設 計 自 動 化 74－4
（1 9 9 5. 3. 8）

GRMIN: 一般化リード・マラー論理式の簡単化プログラムについて

デブナス デバトシュ, 笹尾 勤
九州工業大学情報工学部電子情報工学科
〒820 飯塚市大字川津 680-4

あらまし: 一般化リード・マラー論理式 (GRM) は, AND-EXOR 論理式の一つのクラスである. GRM では, 肯定リテラルと否定リテラルが同時に現れてもよい. GRM を用いて検査容易な回路が設計できる. 本論文は, 多出力関数を表現する GRM を簡単化するヒューリスティックアルゴリズムを示す. アルゴリズムは 7 つの簡単化規則をもちいており, 積項数とリテラル数を減らす. 実験の結果, ほとんどの関数に対して, GRM では SOP(AND-OR 論理和形) に比べ積項数が少なくてよいことが明らかになった.

和文キーワード: 論理式簡単化, リード・マラー論理式, AND-EXOR, 検査容易設計.

GRMIN: A Heuristic Simplification Algorithm for Generalized Reed-Muller Expressions

Debatosh DEBNATH and Tsutomu SASAO
Department of Computer Science and Electronics
Kyushu Institute of Technology
680-4 Kawazu, Iizuka 820, Japan

Abstract: Generalized Reed-Muller expressions (GRMs) is a class of AND-EXOR expressions. In a GRM, each variable may appear both complemented and uncomplemented form. Networks realized using GRMs have easily testable property. This paper presents a heuristic simplification algorithm for GRMs for multiple-output functions. The algorithm uses seven rules. As the primary objective, it reduces the number of products, and as the secondary objective, it reduces the number of literals. Experimental results show that, in most cases, GRMs require fewer products than conventional sum-of-products expressions (SOPs). Our algorithm outperforms existing algorithms.

Key words: Logic minimization, Reed-Muller Expression, AND-EXOR, Easily testable networks.

1 Introduction

Logic networks are usually designed by using AND and OR gates. However, networks utilizing exclusive-OR (EXOR) gates have some advantages over the conventional AND-OR networks. Firstly, such networks often require fewer gates and interconnections than the ones designed using AND and OR gates [14]. Examples of such networks include, arithmetic, telecommunication, and error correcting circuits. Secondly, they can be made easily testable.

Various classes exist in AND-EXOR expressions [7, 13]. Among them, positive polarity Reed-Muller expressions (PPRMs) are well known: a PPRM, an EXOR sum-of-products with positive literals, uniquely represents an arbitrary logic function. Networks based on PPRMs are easily testable [10, 11], but they often require more products than ones based on other expressions. Generalized Reed-Muller expressions (GRMs) [4] are generalization of PPRMs. They never require more products than PPRMs, and often require many fewer products than PPRMs. GRMs were studied many years ago [2], but no practical applications have been reported. Recently, easily testable realizations for GRMs have been developed [15]. Because GRMs often require many fewer products than PPRMs and have very good testability, the GRM based design have practical importance. An exact minimization algorithm for GRMs is available [16], but for the functions with more than six variables it is very time and memory consuming. For heuristic simplification of GRMs, only a few algorithms [3] exist. In this paper, we present a heuristic simplification algorithm of GRMs for multiple-output function. The experimental results show the effectiveness of our approach.

2 Definitions and Basic Properties

2.1 Positional Cube Notation

A positional cube [8] is convenient for the manipulation of logic expressions by computers. In positional cube notation, an uncomplemented literal such as x_i is represented by 01, a complemented literal $\bar{x}_i$ is represented by 10, and a don't care (missing variable in a product) is represented by 11. 00 represents no value of the variable, and any cube containing a 00 for any variable position denotes a null cube.

For m-output functions $(f_0, f_1, \ldots, f_{m-1})$ $(m \geq 1)$, an m-bit tag field is catenated to the cube to denote the output part. If the ith bit of the tag field of a cube is 1, the ith output is occupied by the cube. In this case, we think the outputs as an m-valued variable.

In the positional cube notation, each variable constitute a *part*. Thus, in the representation of an n-variable function there are $n+1$ parts.

Definition 2.1 *A list of cubes is called* **OR array** *if it represents the OR of cubes, and called* **EXOR array** *if it represents the EXOR of cubes.*

Example 2.1 *The three-variable function* $f_0 = x_1x_2 \vee x_1\bar{x}_3 \vee \bar{x}_1\bar{x}_3$ *is represented by the OR array:*

x_1	x_2	x_3	f_0
01 –	01 –	11 –	1
01 –	11 –	10 –	1
10 –	11 –	10 –	1.

It has four parts. *(End of Example)*

Example 2.2 *The three-variable three-output function* $f_0 = x_1\bar{x}_3 \oplus x_2x_3$, $f_1 = x_2x_3$, *and* $f_2 = x_1\bar{x}_3$ *is represented by the EXOR array:*

x_1	x_2	x_3	$f_0f_1f_2$
01 –	11 –	10 –	101
11 –	01 –	01 –	110.

It has four parts. *(End of Example)*

From now on, unless otherwise specified, a *list* of cube represents an EXOR array. Also, all the inputs are two-valued.

Definition 2.2 *The* **c-distance** *between two cubes is the number of parts in which they differ.*

Example 2.3 *The c-distance between the first two cubes in Example 2.1 is two, and the c-distance between the two cubes in Example 2.2 is four.* *(End of Example)*

2.2 Logical Expression

An alternate representation of a logic function is a logic expression. In this representation, position of 1's in a part of the positional cube notation of a cube are shown as the exponent of the corresponding variable. In Section 4.3, this representation is used to show the simplification rules for GRMs. From now, we will consider outputs as a multiple-valued variable and represent it by z.

Example 2.4 *The three-variable multiple-output function shown in Example 2.2 can be represented by* $x_1^{\{1\}}x_2^{\{01\}}x_3^{\{0\}}z^{\{02\}} \oplus x_1^{\{01\}}x_2^{\{1\}}x_3^{\{1\}}z^{\{01\}}$. *Because it is an EXOR array, the* $\oplus$ *operator is used.* *(End of Example)*

2.3 PPRM, FPRM and GRM

In this part, we will define three classes of AND-EXOR expressions. The following lemma is the basis of the EXOR-based expansion:

Lemma 2.1 *An arbitrary logic function* $f(x_1, x_2, \ldots, x_n)$ *can be expanded as*

$$f = \bar{x}_1 f_0 \oplus x_1 f_1, \tag{2.1}$$

$$f = f_0 \oplus x_1 f_2, \tag{2.2}$$

$$f = f_1 \oplus \bar{x}_1 f_2, \tag{2.3}$$

where $f_0 = f(0, x_2, \ldots, x_n)$, $f_1 = f(1, x_2, \ldots, x_n)$, *and* $f_2 = f_0 \oplus f_1$.

(2.1), (2.2), and (2.3) are called the *Shannon expansion*, the *positive Davio expansion*, and the *negative Davio expansion*, respectively. If we use (2.2) recursively to a function f, then we have the following:

Lemma 2.2 *An arbitrary n-variable function* $f(x_1, x_2, \ldots, x_n)$ *can be represented as*

$$\begin{aligned} f = {} & a_0 \oplus a_1x_1 \oplus a_2x_2 \oplus \cdots \oplus a_nx_n \\ & \oplus a_{12}x_1x_2 \oplus a_{13}x_1x_3 \oplus \cdots \oplus a_{n-1\,n}x_{n-1}x_n \oplus \\ & \cdots\cdots\cdots\cdots\cdots\cdots \\ & \oplus a_{12\cdots n}x_1x_2x_3\cdots x_n, \end{aligned} \tag{2.4}$$

where a's are either 0 or 1.

(2.4) is called a *positive polarity Reed-Muller expression* (PPRM). For a given function f, the coefficients $a_0, a_1, a_2, \ldots, a_{12\cdots n}$ are uniquely determined. Thus, the PPRM is a canonical representation. The number of products in (2.4) is at most 2^n, and all the literals are positive (uncomplemented).

In (2.4), for each variable x_i $(i = 1, 2, \ldots, n)$, if we use either a positive literal (x_i) throughout or a negative literal $(\bar{x}_i)$ throughout, then we have a *fixed polarity Reed-Muller expression* (FPRM). For each variable x_i, there are two ways of choosing the polarities: positive (x_i) or negative $(\bar{x}_i)$. Thus, 2^n different set of polarities exist for an n-variable function. For a given function and a given set of polarities, a unique set of coefficients $(a_0, a_1, \ldots, a_{12\cdots n})$ exists. Thus, an FPRM is a canonical representation.

In (2.4), if we can freely choose the polarity for each literal, then we have a *generalized Reed-Muller expression* (GRM). Unlike FPRMs, both x_i and $\bar{x}_i$ can appear in a GRM. Some authors use GRMs to represent another class of expressions [6], thus the terminology is not unified. There are $n2^{n-1}$ literals in (2.4), so $2^{n2^{n-1}}$ different set of polarities exist for an n-variable function. For a given set of polarities, a unique set of coefficients $(a_0, a_1, \ldots, a_{12\cdots n})$ exists. Thus, a GRM is a canonical representation for a logic function.

A GRM for a multiple-output function is defined as follows:

Definition 2.3 *An array represents a multiple-output GRM, if for each output, the corresponding cubes represent a GRM.*

Example 2.5 *Consider an array representing a three-variable two-output function:*

x_1	x_2	x_3	$f_0 f_1$
10 –	10 –	10 –	10
10 –	01 –	01 –	01
01 –	01 –	11 –	11.

Both $f_0 = \bar{x}_1\bar{x}_2\bar{x}_3 \oplus x_1x_2$, *and* $f_1 = \bar{x}_1x_2x_3 \oplus x_1x_2$ *are GRMs. Thus, the array represents a multiple-output GRM.* *(End of Example)*

2.4 PSDRM, KRO, PSDKRO and ESOP

Before studying the simplification method for GRMs, it is convenient to define other classes of expressions.

Suppose that we are given a three-variable function $f(x_1, x_2, x_3)$. When we expand f by using the positive Davio expansion with respect to x_1, we have

$$f = f_0 \oplus x_1 f_2.$$

Next, when we expand f_0 and f_2 in the similar way with respect to x_2, we have

$$f_0 = f_{00} \oplus x_2 f_{02}, \quad f_2 = f_{20} \oplus x_2 f_{22}.$$

Furthermore, when we use similar expansions with respect to x_3, we have

$$f_{00} = f_{000} \oplus x_3 f_{002}, \quad f_{02} = f_{020} \oplus x_3 f_{022},$$
$$f_{20} = f_{200} \oplus x_3 f_{202}, \quad f_{22} = f_{220} \oplus x_3 f_{222}.$$

The expansion tree in Fig. 2.1 illustrates this process. A path from the root node to a terminal node represents a product of an expression, where a label of an edge shows the literal for the corresponding variable. For example, the path from the root node to f_{000} represents the product $1 \cdot 1 \cdot 1 \cdot f_{000} = f_{000}$, the path to f_{002} represents $1 \cdot 1 \cdot x_3 f_{002} =$

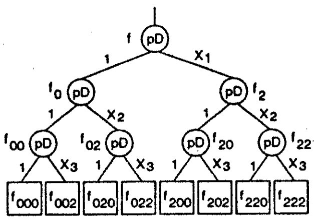

Figure 2.1: Representation of a logic function using positive Davio expansions.

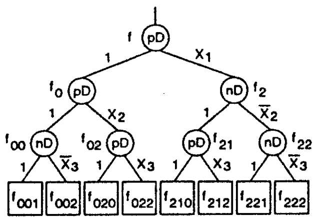

Figure 2.2: Representation of a logic function using pseudo Reed-Muller expansions.

$x_3 f_{002}$, and the path to f_{222} represents $x_1x_2x_3f_{222}$. Thus, the tree in Fig. 2.1 shows the PPRM:

$$f = f_{000} \oplus x_3 f_{002} \oplus x_2 f_{020} \oplus x_2x_3 f_{022} \oplus x_1 f_{200}$$
$$\oplus\, x_1x_3 f_{202} \oplus x_1x_2 f_{220} \oplus x_1x_2x_3 f_{222}.$$

Each node has a label pD, which shows the positive Davio expansion. In Fig. 2.1, only the positive Davio expansions are used. However, if we use either the positive or the negative Davio expansion for each variable, then we have a more general tree. Such a tree represents an FPRM. If we use either the positive or the negative Davio expansion for each node, then we have a more general tree. Such a tree represents a *pseudo Reed-Muller expression* (PSDRM). For example, in Fig. 2.2, f, f_0, f_{02}, and f_{21} use the positive Davio expansions, while f_2, f_{00}, and f_{22} use the negative Davio expansions. Nodes with label nD denotes the negative Davio expansion. Note that the tree in Fig. 2.2 shows the PSDRM:

$$f = 1 \cdot 1 \cdot 1 \cdot f_{001} \oplus 1 \cdot 1 \cdot \bar{x}_3 f_{002} \oplus 1 \cdot x_2 \cdot 1 \cdot f_{020}$$
$$\oplus\, 1 \cdot x_2x_3 f_{022} \oplus x_1 \cdot 1 \cdot 1 \cdot f_{210} \oplus x_1 \cdot 1 \cdot x_3 f_{212}$$
$$\oplus\, x_1\bar{x}_2 \cdot 1 \cdot f_{221} \oplus x_1\bar{x}_2\bar{x}_3 f_{222}.$$

There are 7 nodes in the tree, and each node represents either the positive Davio (pD) or the negative Davio (nD) expansion. From the definitions, clearly FPRMs are the special class of PSDRMs.

In Fig. 2.1, if we use either the Shannon, the positive Davio, or the negative Davio expansion for each variable, then we have another class of trees. Such a tree represents a *Kronecker expression* (KRO).

In Fig. 2.1, if we use either the Shannon, the positive Davio, or the negative Davio expansion for each node, then we have yet another class of trees. Such a tree represents a *pseudo Kronecker expression* (PSDKRO). By definitions, clearly FPRMs form a special class of KROs, and KROs form a special class of PSDKROs.

Arbitrary product terms combined by EXORs is called an *Exclusive-or Sum-of-Products Expression* (ESOP). The ESOP is the most general AND-EXOR expression. Arbitrary product terms combined by ORs is called a *Sum-of-Products Expression* (SOP).

Example 2.6

1. $x_1 \oplus x_2 \oplus x_1x_2$ *is a PPRM (all literals are uncomplemented).*
2. $x_1 \oplus \bar{x}_2 \oplus x_1\bar{x}_2$ *is an FPRM, but not a PPRM (x_2 have complemented literals).*
3. $x_2 \oplus x_1\bar{x}_2$ *is a PSDRM, but not an FPRM (x_2 have both complemented and uncomplemented literals).*
4. $x_1 \oplus x_2 \oplus \bar{x}_1\bar{x}_2$ *is a GRM, but not a PSDRM (it cannot be generated by an expansion tree for a PSDRM).*

2.5 Properties of GRMs

In this part, we consider some properties of GRMs, which are useful for the simplification of expressions.

Lemma 2.3 *PSDRMs form special class of GRMs.*

From the above arguments, we have the following relations, which are also shown in Fig. 2.3.

Theorem 2.1 *Suppose that $\mathcal{PPRM}$, $\mathcal{FPRM}$, $\mathcal{PSDRM}$, $\mathcal{KRO}$, $\mathcal{PSDKRO}$, $\mathcal{GRM}$ and $\mathcal{ESOP}$ denote the corresponding set of expressions. Then, the following relations hold:*

$$\mathcal{PPRM} \subset \mathcal{FPRM} \subset \mathcal{PSDRM} \subset \mathcal{GRM} \subset \mathcal{ESOP},$$
$$\mathcal{FPRM} \subset \mathcal{KRO} \subset \mathcal{PSDKRO} \subset \mathcal{ESOP},$$
$$\mathcal{PSDRM} \subset \mathcal{PSDKRO}.$$

Definition 2.4 *The* **variable set of a product** p *is denoted by* $V(p) = \{x_i \mid x_i \text{ or } \bar{x}_i \text{ appears in } p\}$. *The* **variable set of a cube** *is the variable set of the product represented by the input parts of the cube.*

Example 2.7 $V(x_1\bar{x}_2\bar{x}_4) = \{x_1, x_2, x_4\}$. *The variable sets of the first and second cubes in Example 2.2 are $\{x_1, x_3\}$, and $\{x_2, x_3\}$, respectively.* *(End of Example)*

Lemma 2.4 *An AND-EXOR expression (for single-output function) is a GRM, if no two products have the same set of variables.*

(Proof) It is obvious from the definition of GRMs (Section 2.3). (Q.E.D.)

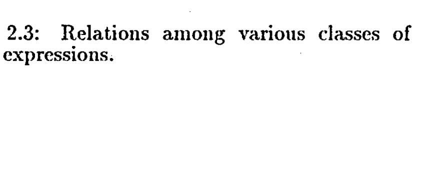

Figure 2.3: Relations among various classes of AND-EXOR expressions.

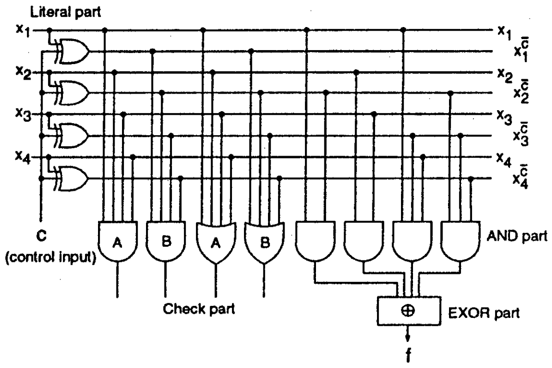

Figure 3.1: An easily testable realization for a GRM.

Example 2.8 *Let* $f = x_4 \oplus x_1x_2x_3 \oplus \bar{x}_1\bar{x}_2\bar{x}_3$. *Then* $V(x_1x_2x_3) = \{x_1, x_2, x_3\}$, *and* $V(\bar{x}_1\bar{x}_2\bar{x}_3) = \{x_1, x_2, x_3\}$. *Thus, f is not a GRM because two products have the same set of variables.* *(End of Example)*

In a GRM for multiple-output function, more than one cube may have the same set of variables.

Lemma 2.5 *An array represents a multiple-output GRM, if for any output, no two cubes have the same set of variables.*

(Proof) Let for a output, two cubes have the same set of variables, then the corresponding function represents a non-GRM (Lemma 2.4). Thus, according to the Definition 2.3, the array is not a GRM. Hence, we have the lemma. (Q.E.D.)

Corollary 2.1 *In an array for a multiple-output function, if cubes having the same set of variables have non-disjoint output parts, then the array does not represent a GRM.*

Example 2.9 *Consider the array in Example 2.5. The first and second cubes have the same set of variables, but their output parts are disjoint. Thus, the array represents a multiple-output GRM.* *(End of Example)*

3 Easily Testable Realizations for GRMs

We can design easily testable networks by employing EXOR gates [5]. Reddy showed that if a network is realized using PPRMs, and only a single stuck-at fault is present, at most $(n+4)$ fault detecting tests are sufficient, where n is the number of input variables [10]. The method require a small number of test sets. But the networks based on PPRMs require excessive amount of hardware.

Recently, an easily testable realizations for GRMs have been developed [15]. The number of products for GRMs is, on the average, less than a half of that for PPRMs [16]. Thus, the network require less hardware. The method uses four extra gates and detects multiple stuck-at faults.

Let we consider the GRM: $f = x_1\bar{x}_2 \oplus x_2x_3 \oplus x_1\bar{x}_3x_4 \oplus \bar{x}_2\bar{x}_3\bar{x}_4$. The testable network of this GRM is shown in

Fig. 3.1, where the literal part has a control input c. During the normal operation, the control input is set to one, and the literal part produces the positive (x_i) and the negative ($\bar{x}_i$) literals. During the test mode, the control input is set to zero, and the network realizes a PPRM. Thus, we can test the network in a similar way to [10].

4 Simplification Algorithm

4.1 Outline of the Algorithm

The GRM simplification algorithm in this paper have the following features:

1. As an input, it accepts a GRM.
2. It simplifies multiple-output functions.
3. It uses seven rules iteratively to reduce the number of products.
4. It modifies the cubes repeatedly by replacing a pair of cubes with another one, while keeping the array to represent a GRM.
5. It never increases the number of products.

4.2 Initial Solution

Because PPRMs, FPRMs and PSDRMs are special class of GRMs, any of them can be used as an initial solution for the GRM. We use a PSDRM as an initial solution, because minimal PSDRMs are easy to derive and usually require fewer products than PPRMs and FPRMs. We used the algorithm in [13] to obtain PSDRMs.

4.3 Simplification Rules

For simplification of ESOPs, 10 rules are used in EXMIN2 [14]. However for simplification of GRMs only seven rules are used.

Definition 4.1 *$\cap$ and $\cup$ denote the intersection and union operation between two sets, respectively. Also '$-$' (overline) denote the complement of a set. If A and B are sets, $A \oplus B = (A \cap \overline{B}) \cup (\overline{A} \cap B)$. The symbol $\oplus$ is also used to denote the EXOR of two logic functions. A set is* null (ϕ) *if it contains no element.*

Let $A, B, C, D \subseteq P$, where $P = \{0, 1\}$. Then, the simplification rules for GRMs are as follows:

1. X-MERGE
$$X^A \oplus X^B \Rightarrow X^{(A \oplus B)}$$
2. RESHAPE
$$X^A Y^B \oplus X^C Y^D \Rightarrow X^A Y^{(B \cap \overline{D})} \oplus X^{(A \cup C)} Y^D$$
$$\text{if } (A \cap C = \phi,\ B \supset D)$$
3. DUAL-COMPLEMENT
$$X^A Y^B \oplus X^C Y^D \Rightarrow X^{(\overline{A} \cap C)} Y^B \oplus X^C Y^{(B \cap \overline{D})}$$
$$\text{if } (A \subset C,\ B \supset D)$$
4. X-EXPAND-1
$$X^A Y^B \oplus X^C Y^D \Rightarrow X^A Y^{(B \cup D)} \oplus X^{(A \cup C)} Y^D$$
$$\Rightarrow X^{(A \cup C)} Y^B \oplus X^C Y^{(B \cup D)}$$
$$\text{if } (A \cap C = \phi,\ B \cap D = \phi)$$
5. X-EXPAND-2
$$X^A Y^B \oplus X^C Y^D \Rightarrow X^{(A \cup C)} Y^B \oplus X^C Y^{(B \cap \overline{D})}$$
$$\text{if } (A \cap C = \phi,\ B \supset D)$$
6. X-REDUCE-1
$$X^A Y^B \oplus X^C Y^D \Rightarrow X^{(A \cap \overline{C})} Y^B \oplus X^C Y^{(D \cap \overline{B})}$$
$$\text{if } (A \supset C,\ B \subset D)$$
7. X-REDUCE-2
$$X^A Y^B \oplus X^C Y^D \Rightarrow X^{(A \cap \overline{C})} Y^B \oplus X^C Y^{(B \cap \overline{D})}$$
$$\text{if } (A \supset C,\ B \supset D)$$

The representation, analysis, and proof of these rules for multiple-valued input function can be found in [14].

4.4 Examples of Simplification

Example 4.1 *Let an initial GRM be $\bar{x}_1 \oplus x_1 x_2 \oplus \bar{x}_2$. Its positional cubes are:*

x_1	x_2	f_0
10 –	11 –	1
01 –	01 –	1
11 –	10 –	1.

X-MERGE is inapplicable to this array. Applying X-EXPAND-2 to the first two cubes, we have the following array:

x_1	x_2	f_0
11 –	11 –	1
01 –	10 –	1
11 –	10 –	1.

Applying X-MERGE to the last two cubes, we have

x_1	x_2	f_0
11 –	11 –	1
10 –	10 –	1.

Thus, we have a GRM with two products: $1 \oplus \bar{x}_1 \bar{x}_2$.
(End of Example)

Example 4.2 *Consider the following array:*

x_1	x_2	$f_0 f_1$
10 –	11 –	01
01 –	10 –	01
11 –	10 –	10.

X-MERGE is inapplicable to this array, but RESHAPE is applicable to the first two cubes, and we have the following array:

x_1	x_2	$f_0 f_1$
10 –	01 –	01
11 –	10 –	01
11 –	10 –	10.

Merging the last two cubes, we have a GRM with two cubes:

x_1	x_2	$f_0 f_1$
10 –	01 –	01
11 –	10 –	11.

(End of Example)

4.5 Properties of Simplification Rules

To simplify GRMs we use seven rules. Among them, X-MERGE is the only rule that reduces the number of cubes. When X-MERGE is inapplicable, other rules are used to modify the shape of the cubes so that X-MERGE become applicable. For a pair of cubes, the rules are applicable only when the c-distance between them is one or two. Note that we keep the array to represent a GRM always.

Lemma 4.1 *In an array for GRM, if the output part of two cubes differ, then RESHAPE is inapplicable.*

(Proof) Let the original cubes be $pX^A Y^B$ and $pX^C Y^D$, where p is a product common to the two cubes. Note that input variable is two-valued, and output variable is multiple-valued. The rule RESHAPE is:

$$X^AY^B \oplus X^CY^D \Rightarrow X^AY^{(B\cap\bar{D})} \oplus X^{(A\cup C)}Y^D$$
$$\text{if } (A\cap C=\phi,\ B\supset D).$$

When the output parts are disjoint: Let Y be the input variable and X be the output variable. $B\supset D$ is the condition for RESHAPE, so $B=\{0,1\}$, and $D=\{0\}$, or $\{1\}$. Therefore, the new input parts are $Y^{\{1\}}$ and $Y^{\{0\}}$. This represents new cubes have the same set of variables. The new output parts are X^A and $X^{(A\cup C)}$. But $A\cap(A\cup C)\neq\phi$, i.e., outputs of the new cubes are non-disjoint. Thus, by Corollary 2.1, the cubes no longer represent a GRM.

When input parts are disjoint: Let X be the input variable and Y be the output variable. By the condition for RESHAPE, the given input parts are $X^{\{0\}}$ and $X^{\{1\}}$, i.e., the cubes have the same set of variables. Also, the condition $B\supset D$ implies that outputs are non-disjoint. Thus, from Corollary 2.1, we have the lemma. (Q.E.D.)

Lemma 4.2 *For a pair of cubes in an array for GRM, if two input parts differ, then X-EXPAND-1 is inapplicable.*

(Proof) Let pX^AY^B and pX^CY^D be the original cubes, where p is a product common to the both cubes. Note that both X and Y are input variables, and they are two-valued. From Section 4.3, the condition for X-EXPAND-1 is $A\cap C=\phi$ and $B\cap D=\phi$, i.e., input parts are disjoint. This implies that both cubes have the same set of variables. Also, the outputs are the same in both cubes. Thus, from Corollary 2.1, we have the lemma. (Q.E.D.)

Lemma 4.3 *For a pair of cubes in an array for GRM, if they are disjoint in an input part, and differs in the output part, then X-EXPAND-2 is inapplicable.*

(Proof) Let the original cubes be pX^AY^B and pX^CY^D, where p is a product common to the both cubes, X be a two-valued input variable, and Y be the multiple-valued variable representing the output part. From Section 4.3, the condition for X-EXPAND-2 ($A\cap C=\phi$, $B\supset D$) implies that they are non-disjoint in the output parts. Thus, by Corollary 2.1, we have the lemma. (Q.E.D.)

Lemma 4.4 *In an array for GRM, if a pair of cubes differ in two input parts, then X-REDUCE-1 is inapplicable.*

(Proof) Let pX^AY^B and pX^CY^D be the original cubes, where p is a product common to the both cubes. Note that both X and Y are two-valued input variables. From Section 4.3, X-REDUCE-1 is:

$$X^AY^B \oplus X^CY^D \Rightarrow X^{(A\cap\bar{C})}Y^B \oplus X^CY^{(D\cap\bar{B})}$$
$$\text{if } (A\supset C,\ B\subset D).$$

By the condition $A\supset C$, we have $A=\{0,1\}$, and $C=\{0\}$, or $\{1\}$. So, a new input part of the two cubes are $X^{\{1\}}$ and $X^{\{0\}}$. Similarly, other input part of the two cubes are $Y^{\{0\}}$ and $Y^{\{1\}}$. These imply that new cubes have the same set of variables. Also, both cubes have the same output part. Thus, from Corollary 2.1, the array is a non-GRM. Hence, the lemma. (Q.E.D.)

Rules 2-7 of Section 4.3 may produce cubes with new variable set and modified output parts. Therefore, before applying these rules, we check if the resultant array represents a GRM or not. If it is non-GRM, we discard the operation to keep the array to represent a GRM.

Example 4.3 *Consider an array for a GRM:*

x_1	x_2	x_3	$f_0f_1f_2$
10 −	10 −	11 −	110
10 −	01 −	01 −	110
01 −	11 −	10 −	011.

By applying RESHAPE to the first two cubes, we have

x_1	x_2	x_3	$f_0f_1f_2$
10 −	10 −	10 −	110
10 −	11 −	01 −	110
01 −	11 −	10 −	011.

The last two cubes of this array have identical variable sets, i.e., $\{x_1, x_3\}$, and the outputs are non-disjoint. Thus, the array no longer represents a GRM, and we discard this operation to keep the array to represent a GRM. *(End of Example)*

Lemma 4.5 *In an array for GRM, if a rule does not change the variable set of a cube, and produces the outputs which is a subset of the original outputs, then the cube can be modified without checking the whole array.*

(Proof) The given array represents a GRM. So, for any output, no two cubes in the array have the same set of variables (Lemma 2.5). By the hypothesis of the lemma, variable set of the cube remains same, and no new output is produced. This implies that the array still satisfy the condition of Lemma 2.5, and is a GRM. Thus, we can modify the cube without checking the whole array. Hence, we have the lemma. (Q.E.D.)

Example 4.4 *Consider an array of k ($k>3$) cubes for a GRM. Let three of the cubes in the array are:*

x_1	x_2	x_3	$f_0f_1f_2$
01 −	10 −	01 −	101
01 −	10 −	11 −	001
10 −	01 −	11 −	110.

By applying DUAL-COMPLEMENT to the first two cubes, we have

x_1	x_2	x_3	$f_0f_1f_2$
01 −	10 −	10 −	101
01 −	10 −	11 −	100
10 −	01 −	11 −	110.

In the first cube, variable set remains the same and outputs are the subset of the original outputs (equal here). Thus, we can modify the first cube without checking the whole array. Note that the polarity of the literal x_3 have been changed.

In the second cube, variable set remains same but the outputs are not a subset of the original outputs. So, we have to check if the resultant array represents a GRM or not. The second and third cube have the same set of variables and the outputs are non-disjoint. Thus, the array no longer represent a GRM, and we have to discard this operation to keep the array represent a GRM. *(End of Example)*

Lemma 4.6 *In an array for GRM, if a pair of cubes differ in two input parts, and DUAL-COMPLEMENT is applied, then the cubes can be modified without checking other cubes in the array.*

(Proof) Let pX^AY^B and pX^CY^D be the original cubes, where p is a product common to the both cubes. Note that both X and Y are two-valued input variables. From Section 4.3, the rule DUAL-COMPLEMENT is:

$$X^{A}Y^{B} \oplus X^{C}Y^{D} \Rightarrow X^{(\overline{A}\cap C)}Y^{B} \oplus X^{C}Y^{(B\cap\overline{D})}$$
$$\text{if } (A \subset C,\ B \supset D).$$

By the condition $A \subset C$, we have $C = \{0,1\}$, and $A = \{0\}$, or $\{1\}$. So, the first cube must be $X^{\{0\}}Y^{B}$, or $X^{\{1\}}Y^{B}$. After DUAL-COMPLEMENT, the first cube becomes $X^{\{1\}}Y^{B}$, or $X^{\{0\}}Y^{B}$, respectively. This implies that the variable set of the cube remains same. Also, the outputs of the cube remain unchanged. Thus, from Lemma 4.5, the cube can be modified without checking the whole array. Similarly, we can show that the same is true for the other cube. Thus, we have the lemma.(Q.E.D.)

4.6 Algorithm

For many functions, the order of the simplification rules influence the final solution. By conducting experiments on the benchmark functions, we found the following heuristic algorithm is effective.

Algorithm 4.1 *(GRMIN: Simplification of GRMs)*

1. *Obtain a PSDRM from the given SOP.*
2. *Rearrange the cubes so that the number of 1's in the positional cube notation are in ascending order.*
3. *For each pair of cubes, check if X-MERGE is applicable. If so, X-MERGE them. Continue this step until reduction of the number of cubes are possible.*
4. *For each pair of cubes, check if RESHAPE, DUAL-COMPLEMENT, X-REDUCE-1, or X-REDUCE-2 is applicable. If so, apply that.*
5. *For each pair of cubes, check if X-MERGE is applicable. If so, merge them. Continue this step until reduction of the number of cubes are possible.*
6. *If the number of cubes is reduced in step 5, then go to step 4.*
7. *For each pair of cubes, check if X-EXPAND-1, X-EXPAND-2, RESHAPE, or DUAL-COMPLEMENT is applicable. If so, apply that.*
8. *For each pair of cubes, check if X-MERGE is applicable. If so, merge them. Continue this step while reduction of the number of cubes are possible.*
9. *If the number of cubes is reduced in step 8, then go to step 7.*
10. *If the number of cubes is reduced between step 4-9, then go to step 4.*
11. *For each pair of cubes, check if X-REDUCE-1, X-REDUCE-2, RESHAPE, or DUAL-COMPLEMENT is applicable. If so, apply that. Continue this step while reduction of the number of connections is possible.*
12. *Check that the simplified cubes represent a GRM, and verify that it is functionally equivalent to the given SOP.*

In steps 3, 5 and 8, more than one merging passes are often require. In our implementation, we used additional fields for each cube. Using these fields, we can avoid many redundant computations. One of these fields store when the cube was modified. For example, during the first merging pass in step 5 of Algorithm 4.1, it checks two cubes if at least one of them are modified in step 4. As stated in Section 4.5, before applying a rule, the algorithm checks the whole array if it represents a GRM. Another field is used to make this checking easier by storing variable set information.

Table 5.1: Comparison with Cannes [3].

Data	In	Out	Cannes	GRMIN	Improvement
5xp1	7	10	60	42	30%
con1	7	2	12	9	25%
misex1	8	7	20	13	35%
rd53	5	3	20	20	0%
rd73	7	3	63	63	0%
sao2	10	4	52	35	33%
squar5	5	8	22	19	14%
sym9	9	1	131	127	3%
xor5	5	1	5	5	0%

Table 5.2: Number of products to realize arithmetic functions.

Data	PPRM	FPRM	KRO	PSD RM	PSD KRO	GRM	ESOP	SOP
adr4	34	34	34	34	34	34	31	75
log8	253	193	171	163	128	117	96	123
mlp4	97	97	97	90	81	72	61	121
nrm4	216	185	157	150	105	120	69	120
rdm8	56	56	56	46	41	35	31	76
rot8	225	118	83	81	44	59	35	57
sqr8	168	168	168	164	146	136	112	178
sym9	210	173	173	127	90	127	51	84
wgt8	107	107	107	107	107	107	58	255

5 Experimental Results

We coded the GRMIN in C. As an initial solution, it accepts a GRM. GRMIN simplifies multiple-output functions. Table 5.1 compares the number of products generated by GRMIN with that of the another heuristic program Cannes [3]. It shows that GRMIN outperforms Cannes. The improvement is up to 35%.

Table 5.2 compares the number of products required to realize arithmetic functions by various AND-EXOR expressions. In this experiment, the PPRMs, FPRMs, KROs, PSDRMs, and PSDKROs were minimized by a program in [13]. ESOPs were simplified by EXMIN2 [14]. Table 5.3 compares the number of products to realize other benchmark functions. Table 5.2 and 5.3 show that, in many cases, GRMs require fewer products than SOPs.

Table 5.4 compares the number of products to realize randomly generated functions. For each value of n, an n-variable pseudo-random function with 2^{n-1} minterms was generated and minimized. Here $|f|$ denotes the number of true minterms of the function. In this experiment, SOPs were simplified by MINI II [13], and other data were obtained by the same programs as mentioned above. This table shows, in most cases, GRMs require fewer products than SOPs.

6 Conclusion and Comments

In this paper, we presented GRMIN, a heuristic simplification algorithm for GRMs. Experimental results show that, in most cases, GRMs require fewer products than SOPs. Also, we showed that GRMIN outperforms existing algorithms.

Table 5.3: Number of products to realize other benchmark functions.

Data	In	Out	FPRM	PSDRM	GRM	ESOP	SOP
5xp1	7	10	61	55	42	32	63
addm4	9	8	160	145	106	91	189
alu1	12	8	31	31	16	16	19
amd	14	24	156	81	72	58	66
br1	12	8	70	47	31	19	19
clip	9	5	206	160	125	67	117
con1	7	2	17	12	9	9	9
dc2	8	7	57	50	42	32	39
life	9	1	100	75	53	49	84
log8mod	8	5	53	49	36	30	38
luc	8	27	57	49	34	28	26
m1	6	12	19	19	16	16	19
m2	8	16	53	51	44	39	47
m3	8	16	75	66	57	51	62
m4	8	16	132	116	102	84	101
max1024	10	6	721	552	380	165	262
max128	7	24	109	102	96	63	78
max46	9	1	206	99	54	41	46
max512	9	6	341	245	175	84	133
misex1	8	7	20	19	13	12	12
misex3	14	14	3536	1186	798	553	696
mlp6	12	12	2047	1826	1314	862	1870
newbyte	5	8	8	8	8	8	8
newcpla1	9	16	76	47	35	33	38
newtag	9	1	6	6	5	5	8
newxcpla	9	23	64	50	35	30	41
rd53	5	3	20	20	20	14	31
rd73	7	3	63	63	63	35	127
risc	8	31	37	36	27	26	27
sao2	10	4	100	62	35	29	58
sqn	7	3	66	58	40	29	38
sqr6	6	12	45	44	35	34	47
squar5	5	8	23	23	19	19	25
sym10	10	1	266	216	157	82	210
sym6	6	1	36	27	14	13	15
t3	12	8	51	34	26	24	33
tial	14	8	3683	1732	1033	487	587
xor5	5	1	5	5	5	5	16

Table 5.4: Number of products to realize randomly generated functions.

n	$\|f\|$	PPRM	FPRM	KRO	PSD RM	PSD KRO	GRM	ESOP	SOP
4	8	6	5	4	4	4	4	3	4
5	16	16	10	8	7	6	5	5	6
6	32	36	17	17	13	12	10	10	13
7	64	64	54	48	30	26	21	19	24
8	128	122	101	100	56	50	36	36	46
9	256	236	226	212	112	99	70	64	86
10	512	528	459	439	235	206	149	142	167
11	1024	1021	956	925	458	391	282	274	331
12	2048	1996	1925	1899	909	775	563	539	611
13	4096	4136	3923	3865	1813	1563	1126	1045	1157
14	8192	8210	7924	7826	3617	3107	2258	2150	2234

Although GRMIN produce good solutions, we believe that significant improvement could be possible for many functions by adding product increasing rules [1]. The product increasing rule SPLIT ($X^P \Rightarrow X^{\bar{A}} \oplus X^A$) introduced in [14] is inapplicable to a GRM.

For several functions we failed to reduce the number of products from the initial solutions. In these cases, either the initial PSDRMs are already minimum GRMs, or the given cubes has awkward shapes that our program could not reduce.

Acknowledgement

This work was supported in part by a Grant in Aid for the Scientific Research of the Ministry of Education, Science and Culture of Japan.

References

[1] D. Brand, and T. Sasao, "Minimization of AND-EXOR expressions using rewrite rules," *IEEE Trans. Comput.*, vol. 42, No. 5, pp. 568-576, May 1993.

[2] M. Cohn, "Inconsistent canonical forms of switching functions," *IRE Trans.*, EC-11, pp. 284-285, Apr. 1962.

[3] L. Csanky, M. A. Perkowski, and I. Schäfer, "Canonical restricted mixed-polarity exclusive-OR sums of products and the efficient algorithm for their minimisation," *IEE Proceedings-E*, vol. 140, No. 1, pp. 69-77, Jan. 1993.

[4] M. Davio, J-P Deschamps, and A. Thayse, *Discrete and Switching Functions*, McGraw-Hill International, New York, 1978.

[5] H. Fujiwara, *Logic Testing and Design for Testability*, The MIT Press, Cambridge, 1985.

[6] D. Green, *Modern Logic Design*, Addison-Wesley Publishing Company, Wokingham, England, 1986.

[7] D. Green, "Families of Reed-Muller canonical forms," *International J. of Electronics*, vol. 70-2, pp. 259-280, 1991.

[8] S. J. Hong, R. G. Cain, and D. L. Ostapko, "MINI: A heuristic approach for logic minimization," *IBM J. Res. & Develop.* pp. 443-458, Sept. 1974.

[9] A. Mukhopadhyay, and G. Schmitz, "Minimization of EXCLUSIVE OR and LOGICAL EQUIVALENCE switching circuits," *IEEE Trans. Comput.*, vol. C-19, pp. 132-140, Feb. 1970.

[10] S. M. Reddy, "Easily testable realizations for logic functions," *IEEE Trans. Comput.*, vol. C-21, No. 11, pp. 1183-1188, Nov. 1972.

[11] K. K. Saluja, and S. M. Reddy, "Fault detecting test sets for Reed-Muller canonic networks," *IEEE Trans. Comput.*, vol. C-24, No. 1, pp. 995-998, Oct. 1975.

[12] T. Sasao, and P. Besslich, "On the complexity of MOD-2 sum PLA's," *IEEE Trans. Comput.*, vol. 39, No. 2, pp. 262-266, Feb. 1990.

[13] T. Sasao, "AND-EXOR expressions and their optimization," in (Sasao e.d.) *Logic Synthesis and Optimization*, Kluwer Academic Publishers, 1993.

[14] T. Sasao, "EXMIN2: A simplification algorithm for exclusive-OR sum-of-products expressions for multiple-valued input two-valued output functions," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, No. 5, pp. 621-632, May 1993.

[15] T. Sasao, "Easily testable realizations for generalized Reed-Muller expressions," *Proc. IEEE The Third Asian Test Symposium*, pp. 157-162, Nov. 1994.

[16] T. Sasao, and D. Debnath, "An exact minimization algorithm for generalized Reed-Muller expressions," *Proc. IEEE Asia-Pacific Conference on Circuits & Systems*, pp. 460-465, Dec. 1994.