Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2013, Article ID 401616, 13 pages
http://dx.doi.org/10.1155/2013/401616

Research Article

A Generalized If-Then-Else Operator for the Representation of Multi-Output Functions

Ilya Levin[1] and Osnat Keren[2]

[1] *School of Education, Tel-Aviv University, Ramat Aviv, Tel Aviv 69978, Israel*
[2] *School of Engineering, Bar-Ilan University, Ramat-Gan 52900, Israel*

Correspondence should be addressed to Ilya Levin; ilia1@post.tau.ac.il

Received 6 September 2012; Revised 30 December 2012; Accepted 22 January 2013

Academic Editor: Bozidar Sarler

The paper deals with fundamentals of systems of Boolean functions called multi-output functions (MOFs). A new approach to representing MOFs is introduced based on a Generalized If-Then-Else (GITE) function. It is shown that known operations on MOFs may be expressed by a GITE function. The GITE forms the algebra of MOFs. We use the properties of this algebra to solve an MOF-decomposition problem. The solution provides a compact representation of MOFs.

1. Introduction

Logic design, as a scientific discipline, has a fascinating history. This field was intensively studied from the 1950s to the 1970s. This produced many remarkable results including the theorem of function completeness, optimization techniques, decomposition techniques, a number of spectral methods, and a theory of multivalued functions [1–7]. However, there are topics in logic design which are still interesting and have untapped potential from a theoretical point of view [8, 9]. One such topic is "systems of logic functions" (logic systems) and their corresponding representations. The most popular representations of logic systems are (a) the matrix (which is actually a Karnaugh-like form) and (b) the Multiterminal Binary Decision Diagram (MTBDD) [10–14]. There are three known ways to treat and optimize systems of logic functions: (a) as a set of single decision diagrams with two terminal nodes that may share conditional nodes (Shared Binary Decision Diagrams (SBDDs)) [15]; (b) as a single, so-called characteristic function, where the output of each function of the system is considered to be an additional input variable of the characteristic function [16]; (c) as a single decision diagram whose terminals are words [14].

In this paper, we present a different approach. We represent a system of logic functions as a network of interconnected subfunctions. In other words, we decompose a given system into a number of subsystems of lower complexity, thus achieving a compact representation of the given system.

We introduce a new Generalized If-Then-Else operator, denoted as GITE, and an algebra based on this operator. A system of logic functions can be described as a GITE formula. In this sense, the conventional algebra of the Boolean formulas is a particular case of the GITE algebra. In turn, the new operator is considered to be a generalization of the known If-Then-Else (ITE) operator that is widely used in computer science.

GITE-algebra makes it possible to formulate and solve various optimization problems. In our work, we formulate a general optimization problem, which is the decomposition of a system into a *compact* network of GITE components with predefined characteristics. We present a solution of this task in the form of an optimization algorithm. The decomposition algorithm is based on a theorem of GITE formula transformation presented in Section 4.

The paper is organized as follows. Basics of the theory of logic functions and Boolean formula are briefly reviewed in Section 2. The GITE operator is introduced in Section 3. A polynomial representation of GITE formulas is presented in Section 4. The optimization problem is formulated in

Section 5. A solution to the optimization problem is given in Section 6. Section 7 includes experimental results. Conclusions are given in Section 8.

2. Preliminaries

In this section we review some fundamentals of logic design underlying our work.

2.1. Representation of a Single Logic Function. A logic function $f(x_1, \ldots, x_n)$ is a function that takes the values of 0 and 1. An expression obtained by substitution of functions into each other followed by renaming variables is called a *formula* that describes this substitution. The formula-based representation of logic functions is an *analytical* expression.

Let f be a set of functions, and let $\mathcal{F}$ be a set of formulas. We define the depth of a formula $F \in \mathcal{F}$ as follows.

Definition 1. Symbols representing the input variables are considered to be formulas of *depth* 0. A formula F has depth $(k+1)$ if F can be expressed as $f_i(F_1, \ldots, F_{n_i})$, where $f_i \in f$, $F_1, \ldots, F_{n_i} \in \mathcal{F}$ are formulas of the maximal depth k, and n_i is the number of arguments of f_i.

The formulas $F_1, \ldots, F_{n_i}$ are called subformulas of F. Function f_i is called the *outside* or the *main* operation of formula F. All subformulas of the $F_1, \ldots, F_{n_i}$ are also considered as subformulas of F.

When talking about a formula corresponding to a specific function, it is acceptable to say that a formula *represents* the function. Unlike the truth table representation of a function, the formula representation is not unique. Formulas representing one and the same function are *equivalent*. Actually, in most cases, we do not deal with functions but rather with formulas representing these functions. In other words, we deal with the algebra of formulas [17, 18].

One of the popular forms for the representation of a logic function is a Binary Decision Diagram (BDD) [17]. A fundamental operator enabling operations between BDDs is the *If-Then-Else (ITE)* operator [19]. In our paper, we use the ITE operator as an operator on the system of logic functions and not just on the set of BDDs.

The ITE operator serves to represent any function of two variables [17, 19] and consequently may be considered as a *universal* basis for the set of logic functions. In other words, any function of n variables can be represented as a substitution of ITEs. We define the depth of an ITE formula as follows.

Definition 2. Symbols representing the input variables are considered ITE formulas of *depth* 0. An ITE formula F has depth $(k+1)$ if F can be expressed as $f_i(F_1, \ldots, F_{n_i})$, where $f_i \in f, F_1, \ldots F_{n_i} \in \mathcal{F}$ are ITE formulas of the maximal depth k.

2.2. Representation of the System of Functions. The present study deals primarily with a system of logic functions and not necessarily a single Boolean function. The system of logic functions can be considered as a single function of n input Boolean variables and m Boolean output variables. Such functions are called *multi-output functions* (MOFs) [10, 11, 20–23]. The domain and range of an MOF are the n-dimensional and m-dimensional Boolean cubes, $\mathbb{B}^n$ and $\mathbb{B}^m$, respectively. We refer to a vertex of a Boolean cube by its corresponding integer value or as a minterm (A literal is a variable or its complement. A minterm is a product of all the literals). An MOF can be defined by a truth vector (also called a truth table); that is, a list of all possible input combinations with the corresponding output values (vectors). In this paper we refer to the output vectors by their corresponding integer values. An *analytical* representation of MOFs differs from the conventional Boolean representation. An analytical MOF representation is the focus of the present study.

Multiterminal Binary Decision Diagram (MTBDD) is a data structure for representation and efficient manipulations with MOFs [10, 24]. Unlike the conventional BDD that has two terminal nodes, the MTBDD has a number of the terminal nodes.

Definition 3. An MTBDD is a directed acyclic graph, representing an MOF or a set of logic functions $f : \mathbb{B}^n \to \mathbb{B}^m$.

Properties of MTBDD and other MOF representations and their applications have been studied intensively [25–33]. To the best of our knowledge, some aspects of the representation of MOFs and especially an algebra for MOFs have never been developed, although two main operations on MOF, the Apply and the ITE, were studied in [10]. The Apply may be used to accomplish a large number of matrix operations. Its definition is $\text{Apply}(f, g, \circ) = f \circ g$, where f and g are MOFs and $\circ$ is any binary operation on two operands, for example, +, −, min, max, and so forth.

The ITE operator has three arguments. The first argument takes 0 or 1. The second and third arguments are MOFs. ITE is defined as follows:

$$\text{ITE}(f, g, h) = \begin{cases} g & \text{if } f = 1 \\ h & \text{if } f = 0. \end{cases} \tag{1}$$

The following example illustrates the above operations. To simplify the explanation we use the truth vector representation of MOFs. That is, a function f of n variables is represented as a vector of length 2^n, $f = [v_0, v_1, \ldots, v_{2^n-1}]$, where v_ω is the value of the function $f(x)$ at $x = \omega$.

Example 4. Let f, g, and h be three MOF functions specified by the following truth vectors:

$$f = [1, 0, 0, 1], \qquad g = [4, 4, 2, 2], \qquad h = [3, 3, 3, 3], \tag{2}$$

and let *op* stand for addition of integers. Then,

$$\begin{aligned} \text{Apply}(f, g, +) &= [5, 4, 2, 3], \\ \text{ITE}(f, g, h) &= [4, 3, 3, 2]. \end{aligned} \tag{3}$$

In the following section we generalize these two operations to define the algebra of MOFs.

2.3. Partition on a Boolean Cube. In this section we describe the concept of partition on Boolean cubes.

Definition 5. A partition π on a set S is a collection of w disjoint subsets of S whose set union is S; that is, $\pi = \{B_i\}_{i=1}^{w}$ such that $B_i \cap B_j = \emptyset$ $(i \neq j)$ and $\cup_{i=1}^{w} B_i = S$.

We refer to the subsets of π as blocks of π.

The fact that two distinct elements s and t are in the same block k of π is denoted as $s \equiv t(\pi)$. In other words, $s \equiv t(\pi)$ if and only if there exists k such that $s \in B_k$ and $t \in B_k$.

Definition 6 (intersection of partitions). Let π_1 and π_2 be two partitions. The product $\pi = \pi_1 * \pi_2$ is a partition comprising intersections of blocks of π_1 and π_2, such that

$$s \equiv t(\pi_1 * \pi_2) \Longleftrightarrow s \equiv t(\pi_2), \qquad s \equiv t(\pi_1). \tag{4}$$

Definition 7 (summation of partitions). Let π_1 and π_2 be two partitions. The sum $\pi = \pi_1 + \pi_2$ consists of blocks for which $s \equiv t(\pi_1 + \pi_2)$ if and only if a chain $s_0, s_1, \ldots, s_n$ exists in S such as: $s = s_0, s_1, \ldots, s_n = t$ for which either $s_i \equiv s_{i+1}(\pi_1)$ or $s_i \equiv s_{i+1}(\pi_2)$, $0 \leq i \leq n-1$.

We denote the product and the sum of partitions as

$$\begin{aligned} \prod_{i=1}^{n} \pi_i &= \pi_1 * \cdots * \pi_n, \\ \sum_{i=1}^{n} \pi_i &= \pi_1 + \cdots + \pi_n, \end{aligned} \tag{5}$$

respectively.

For π_1 and π_2, we say that π_1 is larger or equal to π_2 and write $\pi_1 \geq \pi_2$ if and only if every block of π_2 is contained in a block of π_1. Thus, $\pi_1 \geq \pi_2$ if $\pi_1 * \pi_2 = \pi_2$ and if and only if $\pi_1 + \pi_2 = \pi_1$. The algebraic structure of partitions is known as a lattice. This lattice has both *Zero* (the smallest partition) and *One* (the largest partition). These elements are

$$\begin{aligned} \pi^0 &= \{\{s_1\}; \ldots; \{s_m\}\}, \\ \pi^1 &= \{s_1, \ldots, s_m\}. \end{aligned} \tag{6}$$

Obviously,

$$\pi^0 \leq \pi_1 * \pi_2 \leq \pi_1, \qquad \pi_2 \leq \pi_1 + \pi_2 \leq \pi^1. \tag{7}$$

In this paper, the partition is performed on the n-dimensional Boolean cube $\mathbb{B}^n$. A block of the partition is the subsets of vertices of the cube. Vertices that belong to the same block must have the same output value. The *Zero* partition π^0 on a Boolean cube corresponds to the Sum-Of-Minterms (SOM) representation of a function.

Example 8. Consider a function $f(x_2, x_1, x_0)$ specified by the truth vector

$$f = [v_0, \ldots v_7] = [2, 2, 1, 1, 1, 1, 2, 0] \tag{8}$$

and a partition

$$\pi = \{B_1, B_2, B_3, B_4\} = \{\{0, 1\}; \{2, 3, 4, 5\}; \{6\}; \{7\}\}. \tag{9}$$

Note that all the elements in a block are associated with the same value of the function, for example, $v_0 = v_1 = 2$. Consider a different partition

$$\widehat{\pi} = \{\{0\}; \{1, 6\}; \{2, 3, 4, 5\}; \{7\}\}. \tag{10}$$

The product of partitions is

$$\pi * \widehat{\pi} = \{\{0\}; \{1\}; \{2, 3, 4, 5\}; 6; \{7\}\}. \tag{11}$$

The sum of partitions is

$$\pi + \widehat{\pi} = \{\{0, 1, 6\}; \{2, 3, 4, 5\}; \{7\}\}. \tag{12}$$

Since we are dealing with partitions of the Boolean cube, blocks in the partition can be expressed as Boolean functions. For example, the Boolean functions that correspond to the blocks of

$$\pi = \{\{0, 1\}; \{2, 3, 4, 5\}; \{6\}; \{7\}\} \tag{13}$$

are

$$\begin{aligned} \{0, 1\} &\Longrightarrow \overline{x}_2 \overline{x}_1, \\ \{2, 3, 4, 5\} &\Longrightarrow \overline{x}_2 x_1 + x_2 \overline{x}_1, \\ \{6\} &\Longrightarrow x_2 x_1 \overline{x}_0, \\ \{7\} &\Longrightarrow x_2 x_1 x_0. \end{aligned} \tag{14}$$

3. The Generalized ITE Operator

The first argument in the definition of the ITE operator can be interpreted as a two-block partition of the Boolean cube or as a Boolean function. The present paper is focused on a *Generalized ITE (GITE) operator* which uses an n-block partition instead of the two-block partition.

3.1. Definition of the GITE Operator. First, we define the GITE operator on MOFs that are specified by truth vectors.

Definition 9. Let π be a partition of the n-dimensional Boolean cube comprising w blocks: $\pi = \{B_i\}_{i=1}^{w}$, and let $\{h_i\}_{i=1}^{w}$ be a set of MOFs defined by their truth vectors:

$$h_i = \left[v_{h_i,0}, v_{h_i,1}, \ldots v_{h_i,2^n-1}\right]. \tag{15}$$

Then, the GITE operator is $g = \mathrm{GITE}(\pi, h_1, \ldots h_w)$, where the jth element in the truth vector g is

$$v_{g,j} = \sum_{i=1}^{w} b_{i,j} v_{h_i,j}, \quad 0 \leq j < 2^n, \tag{16}$$

where

$$b_{i,j} = \begin{cases} 1 & \text{if } j \in B_i \\ 0 & \text{if otherwise.} \end{cases} \tag{17}$$

Example 10. Let π be a 4-block partition of $\mathbb{B}^3$, and let the vertices of the Boolean cube be addressed by their corresponding integer value, for example, 6 = (110),

$$\pi = \{B_1, B_2, B_3, B_4\} = \{1; 2; \{3, 4, 7\}; \{0, 5, 6\}\}. \tag{18}$$

Let $g = \mathrm{GITE}(\pi, h_1, h_2, h_3, h_4)$, where

$$\begin{aligned} h_1 &= [1, 1, 2, 1, 1, 1, 2, 2], & h_2 &= [3, 3, 3, 3, 3, 3, 3, 3], \\ h_3 &= [2, 2, 4, 4, 2, 2, 4, 4], & h_4 &= [5, 6, 5, 6, 5, 6, 5, 6]. \end{aligned} \tag{19}$$

Then g is

$$\begin{aligned} g &= \left[v_{g,0}, v_{g,1}, \ldots, v_{g,7}\right] \\ &= \left[v_{h_4,0}, v_{h_1,1}, v_{h_2,2}, v_{h_3,3}, v_{h_3,4}, v_{h_4,5}, v_{h_4,6}, v_{h_3,7}\right] \\ &= [5, 1, 3, 4, 2, 6, 5, 4]. \end{aligned} \tag{20}$$

Now we are ready to formulate the concept of MOF in terms of GITE.

Definition 11. An MOF is a mapping from $\mathbb{B}^n$ to $\mathbb{B}^m$, which is a GITE(π, Y) operation defined on two sets: the set of partitions and the set Y, $Y \subseteq \mathbb{B}^m$ of terminals (In [10], the values of an MOF, that is, the Y's, are called *terminal nodes* or terminals (this is different from the internal nodes that correspond to variables). In Algorithmic State Machine theory, the values of the MOF represent operations to be performed and hence are called *operators* [34]. In this paper we use the term "terminals.").

In other words, the GITE comprises a *partition* portion and a *terminals* portion. The GITE operation maps a partition of the Boolean cube $\mathbb{B}^n$ on the predefined set Y.

3.2. GITE Formulas. In this section we introduce the algebra of GITE formulas. The elements of the algebra are MOFs, and the basic operator is the GITE. We define the depth of the GITE formula as follows.

Definition 12. Symbols of given terminals from the finite set $Y = \{Y_1, \ldots, Y_n\} \subseteq \mathbb{B}^m$ are considered as GITE formulas of *depth* 0. A GITE formula G has depth $(k + 1)$ if G can be expressed as $G = \mathrm{GITE}(\pi, (G_1, \ldots, G_{n_i}))$, where $G_1, \ldots, G_{n_i}$ are formulas of the maximal depth k and n_i is the number of blocks in the partition π.

Unlike the case of the algebra of Boolean functions, the algebra of GITE formulas contains an additional operation *composition*. The composition operation (*Compose*) corresponds to the known Apply operation in the algebra of ITE. Composition means to construct a GITE from two other GITEs. The values of the composed GITE are calculated by performing a bitwise operation denoted by "$\circ$" on the values of the given GITEs.

Definition 13 (composition). Let

$$\begin{aligned} G_1 &= \mathrm{GITE}\left(\pi_1, Y_{1,1}, \ldots, Y_{1,k}\right), \\ G_2 &= \mathrm{GITE}\left(\pi_2, Y_{2,1}, \ldots, Y_{2,m}\right), \end{aligned} \tag{21}$$

then,

$$\begin{aligned} &\mathrm{Compose}\left(G_1, G_2\right) \\ &\quad \stackrel{\mathrm{def}}{=} G_1 \circ G_2 \\ &\quad = \mathrm{GITE}\left(\pi_1 * \pi_2; Y_{1,1} \circ Y_{2,1}, \ldots, Y_{1,k} \circ Y_{2,m}\right). \end{aligned} \tag{22}$$

In other words, the composition of the GITE is performed by multiplying the corresponding partitions π_1 and π_2 and pairwise "$\circ$" operations on terminal Y.

Example 14. Let

$$\begin{aligned} \pi_1 &= \left\{B_1^1, B_2^1, B_3^1, B_4^1\right\} = \{1; 2; \{3, 4, 7\}; \{0, 5, 6\}\}, \\ \pi_2 &= \left\{B_1^2, B_2^2, B_0^2\right\} = \{\{1, 2, 3\}; \{0, 4, 7\}; \{5, 6\}\}, \end{aligned} \tag{23}$$

and let

$$\begin{aligned} G_1 &= \mathrm{GITE}\left(\pi_1, Y_{1,1}, Y_{1,2}, Y_{1,3}, Y_{1,4}\right), \\ G_2 &= \mathrm{GITE}\left(\pi_2, Y_{2,1}, Y_{2,2}, Y_{2,0}\right), \end{aligned} \tag{24}$$

where

$$\begin{aligned} Y_{1,1} = 2, \quad Y_{1,2} = 6, \quad Y_{1,3} = 5, \quad Y_{1,4} = 11, \\ Y_{2,1} = 3, \quad Y_{2,2} = 8, \quad Y_{2,0} = 0. \end{aligned} \tag{25}$$

Let the "$\circ$" stand for a bitwise XOR between two integers represented in radix two. Then,

$$\begin{aligned} \pi = \pi_1 * \pi_2 &= \{0; 1; 2; 3; \{4, 7\}; \{5, 6\}\}, \\ G_1 \circ G_2 &= \mathrm{Compose}\left(G_1, G_2\right) \\ &= \mathrm{GITE}\left(\pi; Y_{1,4} \circ Y_{2,2}, Y_{1,1} \circ Y_{2,1}, Y_{1,2} \circ Y_{2,1}, \right. \\ &\qquad \left. Y_{1,3} \circ Y_{2,1}, Y_{1,3} \circ Y_{2,2}, Y_{1,4} \circ Y_{2,0}\right) \\ &= \mathrm{GITE}\left(\pi; 3, 1, 5, 6, 13, 11\right). \end{aligned} \tag{26}$$

4. *D*-Polynomials Analytical Representation of GITE Formulas

In this section we describe a special type of GITE formulas. This type corresponds to the case where a GITE formula is defined by partitions expressed in an analytic form—the form of a Boolean expression—the Sum-of-Products (SOP) form. We call this analytical representation a *D*-polynomial representation [35–37].

Definition 15. Let $B_1, \ldots, B_m$ be Boolean functions expressed in an SOP form, and let $\pi = \{B_1, \ldots, B_m, B_0\}$ be a partition, where $B_0 = \overline{\vee_{i=1}^{m} B_i}$. A D-polynomial is defined as follows:

$$\begin{aligned} D &\stackrel{\text{def}}{=} \sum_{i=1}^{m} B_i Y_i + B_0 Y_0 \\ &= \text{GITE}(\pi, Y) \\ &= \text{GITE}(B_1, \ldots, B_m, B_0; Y_1, \ldots, Y_m, Y_0), \end{aligned} \tag{27}$$

where $Y_0 = (0, \ldots, 0) \in \mathbb{B}^m$.

A D-polynomial D can be interpreted as follows. If B_i evaluates to 1, then $D = Y_i$. If all of the explicit functions B_i are equal to 0, then $D = Y_0$.

A *D-polynomial of depth* 0 is a formula defined over a set of terminal Y. D-polynomials of *depth k* are defined in the same way as in the general case of GITE formulas.

Since the D-polynomials are defined in the SOP-based analytical form, the *Compose* operation on the set of D-polynomials may be easier to perform than in the general case of GITE, namely, by using Boolean operations. In turn, the *Compose* operation is interpreted as a composition of D-polynomials.

Definition 16 (composition of D-polynomials). Let

$$D_1 = \sum_{i=1}^{m} B_i^1 Y_i + B_0^1 Y_0, \qquad D_2 = \sum_{i=1}^{m} B_i^2 Y_i + B_0^2 Y_0. \tag{28}$$

The composition of D_1 and D_2 denoted by $D_1 \circ D_2$ is defined as

$$D_1 \circ D_2 = \sum_{i,j} \left(B_i^1 \cdot B_j^2\right) \left\{Y_i \circ Y_j\right\} + \left(B_0^1 \cdot B_0^2\right) Y_0, \tag{29}$$

over each pair of products from D_1 and D_2, including the implicit terms $B_0^1 Y_0$ and $B_0^2 Y_0$.

Here $B_i^1 \cdot B_j^2$ is a logic product (AND) of the corresponding functions, and $Y_i \circ Y_j$ is a predefined bitwise operation between Y_i and Y_j. In other words, when $B_i^1 \cdot B_j^2$ evaluates to 1, the operation between Y_i and Y_j is performed.

Note that partition algebra and Boolean algebra use different terminologies. Consequently, the terminologies of the GITE and D-polynomials are also different. In particular, if two partition blocks are disjoint, their corresponding Boolean functions are *orthogonal*.

According to this definition, the composition operation corresponds to the *product of the partition* of D-polynomials. Next we show that the D-polynomial operation that corresponds to the *sum of the partition portions* is not a sum of D-polynomials (as may be expected), but is a *factorization*.

Without loss of generality, let a D-polynomial D be represented as $D = D_1 \circ D_2$.

Definition 17 (factorization of D-polynomials). The factorization of D with respect to D_1 and D_2 is its representation in the following form:

$$D = \text{GITE}\left(\pi_1 + \pi_2; \widehat{D}_1, \ldots, \widehat{D}_l\right), \tag{30}$$

where l is a number of blocks in $(\pi_1 + \pi_2)$ and $\widehat{D}_1, \ldots, \widehat{D}_l$ stand for D-polynomials corresponding to the remaining functions.

Note that sum of the partition is equivalent to the max operation in lattices [22]. In this sense the sum is the minimal partition that is larger than both of them; that is, $\pi_1 + \pi_2 \geq \pi_1, \pi_2$ (see Definition 7). Hence, the sum can be interpreted as a common factor.

Example 18. Consider two partitions:

$$\begin{aligned} \pi_1 &= \left\{B_1^1, B_2^1, B_3^1, B_0^1\right\} = \left\{x_1, \overline{x}_1 \overline{x}_2, \overline{x}_1 x_2, 0\right\}, \\ \pi_2 &= \left\{B_1^2, B_2^2, B_3^2, B_0^2\right\} = \left\{\overline{x}_1, x_1 \overline{x}_3, x_1 x_3, 0\right\}. \end{aligned} \tag{31}$$

Let

$$\begin{aligned} D_1 &= \text{GITE}(\pi_1, Y_1, Y_2, Y_3, Y_0) \\ &= x_1 Y_1 + \overline{x}_1 \overline{x}_2 Y_2 + \overline{x}_1 x_2 Y_3, \\ D_2 &= \text{GITE}(\pi_1, Y_4, Y_5, Y_6, Y_0) \\ &= \overline{x}_1 Y_4 + x_1 \overline{x}_3 Y_5 + x_1 x_3 Y_6. \end{aligned} \tag{32}$$

Then, $\pi_1 + \pi_2 = \{x_1, \overline{x}_1, 0\}$, and

$$\begin{aligned} D_1 \circ D_2 &= \text{GITE}(\pi_1 + \pi_2; D_{156}, D_{234}, Y_0) \\ &= x_1 D_{156} + \overline{x}_1 D_{234}, \end{aligned} \tag{33}$$

where

$$\begin{aligned} D_{156} &= Y_1 \circ \left(\overline{x}_3 Y_5 + x_3 Y_6\right), \\ D_{234} &= \left(\overline{x}_2 Y_2 + x_2 Y_3\right) \circ Y_4. \end{aligned} \tag{34}$$

Laws and Properties of D-Polynomials. The D-polynomials are the special class of GITE formulas. In what follows we use the D-polynomial notation to represent and manipulate MOFs. There are two main reasons for using the D-polynomial notation: (a) D-polynomials are formulas; (b) D-polynomials support SOP function representations.

We now introduce a substitution of certain D-polynomials into another D-polynomial as follows. Let D_1, D_2, and D_3 be MOFs:

$$\begin{aligned} D_1 &= \text{GITE}(\pi_1; Y_{11}, Y_{12}), \\ D_2 &= \text{GITE}(\pi_2; Y_{21}, \ldots, Y_{2m}), \\ D_3 &= \text{GITE}(\pi_3; Y_{31}, \ldots, Y_{3k}). \end{aligned} \tag{35}$$

After substitution $Y_{11} \leftarrow D_2, Y_{12} \leftarrow D_3$ we have $D_1' = \text{GITE}(\pi_1; D_2, D_3)$ which is a hierarchical structure comprising a number of D-polynomials.

In general, let $D = \mathrm{GITE}(\pi; Y_1, \ldots, Y_m)$ be a formula representing a multi-output function f. A substitution of formula $D_1, \ldots, D_m$ in the place of the terminals gives a new formula

$$D' = \mathrm{GITE}\left(\pi; D_1, \ldots, D_m\right). \tag{36}$$

Obviously, $D' \neq D$ but we consider them as the same function (f) with different arguments. As a result, we can deal with the algebra of functions, which are determined by their partition portion of the corresponding GITE formula. The same situation is found in the conventional Boolean algebra of logic functions where we usually talk about logic operations (AND, OR, NOT, etc.) on the set of logic functions.

A special class of D-polynomials is the class of atomic D-polynomials; that is, D-binomials.

Definition 19. A D-binomial is a special case of D-polynomials, including exactly one product: $D = B_1Y_1 + B_0Y_0$.

The following theorem states that any D-polynomial can be represented as a composition of all of its D-binomials.

Theorem 20. *An arbitrary D-polynomial* $D = \sum_j B_jY_j + B_0Y_0$ *can be represented as a composition of D-binomials* $D_j = (B_jY_j + B_0^jY_0)$, *where* $B_0 = \prod_j B_0^j$.

The proof of this theorem is given in the Appendix.

5. The Decomposition Problem

Each class of hardware technology requires its own specific optimization criterion. Both the technology and the corresponding optimization criteria are changing continuously as a function of progress in the field of hardware and updates in design requirements. However, universal criteria of optimization do exist. These universal criteria relate to a measure of complexity of the formulas for the representation of a discrete function. Complexity may be considered *quantitatively* and *qualitatively*. Here we consider the *compactness* of the formula representation as a quantitative complexity criterion and the *modularity* as the qualitatively complexity criterion. Compactness is an important parameter for storing and remote communication of information, as well as for software representations of discrete functions. In our experiments, we assessed compactness as the number of nodes in the corresponding decision diagram.

The universal qualitative criterion for complexity is modularity. We represent a given discrete function as a hierarchical network of modular components, each performing part of a common functionality. There are number of reasons to prefer a modular (structured) representation. First of all, a structured representation simplifies the process of debugging and testing; further, modules are potentially reusable. Moreover, structured representations usually correspond to their specifications. This correspondence is highly desirable, since it helps understand and interpret the realization of the function and can be seen as a powerful measure of complexity of the representation of the function.

Table 1: The MOF for Example 21.

i	x_0	x_1	x_2	x_3	x_4	F_0	F_1	F_2	F_3	Y
1	0	1	—	0	—	1	0	0	0	8
2	0	1	—	1	—	0	1	1	0	6
3	—	—	1	—	0	0	0	1	1	3
4	—	—	—	—	1	0	0	0	1	1
5	1	0	—	1	—	1	1	0	0	12

Below we discuss methods for transforming a system of logic functions into a structured hierarchical network of interconnected components. We decompose the system while minimizing the total number of nodes in the resulting structure.

Let us consider two extreme cases of decomposition of D-polynomials that are the D-binomials (i.e., the *binomial representation*) and the *monolith*. The monolith corresponds to the D-polynomial $D = \mathrm{GITE}(\pi, Y_1, \ldots, Y_k)$, where π is a partition corresponding to the Reduced Ordered MTBDD representation of MOF [17, 19, 20, 32]. The following example illustrates these two extreme cases.

Example 21. Consider the MOF specified in Table 1. The function has five inputs and four outputs. Each row of the table corresponds to one D-binomial. Hence,

$$D = D_1 \circ D_2 \circ D_3 \circ D_4 \circ D_5, \tag{37}$$

where D_i is the D-binomial that corresponds to the ith row in the table. For example, for $i = 1$,

$$\pi_1 = \left\{B_1, B_0^1\right\} = \left\{\overline{x}_0x_1\overline{x}_3, \overline{\overline{x}_0\overline{x}_1\overline{x}_3}\right\}, \tag{38}$$

and therefore $D_1 = 8B_1 + 0B_0^1$. The binomial representation of the function is shown in Figure 1, and the corresponding monolith is presented in Figure 2. The terminal nodes of the decision diagrams are marked by the decimal numbers of corresponding outputs. Note that, as a result of the composition operation, the sets of terminal nodes in the diagrams are not the same. In this example, let the $\circ$ be a bitwise-OR between corresponding output vectors. For example, terminal node "7" in Figure 2 corresponds to the two terminal nodes "6" and "3" in Figure 1. Such cases where new values are being created stem from the nonorthogonality of products (in our case, B_2 and B_3 are nonorthogonal).

Note that the monolith can be derived by composing all the D-binomials. However, this procedure has high complexity. For example to obtain the monolith representation in Example 21 one has to compose the five D-binomials of D. The first Compose is quite simple since B_1 and B_2 are orthogonal ($B_1B_2 = 0$); that is,

$$D_1 \circ D_2 = 8B_1 + 6B_2 + 0B_0^{1,2}. \tag{39}$$

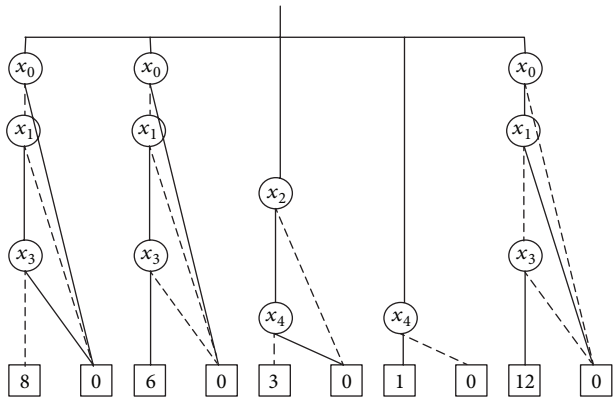

Figure 1: Binomial representation of the MOF in Example 21.

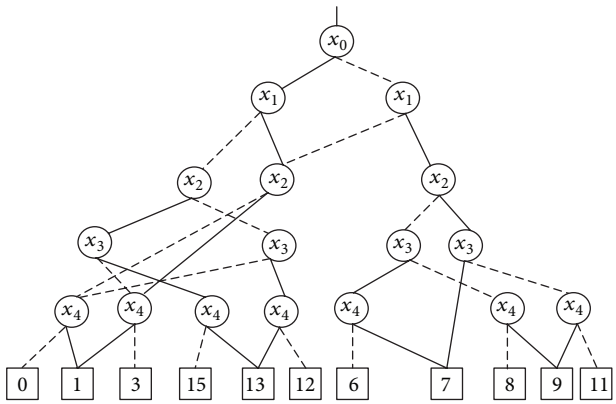

Figure 2: Monolith representation of the MOF in Example 21.

The second Compose is

$$\begin{aligned}(D_1 \circ D_2) \circ D_3 = {} & (8 \circ 3)\, B_1 B_3 + (6 \circ 3)\, B_2 B_3 \\ & + (0 \circ 3)\, B_0^{1,2} B_3 + (8 \circ 0)\, B_1 B_0^3 \\ & + (6 \circ 0)\, B_2 B_0^3 + 0 B_0^{1,2} B_3 \\ = {} & 11 B_1 B_3 + 7 B_2 B_3 + 3 B_0^{1,2} B_3 \\ & + 8 B_1 B_0^3 + 6 B_2 B_0^3 + 0 B_0^{1,2} B_3.\end{aligned} \tag{40}$$

It is possible to avoid this complex calculation and, at the same time, reduce the resulting number of MTBDD nodes by factorization on subsets of D-binomials. The proposed decomposition presented below is based on this principle.

There are many ways to group D-binomials to form a network of D-polynomials. Different groupings of the binomials yield different representations of MOF. Each D-polynomial in the network has its own optimal structure, that is, its own optimal header. This fact forms the basis of our decomposition approach. The decomposition goal is to represent the given D-polynomial in a compact form so as to optimize a certain cost function, for example, the number of nodes in the corresponding decision diagram. The number of nodes in the decomposed D-polynomial is upper bounded by $\min(2^n, nm)$.

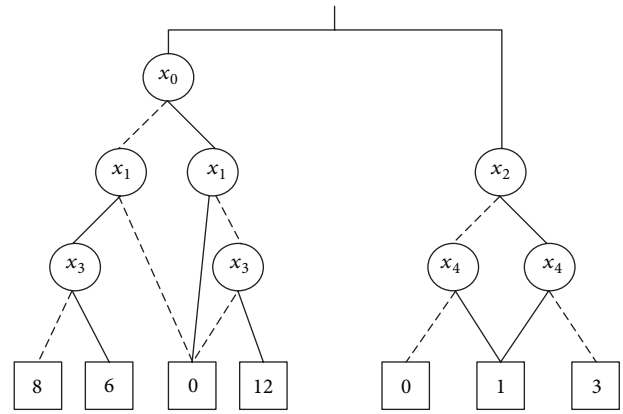

Figure 3: Decomposed MTBDD corresponding to Example 21.

In our example, Figure 3 shows a compact representation of the MOF as a network of MTBDDs. Namely,

$$D = D_{\text{left}} \circ D_{\text{right}} = \left(D_1 \circ D_2 \circ D_5\right) \circ \left(D_3 \circ D_4\right). \tag{41}$$

For comparison, in this example, the monolith MTBDD has 17 nonterminal nodes (NTNs), whereas the decomposition allows to reduce the number of nonterminal nodes to 8.

6. Decomposition Algorithm

The main decomposition algorithm is a recursive grouping of the set of products representing the given D-polynomial to a number of portions. The main decomposition algorithm is presented in Figure 4. Each recursion step comprises a fragmentation of current portion into a *block* and a *remainder*. In turn, a block is divided into a *block header* and a number of *block fragments* (*tails*). The fragmentation algorithm is described below and presented in Figure 5.

The main decomposition algorithm uses a stack for saving a current portion of the given D-polynomial. After the fragmentation of the current portion, each of the resulting tails the remainder are saved in the stack sequentially. If the remainder portion is empty, which means that all products of the current portion of the D-polynomial are included in the block; then, obviously, nothing is saved into the stack. Similarly, if a certain resulting tail comprises just one product (one terminal), then nothing is saved into the stack. The main decomposition algorithm stops when the stack is empty. Clearly, this happens when all products of the given D-polynomial are distributed between blocks.

6.1. Fragmentation Algorithm. In each step, the fragmentation algorithm divides the set of D-binomials (which is the set of products) into two subsets. One subset is called T_B, and the second subset is called T_R; T_B consists of the products that determine the *block* D-polynomial—B, and T_R contains the products that form the *reminder* D-polynomial—R. Formally,

$$\widehat{D} = B \circ R = \text{GITE}\left(\pi_h, D_1, \ldots D_j\right) \circ R, \tag{42}$$

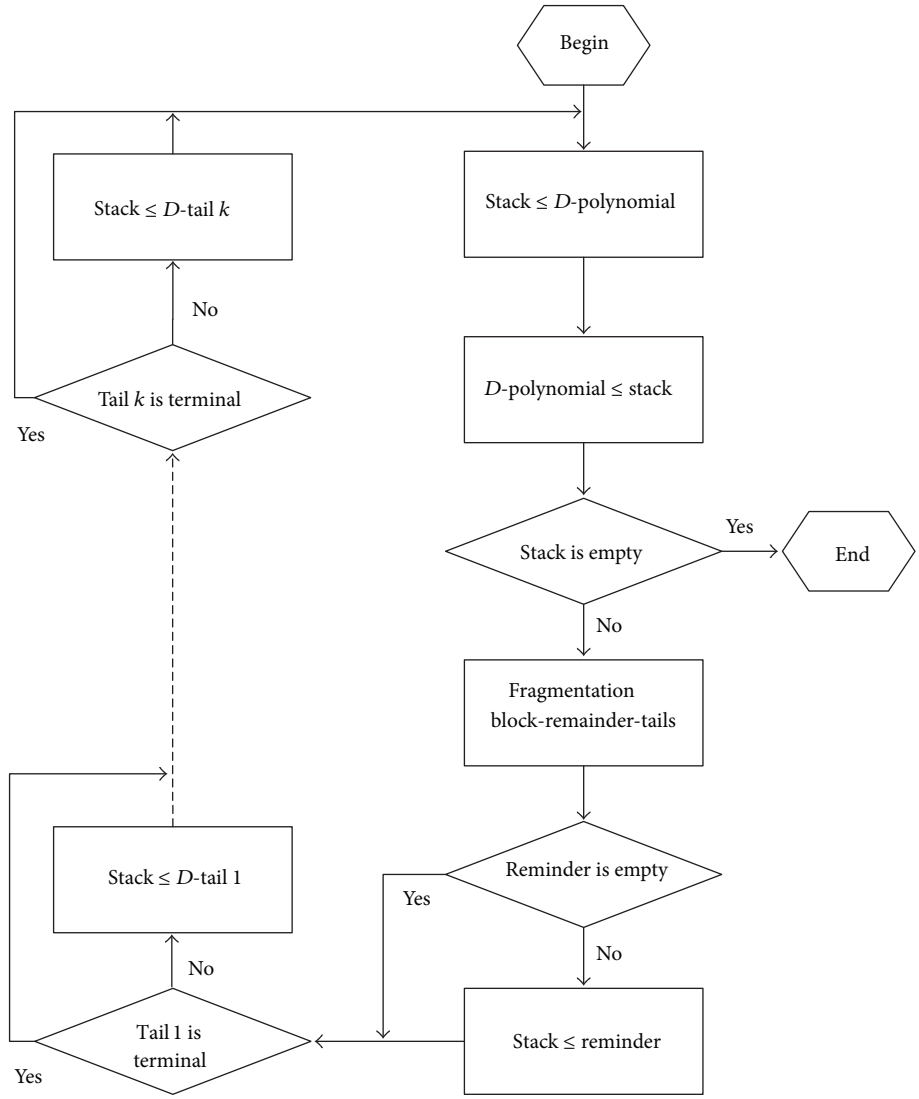

Figure 4: Block diagram of the main decomposition procedure.

where the partition π_h determines a common factor (block header) and D_i's (block fragments) of B. The block header is selected in such a way so as to provide a minimization of the resulting structure.

Definition 22 (prefix). Let P be a product in a block. Each product P' covering the product P is called a *prefix*.

The set of all prefixes associated with the products of the block defines the block *header*.

The block header can be represented as a monolith whose internal nodes are associated with the prefix variables. The terminal nodes of the monolith correspond to the block fragments representing a D-polynomial with the remaining input variables. We call such fragments *tails*.

In what follows we describe fragmentation algorithm.

In each iteration the fragmentation algorithm chooses the prefixes for the current block. The prefixes that form the block header are chosen one by one. Each newly added prefix must be orthogonal to all prefixes accumulated so far. The algorithm is depicted in Figure 5.

Each iteration starts by preparing a list of candidate prefixes (see Section 6.3). Then, the first (basic) prefix is chosen. The basic prefix defines the set of input variables that will determine the partition π_h. Therefore it has special significance. The left hand side of Figure 5 shows the steps related to choosing the basic prefix. The set of criteria for ranking the candidate prefixes is described in Section 6.4. After choosing the prefix that is ranked the highest as the basic prefix, the algorithm constructs the block header by adding secondary prefixes. The right hand side of Figure 5 shows the algorithm that gathers all the prefixes that form the block header. The set of criteria for ranking the candidate secondary prefixes is described in Section 6.5. Let us start by presenting the notations.

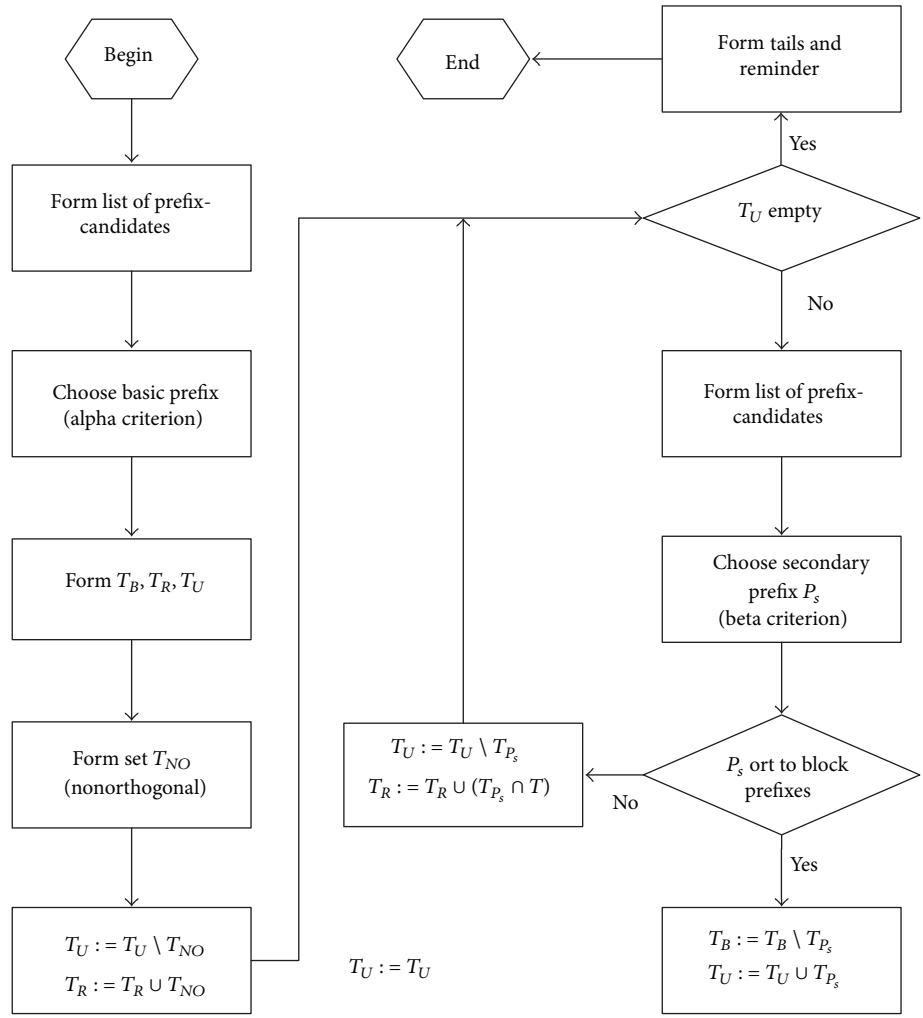

Figure 5: Block diagram of the fragmentation Algorithm.

6.2. Notations

(i) T—the set of products for a given MOF.

(ii) P—the prefix under consideration, that is, the basic prefix in the block header.

(iii) P_s—a secondary prefix to be added to the block header.

(iv) M—the number of variables in the prefix.

(v) $T_B(P)$—the set of products having prefix P. The set $T_B(P)$ is called the family of the prefix. When it is clear from the context we write T_B instead of $T_B(P)$.

(vi) $T_R(P)$—the set of the products that do not depend on any of the prefix variables.

(vii) $T_U(P)$—the set of products depending on some of the prefix variables, $T_U = R \setminus (T_P \cup T_R)$. Set T_U is the "undecided" set, since these products are neither in the prefix family nor in its remainder.

(viii) $T_O(P)$—the set of products orthogonal to the prefix. Clearly, $T_O \subseteq T_U$.

(ix) $T_{NO}(P)$—the set of products that are not orthogonal to the prefix, $T_{NO} = T_U \setminus T_O$.

(x) $X(A)$—the set of variables in all the products in a set A.

(xi) $S(A)$—the number of literals in all the products in a set A.

(xii) $Y(A)$—the set of outputs corresponding to all the products in a set A.

Example 23. Consider the function specified in Table 1. The function is specified by five products.

Let $P = \overline{x}_0 x_1$ be a prefix under consideration. The number of variables in P is $M = 2$. The family of P, that is, the set of products having prefix P, is $T_B = \{B_1, B_2\}$. There are two products in T_B. Note that the set of variables in all the products of T_B is $X(T_B) = \{x_0, x_1, x_3\}$, the number of literals in all the products of T_B is $S(T_B) = 2 \cdot 3 = 6$, and the set of

INPUT: List of products T_U.
OUTPUT: List of candidates.
Output ← *Input*
*Temp*1 ← *Output*
REPEAT
  *Temp*2 ← Φ
  Compose *common* to every pair of products from *Temp*1.
If the result is not empty, add it to *Temp*2.
  Output ← *Output* ∪ *Temp*2
  *Temp*1 ← *Temp*2
UNTIL |*Temp*1| = 0

Algorithm 1: Constructing the list of candidates.

outputs corresponding to all the products of T_B is $Y(T_B) = \{8, 6\}$.

The set of the products that do not depend on any of the prefix variables is $T_R = \{B_3, B_4\}$, and the set of products depending on some of the prefix variables is $T_U = \{B_5\}$. Note that B_5 is orthogonal to the prefix; hence $T_0 = \{B_5\}$.

6.3. Preparing the List of Candidates. The prefix can be either a product or a product that covers it. A straightforward procedure is proposed below for constructing the list of the candidates from the products in $T_U(P)$.

Let x, y be variables with values from $\{0, 1, -\}$. Define an operator $\Psi(x, y)$, that compares these two Boolean variables x and y, and returns the value of one of them if they are equal, and otherwise it returns a "–". The function *Common* (T_1, T_2) accepts two products T_1 and T_2 and applies Ψ in a bitwise manner to each of the variables in the set $X(T_1) \cup X(T_2)$. The suggested procedure for constructing the list of candidates is presented in Algorithm 1.

6.4. Choosing the Basic Prefix. The basic prefix is the foundation of a block. It is chosen to simplify the representation of the block header. For this, the basic prefix has to attract the secondary prefixes "close" to it and repel those "far" from it.

There are three main concerns to consider here: the input variables, the output functions, and the length of the prefix. In addition, since the secondary prefixes will be chosen from set T_O, it is imperative to measure the orthogonality of T_U. The four criteria are as follows.

The first criterion fulfills the input requirement:

$$\alpha_X = 1 - \frac{\left|X\left(T_B\right) \cap X\left(T_R\right)\right|}{\left|\left(X\left(T_B\right) \setminus X\left(P\right)\right) \cup X\left(T_R\right)\right|}. \tag{43}$$

It counts the variables common to the tail and the remainder corresponding to the prefix. The ratio must be reduced as much as possible to separate the block (with its tails) from the remainder. This criterion has values in the $[0, 1]$ interval, where 0 corresponds to the case, where all the remainder variables are present in the tail, and 1 to the opposite.

The second criterion responds to the output requirement:

$$\alpha_Y = 1 - \frac{\left|Y\left(T_B\right) \cap Y\left(T_R\right)\right|}{\left|Y\left(T_B\right) \cup Y\left(T_R\right)\right|}. \tag{44}$$

It counts the outputs common to the tail and the remainder corresponding to the prefix. The rationale here is the same as for the input requirement.

The third criterion, called *Prefix Significance*, measures the percentage of literals in the products of the prefix family:

$$\alpha_P = \frac{M \cdot \left|T_B\right|}{S\left(T_B\right)}. \tag{45}$$

The reason for this is simple: the longer the basic prefix, the longer the list of candidates for the secondary prefixes.

The last criterion, called *Orthogonality*, responds to an additional requirement. It counts the number of literals in the products orthogonal to the prefix relative to the number of literals in all the candidates:

$$\alpha_T = \frac{S\left(T_O\right)}{S\left(T_U\right)}. \tag{46}$$

The weighted grade of a candidate prefix is defined as $\alpha = a_X\alpha_X + a_Y\alpha_Y + a_P\alpha_P + a_T\alpha_T$. When choosing the basic prefix, the candidate with the highest α is taken. Note that the coefficients of the criteria should be chosen so as to reflect the relative significance/contribution of each criterion to the quality of the overall solution. In this paper (since it is conceptual) we assumed that all the criteria were equally significant; that is, the experimental results described in Section 7 were produced with $a_X = a_Y = a_P = a_T = 1$. Therefore, the results are suboptimal: they can be further improved. This however is left for future study.

6.5. Construction of the Block Header by Choosing the Secondary Prefixes. In the following equations, the superscript indices i and $i + 1$ stand for "current situation" and "after adding the target prefix," respectively.

The first criterion, called *Additional Inputs*, counts the number of variables common to the tail and to the remainder of the target prefix, but only those not yet present in the block,

$$\beta_X = 1 - \frac{\left|X^{i+1}(T_B) \cap X^{i+1}(T_R)\right| - \left|X^{i}(T_B) \cap X^{i}(T_R)\right|}{\left|\left(X^{i+1}(T_B) \setminus X^{i+1}(P)\right) \cup X^{i+1}(T_R)\right|}. \tag{47}$$

The second criterion, called *Additional Outputs*, counts the number of output functions common to the tail and to the remainder of the prefix:

$$\beta_Y = 1 - \frac{\left|Y^{i+1}(T_B) \cap Y^{i+1}(T_R)\right| - \left|Y^{i}(T_B) \cap Y^{i}(T_R)\right|}{\left|Y^{i+1}(T_B) \cup Y^{i+1}(T_R)\right|}. \tag{48}$$

Here, as in the previous criterion, only the newly added outputs are considered.

The third criterion, called *Overhead*, measures the literal overhead introduced to the block and removed from the remainder by selecting the target prefix,

$$\beta_S = \frac{S^{i+1}(T_B) - S^{i}(T_B)}{S^{i+1}(T_B)} - \frac{S^{i+1}(T_R) - S^{i}(T_R)}{S^{i+1}(T_R)}. \tag{49}$$

This equation can be rewritten as follows:

$$\begin{aligned} \beta_S &= 1 - \frac{S^{i}(T_B)}{S^{i+1}(T_B)} - \left(1 - \frac{S^{i}(T_R)}{S^{i+1}(T_R)}\right) \\ &= \frac{S^{i}(T_R)}{S^{i+1}(T_R)} - \frac{S^{i}(T_B)}{S^{i+1}(T_B)}. \end{aligned} \tag{50}$$

Each of the two fractions is limited to the interval $[0, 1]$, but the total value of β_S is in the interval $[-1, 1]$.

The weighted grade of a candidate prefix is defined as $\beta = b_X\beta_X + b_Y\beta_Y + b_S\beta_S$. The candidate with the highest β is taken and added to the set of prefixes that form the block header.

The complexity of the algorithm can be estimated as follows. Denote by N the number of products in the given D-polynomial. Unlike the fragmentation algorithm that deals with the partitions, the main decomposition algorithm considers only the values of the MOF. Since the main decomposition algorithm separates the N products by using a binary tree, its complexity is of order $\mathcal{O}(N)$. The complexity of the fragmentation algorithm is of order $\mathcal{O}(N^2)$, since its main task is the generation of the set of secondary prefixes.

7. Experimental Results

The efficiency of the proposed approach was tested experimentally by applying the above decomposition algorithm to a number of benchmark functions. The effectiveness of the method was evaluated by comparing the compactness of a monolith MTBDD which corresponds to the given MOF with the compactness of the proposed decomposed network. In the experiments, PLA-like representations of the standard combinatorial-circuit benchmarks (LGSYNTH93) were used.

Table 2: Experimental results in which the decomposed network is simpler than the monolith MTBDD.

Title	$\|X\|$	D%	N_{mon}	N_{net}	Ratio
ALU1	12	18	982	25	0.02
B12	15	29	155	145	0.93
DK48	15	31	3428	58	0.02
DK27	9	34	79	22	0.28
CON1	7	37	16	15	0.94
ALU2	10	39	264	150	0.57
DUKE2	22	40	1435	326	0.23
ALU3	10	42	278	151	0.54
MISEX3C	14	43	10875	705	0.06
WIM	4	50	15	10	0.67
F51M	8	53	255	155	0.61
DK17	10	57	160	55	0.34
APLA	10	64	128	85	0.66
INC	7	79	39	35	0.90

Table 3: Experimental results in which the monolith MTBDD is simpler than the decomposed network.

Title	$\|X\|$	D%	N_{mon}	N_{net}	Ratio
ADD6	12	52	504	731	1.45
RADD	8	57	90	143	1.59
CLIP	9	59	189	376	1.99
Z4	7	61	52	101	1.94
ROOT	8	65	72	134	1.86
SQR6	6	67	63	85	1.35
SQN	7	69	81	116	1.43
MLP4	8	73	240	345	1.44
SAO2	10	73	95	157	1.65
DIST	8	73	125	326	2.61
BW	5	80	25	58	2.32
RD53	5	90	15	53	3.53

The experiments demonstrate that the proposed decomposition, when successful, greatly reduces the size of the decision diagram as compared to the monolith solution.

To analyze the experimental results, we defined a *block density*—a specific parameter of a block. This parameter corresponds to the number of literals in the block's products normalized by the maximal possible number of literals in this block. The success of the decomposition strongly depends on this defined density. Consequently, the effectiveness of the decomposition can be predicted quite reliably by making a preliminary study of the given MOF.

The experimental results are shown in Tables 2 and 3. Table 2 lists the benchmarks for which the decomposition network was simpler than the monolith MTBDD of the MOF. Table 3 shows the opposite cases. The columns in the tables are as follows: $|X|$ is the number of inputs, D is the benchmark's density D, and N_{mon} and N_{net} are the number of nodes in the monolith MTBDD and in the decomposition network. The last column shows the ratio $N_{\text{net}}/N_{\text{mon}}$. Both tables are arranged by ascending density.

The results show that density is a consistent indicator of the success of the decomposition. The successful cases are mostly in the low-density area (density up to 45%), and the unsuccessful ones are mostly in the high-density area (density of at least 60%). The middle functions (density 40–60%) are divided more or less evenly between the successes and the failures. Moreover, there are several examples where the high-density functions are successfully decomposed and no examples where the method failed to work on low-density functions.

The proposed decomposition, on the other hand, relies upon extracting dense blocks from the given MOF and treating the sparse remainders and tails separately. Therefore, a sparse MOF can be easily dealt with by splitting them into a network of component MTBDDs. With dense MOFs, choosing suitable blocks is difficult, and arbitrary choices lead to an ineffective resulting network.

8. Conclusions

Despite extensive research on building the fundamentals of logic design, some of its topics have yet to be examined. One of these topics relates to representation of systems of Boolean functions (multioutput functions) by decision diagrams. Specifically, the conceptual transition from the Boolean function domain to the multi-output functions is considered hard. Although introducing the If-Then-Else (ITE) operator on the Boolean domain makes it possible to construct the decision diagram of a logic function in a very clear way, the analogous procedure for multi-output functions was unknown. This work fills this gap. The main results can be summarized as follows.

(i) A GITE operator was introduced. The GITE operator is a generalization of the ITE operator on the Boolean domain.

(ii) Based on the GITE operator, an algebra of the GITE formula was developed and studied.

(iii) The concept of D-polynomials as a compact analytical representation of the GITE formula was presented. The problem of the compact representation of multi-output functions was then formulated as a problem of decomposition of D-polynomials.

(iv) Finally, a solution to this problem, based on the GITE algebra and its properties, was introduced.

Experimental results obtained on a number of benchmarks are promising. We believe that the present work will initiate future research on the GITE algebra and its possible applications in logic design.

Appendix

Proof of Theorem 20

Theorem 20 states that any D-polynomial can be represented as the composition of its all D-binomials. To prove the theorem we show that the composition of D-binomials $(B_jY_j + B_0^jY_0)$ is equal to the D-polynomial with the same coefficients.

When composing two D-binomials, one of the following cases can occur.

(1) The products B_j are pairwise orthogonal: $B_kY_k \cdot B_jY_j = 0$, for $k \neq j$.

In this case, the orthogonality of the products and the completeness condition yield $B_0^j = \overline{B_j}$. Hence, $B_j \cdot B_0^k = B_j$.

Therefore,

$$\begin{aligned}
&\prod_{j=1}^{m}\left(B_jY_j + B_0^jY_0\right)\\
&\quad = \left(B_1 \cdot B_2\{Y_1 \circ Y_2\} + B_1 \cdot B_0^2\{Y_1 \circ Y_0\}\right.\\
&\qquad \left. + B_2 \cdot B_0^1\{Y_2 \circ Y_0\} + B_0^1 \cdot B_0^2\{Y_0 \circ Y_0\}\right),\\
&\prod_{j=3}^{m}\left(B_jY_j + B_0^jY_0\right)\\
&\quad = \left(B_1Y_1 + B_2Y_2 + B_0^1 \cdot B_0^2Y_0\right)\prod_{j=3}^{m}\left(B_jY_j + B_0^jY_0\right)\\
&\quad = \cdots\\
&\quad = \left(B_1Y_1 + B_2Y_2 + \cdots + B_mY_m + B_0^1 \cdot B_0^2 \cdot \cdots \cdot B_0^mY_0\right)\\
&\quad = \left(B_1Y_1 + B_2Y_2 + \cdots + B_mY_m + B_0Y_0\right)\\
&\quad = \sum_j B_jY_j + B_0Y_0.
\end{aligned} \tag{A.1}$$

(2) Products B_j and B_j are not orthogonal: $B_kY_k \cdot B_jY_j \neq 0$, for $k \neq j$.

Note that the nonorthogonal products are associated with one and the same terminal Y.

Let

$$D = \left(B_jY + B_0^jY_0\right) \circ \left(B_kY + B_0^kY_0\right). \tag{A.2}$$

After composition we have

$$\begin{aligned}
D &= B_j \cdot B_k\{Y_1 \circ Y_1\} + B_j \cdot B_k^0\{Y_1 \circ Y_0\}\\
&\quad + B_0 \cdot B_k\{Y_0 \circ Y_1\} + B_j^0 \cdot B_k^0Y_0\\
&= \left(B_j \cdot B_k + B_j \cdot B_k^0 + B_j^0 \cdot B_k\right)Y_1\\
&\quad + B_j^0 \cdot B_k^0Y_0\\
&= \left(B_j \cdot B_k + B_j \cdot \overline{B}_k + \overline{B}_j \cdot B_k\right)Y_1\\
&\quad + B_j^0 \cdot B_k^0Y_0\\
&= \left(B_j + B_k\right)Y_1 + B_0Y_0.
\end{aligned} \tag{A.3}$$

Acknowledgment

This paper was partially supported by the Israel Science Foundation (Grant no. 1200/12).

References

[1] R. L. Ashenhurst, "The decomposition of switching functions," in *Procceedings of an International Symposium on the Theory of Switching*, pp. 74–116, April 1957.

[2] G. de Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill Higher Education,, 1994.

[3] J. P. Hayes, *Introduction to Digital Logic Design*, Addison-Wesley Longman Publishing, Boston, Mass, USA, 1993.

[4] M. G. Karpovsky, *Finite Orthogonal Series in the Design of Digital Devices*, John Wiley & Sons, New York, NY, USA, 1976.

[5] Z. Kohavi, *Switching and Finite Automata Theory*, McGraw-Hill, 1970.

[6] E. J. McCluskey, *Logic Design Principles*, Prentice-Hall, Englewood Cliffs, NJ, USA, 1986.

[7] R. E. Miller, *Switching Theory*, John Wiley & Sons, New York, NY, USA, 1965.

[8] R. Brayton, "The future of logic synthesis and verification," in *Logic Synthesis and Verification*, pp. 403–4434, Kluwer Academic Publishers, Norwell, Mass, USA, 2002.

[9] M. A. Perkowski, "A survey of literature on function decomposition," Technical Report, GSRP Wright Laboratories, 1995.

[10] R. I. Bahar, E. A. Frohm, C. M. Gaona et al., "Algebraic decision diagrams and their applications," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 188–191, November 1993.

[11] C. M. Files and M. A. Perkowski, "New multivalued functional decomposition algorithms based on MDDs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 9, pp. 1081–1086, 2000.

[12] Y. Iguchi, T. Sasao, and M. Matasuura, "Evaluation of multiple-output logic functions using decision diagrams," in *Proceedings of the Asia and South Pacific Design Automation Conference*, 2003.

[13] T. Sasao, Y. Iguchi, and M. Matsuura, "Comparison of decision diagrams for multiple-output functions," in *Proceedings of the International Workshop on Logic and Synthesis*, 2002.

[14] S. N. Yanushkevich, D. M. Miller, V. P. Shmerko, and R. S. Stankovic, *Decision Diagram Techniques For Micro- and Nanoelectronic Design Handbook*, CRC Press, 2005.

[15] T. Sasao, *Memory-Based Logic Synthesis*, Springer, 2011.

[16] T. Sasao and M. Matsuura, "A method to decompose multiple-output logic functions," in *Proceedings of the 41st Design Automation Conference*, pp. 428–433, San Diego, Calif, USA, June 2004.

[17] R. E. Bryant, "Symbolic boolean manipulation with ordered binary decision diagrams," *ACM Computing Surveys*, vol. 24, no. 3, pp. 293–318, 1992.

[18] P. G. Hinman, *Fundamentals of Mathematical Logic*, A K Peters, 2005.

[19] G. D. Hachtel and F. Somenzi, *Logic Synthesis and Verification Algorithms*, Kluwer Academic Publisher, 2005.

[20] S. Hassoun and T. Sasao, Eds., *Logic Synthesis and Verification*, vol. 654 of *The Springer International Series in Engineering and Computer Science*, 2002.

[21] S. Nagayama and T. Sasao, "Compact representations of logic functions using heterogeneous MDDs," in *Proceedings of the 33rd International Symposium on Multiple-Valued Logic*, pp. 247–252, May 2003.

[22] T. Sasao, *Switching Theory For Logic Synthesis*, Kluwer Academic Publishers, 1999.

[23] T. Sasao and M. Fujita, *Representations of Discrete Functions*, Kluwer Academic Publishers, 1996.

[24] C. Baier and E. Clarke, "The algebraic Mu-calculus and MTBDDs," in *Proceedings of the 5th Workshop on Logic, Language, Information and Computation (WoLLIC '98)*, pp. 27–38, 1998.

[25] B. Chen and C. L. Lee, "Complement-based fast algorithm to generate universal test sets for multi-output functions," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 3, pp. 370–377, 1994.

[26] R. Drechsler, J. Shi, and G. Fey, "Synthesis of fully testable circuits from BDDs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 3, pp. 440–443, 2004.

[27] G. Fey and R. Drechsler, "Minimizing the number of paths in BDDs: theory and algorithm," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 1, pp. 4–11, 2006.

[28] W. N. N. Hung, X. Song, G. Yang, J. Yang, and M. Perkowski, "Optimal synthesis of multiple output Boolean functions using a set of quantum gates by symbolic reachability analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 9, pp. 1652–1663, 2006.

[29] M. G. Karpovsky, R. S. Stankovic, and J. T. Astola, "Reduction of sizes of decision diagrams by autocorrelation functions," *IEEE Transactions on Computers*, vol. 52, no. 5, pp. 592–606, 2003.

[30] O. Keren, "Reduction of the average path length in binary decision diagrams by spectral methods," *IEEE Transactions on Computers*, vol. 57, no. 4, pp. 520–531, 2008.

[31] O. Keren I. Levin and R. S. Stankovic, "Minimization of the number of paths in binary decision diagrams by using autocorrelation coefficients," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 1, pp. 31–44, 2011.

[32] C. Meinel, F. Somenzi, and T. Theobald, "Linear sifting of decision diagrams and its application synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 5, pp. 521–533, 2000.

[33] C. Yang and M. Ciesielski, "BDS: A BDD-based logic optimization system," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 7, pp. 866–876, 2002.

[34] S. Baranov, *Logic and System Design of Digital Systems*, TUT Press, 2008.

[35] I. Levin, O. Keren, V. Ostrovsky, and G. Kolotov, "Concurrent decomposition of multi-terminal BDDs," pp. 165–169, Proceedings of the 7th International Workshop on Boolean Problems, Freiberg, Germany, September 2006.

[36] I. Levin and O. Keren, "Split multi-terminal binary decision diagrams," in *Proceedings of the 8th International Workshop on Boolean Problems*, pp. 161–167, 2008.

[37] I. Levin and O. Keren, "Generalized if-then-else operator for compact polynomial representation of multi output functions," in *Proceedings of the 14th Euromicro Conference on Digital System Design (DSD '11)*, pp. 15–20, 2011.