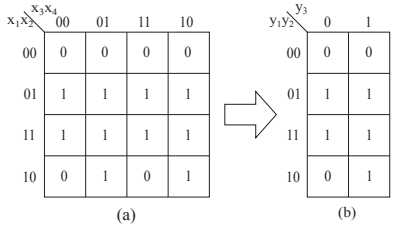

Fig. 1. *An autosymmetric completely specified Boolean function f with autosymmetry degree $k = 1$, in a Karnaugh map of four variables: x_1, x_2, x_3, and x_4 (on the left). And the restriction f_k that depends on the variables y_1, y_2 , and y_3 (on the right).*

Another very important characteristic of autosymmetric functions is that they can be efficiently identified: the restriction f_k and the linear equations that define the new $n-k$ variables can be derived in time polynomial in the number of variables and minterms [8]. This allows to exploit the autosymmetry in the minimization process: the idea is to minimize the restriction f_k, which depends on a smaller number of variables and has only $|f|/2^k$ points, instead of $|f|$, and then to derive a minimal form for f composing the linear equations defining the new variables with a minimal form for f_k. This final step can be also implemented efficiently, by simply adding a XOR layer to a circuit for f_k.

Motivated by all these properties (frequency, efficiency of the autosymmetry test, possibility to derive compact logic representations in shorter time), in this paper we focus on the definition of autosymmetry for incompletely specified functions. Recall that a *completely specified Boolean function* f in n input variables, $x_1, \ldots, x_n$, is a function $f : \{0,1\}^n \rightarrow \{0,1\}$, where the outputs can assume the values 0 and 1. An *incompletely specified Boolean function* f in n input variables is a function $f : \{0,1\}^n \rightarrow \{0,1,-\}$, where the outputs can assume the values 0, 1, and the don't care value $-$. A don't care means that the value of the function, on that particular input, could be either 1 or 0, and the final value is usually decided by the minimization algorithm used to synthesize the function. Note that a function with t don't cares actually represents a family of 2^t different functions, where

978-3-9819263-5-4/DATE21/©2021 EDAA

each function corresponds to one of the possible assignments of the values 0 and 1 to the t don't cares.

In a couple of previous works on this subject [3], [7], autosymmetry was extended to incompletely specified functions in a quite simple yet restrictive way, i.e., assigning the same value 1 (or 0) to all don't care conditions, and testing the resulting function for autosymmetry. In this paper, we follow a different approach: our aim is now to select the function with the highest autosymmetry degree among the 2^t completely specified functions associated to an incompletely specified function with t don't cares. To this aim, we propose a strategy for performing such a selection in an efficient way, without generating and studying the 2^t possible functions, and we discuss a heuristic implementation.

We validate the proposed approach through a set of experiments, which show that the proposed method guarantees an increase of the autosymmetry degree in the 77% of the benchmarks. The degree is, in average, more than doubled.

The paper is organized as follows. Preliminaries on autosymmetric functions are described in Section II. Section III presents the strategy for enhancing the autosymmetry of incompletely specified functions, and Section IV reports the experimental results. Finally, Section V concludes the work.

II. Autosymmetric functions: preliminaries

In this section we summarize the definitions and the main theoretical results for *completely specified autosymmetric* functions [7]. These preliminary results are the basis for the new definition of autosymmetry of incompletely specified functions, introduced and discussed in the next sections.

Let f be a Boolean function. The *on-set* (i.e., f^{on}) of f is the set of input vectors x such that $f(x) = 1$; the *off-set* (i.e., f^{off}) is the set of input vectors x such that $f(x) = 0$; and the *don't-care set* (i.e., f^{dc}) is the set of input vectors x such that $f(x) = -$. Thus, we denote a completely specified Boolean function f with the couple (f^{on}, f^{off}), and we describe an incompletely specified Boolean function f using the triplet $(f^{on}, f^{dc}, f^{off})$.

We can extend the symbol $\oplus$ to denote the element-wise EXOR between two Boolean vectors, i.e., $\alpha, \beta \in \{0,1\}^n$, in the following way: $\alpha \oplus \beta$ is the vector obtained from β complementing in it the elements corresponding to the 1's of α. For example, $0011 \oplus 1010 = 1001$. Recall that $(\{0,1\}^n, \oplus)$ is a vector space, and that a *vector subspace* V is a subset of $\{0,1\}^n$ containing the zero vector, such that for each couple of vectors v_1 and v_2 in V, also the vector $v_1 \oplus v_2$ is in V. V contains 2^k vectors, k is the *dimension* of V, and V is generated by a basis B containing k vectors. The basis B is a minimal set of vectors of V such that each point of V is an EXOR combination of some vectors in B.

A completely specified Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$ is *closed under* a vector $\alpha \in \{0,1\}^n$, if for each vector $w \in f^{on}$, we have $w \oplus \alpha \in f^{on}$.

For example, the function $f : \{0,1\}^3 \rightarrow \{0,1\}$, such that $f^{on} == \{001, 010, 100, 111\}$, is closed under 011. We can simply verify the closure since $001 \oplus 011 = 010 \in f^{on}$, $010 \oplus 011 = 001 \in f^{on}$, $100 \oplus 011 = 111 \in f^{on}$, and $111 \oplus 011 = 100 \in f^{on}$.

Let us consider the set of vectors: $L_f = \{\alpha \in \{0,1\}^n \mid f \text{ is closed under } \alpha\}$. Note that, any function f is obviously closed under the zero vector $\mathbf{0}$, i.e., $\mathbf{0} \in L_f$. Moreover, if a function f is closed under two different vectors α_1, $\alpha_2 \in \{0,1\}^n$, it is also closed under $\alpha_1 \oplus \alpha_2$. Therefore, the set L_f is a vector subspace of $(\{0,1\}^n, \oplus)$. The set L_f is called the *vector space* of f. L_f contains 2^k vectors and it has dimension $k = \log_2 |L_f|$. For instance, consider again the function f in the previous example, f is closed under the vectors 000, 011, 101, and 110. Thus, $L_f = \{000, 011, 101, 110\}$.

We can now give the definition of autosymmetry for a completely specified Boolean function:

Definition 1 ([7]): A completely specified Boolean function f is *k-autosymmetric*, $0 \leq k \leq n$, if its vector space L_f has dimension k. A function f is *autosymmetric* if it is k-autosymmetric with $k \geq 1$.

For example, the function f, described in the previous examples, is 2-autosymmetric, since the dimension of the vector space L_f is 2.

In order to represent L_f, we can choose a particular basis, called canonical, in the following way. Consider a $2^k \times n$ matrix M whose rows correspond to the points of the vector space L_f of dimension k, and whose columns correspond to the variables $x_1, x_2, \ldots, x_n$. Let the row indices of M be numbered from 0 to $2^k - 1$. We say that L_f is in *binary order* if the rows of M are sorted as increasing binary numbers. The *canonical basis* B_V of L_f is the set of points corresponding to the rows of M with indices $2^0, 2^1, \ldots, 2^{k-1}$. The variables corresponding to the first 1 from the left of each row of the canonical basis are called *canonical variables* of L_f, while the other variables are called *non-canonical*. Informally, the canonical variables are the ones that have all the possible combinations of values in the vectors of the vector space L_f, while the non-canonical variables are the variables that, on L_f, have a constant value or are a linear combination of the canonical ones.

It can be easily proved that the canonical basis is indeed a vector basis. The canonical variables of L_f are also called canonical variables of f. Consider, for example, the vector space L_f of the function f of the previous example. We can arrange its vectors in a matrix in binary order:

	x_1	x_2	x_3
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	0

The canonical basis is composed of the vectors in position **1** and **2**, that are the vectors 011 and 101. The canonical variables of f are x_2 (corresponding to the first 1 in 011) and x_1 (corresponding to the first 1 in 101). The remaining variable, x_3, is non-canonical.

Consider now a vector $\alpha \in \{0,1\}^n$ and a vector subspace V of $(\{0,1\}^n, \oplus)$, we can "translate" V with respect to α and we obtain the set $A = \alpha \oplus V$, which is the *affine space* over V with *translation point* α. On the other hand, given an affine space A, there is a simple formula that characterizes the vector space V associated to a A: $V = \alpha \oplus A$ choosing α as any point in A. As show in [2], the points of a k-autosymmetric function f can be partitioned into $\ell = |f^{on}|/2^k$ disjoint sets, where $|f^{on}|$ denotes the number of vectors in the on-set of f. All these sets are affine spaces over the vector space L_f, i.e., $S = \alpha \oplus L_f$, where $\alpha \in f^{on}$. Thus, $f^{on} = \bigcup_{i=1}^{\ell}(w^i \oplus L_f)$ and for each i, j, $i \neq j$, $(w^i \oplus L_f) \cap (w^j \oplus L_f) = \emptyset$. The vectors $w^1, \ldots, w^\ell$ are chosen as all vectors of f^{on} where all the canonical variables have value 0.

Example 1: Consider the function f with on-set $f^{on} = \{0100, 0101, 0110, 0111, 1001, 1010, 1100, 1101, 1110, 1111\}$ depicted in Figure 1(a). It is easy to verify that $L_f = \{0000, 0011\}$. Therefore, f is 1-autosymmetric and the canonical variable is x_3. Thus, if we take the points of f^{on} with the canonical variable set to 0, i.e., $w^1 = 0100$, $w^2 = 0101$, $w^3 = 1001$, $w^4 = 1100$, and $w^5 = 1101$, we have $f^{on} = (0100 \oplus L_f) \cup (0101 \oplus L_f) \cup (1001 \oplus L_f) \cup (1100 \oplus L_f) \cup (1101 \oplus L_f)$.

Autosymmetric functions can be reduced to "equivalent, but smaller" functions. Indeed, if a function f is k-autosymmetric, then there exists a function f_k over $n-k$ variables, $y_1, y_2, \ldots, y_{n-k}$, such that $f(x_1, \ldots, x_n) = f_k(y_1, \ldots, y_{n-k})$, where each y_i is an EXOR combination of a subset of x_i's. These combinations are denoted $EXOR(X_i)$, where $X_i \subseteq X$, and the equations $y_i = EXOR(X_i)$, $i = 1, \ldots, n-k$, are called *reduction equations*. The function f_k is called a *restriction* of f; indeed f_k is "equivalent" to, but smaller than f, and has $|f|/2^k$ points only.

The restriction f_k can be computed from f and its vector space L_f by first identifying the canonical variables, and then deriving the projection of f in the subspace where all the canonical variables are set to 0 (see [2] and [8] for more details). The reduction equations correspond to the homogeneous system of linear equations whose solutions define the vector space L_f, and they can be derived applying standard linear algebra techniques as shown in [2], [8]. Notice that, in general, an autosymmetric function f depends on all its input variables.

Example 2: Consider the 1-autosymmetric function f in the running Example 1, with $L_f = \{0000, 0011\}$ and canonical variable x_3. We can build f_k by projecting the points of f in the space where $x_3 = 0$: $f_{x_3=0} = \{010, 011, 101, 110, 111\}$, that contains only 5 points and can be represented in algebraic form $f_k(y_1, y_2, y_3) = y_2 + y_1 y_3$ (see, Figure 1(b)). The homogeneous system whose solutions are $\{0000, 0011\}$ is:

$$\begin{cases} x_1 = 0 \\ x_2 = 0 \\ x_3 \oplus x_4 = 0 \end{cases}$$

Thus, the reduction equations are given by: $y_1 = x_1$, $y_2 = x_2$, and $y_3 = x_3 \oplus x_4$. Finally, we can reconstruct f from f_k in the following way: $f(x_1, x_2, x_3, x_4) = f_k(x_1, x_2, x_3 \oplus x_4) = x_2 + x_1(x_3 \oplus x_4)$. Observe that f depends on all the 4 input variables.

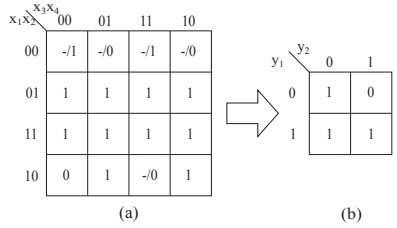

Fig. 2. *An autosymmetric incompletely specified Boolean function f with autosymmetry degree $k = 2$, in a Karnaugh map of four variables (x_1, x_2, x_3, x_4), on the left. And the restriction h_2 that depends on the variables y_1 and y_2, on the right. In the map on the left, the cells containing $-/0$ (resp., $-/1$) are don't care points that are specified to 0 (resp., to 1) in the construction of the function h.*

In summary, the vector space L_f provides the essential information to compute the autosymmetry degree, the restriction f_k, and the reduction equations of f.

III. Incompletely specified functions

In this section we show how the notion of autosymmetry can be extended from completely specified to incompletely specified Boolean functions.

In a previous work on this subject [7], autosymmetry was extended to incompletely specified functions in a quite restrictive way, by moving all the don't cares of a given function $f = (f^{on}, f^{dc}, f^{off})$ to its on-set, thus replacing f with the new completely specified function f_1 with on-set $f_1^{on} = f^{on} \cup f^{dc}$ and off-set $f_1^{off} = f^{off}$. In this scenario, f can be considered k-autosymmetric if and only if f_1 is k-autosymmetric. Similarly, one could move all the don't cares of f to the off-set and consider the completely specified function f_0, with on-set $f_0^{on} = f^{on}$ and off-set $f_0^{off} = f^{off} \cup f^{dc}$. In this case, f is k-autosymmetric if and only if f_0 is k-autosymmetric. In general, one can consider both cases, and select the function with highest autosymmetry degree between f_1 and f_0.

In this paper, we follow a different approach: our aim is now to select only a subset of don't cares of f to be moved to the on-set, in order to maximize the autosymmetry degree. More precisely, our idea is to derive a new completely specified function h such that

(i) $f^{on} \subseteq h^{on} \subseteq f^{on} \cup f^{dc}$

(ii) the autosymmetry degree of h is greater or equal to the autosymmetry degrees of the two functions f_1 and f_0.

Example 3: Consider again the function f from Example 1, and replace some off-set points with don't cares, as shown in Figure 2(a), so that only the vector 1000 is left in the off-set. If we move all don't care minterms to the on-set, we obtain a function f_1 that is not autosymmetric, as it can be easily verified since f_1^{on} contains an odd number of vectors. On the other hand, the function f_0 with on-set $f_0^{on} = f^{on}$ is 1-autosymmetric, as shown in the previous Example 1. However, by moving to the on-set the two don't care points 0000 and 0011 and to the off-set the other don't cares, we can derive a new completely specified function h such that: (i) *h correctly implements f, since $f^{on} \subseteq h^{on} \subseteq f^{on} \cup f^{dc}$,* (ii) *$h$ is 2-autosymmetric with vector*

space $L_h = \{0000, 0011, 1001, 1010\}$, and (iii) *the restriction h_2 depends on two variables only (see Figure 2(b)).*

In order to better exploit the presence of don't cares, we first generalize the definition of closure with respect to a vector as follows. Let $f = (f^{on}, f^{dc}, f^{off})$ be an incompletely specified function depending on n binary variables.

Definition 2: f is *closed* under a vector $\alpha \in \{0,1\}^n$, if for each vector $w \in f^{on}$, $w \oplus \alpha \in f^{on} \cup f^{dc}$.

Definition 3: The *closure set* of f is the set of all vectors α such that f is closed under α:

$$S_f = \{\alpha \in \{0,1\}^n \mid \forall w \in f^{on},\ w \oplus \alpha \in f^{on} \cup f^{dc}\}.$$

Unlike the case of completely specified functions, the set S_f is not in general a vector subspace.

Example 4: For the incompletely specified function f derived from our running example, and depicted in Figure 2, the closure set is given by $S_f = \{0000, 0011, 1000, 1001, 1010, 1011\}$, which is not a vector subspace (e.g., $1000 \oplus 1001 = 0001 \notin S_f$).

On the other hand, S_f contains the vector subspaces L_0 and L_1 associated to the completely specified functions f_0 and f_1 with on-set f^{on} and $f^{on} \cup f^{dc}$, respectively.

Lemma 1: Let $L_0 = \{\alpha \in \{0,1\}^n \mid \forall w \in f^{on},\ w \oplus \alpha \in f^{on}\}$ and $L_1 = \{\alpha \in \{0,1\}^n \mid \forall w \in f^{on} \cup f^{dc},\ w \oplus \alpha \in f^{on} \cup f^{dc}\}$. Then $L_0 \subseteq S_f$ and $L_1 \subseteq S_f$.

Proof. Let $\alpha \in L_0$. Then, for all $w \in f^{on}$, $w \oplus \alpha \in f^{on}$. Thus, $\alpha \in S_f$ since $f^{on} \subseteq f^{on} \cup f^{dc}$. Now, let $\alpha \in L_1$. We have that $\alpha \in S_f$ since in this case the closure property holds for all $w \in f^{on} \cup f^{dc}$, and in particular for all $w \in f^{on}$. ■

In order to apply the theory of autosymmetry, reviewed in Section II, to the incompletely specified function f and to define its restriction f_k and the corresponding reduction equations, we need to "extract" a vector subspace from S_f. We therefore propose the following definition.

Definition 4: Let f be an incompletely specified function, and let S_f be its closure set. f is *k-autosymmetric*, $0 \leq k \leq n$, if the largest vector subspace L_S contained in S_f has dimension k.

Before presenting the algorithms for computing L_S, the restriction of f and the reduction equations, let us discuss how this definition extends the previous one given in [7], allowing to obtain a greater degree of autosymmetry. Let k_0 and k_1 be the autosymmetry degrees of the completely specified functions f_0 and f_1 with on-set f^{on} and $f^{on} \cup f^{dc}$, respectively.

Proposition 1: The dimension k of L_S is greater of equal than k_0 and k_1.

Proof. Follows immediately from Lemma 1 and from the fact that L_S is a vector subspace of highest dimension contained in S_f. ■

Finally, notice that the largest vector subspace included in a given set is not unique. For instance, the set $S = \{00, 01, 10\}$ contains the two vector subspaces $\{00, 01\}$ and $\{00, 10\}$, both of highest dimension 1. However, regardless of which subspace is selected, the degree of autosymmetry does not change.

A. Computation of S and L_S

Let $f = (f^{on}, f^{dc}, f^{off})$ be an incompletely specified function depending on n binary variables. The closure set S_f of f can be computed implicitly in a very efficient way using Binary Decision Diagrams (BDDs), extending the approach proposed in [8].

The algorithm starts from two BDDs representing the completely specified functions f_0 and f_1 with on-set $f_0^{on} = f^{on}$ and $f_1^{on} = f^{on} \cup f^{dc}$, respectively.

Algorithm 1: Algorithm for computing the closure set of an incompletely specified function f.

ClosureSet (f_0, f_1)

Input: The functions f_0 and f_1, in BDD forms.
Output: The reduced BDD representation of S_f

Let $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ be a Boolean vector
$u \leftarrow$ BDD for $f_1(x_1 \oplus \alpha_1, \ldots, x_n \oplus \alpha_n)$
$v \leftarrow$ BDD for $f_0 \Rightarrow u$
$S_f \leftarrow$ BDD for $\forall (x_1, \ldots, x_n)\ v$
return S_f

This algorithm makes use of BDD operators included in standard synthesis tools that use BDDs, and its complexity is polynomial in the size of the BDD for f^{on} and $f^{on} \cup f^{dc}$. More precisely, the time complexity is $O(n \cdot (G_0 \cdot G_1)^2)$, where n is the number of input variables, and G_0 and G_1 denote the sizes of the BDDs for f_0 and f_1. Indeed, if we choose the ordering $\alpha_1, x_1, \alpha_2, x_2, \ldots, \alpha_n, x_n$, then the size G_u of the BDD u for $f_1(x_1 \oplus \alpha_1, \ldots, x_n \oplus \alpha_n)$ is at most $O(G_1)$, since each node x_i in the BDD for $f_1(x_1, \ldots, x_n)$ can be replaced with three nodes representing the EXOR between x_i and α_i. Finally, the *imply* operator has complexity $O(G_0 \cdot G_u) = O(G_0 \cdot G_1)$, and the *forall* operator has complexity $O(n \cdot (G_0 \cdot G_1)^2)$ (see [8] for more details).

Once S_f has been computed, we must extract the *largest vector space L_S contained* in S_f. This can be a quite difficult task, unlike the 'complementary' problem of computing the *smallest* (and unique) *vector space V containing S_f*, problem that can be solved in polynomial time applying the *Gauss-Jordan elimination* procedure [14] to compute a basis for V.

An enumerative algorithm for this problem could consist in an exhaustive search of a vector subspace included in S_f, in order of decreasing dimension starting from $d = \lfloor \log_2 |S_f| \rfloor$. This algorithm has a time complexity that grows exponentially, as it requires to enumerate the subsets of i linearly independent vectors in S_f, whose number is upper bounded by the *Gaussian binomial coefficient* $\binom{r}{i}_2 = O(2^{i(r-i)})$, where r is the maximum number of linearly independent vectors in S_f, and $1 \leq i \leq d$ (see [10], [11] for more details).

Example 5: Consider the set S_f from Example 4. Since $\lfloor \log_2 |S_f| \rfloor = \lfloor \log_2 6 \rfloor = 2$, the algorithm starts considering the vector subspaces generated by sets of 2 independent vectors in S_f. It stops at the subset $B = \{0011, 1001\}$ that generates the 2-dimensional vector subspace $L_S = \{0000, 0011, 1001, 1010\}$, which is contained in S_f.

Alternatively, one can build L_S incrementally, in an heuristic way. Recall from Lemma 1 that S_f contains the two vector subspaces L_0 and L_1 associated to the functions f_0 and f_1 with on-set $f_0^{on} = f^{on}$ and $f_1^{on} = f^{on} \cup f^{dc}$, respectively.

Let us denote by L the largest subspace between L_0 and L_1. Thus, the idea is to start from L, and to increase, if possible, its dimension adding to it the affine space $v \oplus L$, where $v \in S_f \backslash L$.

Algorithm 2: Heuristic algorithm for computing L_S.

VECTORSUBSPACE (S_f, L_0, L_1)

Input: Closure set S_f, vector subspaces L_0, L_1
Output: A vector subspace L contained in S_f

$L \leftarrow$ largest subspace between L_0 and L_1
$S \leftarrow S_f \setminus L$
while ($S \neq \emptyset$)
  Let β be a vector in S
  $A \leftarrow \beta \oplus L$
  if ($A \subseteq S_f$) $L \leftarrow L \cup A$
  $S \leftarrow S \setminus A$
return L

Observe that the test and the body of the *while* loop can be evaluated and executed in time polynomial in the size of the BDD representing S, L and S_f. Thus, the time complexity of the heuristic mainly depends on the number of iterations of the *while* loop, which can be $O(|S_f|)$, and the worst case occurs when L_0 and L_1 contain only the zero vector.

Example 6: Consider again our running example. Since the function f_0, with on-set f^{on} is 1-autosymmetric with vector space $L_0 = \{0000, 0011\}$, while the function f_1 with $f_1^{on} = f^{on} \cup f^{dc}$ is not autosymmetric (see Examples 1 and 3), Algorithm 2 tries to expand L_0. If at the first iteration of the while, the algorithm chooses the vector $\beta = 1001$ from S, it computes the affine space $A = 1001 \oplus L_t = \{1001, 1010\}$ which is contained in S_f. So, the algorithm builds the 2-dimensional vector space $L = \{0000, 0011, 1001, 1010\}$. In this case, Algorithm 2 finds the largest subspace contained in S_f. The same happens for the choice $\beta = 1000$, which leads to the 2-dimensional vector space $L' = \{0000, 0011, 1000, 1011\}$, also contained in S_f.

Unlike the enumerative algorithm, this heuristic may find a not-optimal solution. However, even if the vector space L_S computed using Algorithm 2 is not the largest one, its dimension is by construction greater or equal to the dimension of L_0 and L_1, so that the autosymmetry degree derived for f (see Definition 4) is always greater or equal than the autosymmetry degrees of the two completely specified functions with on-set f^{on} and $f^{on} \cup f^{dc}$.

B. Computation of the restriction f_k

Once L_S has been computed, the canonical variables and the reduction equations can be immediately derived applying the algorithms shown in [7], [8]. Instead, the computation of the restriction of f requires some modifications, as it is no longer sufficient to project the points of the on-set of f in the subspace where all the canonical variables are set to 0.

Consider for instance the vector subspace L_S from Example 5. The canonical variables of L_S are x_1 and x_3. However, there are only two points in f^{on} where all canonical variables are 0, and the union of two affine spaces over L_S would contain 8 minterms only, while $|f^{on}| = 10$. On the other hand, if we consider also the don't care-set, we have two additional minterms with both canonical variables set to 0, but now the union of four affine spaces over L_S would contain 16 minterms, while $|f^{on} \cup f^{dc}| = 15$. Therefore, before executing the projection, we need to select the don't cares of f that should be moved to the on-set in order to derive a completely specified autosymmetric function h with on-set $f^{on} \subseteq h^{on} \subseteq f^{on} \cup f^{dc}$ and vector space $L_h = L_S$. This task can be done exploiting the knowledge of the vector space L_S. The idea is to build h adding to f^{on} the affine subspaces of L_S computed using *all* on-set points of f, as shown in the following algorithm.

Algorithm 3: Algorithm for computing the k-autosymmetric function h s.t. $f^{on} \subseteq h^{on} \subseteq f^{on} \cup f^{dc}$ and $L_h = L_S$.

BUILDNEWFUNCTION (f^{on}, L_S)

Input: The on-set f^{on}, and L_S (in BDD forms)
Output: The BDD representation of the k-autosymmetric function h

$h^{on} \leftarrow \emptyset$
$t \leftarrow f^{on}$
while ($t \neq \emptyset$)
  Let α be a vector in t
  $A \leftarrow \alpha \oplus L_S$
  $h^{on} \leftarrow h^{on} \cup A$
  $t \leftarrow t \setminus A$
return h

The body of *while* loop can be executed in time polynomial in the size of the BDD representing the sets f_0, h, and L_S. Thus, the time complexity mainly depends on the number of iteration of the *while* loop, which can be upper bounded by $O(|f^{on}|)$, with the worst case occurring when L_S has a low dimension. The correctness of Algorithm 3 is proved in the following theorem.

Theorem 1: Let $f = (f^{on}, f^{dc}, f^{off})$ be an incompletely specified function, let f_0 be the completely specified function with on-set $f_0^{on} = f^{on}$, and let L_S be a k-dimensional vector space contained in the closure set S_f of f. Algorithm 3 computes a k-autosymmetric function $h = (h^{on}, h^{off})$ s.t. $f^{on} \subseteq h^{on} \subseteq f^{on} \cup f^{dc}$ and $L_h = L_S$.

Proof. The function h is k-autosymmetric with $L_h = L_S$ as its on-set is, by construction, the union of affine spaces over the same vector space L_S. Thus, we only need to prove that $f^{on} \subseteq h^{on} \subseteq f^{on} \cup f^{dc}$. We first prove that $f^{on} \subseteq h^{on}$. Let $w \in f^{on}$. If w is explicitly chosen in the algorithm to build the affine space A (t is equal to f^{on} at the beginning of the algorithm), then we immediately have $w \in h^{on}$. Otherwise, if w is not chosen, it means that it was included in another affine space A already eliminated from t and added to h^{on}. Thus, $w \in h^{on}$. Finally, the inclusion $h^{on} \subseteq f^{on} \cup f^{dc}$ follows from the fact that $L_s \subseteq S_f$ and from the definition of the closure set S_f (see Definition 3). ■

Finally, once h has been derived, the restriction f_k of the original function f can be immediately computed projecting the points of the on-set of h in the subspace where all the canonical variables of L_S are set to 0.

Example 7: Consider the function f in our running example. If we take the vector $\alpha = 0100$ from its on-set, we compute the space $A = 0100 \oplus L_S = \{0100, 0111, 1101, 1110\}$, which

TABLE I
COMPARISON OF THE PROPOSED METHOD AND THE ALL DON'T CARES TO 0 OR TO 1 STRATEGIES.

	All don't' cares to 0 [7], [9]			All don't' cares to 1 [7], [9]			Proposed heuristic		
Bench(in, out)	**Average k**	**Max k**	**Time (ms)**	**Average k**	**Max k**	**Time (ms)**	**Average k**	**Max k**	**Time (s)**
alu2(10. 8)	2.750	6	1.805	1.125	6	2.513	3.375	7	8.026
alu3 (10. 8)	2.750	6	1.095	0.625	2	2.836	2.750	6	2.313
apla(10. 12)	0.166	1	1.439	0.083	1	2.713	4.417	6	38.419
b10(15. 11)	2.545	15	5.534	1.181	3	6.710	2.636	15	8.060
bcc(26. 45)	10.755	14	36.243	10.755	14	39.36	10.755	14	70.267
bench1(9. 9)	0.000	0	8.855	0.000	0	19.351	0.000	0	0.001
dk17(10. 11)	0.090	1	0.877	0.090	1	1.029	5.727	7	25.767
dk27(9. 9)	0.333	1	0.468	0.111	1	0.503	6.444	8	5.346
dk48(15. 17)	0.176	2	1.620	0.058	1	1.805	11.588	12	119.311
ex1010(10. 10)	0.000	0	44.253	0.000	0	44.957	0.000	0	0.144
exam(10. 10)	1.000	10	3.342	1.100	10	4.545	4.200	10	33.144
exp(8. 18)	1.111	8	1.605	0.000	0	3.232	1.444	8	2.133
exps(8 . 38)	0.184	1	5.946	0.131	1	7.379	0.395	4	2.490
l8err(8. 8)	3.000	8	0.884	4.000	8	0.501	4.000	8	0.520
p3(8. 14)	1.428	8	1.153	0.000	0	1.625	3.786	8	9.376
pdc(16. 40)	3.275	14	10.387	0.000	0	22.038	9.875	14	757.339
t2(17. 16)	8.625	12	1.213	8.187	11	1.248	9.500	13	30.687
t4(12. 8)	2.250	3	0.810	2.250	3	0.900	6.875	8	20.698
test3(10. 35)	0.000	0	104.855	0.000	0	227.414	0.000	0	550.411
test4(8. 30)	0.000	0	19.946	0.000	0	8.826	0.633	1	29.652
x1dn(27. 6)	9.166	13	2.894	3.000	3	5.923	12.500	16	557.006
Average values	1.798	5	9.105	1.186	2	14.513	3.864	6	79.611

is added to h^{on} and subtracted from f^{on}. Suppose that at the second iteration, the algorithms picks $\alpha = 0101$. Then, it computes $A = \{0101, 0110, 1100, 1111\}$, which is again added to h^{on} and subtracted from f^{on}. At the third iteration, there are only two points left in f^{on}: 1001 *and* 1010*. If the algorithm picks* 1001*, it computes $A = \{1001, 1010, 0000, 0011\}$. Observe that adding A to h^{on} has the effect of moving the two don't cares* 0000 *and* 0011 *to the on-set. At this point $f^{on} = \emptyset$, so the algorithm terminates and returns h^{on}. The function h now contains three vectors with the two canonical variables x_1 and x_3 set to* 0 *(*0000*,* 0100*, and* 0101*). The restriction of h is then given by $h_2 = \{00, 10, 11\}$, as depicted in Figure 2.*

IV. EXPERIMENTAL RESULTS

In this section we discuss the computational results obtained by testing the autosymmetry on incompletely specified Boolean functions. In order to test incompletely specified functions, we have considered the classical benchmark suite described in [18] and the computational experiments were performed on an Intel Core i5-7200U (2.50 GHz x 2) PC with 4 GB of RAM on a virtual machine running OS Ubuntu 64-bit. The experiments consider the subset of benchmarks containing outputs with don't care conditions. The main aim is to compare the autosymmetry degree computed by the proposed algorithm and the degree computed by considering all don't care conditions in the off-set or all in the on-set (using the autosymmetry test described in [7], [9]). Table I reports this comparison. The first column reports the name of the function considered (benchmark function together with the number of inputs and outputs). The following three columns report the average autosymmetry degree, the maximum autosymmetry degree and the computational time for the strategy where the don't cares are fixed to 0. The following three columns contains the same results for the case where the don't cares are fixed to 1. Finally, the last three columns describe the results for our proposed strategy. We can note that the experimental results are coherent with the theoretical intuition, which states that an accurate choice of the don't cares to be set to 1, and to be set to 0, can considerably help in the maximization of the autosymmetry degree of a Boolean function. In summary, the 77% of the considered benchmarks has an improvement in the average autosymmetric degree with respect to the constant cases. While the average autosymmetric degree for the case where the don't care are set to 0 (resp., 1) is 1.8 (resp., 1.2), the average autosymmetric degree using our strategy is 3.9, at the obvious cost of an increased running time (see Table I).

V. CONCLUSION

This paper has defined and studied the autosymmetry degree of incompletely specified Boolean functions. A very interesting theoretical problem remains, i.e., the (efficient) definition of the bigger vector space contained in a given set of Boolean vectors. Another possible future direction is the study of the properties of BDDs representing affine spaces, in order to further improve the efficiency of the proposed methods, i.e., by the definition of an ad hoc variable ordering.

REFERENCES

[1] D. Alexandrescu, M. Altun, L. Anghel, A. Bernasconi, V. Ciriani, L. Frontini, and M. B. Tahoori, "Logic synthesis and testing techniques for switching nano-crossbar arrays," *Microprocess. Microsystems*, vol. 54, pp. 14–25, 2017.

[2] A. Bernasconi, V. Ciriani, F. Luccio, and L. Pagli, "Fast Three-Level Logic Minimization Based on Autosymmetry," in *ACM/IEEE 39th Design Automation Conference (DAC)*, 2002, pp. 425–430.

[3] A. Bernasconi, S. Cimato, V. Ciriani, and M. C. Molteni, "Multiplicative Complexity of Autosymmetric Functions: Theory and Applications to Security," in *ACM/IEEE 58th Design Automation Conference (DAC)*, 2020.

[4] A. Bernasconi, V. Ciriani, L. Frontini, V. Liberali, G. Trucco, and T. Villa, "Logic synthesis for switching lattices by decomposition with p-circuits," in *2016 Euromicro Conference on Digital System Design, DSD*, 2016, pp. 423–430.

[5] A. Bernasconi, V. Ciriani, L. Frontini, and G. Trucco, "Composition of switching lattices and autosymmetric boolean function synthesis," in *Euromicro Conference on Digital System Design, DSD 2017, Vienna, Austria, August 30 - Sept. 1, 2017*, 2017, pp. 137–144.

[6] ——, "Composition of switching lattices for regular and for decomposed functions," *Microprocess. Microsystems*, vol. 60, pp. 207–218, 2018.

[7] A. Bernasconi, V. Ciriani, F. Luccio, and L. Pagli, "Three-Level Logic Minimization Based on Function Regularities," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 22, no. 8, pp. 1005–1016, 2003.

[8] ——, "Exploiting regularities for boolean function synthesis," *Theory Comput. Syst.*, vol. 39, no. 4, pp. 485–501, 2006.

[9] ——, "Synthesis of autosymmetric functions in a new three-level form," *Theory Comput. Syst.*, vol. 42, no. 4, pp. 450–464, 2008.

[10] J. Goldman and G. Rota, *The Number of Subspaces of a Vector Space*. Defense Technical Information Center, 1969.

[11] D. E. Knuth, "Subspaces, subsets, and partitions," *Journal of Combinatorial Theory, Series A*, vol. 10, no. 2, pp. 178 – 180, 1971.

[12] V. N. Kravets and K. A. Sakallah, "Constructive library-aware synthesis using symmetries," in *2000 Design, Automation and Test in Europe (DATE)*. IEEE Computer Society / ACM, 2000, pp. 208–213.

[13] V. Kravets and K. Sakallah, "Generalized Symmetries of Boolean Functions," in *International Conference on Computer-Aided Design (ICCAD)*, 2000, pp. 526–532.

[14] R. Liebler, *Basic Matrix Algebra with Algorithms and Applications*. Chapman & Hall/CRC., 2003.

[15] F. Luccio and L. Pagli, "On a New Boolean Function with Applications," *IEEE Transactions on Computers*, vol. 48, no. 3, pp. 296–310, 1999.

[16] E. Testa, S. L. Noor, O. Zografos, M. Soeken, F. Catthoor, A. Naeemi, and G. D. Micheli, "Multiplier architectures: Challenges and opportunities with plasmonic-based logic : (special session paper)," in *2020 Design, Automation & Test in Europe Conference & Exhibition, DATE 2020, Grenoble, France, March 9-13, 2020*. IEEE, 2020, pp. 133–138.

[17] E. Testa, M. Soeken, L. G. Amarù, and G. D. Micheli, "Reducing the multiplicative complexity in logic networks for cryptography and security applications," in *Proceedings of the 56th Annual Design Automation Conference 2019, DAC 2019, Las Vegas, NV, USA*, 2019, p. 74.

[18] S. Yang, "Logic synthesis and optimization benchmarks user guide version 3.0," Microelectronic Center, User Guide, 1991.