Decomposition of Boolean Functions Specified by Cubes*

J. A. BRZOZOWSKI
Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1
email: brzozo@uwaterloo.ca

T. ŁUBA
Institute of Telecommunications
Warsaw University of Technology
Nowowiejska 15/19
00-665 Warsaw, Poland
email: luba@tele.pw.edu.pl

ftp://cs-archive.uwaterloo.ca/cs-archive/CS-97-01/CS-97-01.ps.Z

Abstract We study the problem of decomposing a Boolean function into a set of functions with fewer arguments. This problem has considerable practical importance in VLSI. For example, for digital circuits designed with field-programmable gate arrays, it is necessary to express Boolean functions in terms of 'smaller' functions that fit the cells of the array. The decomposition problem is old, and well understood when the function to be decomposed is specified by a truth table listing the function's minterms, or has one output only. However, modern design tools, such as Berkeley's Espresso, handle functions with many outputs and represent them by Boolean cubes rather than minterms, for reasons of efficiency. In this paper we develop a decomposition theory for multiple-output, partially specified Boolean functions represented by cubes. The theory uses ternary algebra and generalized set systems.
Keywords: blanket, Boolean function, cube, decomposition, disjunctive, "don't care," multiple-output, set system, ternary algebra, type *fr*

*This research was supported by the Natural Sciences and Engineering Research Council of Canada under grant No. OGP0000871. This work was done while T. Łuba was a Visiting Professor at the University of Waterloo.

1 Introduction

Decomposition is a fundamental problem in modern logic synthesis. Its goal is to break a logic circuit into a set of smaller interacting components. Such an implementation is desirable for a number of reasons. In the case of designs using field-programmable gate arrays (FPGAs), particularly those with look-up table structures [4], decomposition is a necessity, since FPGA cells can only accomodate functions with very few inputs and outputs. In PLA- and PLD-oriented designs, a decomposed circuit often leads to a smaller silicon area and shorter signal delays [6]. Consequently, decomposition methods are increasingly exploited in today's logic synthesis systems [17].

Mathematically, decomposition is the process of expressing a function of n variables as a function of functions of fewer variables [7]. For example, a function $F(X)$ is decomposable if it can be expressed as $F = H(U, G(V))$, where U and V are proper subsets of the set X of input variables, and G and H have fewer input variables than F.

Numerous decomposition algorithms have been developed. Ashenhurst, in his fundamental paper [1], stated the disjunctive decomposition theorem based on decomposition charts. Curtis extended Ashenhurst's results to multiple decompositions [7], of the form $F = H(U, G_1(V), \ldots, G_k(V))$. The use of charts for decomposition of logic networks is applicable only to restricted classes of functions. To remedy this, Roth and Karp used a more compact representation of a function in the form of a cover of the on-set and a cover of the off-set [15]. However, their method does not deal directly with multiple-output functions.

In the early 1980's, functional decomposition methods received less attention because of the rapid development of synthesis techniques for multi-level logic. Algebraic division of sum-of-product expressions represented by sets of cubes has been a basic operation in the procedures of so-called kernel extraction widely used in multi-level synthesis of Boolean functions [3].

A renewed interest in functional decomposition in recent years was caused by the introduction of field programmable gate arrays by Xilinx in 1986, and other companies (AT&T, Actel, Altera) in succeeding years. FPGAs are modern logic devices which can be programmed by the users to implement any logic circuit [4]. Because of their short turnaround time compared with that of the standard ASIC process, they have recently become very popular in rapid system prototyping. Many FPGA architectures have been proposed; the architecture based on look-up tables is one of the most popular ones. It consists of many identical logic blocks, each of which can implement any Boolean function of k inputs, where k is a small integer. For example, in Xilinx XC3000 architecture, k is 5.

When they found that the earlier algebraic methods could not be easily adapted to the look-up-table model, researchers switched to decomposition methods based on functional dependencies [11, 12, 14, 17, 19, 20, 21, 23]. However, despite the fundamental nature of the functional decomposition problem and its many applications, there does not exist any uniform approach to decomposition which could be applied to completely or incompletely specified multiple-output Boolean functions represented compactly by covers using Boolean cubes.

In this paper we extend previous results concerning functional decomposition to the case where the given function is specified by cubes in the so-called *fr* type description [2]. We develop a new framework, based on ternary algebra, and on certain generalizations of set systems, which we call "blankets." The remainder of the paper is structured as follows. Section 2 presents some notions from ternary algebra. In Section 3 we discuss two generalizations of partitions: blankets and set systems. Multiple-output Boolean functions and their representations are presented in Section 4. The use of blankets for Boolean functions is described in Section 5. The notion of "disjunctive serial separation"—a functional decomposition without any constraints on the size of the components—is defined in Section 6, where we also prove a generalized version of a theorem of Curtis [7]. Section 7 extends our methods to nondisjunctive decompositions. Section 8, discusses decompositions with constraints on the number of arguments of the components. The use of cubes in the decomposition process is illustrated in Section 9. The overall decomposition algorithm is next described in Section 10, and Section 11 concludes the paper.

2 Notions from Ternary Algebra

In this section we present for later use some concepts and notation from ternary algebra. For more details see [5].

We use 0 and 1 to denote the usual logic values, and Φ to denote a third value, which will have several interpretations. The *uncertainty partial order* $\sqsubseteq$ on the set $\{0, \Phi, 1\}$ is defined as follows:

$$0 \sqsubseteq 0,\ \Phi \sqsubseteq \Phi,\ 1 \sqsubseteq 1,\ 0 \sqsubseteq \Phi,\ 1 \sqsubseteq \Phi,$$

and no other pairs are related by $\sqsubseteq$. The value Φ is considered *uncertain*, whereas 0 and 1 are *certain*. The converse of $\sqsubseteq$ is denoted by $\sqsupseteq$; the notation $t \sqsubseteq t'$ is used interchangeably with $t' \sqsupseteq t$.

In general, n-tuples will be denoted by unsubscripted letters, for example t, and their components by subscripted letters, for example t_i. Also, to simplify the notation we often write tuples without parentheses and commas. For example, $(\Phi, 1, \Phi, 0, 1)$ will be written $\Phi 1 \Phi 01$.

The partial order $\sqsubseteq$ is extended to $\{0, \Phi, 1\}^n$:

$$t \sqsubseteq t' \text{ if and only if } t_i \sqsubseteq t'_i \text{ for all } i,\ \ 1 \leq i \leq n,$$

where $t = (t_1, \ldots, t_n)$ and $t' = (t'_1, \ldots, t'_n)$. For example, $01\Phi \sqsubseteq \Phi 1 \Phi$.

In the partially ordered set $\langle \{0, \Phi, 1\}, \sqsubseteq \rangle$, the concept of least upper bound *lub* is defined as usual. Thus, $lub\{0\} = 0$, $lub\{1\} = 1$, and the *lub* of every other nonempty subset of $\{0, \Phi, 1\}$ is equal to Φ. The definition of *lub* is also extended to $\{0, \Phi, 1\}^n$ to be the component-by-component least upper bound. For example,

$$lub\{\Phi 001, 1101, 0101\} = \Phi\Phi 01.$$

As usual, we define a binary operation, $\sqcup$-*addition*, on $\{0, \Phi, 1\}$ to correspond to the least upper bound, as follows:

$$t \sqcup t' = lub\{t, t'\}.$$

Then we have

$$t \sqsubseteq t' \text{ if and only if } t \sqcup t' = t'.$$

The $\sqcup$-addition is idempotent, commutative and associative. For any nonempty subset S of $\{0, \Phi, 1\}^n$, we use the notation

$$lub\, S = \bigsqcup_{t \in S} t.$$

We define the following *compatibility relation* $\sim$ on $\{0, \Phi, 1\}$:

$$0 \sim 0,\ \Phi \sim \Phi,\ 1 \sim 1,\ 0 \sim \Phi,\ 1 \sim \Phi,\ \Phi \sim 0,\ \Phi \sim 1,$$

but the pairs $(0, 1)$ and $(1, 0)$ are not related by $\sim$. Compatibility is reflexive and symmetric, that is, for all $t, t' \in \{0, \Phi, 1\}$, $t \sim t$, and $t \sim t'$ implies $t' \sim t$. However, it is not transitive, since $0 \sim \Phi$ and $\Phi \sim 1$, but $0 \not\sim 1$. Note that

$$t \sqsubseteq t' \text{ implies } t \sim t',$$

and

$$t \sim t' \text{ implies that either } t \sqsubseteq t' \text{ or } t' \sqsubseteq t.$$

The compatibility relation $\sim$ is extended to $\{0, \Phi, 1\}^n$:

$$t \sim t' \text{ if and only if } t_i \sim t'_i \text{ for all } i, \ \ 1 \leq i \leq n.$$

For example, $01\Phi\Phi \sim \Phi10\Phi$.

The greatest lower bound, *glb*, exists only for some subsets of $\{0, \Phi, 1\}$. Thus $glb\{0, 1\}$ does not exist, but $glb\{0, \Phi\} = 0$. The binary operation, $\sqcap$-*product*, corresponding to *glb* is given in Table 1; $t \sqcap t'$ is defined only if t and t' are compatible. When we $\sqcap$-multiply a *certain* value by an *uncertain* value, the *certain* value prevails. Also,

Table 1: The operation $\sqcap$.

	$t \sqcap t'$	t': 0	Φ	1
	0	0	0	–
t	Φ	0	Φ	1
	1	–	1	1

$$t \sqsubseteq t' \text{ if and only if } t \sqcap t' = t.$$

If t, t', and t'' are pairwise compatible, then the $\sqcap$-product $(t \sqcap t') \sqcap t''$ is defined and equal to $t \sqcap (t' \sqcap t'')$. Thus $\sqcap$ is associative. The $\sqcap$-product can be extended to any nonempty set C of pairwise compatible elements:

$$glb\, C = \textstyle\bigsqcap_{t \in C} t.$$

The $\sqcap$-product can be further extended to compatible tuples. For example $01\Phi\Phi \sqcap \Phi10\Phi = 010\Phi$.

With each $t \in \{0, \Phi, 1\}^n$ we associate a set $bin\,t$ of binary tuples:

$$bin\,t = \{b \in \{0, 1\}^n \mid b \sqsubseteq t\}.$$

For example, if $t = 01\Phi\Phi$, then $bin\,t = \{0100, 0101, 0110, 0111\}$. If b is in $\{0, 1\}^n$ and $b \sqsubseteq t$ for some tuple $t \in \{0, \Phi, 1\}^n$, then $b \in bin\,t$ and we say that b *belongs* to t, or that t *contains* b. One verifies that

$$t = lub\, bin\,t.$$

Proposition 1 *Let $C \subseteq \{0, \Phi, 1\}^n$. The following two conditions are equivalent:*

- $\bigsqcap_{t \in C} t$ *is defined, that is, the elements of C are pairwise compatible.*
- $\bigcap_{t \in C} bint$ *is nonempty, that is, there exists $b \in \{0, 1\}^n$ such that $b \sqsubseteq t$ for each $t \in C$.*

If the conditions of the proposition are satisfied, then

$$\bigsqcap_{t \in C} t = \bigsqcup\left(\bigcap_{t \in C} bint\right).$$

For example, let $C = \{0\Phi1\Phi, \Phi11\Phi, 01\Phi\Phi\}$; then $\bigsqcap_{t \in C} t = 011\Phi$,

$$\bigcap_{t \in C} bint = \{0010, 0011, 0110, 0111\} \cap \{0110, 0111, 1110, 1111\} \cap$$

$$\{0100, 0101, 0110, 0111\} = \{0110, 0111\},$$

and $0110 \sqcup 0111 = 011\Phi$.

3 Blankets and Set Systems

A *blanket* on a set S is a collection[1] $\beta = \{B_1, \ldots, B_k\}$ of nonempty and distinct subsets of S, called *blocks*, whose union is S. For example, if $S = \{1, 2, 3\}$, then $\beta = \{\{1, 2\}, \{2, 3\}\}$ and $\beta' = \{\{1\}, \{1, 2\}, \{2, 3\}\}$ are blankets on S. To improve the notation, we often write $\beta = \{\overline{1, 2}; \overline{2, 3}\}$, instead of $\beta = \{\{1, 2\}, \{2, 3\}\}$, etc. Also, when it is possible to avoid reference to the number k of blocks in a blanket $\beta = \{B_1, \ldots, B_k\}$, we simply write $\beta = \{B_i\}$.

Define the "nonempty" operator ne as follows. For any set $\{S_i\}$ of subsets of a set S,

$$ne\{S_i\} = \{S_i\} - \{\emptyset\},$$

is the set $\{S_i\}$ with the empty subset removed, if it was originally present. The *product* $\beta * \beta'$ of two blankets is defined as follows:

$$\beta * \beta' = ne\{B_i \cap B_j \mid B_i \in \beta \text{ and } B_j \in \beta'\}.$$

[1]We call such a collection a blanket, because it covers S.

For example,

$$\{\overline{1,2,3};\ \overline{3,4,5}\}*\{\overline{1,3,4};\ \overline{1,5};\ \overline{2,3,4}\} = \{\overline{1,3};\ \overline{1};\ \overline{2,3};\ \overline{3,4};\ \overline{5}\},$$

and

$$\{\overline{1,2,3};\ \overline{3,4,5}\}*\{\overline{1,2,3};\ \overline{3,4,5}\} = \{\overline{1,2,3};\ \overline{3};\ \overline{3,4,5}\}.$$

The last example shows that blanket product is not idempotent.

Let $\mathcal{B}_S$ be the set of all blankets on a set S. Let $1_S = \{S\}$ be the one-block blanket, and let $0_S = \{\{s_i\} \mid s_i \in S\}$ be the blanket consisting of one-element blocks. Then the algebraic system $\langle \mathcal{B}_S, *, 1_S, 0_S\rangle$ is a commutative monoid with unit 1_S and zero 0_S, that is, for all β, β', β'', we have $\beta*\beta' = \beta'*\beta$, $(\beta*\beta')*\beta'' = \beta*(\beta'*\beta'')$, $\beta*1_S = 1_S*\beta = \beta$, and $\beta*0_S = 0_S*\beta = 0_S$.

If β and β' are blankets on S, we write $\beta \leq \beta'$ if and only if for each B_i in β there exists a B_j in β' such that $B_i \subseteq B_j$. For example, $\{\overline{1};\ \overline{2};\ \overline{3}\} \leq \{\overline{1,2};\ \overline{2,3}\}$, and $\{\overline{1,2};\ \overline{2,3}\} \leq \{\overline{1,2};\ \overline{2,3};\ \overline{1,3}\}$.

The relation $\leq$ is reflexive and transitive, that is, for all β, β', β'', we have $\beta \leq \beta$, and $\beta \leq \beta'$ and $\beta' \leq \beta''$ implies $\beta \leq \beta''$. The relation is not antisymmetric, since $\{\overline{1};\ \overline{1,2}\} \leq \{\overline{1,2}\}$ and $\{\overline{1,2}\} \leq \{\overline{1};\ \overline{1,2}\}$. Also, we have the following property:

$$\beta*\beta' \leq \beta \text{ and } \beta*\beta' \leq \beta'.$$

Next, we consider some special blankets called set systems [8]. A *set system* on a set S is a blanket $\sigma = \{B_1, \ldots, B_k\}$ in which the blocks are maximal in the sense that $B_i \subseteq B_j$ implies $i = j$. For example, if $S = \{1,2,3\}$, then $\sigma = \{\{1,2\},\{2,3\}\}$ is a set system on S. As above, we also write $\sigma = \{\overline{1,2}; \overline{2,3}\}$. Note that set systems are not closed under the blanket product. For example $\{\overline{1,2};\ \overline{2,3}\}*\{\overline{1,2};\ \overline{2,3}\}$ is not a set system.

If $\beta = \{B_i\}$ is any blanket on S, then

$$max\,\beta = \{B \subseteq S \mid B = B_i \text{ for some } i, \text{ and } B \subseteq B_j \text{ implies } B_j = B\}.$$

Note that *max* maps blankets to set systems. The *product* $\sigma \circ \sigma'$ of two set systems [8] is defined as follows:

$$\sigma \circ \sigma' = max(\sigma*\sigma').$$

For example,

$$\{\overline{1,2,3};\ \overline{3,4,5}\} \circ \{\overline{1,3,4};\ \overline{1,5};\ \overline{2,3,4}\} = \{\overline{1,3};\ \overline{2,3};\ \overline{3,4};\ \overline{5}\}.$$

One verifies that set system product is idempotent, commutative, and associative.

A *partition* on a set S is a set system $\pi = \{B_i\}$, where the blocks are disjoint, that is, $B_i \cap B_j = \emptyset$, if $i \neq j$. For example, if $S = \{1, 2, 3\}$, then $\pi = \{\{1, 2\}, \{3\}\}$ is a partition on S.

In the sequel, we will be using blankets on rows of ternary matrices. Let $\mathcal{M}$ be any $h \times k$ matrix with entries from $\{0, \Phi, 1\}$. Assume that the rows of $\mathcal{M}$ are numbered $1, \ldots, h$. Let $\mathcal{T}$ be the set of rows of $\mathcal{M}$; then we denote blankets on $\mathcal{T}$ by blankets on the set $\{1, \ldots, h\}$. Borrowing the terminology from logic design, we refer to ternary k-tuples as *cubes*.

In particular, we define a *row blanket* $\beta = ne\{\mathcal{T}_{\sqsupseteq b}\}$, where $b \in \{0, 1\}^k$ and

$$\mathcal{T}_{\sqsupseteq b} = \{t \in \mathcal{T} \mid t \sqsupseteq b\}.$$

Thus $\mathcal{T}_{\sqsupseteq b}$ is the set of all cubes in $\mathcal{T}$ that "cover" b, in the sense of the partial order $\sqsupseteq$. For example, consider the matrix $\mathcal{M}$ in Table 2. One verifies that $\mathcal{T}_{\sqsupseteq 000} = \{3, 5, 6\}$, $\mathcal{T}_{\sqsupseteq 001} = \{5, 6\}$, $\mathcal{T}_{\sqsupseteq 010} = \{3, 5\}$, $\mathcal{T}_{\sqsupseteq 011} = \{5\}$, $\mathcal{T}_{\sqsupseteq 100} = \{1, 3, 4, 6\}$, $\mathcal{T}_{\sqsupseteq 101} = \{6\}$, $\mathcal{T}_{\sqsupseteq 110} = \{1, 2, 3, 4\}$, $\mathcal{T}_{\sqsupseteq 111} = \emptyset$. The blanket β is therefore

$$\beta = \{\overset{B_1}{\overline{3, 5, 6}};\ \overset{B_2}{\overline{5, 6}};\ \overset{B_3}{\overline{3, 5}};\ \overset{B_4}{\overline{5}};\ \overset{B_5}{\overline{1, 3, 4, 6}};\ \overset{B_6}{\overline{6}};\ \overset{B_7}{\overline{1, 2, 3, 4}}\},$$

where the blocks have been numbered as shown.

Table 2: Matrix $\mathcal{M}$.

1	1	Φ	0
2	1	1	0
3	Φ	Φ	0
4	1	Φ	0
5	0	Φ	Φ
6	Φ	0	Φ

The *representative* r_i of block B_i of a row blanket β is the *glb* of the cubes in B_i. Thus, in our example, $r_1 = \Phi\Phi 0 \sqcap 0\Phi\Phi \sqcap \Phi 0\Phi = 000$, $r_2 = 00\Phi$, $r_3 = 0\Phi 0$, $r_4 = 0\Phi\Phi$, $r_5 = 100$, $r_6 = \Phi 0\Phi$, and $r_7 = 110$. Let $\mathcal{R}$ be the set of block representatives of β; in our example, $\mathcal{R} = \{r_1, \ldots, r_7\}$. One verifies that the mapping $\mu : \beta \rightarrow \mathcal{R}$ is a one-to-one correspondence. In fact, we have the following result:

Proposition 2 *Let β be the row blanket of a matrix $\mathcal{M}$. Let $\langle \beta, \supseteq \rangle$ be the set of blocks of β partially ordered by inclusion, and let $\langle \mathcal{R}, \sqsubseteq \rangle$ be the set of representatives partially ordered by the relation $\sqsubseteq$. Then the two partially ordered sets are isomorphic.*

It is a consequence of the proposition that $r_i \sqsubseteq r_j$ if and only if $B_i \supseteq B_j$. Two representatives r_i and r_j have a *glb* if and only if $B_i \cup B_j$ is a block of β. In fact, $r_i \sqcap r_j = r_k$, if and only if $B_i \cup B_j = B_k$. For example, $r_1 \sqcap r_2 = r_1$ and $B_1 \cup B_2 = B_1$, $r_2 \sqcap r_3 = r_1$ and $B_2 \cup B_3 = B_1$.

4 Multiple-Output Boolean Functions

We will be dealing with functions of the form

$$F : D \rightarrow (\{0, \Phi, 1\}^m - \{(\Phi, \ldots, \Phi)\}),$$

where $D \subseteq \{0,1\}^n$, and $(\Phi, \ldots, \Phi)$ is the tuple of n Φs. Such a function will be used to specify a combinational logic circuit with n inputs, $x_1, \ldots, x_n$, and m outputs, $y_1, \ldots, y_m$. Since "don't care" conditions frequently arise in logic design, the function outputs may be undefined for some input n-tuples. Hence the domain D of F is, in general, a subset of $\{0,1\}^n$. Also, for some input n-tuples, some of the outputs may be undefined, although at least one of the outputs has to be defined. Such undefined output values will be represented by Φs. For example, the (partial) truth table for a 4-input, 2-output function F is shown in Table 3. We will use this function as our running example. For two of the input 4-tuples, 0100 and 1100, no outputs are defined, so these rows are omitted from the table.

An element of $\{0,1\}^n$ is called a *minterm*. As usual, a minterm may be interpreted as a zero-dimensional Boolean cube, or as a product of variables $x_1, \ldots, x_n$ and their complements. For example, if $n = 4$, the 4-tuple 0111 corresponds to the product $\overline{x_1} * x_2 * x_3 * x_4$, where $\overline{x}$ is the complement of x, and $*$ denotes Boolean multiplication. We usually denote Boolean multiplication by juxtaposition. For example, we write $\overline{x_1}\, x_2\, x_3\, x_4$. An element c of $\{0, \Phi, 1\}^n$ is called an *input cube*; if c has p components that are Φ (and hence $n - p$ components that are either 0 or 1), then it is a p-dimensional

Table 3: Truth table of F.

x_1	x_2	x_3	x_4	y_1	y_2
0	0	0	0	1	1
0	0	0	1	1	Φ
0	0	1	0	1	1
0	0	1	1	0	Φ
0	1	0	1	1	1
0	1	1	0	0	0
0	1	1	1	0	1
1	0	0	0	1	0
1	0	0	1	1	Φ
1	0	1	0	1	0
1	0	1	1	0	Φ
1	1	0	1	Φ	1
1	1	1	0	0	0
1	1	1	1	0	1

input cube. Such a cube may be interpreted as a product of $n-p$ variables in the usual way. For example, $0\Phi10$ is a one-dimensional input cube corresponding to the product $\overline{x_1}\,x_3\,\overline{x_4}$.

It is clear that the partial order $\sqsubseteq$ represents the usual containment relation between cubes. For example, we have $\Phi010 \sqsubseteq \Phi0\Phi0$, which corresponds to the fact that cube $\overline{x_2}\,x_3\,\overline{x_4}$ is contained in cube $\overline{x_2}\,\overline{x_4}$. If c is a cube, then $bin\, c$ is the set of minterms contained in that cube. The compatibility relation applied to cubes has the following meaning. Two cubes c and c' are compatible if and only if they have a nonempty common subcube. For example, $01\Phi\Phi \sim \Phi10\Phi$ means that cubes $\overline{x_1}\,x_2$ and $x_2\,\overline{x_3}$ have a nonempty common subcube. In fact, this subcube, $\overline{x_1}\,x_2\,\overline{x_3}$, is denoted by the tuple $01\Phi\Phi \sqcap \Phi10\Phi = 010\Phi$.

We now introduce our notation for incompletely specified multiple-output Boolean functions. A function F, defined as above, with n inputs and m outputs will be specified as a set $\mathcal{F}$ of $(n+m)$-tuples, called *function cubes*, in which the first n components correspond to the inputs and the last m components, to the outputs. We usually represent $\mathcal{F}$ as a matrix, as illustrated below.

Example 1 The seven 6-tuples numbered 1 through 7 in Table 4 define a 4-input, 2-output function. In particular, the table specifies that $y_1 = 1$ for

Table 4: An *fr* function.

Number	x_1	x_2	x_3	x_4	y_1	y_2
1	0	0	Φ	0	1	1
2	1	0	Φ	0	1	0
3	Φ	0	0	Φ	1	Φ
4	Φ	Φ	1	1	0	Φ
5	Φ	1	1	0	0	0
6	Φ	1	Φ	1	Φ	1
7	0	Φ	0	1	1	Φ

the input cubes $00\Phi0$, $10\Phi0$, $\Phi00\Phi$, and $0\Phi01$. Thus the set of minterms for which y_1 must be 1, which is called the *on-set* of y_1, can be described by the Boolean expression

$$y_1^{on} = \overline{x_1}\,\overline{x_2}\,\overline{x_4} + x_1\,\overline{x_2}\,\overline{x_4} + \overline{x_2}\,\overline{x_3} + \overline{x_1}\,\overline{x_3}\,x_4.$$

Similarly, the *off-set* of y_1, the set of minterms for which y_1 is 0, is described by the expression

$$y_1^{off} = x_3\,x_4 + x_2\,x_3\,\overline{x_4}.$$

One verifies that these two expressions are never 1 at the same time; hence, the specification is proper. Finally, all minterms that are not described by y_1^{on} or y_1^{off} belong to the "don't care" set y_1^{dc}. Similarly, we obtain the following expressions for y_2:

$$y_2^{on} = \overline{x_1}\,\overline{x_2}\,\overline{x_4} + x_2\,x_4,$$

and

$$y_2^{off} = x_1\,\overline{x_2}\,\overline{x_4} + x_2\,x_3\,\overline{x_4}.$$

$\square$

For an $(n+m)$-tuple $t = (t_1, \ldots, t_{n+m}) \in \{0, \Phi, 1\}^{n+m}$, the *input projection* of t is $t_\downarrow = (t_1, \ldots, t_n)$, and the *output projection* is $t^\uparrow = (t_{n+1}, \ldots, t_{n+m})$.

A set of $(n+m)$-tuples does not necessarily define a Boolean function, because it is possible to assign conflicting output values. An *fr function* [2], is any set $\mathcal{F}$ of $(n+m)$-tuples from $\{0, \Phi, 1\}^{n+m}$ that satisfies the following *consistency condition*, which guarantees that there are no contradictions. For all $t, t' \in \mathcal{F}$,

$$\text{if } t_\downarrow \sim t'_\downarrow, \text{ then } t^\uparrow \sim t'^\uparrow. \tag{1}$$

Furthermore, $\mathcal{F}$ does not contain any $(n+m)$-tuple t such that $t^\uparrow$ consists of Φs alone.

We say that a minterm b is *involved* in a function cube t, if $b \sqsubseteq t_\downarrow$. We call a minterm b *relevant to* $\mathcal{F}$ if b is involved in some function cube $t \in \mathcal{F}$. The consistency condition on $\mathcal{F}$ implies that, if a minterm b is involved in two cubes t and t' of $\mathcal{F}$, then, for every output y_i, the values t_{n+i} and t'_{n+i} specified for y_i by the two cubes are either equal or one of them is Φ. It follows that, for all the cubes of $\mathcal{F}$ involving b, the $(n+i)$th entries that are not Φ must be identical.

From any *fr* function $\mathcal{F}$, we can construct its truth table in the following manner. If b is not relevant to $\mathcal{F}$, then b is absent from the table. Otherwise, let

$$\mathcal{F}_{X \sqsupseteq b} = \{t \in \mathcal{F} \mid t_\downarrow \sqsupseteq b\},$$

be the set of all cubes involving b. By the consistency condition, all these cubes are pairwise compatible, and their *glb* exists. The value of the function F for minterm b is

$$F(b) = \textstyle\bigsqcap_{t \in \mathcal{F}_{X \sqsupseteq b}} t^\uparrow.$$

To illustrate this, we consider the *fr* function of Table 4. Minterm 0000 is involved in cubes 1 and 3; thus $\mathcal{F}_{X \sqsupseteq 0000} = \{1, 3\}$. The output projections of these two cubes are 11 and 1Φ. Thus $F(0000) = 11 \sqcap 1\Phi = 11$. Minterm 0111 is involved in cubes 4 and 6. The values assigned to $y_1 y_2$ for these two minterms are 0Φ and $\Phi 1$. Hence, in the truth table, the outputs are $0\Phi \sqcap \Phi 1 = 01$. Also, $F(1001) = 1\Phi$, etc. One verifies that the completed truth table is identical to that in Table 3.

In this paper we study the decomposition of *fr* functions. A specification in the form of Table 4 can be viewed as an abbreviated version of the truth table of Table 3. Such specifications are widely used; for example, the function representations of type *fr* used with the program Espresso [2] satisfy our definition.

5 Blankets for Boolean Functions

We will use blankets to characterize decompositions of *fr* functions. For each input x_i in the table of $\mathcal{F}$, we define a two-block blanket $\beta_i = \{\mathcal{F}_{x_i \sqsupseteq 0}, \mathcal{F}_{x_i \sqsupseteq 1}\}$ on $\mathcal{F}$ as follows:

$$\mathcal{F}_{x_i \sqsupseteq 0} = \{t \in \mathcal{F} \mid t_i \sqsupseteq 0\}$$

and

$$\mathcal{F}_{x_i \sqsupseteq 1} = \{t \in \mathcal{F} \mid t_i \sqsupseteq 1\}.$$

For example, for Table 4 we find

$$\beta_1 = \{\overline{1,3,4,5,6,7};\ \overline{2,3,4,5,6}\},$$

$$\beta_2 = \{\overline{1,2,3,4,7};\ \overline{4,5,6,7}\},$$

$$\beta_3 = \{\overline{1,2,3,6,7};\ \overline{1,2,4,5,6}\},$$

$$\beta_4 = \{\overline{1,2,3,5};\ \overline{3,4,6,7}\}.$$

For any subset V of the set $X = \{x_1, \ldots, x_n\}$ of input variables we define the *(input) blanket of* V

$$\beta_V = \prod_{x_i \in V} \beta_i,$$

where Π is the blanket product $*$. Note that we permit V to be empty, in which case $\beta_V = \beta_\emptyset = \{\mathcal{F}\}$ is the one-block blanket on $\mathcal{F}$.

Suppose $V = \{x_{i_1}, \ldots, x_{i_r}\}$ is a subset of the set X of input variables, and t is a function cube; denote by $t^V_\downarrow$ the r-tuple $(t_{i_1}, \ldots, t_{i_r})$.

Proposition 3 *Blanket β_V can be calculated as follows: $\beta_V = ne\{\mathcal{F}_{V \sqsupseteq d}\}$, where $d \in \{0,1\}^r$, and*

$$\mathcal{F}_{V \sqsupseteq d} = \{t \in \mathcal{F} \mid t^V_\downarrow \sqsupseteq d\}. \tag{2}$$

For example, for the function of Table 4, and $V = \{x_2, x_3, x_4\}$, we have

$$\beta_V = \{\overset{000}{\overline{1,2,3}};\ \overset{001}{\overline{3,7}};\ \overset{010}{\overline{1,2}};\ \overset{011}{\overline{4}};\ \overset{101}{\overline{6,7}};\ \overset{110}{\overline{5}};\ \overset{111}{\overline{4,6}}\},$$

where we have indicated the values of the variables x_2, x_3, x_4 corresponding to each block.

If cubes t and t' both appear in some block of a blanket β, then we write $t\beta t'$. From Proposition 3 it follows that,

$$t\beta_V t' \text{ implies } t^V_\downarrow \sim t'^V_\downarrow.$$

We also use blankets for output variables. We define a two-block blanket $\hat{\beta}_i$ for each output y_i. In the example, $\hat{\beta}_1 = \{\overline{4,5,6};\ \overline{1,2,3,6,7}\}$, and $\hat{\beta}_2 = \{\overline{2,3,4,5,7};\ \overline{1,3,4,6,7}\}$. We will also need a blanket β_F, called the *output blanket* corresponding to the product of all the blankets of output variables. Thus

$$\beta_F = \prod_{i=1}^{m} \hat{\beta}_i.$$

Proposition 4 *The output blanket can be found as follows: $\beta_F = ne\{\mathcal{F}_{Y \sqsupseteq e}\}$, where $e \in \{0,1\}^m$ and*

$$\mathcal{F}_{Y \sqsupseteq e} = \{t \in \mathcal{F} \mid t^{\uparrow} \sqsupseteq e\}. \tag{3}$$

For example, for the function of Table 4,

$$\beta_F = \{\overset{00}{\overline{4,5}};\ \overset{01}{\overline{4,6}};\ \overset{10}{\overline{2,3,7}};\ \overset{11}{\overline{1,3,6,7}}\}.$$

6 Disjunctive Serial Separations

The following is a generalization of the work of [7] and a formalization of the work in [12].

We use the following notation. Let $X = \{x_1, \ldots, x_n\}$ be the set of input variables of an *fr* function $F(x_1, \ldots, x_n)$. Let U and V be two disjoint subsets of X such that $U \cup V = X$. For convenience, and without loss of generality, we assume that the variables $x_1, \ldots, x_n$ have been relabeled in such a way that $U = \{x_1, \ldots, x_r\}$ and $V = \{x_{r+1}, \ldots, x_n\}$. Consequently, for an n-tuple x, the first r components are denoted by x^U, and the last $s = n - r$ components, by x^V.

Definition 1 Let F be an *fr* function, with $n > 0$ inputs and $m > 0$ outputs, and let (U, V) be as above. Assume that F is specified by a set $\mathcal{F}$ of function cubes. Let G be an *fr* function with $s = n - r$ inputs and p outputs, and let H be an *fr* function with $r + p$ inputs and $m > 0$ outputs. The pair (G, H) is a *disjunctive serial separation* of F *with respect to* (U, V), if, for every minterm b relevant to $\mathcal{F}$, $G(b^V)$ is defined, $G(b^V) \in \{0,1\}^p$, and

$$F(b) \sqsupseteq H(b^U, G(b^V)). \tag{4}$$

Note that we allow the case $r = 0$; then (4) becomes $F(b) \sqsupseteq H(G(b))$. We also permit the case $s = 0$; here we have $F(b) \sqsupseteq H(b, G)$, where G is a constant binary p-tuple. Finally, we allow the case $p = 0$; now (4) becomes $F(b) \sqsupseteq H(b^U)$. If $U \neq X$, this would show that the outputs of F are uniquely determined by a proper subset of the input variables.

Theorem 1 *An fr function F specified by a set $\mathcal{F}$ of function cubes has a disjunctive serial separation with respect to (U, V) if and only if there exists a blanket β_G on $\mathcal{F}$ such that*

- $\beta_V \leq \beta_G$, *and*
- $\beta_U * \beta_G \leq \beta_F$,

where β_U and β_V are the input blankets of U and V, and β_F is the output blanket.

Proof: (a) Necessity of the conditions

The reader may wish to look at Example 2 below when following this part of the proof.

Suppose first that a (disjunctive serial) separation (G, H) exists. Then, for every binary s-tuple b^V, such that b is relevant to $\mathcal{F}$, the function G is defined and its outputs are all binary. Define blanket β_G on $\mathcal{F}$ as follows. Let $f \in \{0,1\}^p$, and let $\beta_G = ne\{\mathcal{F}_{G=f}\}$, where

$$\mathcal{F}_{G=f} = \{t \in \mathcal{F} \mid \exists d \in \{0,1\}^s \text{ such that } t_\downarrow^V \sqsupseteq d, \text{ and } G(d) = f\}. \quad (5)$$

We first verify that $\beta_V \leq \beta_G$. Suppose that $t\beta_V t'$ for a pair t and t' of cubes of $\mathcal{F}$. By Proposition 3, there exists a $d \in \{0,1\}^s$ such that $t_\downarrow^V \sqsupseteq d$ and $t'^V_\downarrow \sqsupseteq d$. Let $a \in \{0,1\}^{n-s}$ be such that $t_\downarrow \sqsupseteq ad$; clearly, at least one such a can always be found. Similarly, let $a' \in \{0,1\}^{n-s}$ be such that $t'_\downarrow \sqsupseteq a'd$. Since $b = ad$ and $b' = a'd$ are relevant to $\mathcal{F}$, the value $G(b^V) = G(d) = G(b'^V)$ is defined by definition of separation; suppose that value is f. By construction of β_G, both t and t' belong to block $\mathcal{F}_{G=f}$ of β_G. Hence $\beta_V \leq \beta_G$.

Next, we check whether $\beta_U * \beta_G \leq \beta_F$. If $t\beta_U t'$, then there exists a $c \in \{0,1\}^r$ such that $t_\downarrow^U \sqsupseteq c$, and $t'^U_\downarrow \sqsupseteq c$. If $t\beta_G t'$, then there exist $f \in \{0,1\}^p$, $d \in \{0,1\}^s$, and $d' \in \{0,1\}^s$ such that $t_\downarrow^V \sqsupseteq d$, and $t'^V_\downarrow \sqsupseteq d'$, and $G(d) = G(d') = f$. Consider now the n-tuples $b = cd$ and $b' = cd'$; clearly, $t_\downarrow \sqsupseteq b$, and $t'_\downarrow \sqsupseteq b'$. Now these two n-tuples b and b' agree in the first r components, and result in the same value of G from the last s components, that is $b^U = b'^U$ and $G(b^V) = G(b'^V)$. Consequently, $H(b^U, G(b^V)) = H(b'^U, G(b'^V))$.

Now define blanket β_H as follows. Let $e \in \{0,1\}^m$ and let $\beta_H = ne\{\mathcal{F}_{H \sqsupseteq e}\}$, where

$$\mathcal{F}_{H \sqsupseteq e} = \{t \in \mathcal{F} \mid \exists b \in \{0,1\}^n \text{ such that } t_\downarrow \sqsupseteq b \text{ and } H(b^U, G(b^V)) \sqsupseteq e\}. \tag{6}$$

For t and t' from the previous paragraph, $t\beta_H t'$. Altogether, we have shown that $t\beta_U t'$ and $t\beta_G t'$ together imply $t\beta_H t'$. In other words, $t(\beta_U * \beta_G)t'$ implies $t\beta_H t'$.

We now claim that $t\beta_H t'$ implies $t\beta_F t'$. Suppose that t and t' are in the same block $\mathcal{F}_{H \sqsupseteq e}$ of β_H. Then there exists $b \in \{0,1\}^n$ such that $t_\downarrow \sqsupseteq b$ and $H(b^U, G(b^V)) \sqsupseteq e$. Then b is relevant to $\mathcal{F}$ and, by the assumption that (G, H) is a separation, we have $F(b) \sqsupseteq H(b^U, G(b^V))$. Consequently, $F(b) \sqsupseteq e$. But $F(b) = \bigsqcap_{u \in \mathcal{F}_{X \sqsupseteq b}} u^\uparrow$. Since $t_\downarrow \sqsupseteq b$, we have $t \in \mathcal{F}_{X \sqsupseteq b}$. Since $t^\uparrow$ is one of the factors in the $\sqcap$-product for $F(b)$ it satisfies $t^\uparrow \sqsupseteq F(b)$. Thus $t^\uparrow \sqsupseteq F(b) \sqsupseteq e$, and t is in block $\mathcal{F}_{Y \sqsupseteq e}$ of the output blanket β_F. By the same argument, t' is also in block $\mathcal{F}_{Y \sqsupseteq e}$ of β_F. Thus $t\beta_F t'$. This proves that $\beta_U * \beta_G \leq \beta_F$, and concludes the first part of the proof.

(b) Sufficiency of the conditions

The reader may wish to look at Example 3 below when following the second part of the proof.

Suppose that a blanket β_G exists and has q blocks. We define an *fr* function G with s inputs and $p = \lceil \log_2 q \rceil$ outputs as follows. Encode the q blocks of β_G by p variables in such a way that each block B is assigned a distinct p-tuple $\alpha(B)$; otherwise, the coding is arbitrary. For each $d \in \{0,1\}^s$, find the set $\mathcal{F}_{V \sqsupseteq d} = \{t \in \mathcal{F} \mid t^V_\downarrow \sqsupseteq d\}$. If this set is empty, omit d from the truth table for G. Otherwise, note that β_G satisfies the condition $\beta_V \leq \beta_G$; hence each block $\mathcal{F}_{V \sqsupseteq d}$ of β_V is contained in a block B of β_G. Let $G(d) = \alpha(B)$. (If $\mathcal{F}_{V \sqsupseteq d}$ is contained in more than one block of β_G, pick any block, arbitrarily.) In this way we obtain a (partial) truth table for the function G.

Next we construct a (partial) truth table for a function H with $r + p$ inputs and m outputs. For each $h = cf$ with $c \in \{0,1\}^r$ and $f \in \{0,1\}^p$, we will either assign to h a block B_h of the blanket $\beta_U * \beta_G$, or h will be absent from the table of H. Find the set $\mathcal{F}_{U \sqsupseteq c} = \{t \in \mathcal{F} \mid t^U_\downarrow \sqsupseteq c\}$. If this set is empty, omit h from the truth table for H. Otherwise, continue. Either the set $\mathcal{F}_{G=f}$ is nonempty, in which case it is a block of β_G, or not. If $B_h = \mathcal{F}_{U \sqsupseteq c} \cap \mathcal{F}_{G=f} \neq \emptyset$, associate with h that block of $\beta_U * \beta_G$. Otherwise, omit h from the table for H. By the condition $\beta_U * \beta_G \leq \beta_F$, we know that every block B_h of $\beta_U * \beta_G$ is contained in some block $\hat{B}$ of the output blanket β_F. But, if t and t' are in the same block of β_F, then $t^\uparrow \sim t'^\uparrow$. Hence, we know that $\bigsqcap_{t \in B_h} t^\uparrow$ is defined. Let $H(h) = \bigsqcap_{t \in B_h} t^\uparrow$. Thus we obtain the truth table for H.

It remains to be proved that the functions G and H constructed above satisfy the conditions of the definition of disjunctive serial separation. Let b be a minterm relevant to $\mathcal{F}$; then

$$F(b) = \bigsqcap_{t \in \mathcal{F}_{X \sqsupseteq b}} t^\uparrow. \tag{7}$$

Let $d = b^V$; then $\mathcal{F}_{V \sqsupseteq d}$ is not empty, since b is relevant. Suppose that $\mathcal{F}_{V \sqsupseteq d}$ is contained in block B' of β_G, and that $\alpha(B') = f$; by construction, $G(b^V) = f$. Note that

$$\mathcal{F}_{X \sqsupseteq b} \subseteq \mathcal{F}_{V \sqsupseteq d} \subseteq B'.$$

If $c = b^U$, then the input tuple to H is $h = cf$. The set $B = \mathcal{F}_{U \sqsupseteq c} = \{t \mid t_\downarrow^U \sqsupseteq c\}$ is not empty, because $c = b^U$, and b is relevant to $\mathcal{F}$. Note that

$$\mathcal{F}_{X \sqsupseteq b} \subseteq \mathcal{F}_{U \sqsupseteq c} = B;$$

hence, h appears in the table for H, and we assign block $B_h = B \cap B'$ to h. Then we set

$$H(h) = \bigsqcap_{t \in B_h} t^\uparrow. \tag{8}$$

Since $\mathcal{F}_{X \sqsupseteq b} \subseteq B$ and $\mathcal{F}_{X \sqsupseteq b} \subseteq B'$, we have $\mathcal{F}_{X \sqsupseteq b} \subseteq B \cap B' = B_h$. Hence, $F(b) \sqsupseteq H(h)$. This concludes the proof of the claim that $F(b) \sqsupseteq H(b^U, G(b^V))$, and also the proof of the theorem. $\square$

Example 2 In this example we construct a blanket β_G from a given decomposition (G, H) of a function F, thus illustrating the first part of the proof of the theorem. The *fr* function F of Table 4 is decomposed with respect to $U = \{x_1\}$ and $V = \{x_2, x_3, x_4\}$. Table 5 shows the truth table of a function G. Note that the truth table is incomplete, but G does satisfy the condition that $G(b^V)$ is defined and binary whenever b is relevant to $\mathcal{F}$. Consider the triple $d = 000$; we have $G(d) = 00$. The triple d appears in cube 1 of Table 4, in the sense that the last 3 components of cube 1 are compatible with d. Hence, cube 1 is assigned to block $\mathcal{F}_{G=00}$. Triple 000 also appears in cubes 2 and 3; hence these cubes are also assigned to this block. Next, 001 appears in cubes 3 and 7; since $G(001) = 00$, cubes 3 and 7 also belong to $\mathcal{F}_{G=00}$. Using 010, we conclude that cubes 1 and 2 are in $\mathcal{F}_{G=00}$. Since

Table 5: Function G.

x_2	x_3	x_4	g_1	g_2
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	1	1	0
1	1	0	0	1
1	1	1	1	1

G is equal to 00 only for these three triples, we have $\mathcal{F}_{G=00} = \{1, 2, 3, 7\}$. Similarly, we find $\mathcal{F}_{G=01} = \{5\}$, $\mathcal{F}_{G=10} = \{6, 7\}$ and $\mathcal{F}_{G=11} = \{4, 6\}$. Thus, we have the blanket $\beta_G = \{\overline{1, 2, 3, 7};\ \overline{5};\ \overline{6, 7};\ \overline{4, 6}\}$.

One verifies that $\beta_V \leq \beta_G$, where β_V is given after Equation (2). Also, $\beta_U = \{\overline{1, 3, 4, 5, 6, 7};\ \overline{2, 3, 4, 5, 6}\}$, $\beta_F = \{\overline{4, 5};\ \overline{4, 6};\ \overline{2, 3, 7};\ \overline{1, 3, 6, 7}\}$, and

$$\beta_U * \beta_G = \{\overline{1, 3, 7};\ \overline{5};\ \overline{6, 7};\ \overline{4, 6};\ \overline{2, 3};\ \overline{6}\} \leq \beta_F.$$

Thus the conditions of Theorem 1 are satisfied. □

Example 3 In this example we illustrate the second part of the proof of the theorem. We will find a decomposition of the function of Table 4 using

Table 6: Defining function G from β_G.

x_2	x_3	x_4	Block of β_V	Block of β_G	g_1	g_2
0	0	0	$\{1, 2, 3\}$	$\{1, 2, 3, 7\}$	0	0
0	0	1	$\{3, 7\}$	$\{1, 2, 3, 7\}$	0	0
0	1	0	$\{1, 2\}$	$\{1, 2, 3, 7\}$	0	0
0	1	1	$\{4\}$	$\{4, 6\}$	1	1
1	0	0	–	–	–	–
1	0	1	$\{6, 7\}$	$\{6, 7\}$	1	0
1	1	0	$\{5\}$	$\{5\}$	0	1
1	1	1	$\{4, 6\}$	$\{4, 6\}$	1	1

a given blanket $\beta_G = \{\overline{1, 2, 3, 7};\ \overline{5};\ \overline{6, 7};\ \overline{4, 6}\}$. Encode the blocks of β_G using two variables as follows:

$$\beta_G = \{\overset{00}{\overline{1,2,3,7}};\ \overset{01}{\overline{5}};\ \overset{10}{\overline{6,7}};\ \overset{11}{\overline{4,6}}\}.$$

In Table 6, if possible, we first assign (uniquely) a block $\mathcal{F}_{V \sqsupseteq d}$ of β_V to each $d \in \{0,1\}^s$. Then we find a block of β_G containing $\mathcal{F}_{V \sqsupseteq d}$; in this case there is no choice. The resulting table with don't care rows omitted is the same as Table 5.

Next, we construct the function H. The steps of the algorithm are indicated in Table 7. □

Table 7: Defining function H.

x_1	g_1	g_2	β_U	β_G	$\beta_U * \beta_G$	y_1	y_2
0	0	0	$\{1,3,4,5,6,7\}$	$\{1,2,3,7\}$	$\{1,3,7\}$	1	1
0	0	1	$\{1,3,4,5,6,7\}$	$\{5\}$	$\{5\}$	0	0
0	1	0	$\{1,3,4,5,6,7\}$	$\{6,7\}$	$\{6,7\}$	1	1
0	1	1	$\{1,3,4,5,6,7\}$	$\{4,6\}$	$\{4,6\}$	0	1
1	0	0	$\{2,3,4,5,6\}$	$\{1,2,3,7\}$	$\{2,3\}$	1	0
1	0	1	$\{2,3,4,5,6\}$	$\{5\}$	$\{5\}$	0	0
1	1	0	$\{2,3,4,5,6\}$	$\{6,7\}$	$\{6\}$	Φ	1
1	1	1	$\{2,3,4,5,6\}$	$\{4,6\}$	$\{4,6\}$	0	1

If the construction in the proof of Theorem 1 is used, the result of Theorem 1 cannot be strengthened to equality, as we now show. When $x_1, \ldots, x_4 = 1001$, we have $F(1001) = 1\Phi$. We also have $G(001) = 00$, and $H(100) = 10$. Thus F and H are not equal in this case.

We close this section by showing that our results remain true if we use set systems instead of blankets.

Theorem 2 *An fr function F specified by a set $\mathcal{F}$ of function cubes has a disjunctive serial separation with respect to (U,V) if and only if there exists a set system σ_G on $\mathcal{F}$ such that*

- $\sigma_V \leq \sigma_G$, *and*
- $\sigma_U \circ \sigma_G \leq \sigma_F$,

where $\sigma_U = max\beta_U$ and $\sigma_V = max\beta_V$ are the input set systems of U and V, and $\sigma_F = max\beta_F$ is the output set system.

Proof: Suppose F has a separation (G, H). Let $\sigma_G = max\beta_G$, where β_G is defined in the proof of Theorem 1. Since $\beta_V \leq \beta_G$, we also have $max\beta_V \leq max\beta_G$. Similarly, $max\beta_U * max\beta_G \leq max\beta_F$. Consequently, the required conditions are satisfied.

Conversely, suppose a set system σ_G exists. The reader can verify that the sufficiency proof goes through for this case also, in a manner parallel to that of Theorem 1. □

The advantages of using blankets are discussed further in Section 9. Theorem 2 assures us that, if there is a decomposition based on blankets, then there is one based on set systems, but there are many decompositions based on blankets corresponding to the same set system. On the other hand, set systems may be preferable from the computational complexity point of view, since they may have significantly fewer blocks.

7 Nondisjunctive Serial Separations

In this section we consider the decomposition problem when the restriction that U and V be disjoint is removed. Let U and V be two subsets of X such that $U \cup V = X$. Assume that the variables $x_1, \ldots, x_n$ have been relabeled in such a way that $U = \{x_1, \ldots, x_r\}$ and $V = \{x_{n-s+1}, \ldots, x_n\}$. Consequently, for an n-tuple x, the first r components are denoted by x^U, and the last s components, by x^V.

Definition 2 Let F be an *fr* function, with $n > 0$ inputs and $m > 0$ outputs, and let (U, V) be as above. Assume that F is specified by a set $\mathcal{F}$ of function cubes. Let G be an *fr* function with s inputs and p outputs, and let H be an *fr* function with $r + p$ inputs and m outputs. The pair (G, H) is a *serial separation* of F *with respect to* (U, V), if, for every minterm b relevant to $\mathcal{F}$, $G(b^V)$ is defined, $G(b^V) \in \{0, 1\}^p$, and

$$F(b) \sqsupseteq H(b^U, G(b^V)).$$

Theorem 3 *If there exists a blanket β_G on $\mathcal{F}$ such that*

- *$\beta_V \leq \beta_G$, and*

- $\beta_U * \beta_G \leq \beta_F$,

then F has a serial separation with respect to (U, V).

Proof: The construction described in the proof of Theorem 1 (sufficiency) applies equally well here. We leave the details to the reader. (Example 5 below shows that the conditions are not always necessary.) $\square$

Example 4 For the function of Table 4, consider a nondisjunctive decomposition with $U = \{x_1, x_4\}$ and $V = \{x_2, x_3, x_4\}$. We find $\beta_U =$

Table 8: Function G for Example 4.

x_2	x_3	x_4	g
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	1	0
1	1	0	1
1	1	1	1

$\{\overset{00}{\overline{1,3,5}}\ ;\ \overset{01}{\overline{3,4,6,7}};\ \overset{10}{\overline{2,3,5}};\ \overset{11}{\overline{3,4,6}}\}$, $\beta_V = \{\overset{000}{\overline{1,2,3}};\ \overset{001}{\overline{3,7}};\ \overset{010}{\overline{1,2}};\ \overset{011}{\overline{4}}\ ;\ \overset{101}{\overline{6,7}};\ \overset{110}{\overline{5}}\ ;\ \overset{111}{\overline{4,6}}\}$. It is easily verified that $\beta_G = \{\overline{1,2,3,6,7};\ \overline{4,5,6}\}$ satisfies the conditions

Table 9: Function H for Example 4.

x_1	x_4	g	y_1	y_2
0	0	0	1	1
0	0	1	0	0
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	1	1
1	1	1	0	1

of Theorem 3. If we use the encoding $\beta_G = \{\overset{0}{\overline{1,2,3,6,7}}; \overset{1}{\overline{4,5,6}}\}$, the truth tables of functions G and H are as shown in Tables 8 and 9, respectively. In many cases, a serial separation with overlapping sets U and V—sometimes called a nondisjunctive decomposition [7]—is useful. For example, suppose we have an FPGA structure with 3-input, 1-output cells. Then the nondisjunctive decomposition above requires one cell for G and two cells for H (one for each output). Thus, we need a total of three cells. A disjunctive decomposition of the form $H(x_1, x_2, G(x_3, x_4))$, with G having one output would also cost three cells. However, no such decomposition exists. There does exist a disjunctive decomposition of the form $H(x_1, G(x_2, x_3, x_4))$, where G has two outputs, but it would require four cells (two for G and two for H). □

Example 5 The following example shows that the construction used in the proof of the necessity of the conditions of Theorem 1 cannot be used in the nondisjunctive case. Consider function F of Table 10 and functions G and

Table 10: Function F for Example 5.

	x_1	x_2	x_3	x_4	y
1	0	Φ	0	1	0
2	0	Φ	1	1	1

Table 11: Functions G and H for Example 5.

Function G

x_2	x_3	x_4	g
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	0

Function H

x_1	x_2	g	y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0

H of Table 11. Let $U = \{x_1, x_2\}$ and $V = \{x_2, x_3, x_4\}$. One verifies that (G, H) is a separation of F with respect to (U, V). Note, however, that $\beta_U = \{\overline{1,2}\}$, $\beta_V = \{\overline{1}; \overline{2}\}$, and $\beta_F = \{\overline{1}; \overline{2}\}$. The blanket constructed from G is $\beta_G = \{\overline{1,2}\}$. This blanket fails to satisfy the condition $\beta_U * \beta_G \leq \beta_F$.

Still, there *is* a blanket, namely $\beta_K = \{\overline{1};\ \overline{2}\}$, that satisfies both conditions of Theorem 3. However, β_k is not derived from G. $\square$

8 Serial Decompositions

Suppose we have a serial separation (G, H) of a function F of n variables, where G has s inputs and p outputs, and H has $r+p$ inputs and m outputs. Such a separation is not quite a decomposition, because in a decomposition we want G and H to have strictly fewer inputs than F. Thus we should insist that $s < n$. Furthermore, to guarantee that H has fewer inputs than F, we insist that $r + p < n$.

Definition 3 A *serial decomposition* of F *with respect to* (U, V) is a serial separation (G, H), in which $s < n$, and $r + p < n$.

Proposition 5 *If a serial separation (G, H) based on blanket β_G is a serial decomposition, then $s < n$ and the number q of blocks of β_G satisfies*

- $r + \lceil \log_2 q \rceil < n$.

We now develop a useful necessary condition for the existence of a decomposition. Let β be a blanket on a set, and $\beta' = \{B_i'\}$, a blanket on the same set. The *quotient* of a block B of β by β' is a set system on the set B defined by

$$B/\beta' = max\{B \cap B_i'\}.$$

The *block-cover cost* $c(B_1, \beta')$ is the minimum number of blocks of B/β' required to cover B, in the sense that the union of these blocks is B. For example, let

$$\beta = \{\overset{B_1}{\overline{1,2,3}};\ \overset{B_2}{\overline{3,6,7}};\ \overset{B_3}{\overline{1,2,5}};\ \overset{B_4}{\overline{4,6}}\},$$

and

$$\beta' = \{\overline{1,3,6,7}; \overline{2,3,7}; \overline{4,6}; \overline{4,5}\}.$$

To calculate $c(B_1, \beta')$, we first intersect B_1 with all the blocks of β', obtaining $\overline{1,3};\ \overline{2,3};\ \emptyset;\ \emptyset$. Applying the *max* operation, we get $B_1/\beta' = \{\overline{1,3};\ \overline{2,3}\}$. Since both blocks are needed to cover B_1, we have $c(B_1, \beta') = 2$. Similarly, we find $B_2/\beta' = \{\overline{3,6,7}\}$, $B_3/\beta' = \{\overline{1};\ \overline{2};\ \overline{5}\}$, and $B_4/\beta' = \{\overline{4,6}\}$. Thus $c(B_2, \beta') = 1$, $c(B_3, \beta') = 3$, and $c(B_4, \beta') = 1$.

The *cover cost*, $c(\beta, \beta')$, is now defined as the largest block-cover cost, that is,

$$c(\beta, \beta') = max\{c(B, \beta') \mid B \in \beta\},$$

where *max* is used here to denote the largest integer in a set. In the example above, $c(\beta, \beta') = 3$.

Theorem 4 *Let F be an* fr *function, and let U and V be subsets of X, $U \cup V = X$, where X has n elements, U has $r < n$ elements, and V has $s < n$ elements. If a serial decomposition of F based on blanket β_G exists, then*

$$n - r > \lceil \log_2 c(\beta_U, \beta_F) \rceil .$$

Proof: Suppose a decomposition (G, H) based on β_G exists. By Proposition 5, if β_G has q blocks, then

$$n - r > \lceil \log_2 q \rceil .$$

Suppose that, for some $d \in \{0,1\}^r$, there is a block $\mathcal{F}_{U \sqsupseteq d}$ of β_U that requires at least k blocks of β_F to be covered. Let this cover be C_d. If B_e is a block of C_d, then B_e cannot be removed, implying that there is at least one cube t of $\mathcal{F}_{U \sqsupseteq d}$ that is contained only in B_e, but not in any other B_f from C_d. In other words, B_e is essential for t. Hence there must be at least k different cubes t_e in $\mathcal{F}_{U \sqsupseteq d}$, such that each cube is associated with a distinct output value e, in the sense that $t_e^\uparrow \sqsupseteq e$, but $t'^\uparrow \not\sqsupseteq e$, for any t' assigned to a different block of C_d. Hence, for this d, when the input to F is of the form $(x_1, \ldots, x_n) = de$, the output may take on any one of k values depending on the value of e. In the decomposed function, the role of e is taken over by the $\lceil \log_2 q \rceil$ outputs from function G. These outputs can produce at most q different combinations of values for H. The number of these values must be greater than or equal to the cost of covering β_U, that is

$$q \geq c(\beta_U, \beta_F).$$

Consequently,

$$n - r > \lceil \log_2 q \rceil \geq \lceil \log_2 c(\beta_U, \beta_F) \rceil ,$$

as required in the theorem. □

Example 6 Let us find all the sets U for which a decomposition of the function F of Table 4 is not ruled out by the condition of Theorem 4. For convenience, we will use c_0 as a shorthand for $\lceil \log_2 c(\beta_U, \beta_F) \rceil$.

Recall that

$$\beta_F = \{\overline{4,5};\ \overline{4,6};\ \overline{2,3,7};\ \overline{1,3,6,7}\}.$$

For one-variable sets, that is, for $U = \{x_i\}$, we have $n - r = 3$. The condition of the theorem is satisfied, because the number of blocks in β_F is 4; consequently, the cost $c(B, \beta_F)$ cannot be greater than 4 for any block B of β_U. Therefore, $c_0 \leq 2 < n - r = 3$.

For larger sets U, some computation is required. For $U = \{x_1, x_2\}$, we have:

$$\beta_U = \{\overbrace{\overline{1,3,4,7}}^{B_1};\ \overbrace{\overline{4,5,6,7}}^{B_2};\ \overbrace{\overline{2,3,4}}^{B_3};\ \overbrace{\overline{4,5,6}}^{B_4}\},$$

$$B_1/\beta_F = max\{\overline{4};\ \overline{4};\ \overline{3,7};\ \overline{1,3,7}\} = \{\overline{4};\ \overline{1,3,7}\},$$

$$B_2/\beta_F = max\{\overline{4,5};\ \overline{4,6};\ \overline{7};\ \overline{6,7}\} = \{\overline{4,5};\ \overline{4,6};\ \overline{6,7}\},$$

$$B_3/\beta_F = max\{\overline{4};\ \overline{4};\ \overline{2,3};\ \overline{3}\} = \{\overline{4};\ \overline{2,3}\},$$

and

$$B_4/\beta_F = max\{\overline{4,5};\ \overline{4,6};\ \overline{\emptyset};\ \overline{6}\} = \{\overline{4,5};\ \overline{4,6}\}.$$

The corresponding costs are $c(B_1, \beta_F) = 2$, $c(B_2, \beta_F) = 2$, $c(B_3, \beta_F) = 2$, $c(B_4, \beta_F) = 2$. Hence $c_0 = 1 < n - r = 2$, and the condition is satisfied.

For $U = \{x_1, x_3\}$,

$$\beta_U = \{\overbrace{\overline{1,3,6,7}}^{B_1};\ \overbrace{\overline{1,4,5,6}}^{B_2};\ \overbrace{\overline{2,3,6}}^{B_3};\ \overbrace{\overline{2,4,5,6}}^{B_4}\}.$$

Here U fails to satisfy the condition, because

$$B_4/\beta_F = max\{\overline{4,5};\ \overline{4,6};\ \overline{2};\ \overline{6}\} = \{\overline{4,5};\ \overline{4,6};\ \overline{2}\}.$$

Hence, $c_0 = 2 \not< 2 = n - r$.

Altogether, the condition is satisfied for the following subsets of X:

$$\{x_1\}, \{x_2\}, \{x_3\}, \{x_4\}, \{x_1, x_2\}, \{x_1, x_4\}, \text{ and } \{x_2, x_4\}.$$

After the verification of the necessary conditions from Theorem 1, we find that only three of these sets, namely $U_1 = \{x_1\}, U_2 = \{x_2\}, U_3 = \{x_3\}$, lead to disjunctive serial decompositions.

9 Decomposition Method Using Cubes

The decomposition of a function F is completed when truth tables for components G and H have been found. For convenience and simplicity, in the proof of Theorem 1 we used minterms to construct these tables. Thus, we considered $\{0,1\}^s$ and $\{0,1\}^{r+p}$ as inputs to G and H, respectively. Note, however, that F is specified by cubes, and it would be desirable to have a method that handles all functions uniformly as cubes. We now present such a method.

Example 7 Table 12 shows a function F to be decomposed with respect to $U = \{x_1, x_2\}$ and $V = \{x_3, x_4, x_5, x_6\}$. Blanket β_V is

$$\beta_V = \{\overline{7,8}^{B_1};\ \overline{3,4,8}^{B_2};\ \overline{8}^{B_3};\ \overline{4,8}^{B_4};\ \overline{1,5,8}^{B_5};\ \overline{3,5,8}^{B_6};\ \overline{6,8}^{B_7};\ \overline{1,2,5,8}^{B_8}\}.$$

Assume that we have found a blanket σ_G, in this case a set system, that satisfies the conditions of Theorem 1. Assume that the blocks of σ_G are encoded as shown.

$$\sigma_G = \{\overline{3,4,7,8}^{00};\ \overline{3,5,8}^{01};\ \overline{6,8}^{10};\ \overline{1,2,5,8}^{11}\}.$$

To define a function G by a set $\mathcal{G}$ of function cubes, in this case 6-tuples,

Table 12: Function F.

	x_1	x_2	x_3	x_4	x_5	x_6	y_1	y_2	y_3
1	Φ	0	1	Φ	0	0	1	Φ	Φ
2	0	Φ	1	1	0	0	1	Φ	Φ
3	1	Φ	Φ	Φ	0	1	Φ	1	Φ
4	0	Φ	0	Φ	Φ	1	Φ	1	Φ
5	1	Φ	1	Φ	0	Φ	Φ	Φ	1
6	Φ	Φ	1	0	1	Φ	0	0	0
7	Φ	Φ	0	Φ	0	0	Φ	Φ	0
8	1	1	Φ	Φ	Φ	Φ	0	Φ	Φ

we must represent all the minterms relevant to $\mathcal{G}$. These are all the binary 4-tuples that are covered by the V columns of $\mathcal{F}$. Because of row 8, all 16 minterms are relevant to $\mathcal{G}$. The relationship between blocks of β_V and

Table 13: Function F.

Block of β_V	Cubes of $\mathcal{F}$	Minterms of G
B_1	$\{7, 8\}$	$\{0000, 0100\}$
B_2	$\{3, 4, 8\}$	$\{0001, 0101\}$
B_3	$\{8\}$	$\{0010, 0110, 1110, 1111\}$
B_4	$\{4, 8\}$	$\{0011, 0111\}$
B_5	$\{1, 5, 8\}$	$\{1000\}$
B_6	$\{3, 5, 8\}$	$\{1001, 1101\}$
B_7	$\{6, 8\}$	$\{1010, 1011\}$
B_8	$\{1, 2, 5, 8\}$	$\{1100\}$

minterms of G is shown in Table 13, where minterm $d \in \{0,1\}^4$ is assigned to block B if and only if that block is equal to $\mathcal{F}_{V \sqsupseteq d}$.

Consider the following typical situation. Minterm 0011 of G is represented by the two cubes of block B_4: 4 and 8. As far as this block is concerned, we need to worry only about the minterms 0011 and 0111, even though cubes $0\Phi\Phi1$, and $\Phi\Phi\Phi\Phi$ of B_4 cover other minterms. Since each minterm d of $\mathcal{G}$ defines the block $\mathcal{F}_{V \sqsupseteq d}$, other minterms will be accounted for by other blocks. Translating this to $\mathcal{G}$, each minterm d belongs to block $\mathcal{G}_{V \sqsupseteq d}$. Consequently, we lose nothing by using the *glb* of the cubes in $\mathcal{G}_{V \sqsupseteq d}$. Now we have $0\Phi\Phi1 \sqcap \Phi\Phi\Phi\Phi = 0\Phi\Phi1$ as the representative of block B_4. Notice that it covers the two minterms: 0011 and 0111, as required, since $\mathcal{G}_{V \sqsupseteq 0011} = \mathcal{G}_{V \sqsupseteq 0111}$ in this case. Computing the representative for each block, we obtain the second column of Table 14. (For now, ignore the line between B_3 and B_4.) In effect, we have now represented all the relevant minterms of $\mathcal{G}$ by the block representatives in Table 14.

Depending on the set system σ_G used in the decomposition, we assign output values of σ_G to the representatives we have generated. In our example, the outputs will be assigned as in the third column of Table 14. Block $B_1 = \{7, 8\}$ is contained only in block $\{3, 4, 7, 8\}$ of σ_G; hence, its representative should be assigned the output 00. The same type of argument applies to all the other blocks, except B_3, which is contained in all four blocks of σ_G. The representative $\Phi\Phi\Phi\Phi$ of this block has a nonempty $\sqcap$-product with the representative $0\Phi00$ of B_1. This does not matter, since we assign the same output to all the common minterms. But $\Phi\Phi\Phi\Phi$ also shares $1\Phi00$ from B_5, which will be assigned a different output. To avoid conflicts, we must

Table 14: Cubes for G .

Block of β_V	Representative				Outputs	Final Cubes			
B_1	0	Φ	0	0	00	0	Φ	0	0
B_2	0	Φ	0	1	00	0	Φ	0	1
B_3	Φ	Φ	Φ	Φ	00	0	Φ	Φ	Φ
					00	Φ	1	1	Φ
B_4	0	Φ	Φ	1	00	0	Φ	Φ	1
B_5	1	Φ	0	0	11	1	Φ	0	0
B_6	1	Φ	0	1	01	1	Φ	0	1
B_7	1	0	1	Φ	10	1	0	1	Φ
B_8	1	1	0	0	11	1	1	0	0

subtract cubes 1Φ00, 1Φ01, 101Φ, and 1100 from ΦΦΦΦ. The result is two cubes, 0ΦΦΦ and Φ11Φ, and this remainder can be safely assigned output 00.

From Proposition 2, we know that, if $r_1 \sqcap r_2$ is defined for two block representatives r_1 and r_2, then the union $B_1 \cup B_2$ of the corresponding blocks is itself a block. Thus, when conflicts arise, we must subtract from the representative $r(B)$, assigned to block B, all the representatives assigned to blocks containing block B. The subtraction can be easily performed by intersecting $r(B)$ with the complement of the set of all the cubes to be subtracted. An efficient method for complementing a set of cubes can be found in [2].

We can use a similar approach to obtain the table of function H. Here, we deal with blocks of the product $\beta_U * \sigma_G$, where

$$\beta_U = \{\overline{1,2,4,6,7};\ \overline{2,4,6,7};\ \overline{1,3,5,6,7};\ \overline{3,5,6,7,8}\},$$

and

$$\beta_F = \{\overset{111}{\overline{1,2,3,4,5}};\ \overset{110}{\overline{1,2,3,4,7}};\ \overset{101}{\overline{1,2,5}};\ \overset{100}{\overline{1,2,7}};\ \overset{011}{\overline{3,4,5,8}};\ \overset{010}{\overline{3,4,7,8}};\ \overset{001}{\overline{5,8}};\ \overset{000}{\overline{6,7,8}}\}.$$

We find

$$\beta_U * \sigma_G = \{\overset{B_1}{\overline{4,7}};\ \overset{B_2}{\overline{6}};\ \overset{B_3}{\overline{1,2}};\ \overset{B_4}{\overline{2}};\ \overset{B_5}{\overline{3,7}};\ \overset{B_6}{\overline{3,5}};\ \overset{B_7}{\overline{1,5}};\ \overset{B_8}{\overline{3,7,8}};\ \overset{B_9}{\overline{3,5,8}};\ \overset{B_{10}}{\overline{6,8}};\ \overset{B_{11}}{\overline{5,8}}\}.$$

To compute the table for H we consider each block of $\beta_U * \sigma_G$ in turn, and generate Table 15 as follows. For $B_1 = \{4,7\}$, the U cubes from Table 12

Table 15: Cubes for H .

Block	β_U	σ_G	Output
B_1	0Φ	00	$\Phi10$
B_2	$\Phi\Phi$	10	000
B_3	00	11	$1\Phi\Phi$
B_4	0Φ	11	$1\Phi\Phi$
B_5	1Φ	00	$\Phi10$
B_6	1Φ	01	$\Phi11$
B_7	10	11	$1\Phi1$
B_8	11	00	010
B_9	11	01	011
B_{10}	11	10	000
B_{11}	11	$\Phi1$	$0\Phi1$

are $0\Phi \sqcap \Phi\Phi = 0\Phi$, so 0Φ becomes the first part of an input cube of $\mathcal{H}$. Since this block is contained in the block of σ_G that has output 00, then 00 becomes the second part of the input cube. The output value assigned to this cube is $\Phi1\Phi \sqcap \Phi\Phi0 = \Phi10$. The remaining rows are completed in the same fashion. Note that B_{11} appears in blocks 01 and 11 of σ_G; hence, its entry is $\Phi1$ in the σ_G columns. Finally, the outputs are calculated using the *glb* of the output portion of the cubes. For example, the output for block B_1 is $\Phi1\Phi \sqcap \Phi\Phi0 = \Phi10$.

10 Decomposition Procedures

Although the goal of this paper is to extend existing functional decomposition methods to the case of multiple-output *fr* Boolean functions, some discussion of decomposition algorithms will be included, particularly as they relate to logic synthesis optimization algorithms. In this section we show that the main decomposition tasks can be reduced to graph partitioning or graph coloring problems [22], or to the computation of maximal cliques or maximal independent sets in a graph. Other, more detailed tasks are reducible to widely used procedures in modern logic synthesis, for example, column covering or cube complementation [2, 19].

10.1 Maximal Mergeability Classes

The main task in calculating a serial decomposition of a function F with given sets U and V is to find a blanket β_G which satisfies the conditions of Theorem 1. Since β_G must be $\geq \beta_V$, we will construct β_G by merging blocks of β_V as much as possible.

Two blocks B_i and B_j of blanket β_V are *mergeable*, if blanket γ_{ij} obtained from β_V by merging B_i and B_j into a single block satisfies the second condition of Theorem 1, that is, if $\beta_U * \gamma_{ij} \leq \beta_F$. Otherwise blocks B_i and B_j are *unmergeable*. A subset δ of blocks of the input blanket β_V is a *mergeable class* of blocks if the blocks in δ are pairwise mergeable. A mergeable class is *maximal* if it is not contained in any other mergeable class.

Example 8 For the function of Table 4 with $U = \{x_1\}$, we have $\beta_U = \{\overline{1,3,4,5,6,7};\ \overline{2,3,4,5,6}\}$,

$$\beta_V = \{\overset{B_1}{\overline{1,2,3}};\ \overset{B_2}{\overline{3,7}};\ \overset{B_3}{\overline{1,2}};\ \overset{B_4}{\overline{4}};\ \overset{B_5}{\overline{6,7}};\ \overset{B_6}{\overline{5}};\ \overset{B_7}{\overline{4,6}}\},$$

and $\beta_F = \{\overline{4,5};\ \overline{4,6};\ \overline{2,3,7};\ \overline{1,3,6,7}\}$, where the blocks of β_V are labeled as shown.

Let us check whether the pair (B_1, B_2) is mergeable. Denoting the blanket resulting from the merger of B_i and B_j by γ_{ij}, we have $\gamma_{12} = \{\overline{1,2,3,7};\ \ldots\}$, and $\gamma_{12} * \beta_U = \{\overline{1,3,7};\ \overline{2,3};\ \ldots\}$, where we have omitted the rest of blocks in the blankets, since they have no influence on the result. Since $\gamma_{12} * \beta_U \leq \beta_F$, the pair (B_1, B_2) is mergeable. For the pair (B_1, B_4), we have $\gamma_{14} = \{\overline{1,2,3,4};\ \ldots\}$, and $\gamma_{14} * \beta_U = \{\overline{1,3,4};\ \overline{2,3,4};\ \ldots\}$. Here, the inequality $\gamma_{14} * \beta_U \leq \beta_F$ is not satisfied; therefore, the pair (B_1, B_4) is unmergeable.

One verifies that only six pairs are mergeable, namely: (B_1, B_2), (B_1, B_3), (B_2, B_3), (B_2, B_5), (B_4, B_6), and (B_4, B_7). For these pairs there are four maximal classes: $\{B_1, B_2, B_3\}$, $\{B_2, B_5\}$, $\{B_4, B_6\}$, and $\{B_4, B_7\}$.

The problem of finding maximal mergeability classes is identical to that of finding maximal compatibility classes for the reduction of incompletely specified finite-state machines [10]. From the computational point of view, finding maximal mergeability classes is equivalent to finding maximal cliques in a graph $\Gamma = (N, E)$, where the set N of nodes is the set of blocks of β_V, and the set E of edges is formed by the set of mergeable pairs. In the next subsection we describe one method for solving this problem. $\square$

10.2 Finding Maximal Classes Using Unmergeable Blocks

In general, the maximal classes can be formed using *unmergeable* pairs as shown in the following example.

Example 9 Consider β_U, β_V, and β_F given below. In this particular case all three are partitions.

$$\beta_U = \{\overline{1,2,9};\ \overline{3,7};\ \overline{4,5,8};\ \overline{6};\ \overline{10}\},$$

$$\beta_V = \{\overset{B_1}{\overline{1}}\,;\ \overset{B_2}{\overline{2,8}};\ \overset{B_3}{\overline{3}}\,;\ \overset{B_4}{\overline{4}}\,;\ \overset{B_5}{\overline{5}}\,;\ \overset{B_6}{\overline{6,10}};\ \overset{B_7}{\overline{7}}\,;\ \overset{B_8}{\overline{9}}\,\},$$

and

$$\beta_F = \{\overline{1,2,7};\ \overline{3,4,6};\ \overline{5,8};\ \overline{9,10}\}.$$

The following are the unmergeable pairs: (B_1, B_8), (B_2, B_4), (B_2, B_8), (B_3, B_7), and (B_4, B_5). Thus, in forming a maximal class we must omit (either B_1 or B_8) and (either B_2 or B_4), etc. Hence, we have the 2-conjunctive normal form (2-CNF), that is, a product of sums of two variables,

$$(B_1 + B_8)(B_2 + B_4)(B_2 + B_8)(B_3 + B_7)(B_4 + B_5)$$

describing the choices for the blocks that must be omitted. After "multiplying out" this expression and doing some simplifications, we obtain the following disjunctive normal form (DNF), that is, a sum-of-products expression:

$$B_3B_4B_8 + B_2B_3B_5B_8 + B_1B_2B_3B_4 + B_1B_2B_3B_5 + B_4B_7B_8 + B_2B_5B_7B_8 +$$
$$B_1B_2B_4B_7 + B_1B_2B_5B_7.$$

Each product describes a minimal set of blocks to be omitted. For example, we can omit B_3, B_4, and B_8. The maximal classes are obtained by complementing the set of blocks appearing in one product term. Thus, the maximal classes are

$$\{B_1, B_2, B_5, B_6, B_7\}, \{B_1, B_4, B_6, B_7\}, \{B_5, B_6, B_7, B_8\}, \{B_4, B_6, B_7, B_8\},$$

$$\{B_1, B_2, B_3, B_5, B_6\}, \{B_1, B_3, B_4, B_6\}, \{B_3, B_5, B_6, B_8\}, \{B_3, B_4, B_6, B_8\}.$$

□

For a discussion of efficient methods of calculating maximal mergeability classes using unmergeable pairs see [16].

10.3 Minimal Covers for β_V

The next step in calculating β_G is the selection of a set of maximal classes, with minimal cardinality, that covers all the blocks of β_V. The minimal cardinality ensures that the number of blocks of β_G, and hence the number of outputs of the function G, is as small as possible. This covering problem is equivalent to the well-known covering problem encountered in modern logic synthesis. Several algorithms have been developed for efficient calculation of minimal covers. The best-known methods inherit ideas developed for two-level logic minimization algorithms [2, 19]. The minimal cover problem for the set $\{B_i\}$ can be reduced to the column covering problem [2] if we represent rows of a binary matrix M by blocks B_i of β_V and columns of M by maximal mergeability classes C_j. Element m_{ij} of M takes on the value 1 if block B_i is contained in class C_j. Then minimal column covering means that every row of M contains a 1 in some column which appears in a minimal cover.

Covering is not the main procedure in the whole decomposition algorithm; its role is rather auxiliary. In certain heuristic strategies, both procedures (finding maximal mergeability classes and then finding the minimal cover) can be reduced to the graph coloring problem.

We continue with Example 9. A minimal cover of the set $\{B_1, \ldots, B_8\}$ is formed by the classes: $C_1 = \{B_1, B_2, B_5, B_6, B_7\}$ and $C_2 = \{B_3, B_4, B_6, B_8\}$. Another solution is $\{B_1, B_2, B_3, B_5, B_6\}$ and $\{B_4, B_6, B_7, B_8\}$. These classes can be used to construct a blanket β_G. Thus, we find

$$\beta_G = \{\overline{B_1, B_2, B_5, B_6, B_7};\ \overline{B_3, B_4, B_6, B_8}\}$$

or

$$\beta_G' = \{\overline{B_1, B_2, B_3, B_5, B_6};\ \overline{B_4, B_6, B_7, B_8}\}.$$

Translating these to the corresponding subsets of cubes, we finally have $\beta_G = \{\overline{1, 2, 5, 6, 7, 8, 10};\ \overline{3, 4, 6, 9, 10}\}$ and $\beta_G' = \{\overline{1, 2, 3, 5, 6, 8, 10};\ \overline{4, 6, 7, 9, 10}\}$.

As we will now show, we can reduce the task of generating β_G to the graph coloring problem [18]. Blocks of β_V are treated as nodes $v \in N$ and unmergeable pairs $e = (B_i, B_j) \in E$ as edges of a graph $\Gamma = (N, E)$.

A k-coloring of the nodes of a graph is a partition of the set N into k *independent* sets $S_1, \ldots, S_k$, where a set S of nodes is called independent if no two nodes of S are linked by an edge. The smallest integer k for which there exists a k-coloring of graph Γ is the *chromatic number* of Γ. Although, the problem of finding the minimal chromatic number for a given graph Γ is NP-complete, a number of fast heuristics have been developed for it [22].

Calculating β_G corresponds to finding the minimal number k of colors for graph $\Gamma = (N, E)$. In our example, $N = \{B_1, \ldots, B_8\}$ and $E = \{(B_1, B_8), (B_2, B_4), (B_2, B_8), (B_3, B_7), (B_4, B_5)\}$. Two colors are needed here; one for nodes B_1, B_2, B_5, B_6, B_7 and another for nodes B_3, B_4, B_8. The sets of nodes assigned to different colors form the blocks of β_G, as before, although the differently colored blocks are now disjoint.

11 Concluding Remarks

A distinguishing feature of our method is an original calculus based on the representation of a function using generalized set systems called blankets, which permit us to derive functional dependencies. The proposed serial decomposition procedure has already been partly included in the logic synthesis tool DEMAIN dedicated to FPGA-based logic synthesis [13]. Because of the arbitrary number of cell inputs and outputs accepted by DEMAIN, DEMAIN's applicability is wider than that of other FPGA oriented tools. This is because the FPGA logic cell is treated as a universal cell capable of implementing any n input m output Boolean function. For example, in the XILINX 3000 family, each cell can be programmed to implement any single-output function of up to five input variables, or any two-output function of up to five variables with each output depending on at most four input variables. This situation in other algorithms is solved by single-output procedures, the results of which are then merged in special optimization procedures for fitting in one multiple-output cell. Perhaps the most important feature of our method is that it permits us to process incompletely specified multiple-output functions specified by sets of *on* and *off* cubes, as a single n-input, m-output object, for which the result is represented in the same form. Although existing methods [14, 15, 17, 20] deal with multiple-output functions, they do not process them as single objects represented by an Espresso fr matrix. Also, they do not find nondisjunctive decompositions. Our representation of Boolean functions allows for the same treatment of disjunctive and non-disjunctive decompositions.

A comparison of the results achieved by DEMAIN with the other published results shows that the effectiveness of the proposed method does not suffer because of its universality; in fact, it provides better solutions in many cases [12, 13].

Recently, several experiments were performed using the MCNC benchmark circuits to compare the results produced by DEMAIN with those produced by ALTERA's MAX+Plus2 software. For example, the benchmark circuit rd84 was initially compiled directly using ALTERA's compiler. ALTERA's dedicated compiler reported that 147 logic cells were required to implement the benchmark. DEMAIN decomposes rd84 into a netlist of cells, where the cells have four inputs and one output. The decomposed multi-level structure of rd84 was then supplied as input to the MAX+Plus2 compiler; the result was 15 cells. In general, we have an average gain of about 78% over Max+Plus2 on the MCNC benchmarks.

Acknowledgment The authors thank C.-J. Shi of the University of Iowa for his many useful suggestions concerning this work, and Radu Negulescu of the University of Waterloo for his careful reading of the manuscript.

References

[1] R. L. Ashenhurst, The Decomposition of Switching Functions, *Proc. of International Symp. Theory of Switching Functions*, 1959.

[2] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and U. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis,* Kluwer Academic Publishers, Boston, MA, 1984.

[3] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, MIS: Multiple-Level Logic Minimization System, *IEEE Trans. on CAD,* Vol. CAD-6, No. 6, pp. 1062-1081, November 1987.

[4] D. Brown, *Field Programmable Gate Arrays,* Kluwer Academic Publishers, Boston, MA, 1992.

[5] J. A. Brzozowski and C-J. Seger, *Asynchronous Circuits,* Springer-Verlag, New York, NY, 1995.

[6] M. J. Ciesielski and S. Yang, PLADE: A Two-Stage PLA Decomposition, *IEEE Trans. on CAD,* Vol. 11, No. 8, August 1992.

[7] H. A. Curtis, *A New Approach to the Design of Switching Circuits,* D. Van Nostrand Co. Inc., Princeton, NJ, 1962.

[8] J. Hartmanis and R. E. Stearns, *Algebraic Structure Theory of Sequential Machines,* Prentice-Hall, Englewood Cliffs, NJ, 1966.

[9] L. Jóźwiak, General Decomposition and its Use in Digital Circuit Synthesis, *VLSI Design,* Vol. 3, Nos. 3–4, pp. 225–248, 1995.

[10] Z. Kohavi, *Switching and Finite Automata Theory*, McGraw-Hill, New York, 1978.

[11] Y. T. Lai, M. Pedram, and B. K. Vrudhula, EVBDD-Based Algorithms for Integer Linear Programming, Spectral Transformation, and Function Decomposition, *IEEE Trans. on Computer Aided Design,* Vol. 13, No. 8, pp. 959-975, 1994.

[12] T. Łuba and H. Selvaraj, A General Approach to Boolean Function Decomposition and its Application in FPGA-Based Synthesis, *VLSI Design*, Vol. 3, Nos. 3–4, pp. 289–300, 1995.

[13] T. Łuba, H. Selvaraj, M. Nowicka, and A. Krasniewski, Balanced Multi-Level Decomposition and its Applications in FPGA-Based Synthesis, in G. Saucier and A. Mignotte, eds., *Logic and Architecture Synthesis*, pp. 109–115, Chapman and Hall, 1995.

[14] R. Murgai, N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, Improved Logic Synthesis Algorithm for Table Look Up Architectures, *Proc. IEEE International Conf. on Computer Aided Design,* pp. 564–567, 1991.

[15] J. P. Roth and R. M. Karp, Minimization over Boolean Graphs, *IBM Journal of Research and Development,* Vol. 6, pp. 227–238, April 1962.

[16] A. Saldanha, T. Villa, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, Satisfaction of Input and Output Encoding Constraints, *IEEE Trans. on Computer Aided Design,*, Vol. 13, No. 5, pp. 589–602, May 1994.

[17] A. Sangiovanni-Vincentelli, A. Gamal, and J. Rose, Synthesis Methods for Field Programmable Gate Arrays, *Proc. IEEE,* Vol. 81, No. 7, pp. 1057–1083, 1993.

[18] P. Sapiecha, M. Perkowski, and T. Luba, Decomposition of Information Systems Based on Coloring Heuristics, *Proc. Symposium on Modelling, Analysis and Simulation, CESA'96 IMACS Multiconference. Computational Engineering in Systems Applications* , pp. 1101-1106, Lille, France, Gerf EC Lille - Cite Scientifique 1996.

[19] T. Sasao, Logic Synthesis and Optimization, *Kluwer Academic Publishers,* 1993.

[20] T. Stanion and C. Sechen, A Method for Finding Good Ashenhurst Decomposition and Its Application to FPGA Synthesis, *32 Design Automation Conference,* pp. 60–64, San Francisco, 1995.

[21] W. A. Shen, J. D. Huang, and S. M. Chao, Lambda Set Selection in Roth-Karp Decomposition for LUT-Based FPGA Technology Mapping, *32 Design Automation Conference,* pp. 65–69, San Francisco, 1995.

[22] M. Sysło, N. Deo, and J. Kowalik, *Discrete Optimization Algorithms*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1983.

[23] W. Wan and M. A. Perkowski, A New Approach to the Decomposition of Incompletely Specified Multi-Output Function Based on Graph Coloring and Local Transformations and Its Application to FPGA Mapping, *Proc. European Design Automation Conf.,* pp. 230–235, 1992.