Università di Pisa
Dipartimento di Informatica

Technical Report: TR-06-05

DRedSOP: Synthesis of a new class of regular functions

Anna Bernasconi,
Department of Computer Science
University of Pisa
56100 Pisa, Italy
annab@di.unipi.it

Valentina Ciriani,
Department of Information Technologies
University of Milano
26013 Crema (CR), Italy
ciriani@dti.unimi.it

May 29, 2006

address: via F. Buonarroti 2, 56127 Pisa, Italy. tel: +39 050 2212700 fax: +39 050 2212726

DRedSOP: Synthesis of a new class of regular functions

Anna Bernasconi,
Department of Computer Science
University of Pisa
56100 Pisa, Italy
annab@di.unipi.it

Valentina Ciriani,
Department of Information Technologies
University of Milano
26013 Crema (CR), Italy
ciriani@dti.unimi.it

May 29, 2006

Abstract

In this paper we characterize and study a new class of regular Boolean functions called D-reducible. A D-reducible function, depending on all its n input variables, can be studied and synthesized in a space of dimension strictly smaller than n. A D-reducible function can be efficiently decomposed, giving rise to a new logic form, that we have called DRedSOP. This form is shown here to be generally smaller than the corresponding minimum SOP form. Experimental evidence shows that such functions are rather common and D-reducibility can be tested very quickly.

1 Introduction

Synthesis of Boolean functions is a classical problem in Computer Science. We study this problem here for functions exhibiting a particular type of regularity (*D-reducibility*) that, as we will see, is sufficiently common to make the case interesting.

Informally, *D-reducible* (or *Dimention-reducible*) functions are functions whose points are contained in a space A strictly smaller than the whole Boolean cube $\{0,1\}^n$.

The D-reducibility of a function f can be exploited in the minimization process: the idea is to minimize the projection f_A of f onto A, instead of f. This approach thus requires two steps: *(i)* deriving the space A and the projection f_A; *(ii)* minimizing f_A in a given logic frameworks. In this paper we focus on the standard SOP (Sum of Products) minimization, and we prove how our approach to the synthesis of D-reducible functions often turns out to be convenient. Moreover the algorithm deriving the minimum space containing f has time complexity polynomial in the representation of f (i.e., the initial SOP form of f.)

Note that D-reducible functions depend in general on all their n input variables, however we are able to study them in a space of dimension strictly smaller than n.

As this study will need non trivial formal tools, we start here by giving an intuitive presentation of D-reducibility. Consider the function $f = \{0010, 0100, 0110, 1011, 1101\}$ in the Karnaugh map on the left side of Figure 1. The function f is D-reducible, i.e., we can project it into a space of dimension three (the space marked with circles in the Karnaugh map).

We can therefore study the new function f_A that depends only on three variables, represented in the Karnaugh map on the right side of the figure. Notice that f and f_A have the same number of points, but these are now compacted in a smaller space. If we synthesize f and f_A in the classical SOP framework we obtain $f = \overline{x}_1x_3\overline{x}_4 + \overline{x}_1x_2\overline{x}_4 + x_1\overline{x}_2x_3x_4 + x_1x_2\overline{x}_3x_4$, and $f_A = \overline{x}_2x_3 + \overline{x}_1x_2 + x_2\overline{x}_3$. (Note that f depends on all the variables $x_1, \ldots x_4$.) The new and more compact form for f is then $f = (x_1 \oplus \overline{x}_4)(\overline{x}_2x_3 + \overline{x}_1x_2 + x_2\overline{x}_3)$. The EXOR $(x_1 \oplus \overline{x}_4)$ represents the new Boolean space where we study f_A. Figure 2 shows the resulting network for the function f.

The key idea of this paper is that if we project the points of a function in smaller Boolean space we have the chance of reducing the hamming distance between its points in order to merge them in bigger cubes in the final SOP form. For example, consider the point 1101 in the Karnaugh map on the left side of Figure 1, its corresponding product $x_1x_2\overline{x}_3x_4$ is prime since no other point can be merged with 1101. If we project

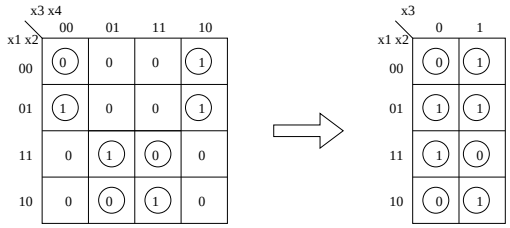

Figure 1: *Karnaugh maps of a D-reducible function f and its corresponding projection f_A.*

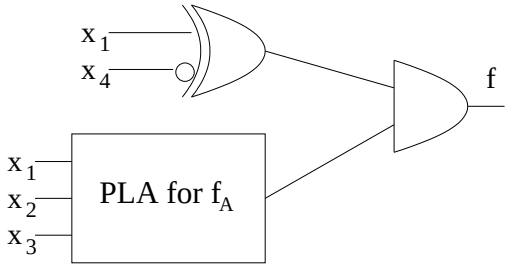

Figure 2: *Network for the D-reducible function f of Figure 1, represented by $(x_1 \oplus \overline{x}_4)(\overline{x}_2 x_3 + \overline{x}_1 x_2 + x_2 \overline{x}_3)$.*

the function in the new space $(x_1 \oplus \overline{x}_4)$ its corresponding point 110 can be merged with 010 giving rise to the prime product $x_2\overline{x}_3$. Observe that simple projections with single literals as $x_i \cdot f$ do not change the hamming distance between points, while projections with EXORs do.

In this paper we describe a simple test that establishes whether a function is D-reducible and computes the smallest space that contains it. We than propose a new three level logic form (*DRedSOP*) for f, which is an AND of some EXOR factors (or literals) representing the projection space A, and the SOP expression for f_A. Figure 2 shows a DRedSOP network.

Our experimental results show that about 70% of the functions in the classical Espresso benchmark suite have at least one output that is D-reducible: although D-reducible functions form a subset of all possible Boolean functions, a great amount of standard functions of practical interest falls in this class. In general we can represent any function as $A \cdot f$ where A is a Boolean subspace of $\{0,1\}^n$. If f is D-reducible the space A is strictly contained in $\{0,1\}^n$, otherwise f is the function $1 \cdot f$ where 1 represents the entire Boolean space $\{0,1\}^n$ (i.e., $A = \{0,1\}^n$.) We can view this synthesis method as a special Boolean factorization where instead to literal terms we have EXORs. Factorization of literal terms is a widely studied field in multi-level logic [3, 17]. Finally note that D-reducibility and autosymmetry (described in [1]) are different regularities, since autosimmetric functions can be studied in a new space whose variables are EXOR combinations of the original ones, and D-reducible functions are studied in a projection space producing an expression where the EXOR gates are in AND with a SOP form. However, D-reducible functions have an interesting connection with autosymmetric functions through their Walsh transform.

The paper is organizes as follows. In the first section we review some basic definition and properties of affine spaces. In Section 3 we formally define D-reducible functions. In section 4 we propose a synthesis algorithm for DRedSOP forms. Finally, in Section 5 we describe our experimental results.

2 Preliminaries

In this section we briefly review some basic notions on affine spaces that are useful in the sequel (for a more detailed introduction on affine spaces see [4, 5]).

We work in a Boolean space $\{0,1\}^n$ described by n variables $x_1, x_2, \ldots, x_n$, where each point is represented by a binary vector of n *components*. Hereafter, we shall use the terms vector and point with the same meaning.

In the space $\{0,1\}^n$, an *EXOR factor* is an EXOR (or modulo 2 sum), denoted by $\oplus$, of variables, one of

which possibly complemented (an EXOR with just one literal corresponds to the literal itself). Let us now extend the symbol $\oplus$ to denote the elementwise EXOR between two vectors. Then $\alpha \oplus \beta$ is the vector obtained from β complementing in it the elements corresponding to the 1's of α. For example $1011 \oplus 0111 = 1100$.

We recall that, a vector subspace V of the vector space $(\{0,1\}^n, \oplus)$ is a subset of $\{0,1\}^n$ containing the zero vector $\mathbf{0} = 00\ldots0$, such that for each v_1 and v_2 in V we have that $v_1 \oplus v_2$ is in V. Note that a vector subspace of a vector space is a vector space itself.

Example 1 *The set $V = \{000, 001, 010, 011\}$ is a vector subspace of $(\{0,1\}^3, \oplus)$. In fact, $\mathbf{0} = 000$ is in V, and $001 \oplus 010 = 011 \in V$, $001 \oplus 011 = 010 \in V$, $010 \oplus 011 = 001 \in V$, $001 \oplus 000 = 001 \in V$, etc.*

Each vector subspace V of $(\{0,1\}^n, \oplus)$ contains 2^k vectors, where k is a positive integer. We say that V has dimension k or is k-dimensional (shortly $\dim(V) = k$). The subspace of Example 1 has 2^2 points, and its dimension is 2.

A k-dimensional vector space V is generated by a basis B containing k vectors. Each vector v in a basis B is linearly independent of all the other vectors in B, i.e., v is not generated by any EXOR combination of the other vectors in B. A vector space, in general, has not a unique basis. In fact, a set of k linearly independent vectors in a vector space V of dimension k always forms a basis of V. For example the vector space $V = \{000, 001, 010, 011\}$ has three different bases, namely $\{010, 011\}$, $\{001, 010\}$, and $\{001, 011\}$.

Given a vector subspace V of $(\{0,1\}^n, \oplus)$, and a point α in $\{0,1\}^n$, we build an *affine space* performing the EXOR between α and each point of V. Formally we pose:

Definition 1 *Let V be a vector subspace of $(\{0,1\}^n, \oplus)$, and let $\alpha \in \{0,1\}^n$ be a Boolean point. The set $A = \alpha \oplus V = \{\alpha \oplus v \mid v \in V\}$ is an* affine space *over V with* translation point *α.*

Example 2 *Consider the vector space $V = \{000, 010, 011, 001\}$ and the vector $\alpha = 100 \in \{0,1\}^3$. The set $A = \alpha \oplus V = 100 \oplus V = \{100, 110, 111, 101\}$ is an affine space over V. Note that we can choose α as any vector of A. In this example $A = 100 \oplus V = 110 \oplus V = 111 \oplus V = 101 \oplus V$.*

An interesting property of affine spaces is that $(\alpha \oplus V \equiv V) \Leftrightarrow \alpha \in V$. For example, let V be the vector space {000, 010, 011, 001}, then $A = 010 \oplus V = \{000, 010, 011, 001\} = V$, because $010 \in V$. Clearly, a vector space is an affine space.

If A is an affine space, there exists a *unique* vector space V such that for all α in A, $A = \alpha \oplus V$. Such space can be computed as $V = \alpha \oplus A$, where α is *any* point of A. Moreover, if A and A' are affine spaces over the same vector space V, then A and A' either coincide or are disjoint.

Example 3 *Consider the affine space $A = \{0010, 0011, 0100, 0101\}$. Our aim is to find the unique vector space V such that $A = \alpha \oplus V$. Choosing $\alpha = 0010 \in A$, we have: $V = \alpha \oplus A = 0010 \oplus A = \{0000, 0001, 0110, 0111\}$. It is easy to verify that choosing a different vector in A as translation point we achieve the same result, i.e., $V = 0010 \oplus A = 0011 \oplus A = 0100 \oplus A = 0101 \oplus A$.*

Let $A = \alpha \oplus V$ be an affine space. The *dimension* of A is the dimension of the vector space V. Since the translation point α can be chosen as any vector of A, and the vector space V can be represented by any of its bases, we must define a unique representation of A. To this end we introduce some notation. A set of k points in $\{0,1\}^n$ (e.g., an affine or vector space) can be arranged in a $k \times n$ matrix whose rows correspond to the points, and whose columns correspond to the variables $x_1, x_2, \ldots, x_n$ (see for example Figure 3).

A matrix of points in $\{0,1\}^n$ is in *binary order* if its rows (points) are sorted as increasing binary numbers. For example the two matrices in Figure 3 are in binary order.

Definition 2 *Let A be an affine space over a vector space V. The* canonical translation point *α_A is the minimum point of A in binary order.*

For example, 00001 is the translation point of the affine space A in Figure 3.

Definition 3 *Let V be a vector space whose matrix is sorted in binary order, with the rows indexed from 0 to $2^k - 1$. And let $A = \alpha \oplus V$ be an affine space over V. The set of points of V with indices $2^0, 2^1, \ldots, 2^{k-1}$ will be called the* canonical basis *B_A of V (or, equivalently, of A).*

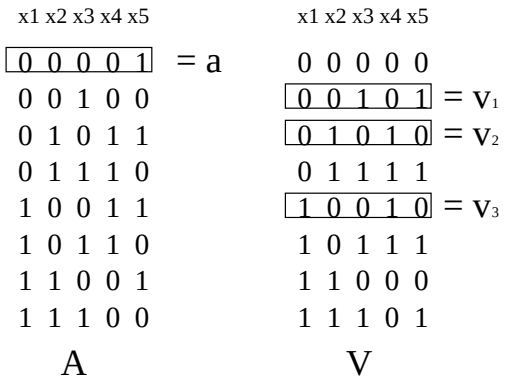

Figure 3: *An affine space $A = a \oplus V$ and the corresponding vector space V. The points v_1, v_2, v_3 form the canonical basis for V, and $a = \alpha_A$ is the canonical translation point of A (note that $V = a \oplus A$).*

For example, the vectors $\{00101, 01010, 10010\}$ in rows $1, 2, 4$ form the canonical basis of the affine space in Figure 3.

As proved in [4], the canonical basis B_A is indeed a basis of V in the algebraic sense, i.e., the points of B_A are linearly independent.

Definition 4 *The* canonical representation (α_A, B_A) *of an affine space is given by its canonical translation point together with its canonical basis.*

For example, the canonical representation of the affine space A in Figure 3 is $\alpha_A = 00001$ and $B_A = \{00101, 01010, 10010\}$.

We can note that the canonical basis corresponds to the basis derived by a matrix in *reduced row echelon form* [5]. The reduced row echelon form of a matrix is unique. Thus the canonical representation uniquely specifies an affine space (see [4] for more details).

We partition now the Boolean variables of an affine space in two sets as follows.

Definition 5 *Let $A = \alpha_A \oplus V$ be an affine space with canonical basis $v_1, \ldots, v_k$. For each v_i, let x be the variable corresponding to the first* 1*-component from left of v_i. The variable x is called* canonical variable*. The variables that are not canonical for any vector in the canonical basis are called* non-canonical*.*

For example in Figure 3, the canonical variables are x_3 for vector $v_1 = 00\mathbf{1}01$, x_2 for vector $v_2 = 0\mathbf{1}010$, and x_1 for vector $v_3 = \mathbf{1}0010$. The non-canonical variables are the remaining variables x_4 and x_5.

Observe that the canonical variables are the truly independent variables in the space A, in the sense that they can assume all possible combinations of 0-1 values. On the contrary, on A the non-canonical variables are not independent because they can be defined as linear combinations (i.e., EXORs) of the canonical ones.

This fact is clearly expressed by the characteristic function of an affine space, represented by an algebraic expression involving AND and EXOR operators. In fact, as shown in [4], an affine space can be represented by a simple expression (called *pseudoproduct*) consisting in an AND of EXORs or literals. For example $x_2\overline{x}_1(x_3 \oplus x_4 \oplus \overline{x}_5)(x_3 \oplus x_7)$ is a pseudoproduct.

The characteristic function of an affine space can be expressed in various ways as a pseudoproducts. Among these forms, a canonical (CEX) expression given in [14] is of particular relevance. In the following definition we explain how to derive the CEX expression of a given affine space A (the direct connection between the canonical basis of an affine space and its CEX expression is detailed in [4]).

Definition 6 *Let $A = \alpha_A \oplus V$ be an affine space. The* CEX(A) expression *of A is given by a product of EXOR factors such that:*

1. *there is an EXOR factor for any non-canonical variable;*
2. *the only variables that appear in the EXOR factor corresponding to the non-canonical variable x are the canonical variables (if any) connected with x in the canonical basis B_A;*

3. *all the canonical variables are not complemented; a non-canonical variable is complemented in its EXOR factor iff its corresponding component in α_A is* 0.

Example 4 *Let $A = \alpha_A \oplus V$ be an affine space with canopnical basis $B_A = \{001010, 010110, 100000\}$ and translation point$\alpha_A = 000100$. The first vector in B_A shows that the canonical variable x_3 is in the EXOR factor corresponding to the non-canonical variable x_5. Vector* 010110 *shows that the canonical variable x_2 is in the EXOR factors corresponding to the non-canonical variables x_4 and x_5. Vector* 100000 *shows that the canonical variable x_1 does not appear in any EXOR factor. By point* 1 *of Definition 6, we have three EXOR factors (one for each non-canonical variable): 1) $x_2 \oplus x_4$, corresponding to the non-canonical variable x_4 and containing the canonical variable x_2; 2) $x_2 \oplus x_3 \oplus \overline{x}_5$, corresponding to the non-canonical variable x_5 and containing the canonical variables x_2 and x_3, and 3) $\overline{x}_6$, corresponding to the non-canonical variable x_6 and not containing any canonical variable. Note that, by point* 3 *of Definition 6, the vector $\alpha_A = 000100$ shows that the non-canonical variable x_4 is not complemented while x_5 and x_6 are complemented. Therefore the CEX expression is $(x_2 \oplus x_4)(x_2 \oplus x_3 \oplus \overline{x}_5)\overline{x}_6$.*

3 D-reducible functions

In this section we define the class of *D-reducible Boolean functions*, and analyze their properties. Informally, D-reducible functions are functions whose points are contained in an affine space strictly smaller than the whole Boolean cube $\{0,1\}^n$.

Definition 7 *The Boolean function $f : \{0,1\}^n \to \{0,1\}$ is* D-reducible *if $f \subseteq A$, where $A \subset \{0,1\}^n$ is an affine space of dimension strictly smaller than n.*

Definition 8 *Let f be a D-reducible function. The smallest affine space containing f is called its* associated affine space.

Proposition 1 *The smallest affine space containing a Boolean function f is unique.*

Proof. Let us suppose that $f \subseteq A_1$ and $f \subseteq A_2$. We first observe that A_1 and A_2 must be affine spaces over the same vector space (this can be verified by some algebraic manipulation). Thus the thesis easily follows since two affine spaces over the same vector space either coincide or are disjoint, and in our case $A_1 \cap A_2 \neq \emptyset$. ■

Note that A can be a vector space. The reason why we consider affine spaces, instead of vector spaces, is that the smallest affine space containing a function f can have dimension a unit smaller than the dimension of the smallest vector space containing f. For instance, the smallest vector space containing the parity function is $\{0,1\}^n$, while the smallest affine space is the set of binary vectors corresponding to the parity itself, i.e., the set of vectors with odd hamming weight, which has dimension $n-1$.

Proposition 2 *Let f be a D-reducible function and A its associated affine space. Then $f = \chi_A \cdot f_A$, where χ_A is the characteristic function of A and f_A is the projection of f on A, i.e., $f_A \subseteq \{0,1\}^{\dim A}$ is the characteristic function of the set $f \cap A$.*

Proof. For any Boolean function f and any subset $A \subseteq \{0,1\}^n$, we can decompose f as $f = \chi_A \cdot f_A + \chi_{\overline{A}} \cdot f_{\overline{A}}$, where f_A and $f_{\overline{A}}$ are the projections of f on A and $\overline{A}$, respectively: $f_A = \chi_{f \cap A}$, $f_{\overline{A}} = \chi_{f \cap \overline{A}}$. Let f be D-reducible, and let A be its associated affine space. Since $f \subseteq A$, $f_{\overline{A}}$ is the constant zero function, and the thesis immediately follows. ■

Corollary 1 *Let f be a D-reducible function, and A its associated affine space. The function f_A depends on the $d = \dim A$ $(< n)$ canonical variables of the affine space V.*

Since A is an affine space, we finally have

Proposition 3 *The function χ_A can be expressed as a pseudoproduct.*

Example 5 *Consider the function* $f = \{0010, 0100, 0110, 1011, 1101\}$ *whose Karnaugh map is shown on the left side of Figure 1.* f *is D-reducible, and its associated affine space is described by the CEX expression* $(x_1 \oplus \overline{x}_4)$*. If we project* f *on the Boolean space of dimension 3, represented in the Karnaugh map on the left side of Figure 1 with circles, we obtain the function* $f_A = \{001, 010, 011, 101, 110\}$*, represented in the Karnaugh map on the right side of the figure, which depends only on the canonical variables of A, i.e.,* x_1, x_2*, and* x_3*.*

3.1 Relations with degenerate and autosymmetric functions

It is important to observe that D-reducible functions depend in general on all their input variables, i.e., they are not degenerate; and degenerate functions are not in general D-reducible. For instance, the function $f(x_1, x_2, x_3) = x_2 \vee x_3$ is degenerate, but not D-reducible; while the function $f(x_1, \ldots, x_n) = x_1 \oplus x_2 \oplus \ldots \oplus \overline{x}_n$ is D-reducible since it is contained in the $n-1$ dimensional space of vectors with even hamming weight, and it is not degenerate.

We now briefly discuss the relationship between the class of D-reducible functions, and the class of *autosymmetric* functions.

Autosymmetric functions, introduced in [14] and studied in [1], exhibit a regular structure that can be exploited by synthesis algorithms. We recall here their definition.

Definition 9 *A Boolean function* f *in* $\{0,1\}^n$ *is* closed under $\boldsymbol{\alpha}$*, with* $\boldsymbol{\alpha} \in \{0,1\}^n$*, if for each* $\boldsymbol{w} \in \{0,1\}^n$*,* $\boldsymbol{w} \oplus \boldsymbol{\alpha} \in f$ *if and only if* $\boldsymbol{w} \in f$ *.*

Each function is obviously closed under the zero vector $\mathbf{0}$. As proved in [14], if a function f is closed under two different vectors $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \in \{0,1\}^n$, it is also closed under $\boldsymbol{\alpha}_1 \oplus \boldsymbol{\alpha}_2$. Therefore the set L_f of all the vectors $\boldsymbol{\beta}$ such that f is closed under $\boldsymbol{\beta}$ is a vector subspace of $\{0,1\}^n$. L_f is called the *linear space* of f, and k is its *dimension.*

Definition 10 *A Boolean function* f *is* k-autosymmetric*, or equivalently* f *has* autosymmetry degree k*,* $0 \leq k \leq n$*, if its linear space* L_f *has dimension* k*.*

The intersection between the set of autosymmetric and D-reducible functions is not empty; for instance the parity function is both autosymmetric and D-reducible. However there exist D-reducible functions that are not autosymmetric, and autosymmetric functions that are not D-reducible. For instance, the function $f(x_1, x_2, x_3) = x_1 x_2 x_3$ is D-reducible but non autosymmetric, and the function $f(x_1, x_2, x_3) = (x_1 \oplus x_2) \vee x_3$ is autosymmetric but not D-reducible.

However, D-reducible functions have an interesting connection with autosymmetric functions, which can be understood by looking at their Walsh transform. The connection is expressed by the following theorems, whose proofs are omitted here for brevity.

Theorem 1 *Let* f *be a D-reducible function, and let* A *be its associated affine space. The function defined by the absolute value of the Walsh transform of* f*,* $|\hat{f}|$*, is a* k*-autosymmetric function, with* $k = n - \dim A$*, and its linear space is* $L_{|\hat{f}|} = V_A^{\perp}$*, where* V_A *denotes the vector space corresponding to* A*.*

Theorem 2 *Let* f *be a* k*-autosymmetric function. Then the characteristic function* $\chi_{\hat{f}}$ *of the support of its Walsh transform* $\hat{f}$ *is a D-reducible function, whose associated affine space is* $L_f^{\perp}$*.*

4 Synthesis of D-reducible functions

In this section we show how the property of D-reducibility can be exploited to perform the synthesis of a Boolean function. Remember that a D-reducible function f can be written as $f = \chi_A \cdot f_A$, where χ_A is the characteristic function of A and f_A is the projection of f on A (see Proposition 2). Intuitively, the idea is that of reducing the minimization of f to the minimization of f_A, which depends on less variables.

We start by showing how to efficiently test whether a function is D-reducible, and derive its associated affine space.

4.1 D-reducibility Test

Given a Boolean function $f : \{0,1\}^n \to \{0,1\}$, we can perform the D-reducibility test by applying a classical linear algebra tool: the Gauss elimination.

Let $m = |f|$. If we execute the Gauss elimination on the $m \times n$ matrix whose rows are the points of the function, we get a basis for the smallest vector space containing f.

As already noted we are interested in getting the smallest *affine* space containing f, since its dimension can be smaller. To this aim, we first note that if the zero vector is a point of f, than A is a vector space (indeed, whenever an affine space contains the zero vector, then it is actually a vector space). Otherwise, f can be contained in an affine space that is not a vector space.

We derive A performing the following steps:

1. We pick any point of f, say v, and compute the set $v \oplus f$. If the zero vector belongs to f, we choose $v = \mathbf{0}$.
2. We compute the smallest vector space V_A containing $v \oplus f$ by Gauss elimination.
3. We finally derive $A \supset f$ from V_A as $A = v \oplus V_A$, where v is the same vector chosen in the first step.

Note that, whenever f do not contain the zero vector, we can choose any point $v \in f$ in the first step without changing the result, i.e. the affine space A. This is a consequence of Proposition 1.

The time complexity of this D-reducibility test is polynomial in n and $|f|$. More precisely the computational cost of the Gauss elimination is $O(n|f|^2)$. Since $|f|$ is often exponential in the number of variables n, the complexity of the test can be exponential in n. In Section 4.4 we discuss a more efficient test that computes the associated affine space of a function f in time polynomial in the original SOP representation of f, and not in the number of its minterms.

4.2 Variable reduction

Recall that a D-reducible function f can be written as $f = \chi_A \cdot f_A$, where f_A depends only on $\dim A$ variables (see Corollary 1). Moreover χ_A, the characteristic function of the affine space A covering f, is a pseudoproduct with $(n - \dim A)$ EXOR factors, each containing a different non-canonical variable. The $\dim A$ variables on which f_A depends are exactly the canonical variables of A. Indeed the non-canonical variables depend, through the EXOR factors of χ_A, on the canonical ones.

4.3 Synthesis

We propose to synthesize a D-reducible function $f = \chi_A \cdot f_A$ as follows. We represent χ_A using its CEX, getting an EXOR-AND network. We can then synthesize f_A in any logical framework (SOP [2, 6], three-level-logic networks [7, 8, 9, 10, 11, 15, 16], etc.). The synthesis of f_A could be easier than the synthesis of f, since f_A depends on $\dim A < n$ variables. Moreover the size of the network for f_A should be smaller than the size of the corresponding network of f. Indeed f and f_A have the same number of points, but f_A is defined in a smaller space and its points are less sparse.

For example consider the function $f = \{0010, 0100, 0110, 1011, 1101\}$, whose Karnaugh map is shown on the left side of Figure 1. f is D-reducible, and its associated affine space is described by the CEX expression $(x_1 \oplus \overline{x}_4)$. We can project f on the Boolean space of dimension 3, represented in the Karnaugh map on the left side of Figure 1 with circles. We can therefore study the function f_A, represented in the Karnaugh map on the right side of the figure. f_A depends only on the canonical variables of A, i.e., x_1, x_2, and x_3.

Notice that we have the same number of points, but these are now compacted in a smaller space, i.e., the points of the function are more adiacent and we have more chance to merge them into cubes. Suppose we want to synthesize f and f_A in the classical SOP framework. We have $f = \overline{x}_1 x_3 \overline{x}_4 + \overline{x}_1 x_2 \overline{x}_4 + x_1 \overline{x}_2 x_3 x_4 + x_1 x_2 \overline{x}_3 x_4$, and $f_A = \overline{x}_2 x_3 + \overline{x}_1 x_2 + x_2 \overline{x}_3$. The new form for f is then $f = (x_1 \oplus \overline{x}_4)(\overline{x}_2 x_3 + \overline{x}_1 x_2 + x_2 \overline{x}_3)$. Figure 2 shows the resulting network for the function f.

4.4 Test from PLAs

The test algorithm described in Section 4.1 considers functions represented by their minterms. Generally Boolean functions are represented by SOP expressions containing cubes and not only minterms (e.g., PLAs). In this section we explain how to perform the D-reducibility test starting from a SOP of a function, *without generating all its minterms*. The complexity of this new version of the test becomes $O(nP^2)$, where P is the number of products in the given SOP for f. Observe that in practical cases, we have often $P << |f|$.

We describe the idea starting with an example. Let f = {000000, 001000, 010001, 010011, 011001, 011011} be represented with the SOP: $\overline{x}_1\overline{x}_2\overline{x}_4\overline{x}_5\overline{x}_6 + \overline{x}_1x_2\overline{x}_4x_6$. The product $\overline{x}_1x_2\overline{x}_4x_6$ is represented in a PLA by the row 01-0-1. For each don't care in the product, we can generate a vector composed by all zeros but a 1 in the position corresponding to the don't care. For instance, in our example we generate the vectors 001000 and 000010. These vectors would be surely generated during the Gauss elimination step. In fact we have: $001000 = 010001 \oplus 011001$ and $000010 = 010001 \oplus 010011$. The matrix to be processed by the Gauss elimination algorithm will then contain: the original vector with 0 instead of the don't cares (010001) and the new generated vectors (001000 and 000010).

Notice that a product is a particular pseudoproduct and represents an affine space $A = \alpha \oplus V_A$ of 2^d points, where d is the number of don't cares. Moreover the basis of V_A is a subset of the *standard basis* of $\{0,1\}^n$, i.e., $e_1 = 100\ldots00$, $e_2 = 010\ldots00$, ..., $e_n = 000\ldots01$. Therefore our idea is to represent a product only with $d+1$ vectors instead of 2^d minterms. These $d+1$ vectors are the basis of V_A, together with α. Moreover, we add a vector of the basis of V_A if and only if it has not been already used for representing another product. In conclusion the P products in the given SOP of f are transformed into at most $P+n$ vectors in input to the Gauss elimination algorithm.

In the former example, the first product can be represented by 000000 and 001000, and the second by 010001, 001000 and 000010. Thus, the input to the Gauss elimination step is given by the set of vectors: {000000, 001000, 010001, 000010}. Note that the vector $e_3 = 001000$ has been written only once.

4.5 Incompletely Specified D-reducible Functions

Let us now briefly discuss how to extend the notion of D-reducibility to functions with don't care points (denoted by $*$.)

The extension to incompletely specified functions is important because synthesis techniques usually benefit from the presence of don't cares; and the synthesis of D-reducible functions would analogously be greatly improved by projecting onto A also the don't care set.

Our current approach to the synthesis of D-reducible functions is rather restrictive, as it projects onto the affine space A only the one-set of a D-reducible function.

In order to keep the dimension of A as small as possible, we still define A as the smallest affine space covering only the one-set of a function.

Definition 11 *An incompletely specified function $f : \{0,1\}^n \rightarrow \{0,1,*\}$ is* D-reducible *if its one-set can be covered by an affine space of dimension strictly smaller than n.*

Once A has been derived, we project onto A non only the ones of f, but also its *don't care set*. The points of the don't care set that are not covered by A are set to 0.

5 Experimental results

In this section we compare the size of the networks described in Section 4.3 (in short DRedSOPs) with the size of the corresponding minimum SOPs. To this end we count the number of literals and the number of gates (OR, AND and EXOR) of an expression. In the multi-level context the cost function is the number of literals in each different gate (see [12, 13]). We observe that in many technologies EXOR and OR (or AND) gates have different costs. In [13] the authors consider a 2-input EXOR gate as $x \oplus y = \overline{x}y + x\overline{y}$. Thus the cost of a 2-input EXOR gate is 6 (4 literals and 2 products), while the cost of the 2-input OR and AND gates is 2. Generally, by the associative property of the EXOR operator, we can always see a k-input EXOR gate as the composition of $(k-1)$ 2-input EXOR gates. (The realization is a tree of

Table 1: *Synthesis times and network costs of DRedSOPs, and exact SOP forms*

Benchmark			Network Cost					Synthesis time	
Name	**n**	$\|P\|$	$\mu_{\mathbf{SOP}}$	$\mu_{\mathbf{SOP'}}$	**AD+E**	$\mu_{\mathbf{DRedSOP}}$	**gain**	**SOP**	**DRedSOP**
addm4	9	189	1407	1380	27	1407	0.00	0.49	0.41
adr4	8	255	415	410	13	423	-1.93	0.07	0.05
alu1	12	19	60	51	15	66	-10.00	0.23	0.34
b2	16	104	1970	1998	19	2017	-2.36	0.81	0.79
chkn	29	153	1744	1544	27	1571	9.92	0.33	0.40
co14	14	14	210	169	81	250	-19.05	0.01	0.01
f51m	8	76	402	396	17	413	-2.74	0.11	0.12
intb	15	629	5911	5259	9	5268	10.88	11.76	9.13
m181	15	430	235	202	14	216	8.06	2.17	21.40
misex2	25	29	213	89	140	239	-12.21	0.01	0.01
mlp4	8	121	869	846	14	860	1.04	0.95	0.30
mp2d	14	123	201	173	64	237	-17.91	1.00	4.97
newapla1	12	4	76	16	53	69	9.21	0.01	0.01
newtpla	15	23	199	112	36	148	25.63	0.01	0.01
sao2	10	58	495	289	55	344	30.51	0.05	0.06
t3	12	33	251	207	29	236	5.98	0.01	0.03
table3	14	175	2643	2737	28	2765	-4.62	0.24	0.23
table5	17	158	2503	2588	92	2680	-7.07	0.25	0.30
vg2	25	110	914	586	118	704	22.98	0.79	0.86
vtx1	27	110	1074	670	116	786	26.82	0.62	2.71
x6dn	39	121	818	737	11	748	8.56	0.57	0.60
x9dn	27	120	1258	704	130	834	33.70	0.69	2.23
xor5	5	16	96	1	26	27	71.88	0.01	0.01

EXOR gates. Note that an EXOR tree for a k-input EXOR can always be balanced, thus its height is $\lceil \log_2 k \rceil$.) Therefore, we can use a cost function μ where a k-input EXOR gate costs $6(k-1)$, and k-input OR/AND gates cost k. With these measures we compare DRedSOP and SOP expressions. Note that, for SOP expressions the cost μ (that we call μ_{SOP}) corresponds to the sum of the number of literals (L) and different AND gates (A) in the SOP expression, i.e., $\mu_{SOP} = L + A$. For the DRedSOP form of a function f, the cost is $\mu_{DRedSOP} = \mu_{SOP'} + AD + E$, where E is the total cost of the EXOR gates, $\mu_{SOP'}$ is the cost of the SOP of the projected function f_A, and AD is the cost of the final AND gate. In facts $AD = nE + 1$, where nE is the number of EXORs, and 1 is the output of the SOP. For example for the DRedSOP: $(x_1 \oplus x_2)(x_1 \oplus x_3)(x_1x_4 + x_6)$ we have $AD = 2 + 1$ and $\mu_{DRedSOP} = 5 + 3 + 2 * 6$.

Our minimization method has been tested on a range of functions taken from the Espresso benchmark suite [18]. CPU times are reported in seconds on a Pentium III $800MHz$ machine with $512MB$ of RAM. The Gauss elimination is computed with *Mathematica* 5.0.

In our experiments, we have first computed the number of functions that have at least one D-reducible output in the benchmark suite. The number of such functions is about 70% of the total. We have then synthesized these functions in order to evaluate whether their DRedSOP network is indeed more compact than the classical minimum SOP form. We have minimized both SOP and DRedSOP forms using Espresso Exact [2]. The size of the resulting networks has been compared using the cost function μ. Table 1 shows a significant subset of our results. The cost of the PLA for the SOP form is reported in the second column (μ_{SOP}) of the table, while the overall cost of the DRedSOP network is in the fourth column ($\mu_{DRedSOP}$).

We can note that the DRedSOP is not always smaller than the minimum SOP form. This is due to different reasons. First, the EXOR part of the network can be expensive in the CMOS technology. Moreover, some functions benefit from the multi-output minimization; after the projection of some outputs, it can happen that the common products are reduced in number.

We have finally compared area and delay of these functions using SIS tool, after the technology mapping. Table 2 shows the results of a significant subset of our experiments.

However we can observe from Table 1 that a significant number of benchmark functions have a reduced size for their DRedSOP form (the functions that have a positive value in the gain column of the table.) Therefore we propose our algorithm as a preprocessing step before the logic synthesis process.

Table 2: *Area/delay costs of DRedSOPs, and exact SOP forms*

Benchmark			**DRedSOP**		**SOP**	
Name	**n**	$\|P\|$	**area**	**delay**	**area**	**delay**
addm4	9	189	1276	42.90	1363	47.90
adr4	8	255	115	12.20	267	19.20
chkn	29	153	866	47.10	819	43.60
f51m	8	76	310	20.70	563	31.60
mp2d	14	123	313	19.70	417	26.00
newtpla	15	23	130	19.70	111	19.70
sao2	10	58	344	27.60	332	27.10
t3	12	33	191	16.90	198	21.50
vg2	25	110	395	22.40	354	18.60
vtx1	27	110	384	25.90	344	21.30
x6dn	39	121	899	34.40	1217	36.80
x9dn	27	120	439	26.90	404	23.00
xor5	5	16	16	9.10	16	9.10

%begintable[t]

6 Conclusion

In this paper we have introduced the notion of D-reducibility of a Boolean function f. This approach supplies a new tool for efficient minimization. For a D-reducible function f, depending on n variables, a new function f_A, depending on less than n variables, can be defined and built in polynomial time. Our experiments have confirmed the foreseen time reduction, and have also shown that a great number of functions of practical importance are indeed D-reducible, thus validating the overall interest of our approach. Our minimization algorithm would probably be greatly improved if formulated on BDD's as its applicability is presently limited by the size of the input. This promising approach is currently under investigation.

References

[1] A. Bernasconi, V. Ciriani, F. Luccio, and L. Pagli. Three-Level Logic Minimization Based on Function Regularities. *IEEE Transactions on TCAD*, 22(8):1005–1016, 2003.

[2] R. Brayton, G. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Ac. Pub., 1984.

[3] G. Caruso. Near Optimal Factorization of Boolean Functions. *IEEE Transactions on CAD*, 10(8):1072–1078, 1991.

[4] V. Ciriani. Synthesis of SPP Three-Level Logic Networks using Affine Spaces. *IEEE Transactions on TCAD*, 22(10):1310–1323, 2003.

[5] P. Cohn. *Algebra Vol. 1.* John Wiley & Sons, 1981.

[6] O. Coudert. Doing Two-Level Logic Minimization 100 Times Faster. In *SODA*, pages 112–121, 1995.

[7] D. Debnath and T. Sasao. An Optimization of AND-OR-EXOR Three-Level Networks. In *Asia and South Pacific Design Automation Conference*, pages 545–550, 1997.

[8] D. Debnath and T. Sasao. Minimization of AND-OR-EXOR Three-Level Networks with AND Gate Sharing. *IEICE Trans. Information and Systems*, E80-D(10):1001–1008, 1997.

[9] D. Debnath and T. Sasao. A Heuristic Algorithm to Design AND-OR-EXOR Three-Level Networks. In *Asia and South Pacific Design Automation Conference*, pages 69–74, 1998.

[10] E. Dubrova, D. Miller, and J. Muzio. Upper Bounds on the Number of Products in AND-OR-XOR Expansion of Logic Functions. *Electronic Letters*, 31(7):541–542, 1995.

[11] E. Dubrova, D. Miller, and J. Muzio. AOXMIN-MV: A Heuristic Algorithm for AND-OR-XOR Minimization. In *4th Int. Workshop on the Applications of the Reed Muller Expansion in circuit Design*, pages 37–54, 1999.

[12] M. Eggerstedt, N. Hendrich, and K. von der Heide. Minimization of Parity-Checked Fault-Secure AND/EXOR Networks. In *IFIP WG 10.2 Workshop on Applications of the Reed-Muller Expansion in Circuit Design*, pages 142–146, 1993.

[13] G. Hachtel and F. Somenzi. *Logic Synthesis and Verification Algorithms*. Kluwer Academy Publishers, 1996.

[14] F. Luccio and L. Pagli. On a New Boolean Function with Applications. *IEEE Transactions on Computers*, 48(3):296–310, 1999.

[15] M. Perkowski. A New Representation of Strongly Unspecified Switching Functions and its Application to Multi-Level AND/OR/EXOR Synthesis. In *IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion*, pages 143–151, 1995.

[16] T. Sasao. A Design Method for AND-OR-EXOR Three Level Networks. In *Int. Workshop on Logic Synthesis*, pages 8:11–8:20, 1995.

[17] T. Sasao. *Switching Theory for Logic Synthesis*. Kluwer Academic Publishers, 1999.

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