Decomposition of Multi-Valued Functions

by

J. J. Lou

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 1998

I hereby declare that I am the sole author of this thesis.

I authorize the University of Waterloo to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the University of Waterloo to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

The University of Waterloo requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Abstract

Decomposition of functions has been studied since the late 1950s; it was mainly used in minimizing circuit designs for switching functions. Recently, decompositions have been used in FPGA synthesis and other applications, and interest in this topic has again increased. A commonly used method decomposes a function $f(x_1, \ldots, x_n)$ into the form $h(x_1, \ldots, x_r, g(x_{r+1}, \ldots, x_n))$, where both g and h are switching functions. In this thesis, we study the decomposition of a very general form of functions, called *set functions*, using this approach. Set functions are multi-valued functions where the outputs can take the value of any non-empty subset of an underlying set E. We represent these functions in a compact cube notation using *consistent set matrices*. A decomposition theory of such functions is presented using *blankets*, which are a generalization of partitions and set systems. We also give a survey of different decomposition methods, including the formalizations of some of these methods using blankets.

Acknowledgements

First, I would like to thank my supervisor Prof. John Brzozowski. Without our discussions on the topic and his constructive comments, this thesis would not have been possible. I would also like to thank him for his patience, guidance and encouragement.

I would like to thank Prof. Fahiem Bacchus and Prof. Grant Weddell for reading my thesis and providing me with many comments. I would also like to thank Prof. Łuba, Prof. Shestakov, and Prof. Perkowski, who provided me insights into the topic of function decomposition.

I thank my family and my friends for believing in me and providing me with encouragement.

Finally, I would like to acknowledge the financial support from the Natural Sciences and Engineering Research Council of Canada through a postgraduate scholarship and through a research assistantship under grant OGP 0000871.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 History of Decomposition

Decomposition of functions has been studied since the late 1950s. In 1959, Ashenhurst published his paper [1] on the so-called disjunctive decomposition of switching functions. His method takes a completely-specified, single-output Boolean function $f(x_1, \ldots, x_n)$ and expresses it as $h(x_1, \ldots, x_r, g(x_{r+1}, \ldots, x_n))$. The methods we will look at generally follow this approach. Ashenhurst used pre-generated charts to perform the decompositions. The size of these charts increases exponentially with the number of inputs n. Hence it is only practical to use this method when n is small.

Curtis [7] later extended Ashenhurst's method to functions with don't care values. He also introduced non-disjunctive decomposition, where the input variables of g and h overlap. Curtis also presented a general theory of decomposition, combining Ashenhurst's work with his own. This theory enables us to enumerate all possible decompositions. However, Curtis still used charts.

Roth and Karp [18] used a more compact notation called cubes to represent switching functions. With this representation, they developed a decomposition algorithm for minimizing the resulting circuit. This algorithm, however, becomes very time consuming when the number of inputs get larger as it attempts to look at all possible decompositions. The efficiency can be improved if we simply try to find a good solution instead of a minimal one. Roth and Karp's method, like Ashenhurst's and Curtis', doesn't deal directly with multiple-output functions. Such functions are merely treated as multiple functions each with a single output.

More recently, Łuba and Selvaraj [16] proposed a decomposition method for Boolean functions using partitions, set systems and their generalizations. Their method has been formalized by Brzozowski and Łuba [5], who introduced the term "blankets" for generalized set systems. A similar method was discovered independently by Shestakov [21, 22], who used Boolean expressions instead of blankets. He also used a tabular method in deriving the decompositions. Lou and Brzozowski [14] formalized Shestakov's method and explained its relation to Łuba and Selvaraj's method.

Decomposition methods of multi-valued functions and relations have been studied in [4, 15, 17]. In [4], the term "set function" is introduced. Roughly speaking, a set function is a multi-valued function where a variable can take any value that is a subset of its domain. For example, for a domain of $\{0, 1, 2\}$, a variable may then take the value $\{0, 2\}$.

Other methods of decomposition have also been proposed. For example, Lai, Pedram and Vrudhula described a decomposition method using binary decision diagrams in [13]. Their method is improved by Stanion and Sechen [24]. Files, Drechsler and Perkowski [8] extended this method to deal with multi-valued functions. Using this approach, a multiple-output function is treated as several single-output functions. To improve the decomposition results, algorithms are used to identify common sub-functions [12, 20].

More details of the above methods will be given in later chapters, after we establish a uniform notation in Chapter 2.

1.2 Applications of Decomposition

Originally, decompositions were used to design economical gate-type combinational switching circuits. For example, with Roth and Karp's minimization algorithm [18], we can, in theory, check all possible implementations of a function and thus are able to find a minimal circuit design.

With the introduction of field programmable gate arrays by Xilinx in 1986, there is a renewed interest in decomposition in recent years. FPGAs are modern logic devices which can be programmed by the users to implement any logic circuit [3]. One of the most popular FPGA architectures is based on look-up tables. It consists of many identical logic blocks, each of which can implement any Boolean functions of k inputs, where k is a small integer. Consequently, to implement a function with large number of inputs, we must decompose it into functions with fewer inputs.

The applications of decomposition are not limited to circuit design. This is especially true when we extend the methods to multi-valued functions; then they also have applications in database management, knowledge discovery, machine learning and pattern recognition. We illustrate some of these applications with the following example.

Example 1.2.1 *Table 1.1 shows how an insurance company assigns car insurance policies for various customers. There are four attributes:*

- *Age: 24 and under* (24−) *or 25 and over (25+);*
- *Gender: Male or Female ;*
- *Marital Status: Single or Married ;*
- *Use of the Car: Business, Pleasure or Farm .*

We use a compact notation where each of the attributes can take any subset of the set of its possible values. A policy numbered 1 through 5 is determined on the basis of the four attributes. For example, Row 1 shows that, for any married female or male customer over the age of 25 who uses the car for a farm or a business, the policy number is 1.

Table 1.1: Car Insurance Policies

	Age	*Gender*	*Marital Status*	*Use*	*Policy*
1	*25+*	*Female, Male*	*Married*	*Farm, Business*	*1*
2	*25+*	*Female*	*Single*	*Farm, Business*	*1*
3	*25+*	*Female, Male*	*Married*	*Pleasure*	*2*
4	*25+*	*Female*	*Single*	*Pleasure*	*2*
5	*25+*	*Male*	*Single*	*Farm*	*2*
6	*25+*	*Male*	*Single*	*Business*	*3*
7	*25+*	*Male*	*Single*	*Pleasure*	*4*
8	*24-*	*Female, Male*	*Single, Married*	*Farm*	*2*
9	*24-*	*Female, Male*	*Single, Married*	*Business*	*3*
10	*24-*	*Female, Male*	*Married*	*Pleasure*	*4*
11	*24-*	*Female, Male*	*Single*	*Pleasure*	*5*

Note that this is not a realistic example, as real car-insurance policies would most likely also depend on car type, owner's driving history, etc. We omit these to keep the example small and use it just as an illustration.

Consider Tables 1.2 and 1.3, which constitute a decomposition of Table 1.1. Here "Person" is a new intermediate attribute, which combines the original attributes of Age, Gender and Marital Status. Take, for example, a customer with the attributes (25+, Female, Married, Business). We look up the first three attributes (25+, Female and Married) in Table 1.2 and get 0 for the value of Person; then we look up (0, Business) in Table 1.3, this gives us policy No. 1. This is the same policy number we get from Table 1.1. We can check that we get the same policy number for all customers whether we are using Table 1.1 or Tables 1.2 and 1.3.

Table 1.2: Personal Information

	Age	*Gender*	*Marital Status*	*Person*
1	*25+*	*Female, Male*	*Married*	*0*
2	*25+*	*Female*	*Single*	*0*
3	*25+*	*Male*	*Single*	*1*
4	*24-*	*Female, Male*	*Married*	*1*
5	*24-*	*Female, Male*	*Single*	*2*

Having two smaller tables instead of a big one can save both space and search time. With the original table small in this case, the savings are marginal, but can be significant for larger tables.

Suppose Row 4 is not present in Table 1.1. Tables 1.2 and 1.3 still constitute a decomposition of our new table. However, in this case, we do not get our original table by combining the two smaller tables. In fact we get one more row, namely

Table 1.3: Car-insurance Policies with Intermediate Value

	Person	*Use*	*Policy*
1	*0*	*Farm, Business*	*1*
2	*0*	*Pleasure*	*2*
3	*1,2*	*Farm*	*2*
4	*1,2*	*Business*	*3*
5	*1*	*Pleasure*	*4*
6	*2*	*Pleasure*	*5*

Row 4 of Table 1.1. This may not be desirable in our case. In a different application, this kind of "generalization" may be good, since often it's not possible to have complete information. Two such applications are machine learning and knowledge discovery.

□

1.3 Contents of Thesis

This thesis is divided into five chapters. The present chapter gave a brief history and some applications of function decompositions. Chapter 2 defines set functions and other concepts used in the decomposition of these functions; it also includes the main theorem on decomposition. In addition, this chapter also discusses the cube-method which has a more compact representation. Chapter 3 presents a formalization of Shestakov's method, in particular his tabular method. This method is very similar to the method in Chapter 2, but not identical. It enables us to find more decompositions. Chapter 4 consists of a survey of various decomposition methods. This includes Ashenhurst's chart method, Curtis' extension of that method, Roth and Karp's minimization algorithm, and the binary decision diagram method proposed by Lai, Pedram and Vrudhula. Chapter 5 contains some concluding remarks.

The method presented in Chapter 2 is based on a paper [4] co-authored with Prof. Brzozowski, which in turn is a generalization of the results by Brzozowski and Łuba [5]. The formalization in Chapter 3 is based on another paper [14] co-authored with Prof. Brzozowski, which is based on the works of Shestakov [21, 22].

Chapter 2

Decomposition Using Blankets

The following conventions will be used in this chapter and all subsequent chapters unless otherwise noted.

Ordered n-tuples are denoted by unsubscripted lower-case letters and their components by subscripted letters, for example, $x = (x_1, \ldots, x_n)$. If $s = (s_1, \ldots, s_n)$ is an ordered n-tuple, then the corresponding set, $\{s_1, \ldots, s_n\}$, is denoted by S. Also, if $S = \{s_1, \ldots, s_n\}$ is a set of elements explicitly represented in that order, then $s = (s_1, \ldots, s_n)$ is the corresponding n-tuple.

2.1 Set Functions

Let E be a finite, nonempty set, and let $D = 2^E - \{\emptyset\}$. Let $\Sigma \subseteq D$ be the set of singleton subsets of E, that is, $\Sigma = \{\{e\} \mid e \in E\}$.

An element of the set D denotes the possible values a variable x can take. If $x = \{e\} \in \Sigma$, then x can only take the value e. If $x = E$, then x can take any value in the set E, and it is treated as a "complete don't care". If $x = S$ where S

is a proper, non-singleton subset of E, then x is viewed as a "limited don't care" value. For example, let $E = \{0, 1, 2\}$; then x can take on singleton set values $\{0\}, \{1\}$, or $\{2\}$ or it can be specified as a complete don't care $\{0, 1, 2\}$ or it may be one of the limited don't cares $\{0, 1\}$, $\{0, 2\}$, and $\{1, 2\}$.

The set D is partially ordered by set inclusion. If the intersection of a set of elements of D is nonempty, then that intersection is the greatest lower bound *glb* of that set of elements.

For $k \geq 1$, an element of D^k will be called a *set vector*. The element $(E, \ldots, E)$ (k Es) of D^k is called the "complete don't care" value. An element in the subset Σ^k of D^k is referred to as a *singleton vector* or, by a slight abuse of terminology, a *minterm*. All other elements of D^k are viewed as 'limited don't care" values. For clarity, we often write set vectors without parentheses or commas. For example, $(\{0, 2\}, \{2\}, \{0, 1\})$ will be written $\{0, 2\}\{2\}\{0, 1\}$. In the case of minterms, we may even omit the $\{\}$ if it does not cause confusion. For example, $(\{0\}, \{1\}, \{2\})$ may be written as 012.

The partial order defined on D is extended to D^k as follows: For $t, t' \in D^k$,

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

For example,

$$\{0, 2\}\{2\}\{0, 1\} \subseteq \{0, 2\}\{1, 2\}\{0, 1, 2\}.$$

The *intersection* of two elements t and t' from D^k, denoted by $t \cap t'$, is defined to be empty if $t_i \cap t'_i$ is empty for some i, $1 \leq i \leq k$; otherwise, $t \cap t'$ is defined to be $(t_1 \cap t'_1, \ldots, t_k \cap t'_k)$. For example

$$\{0, 2\}\{2\}\{0, 1\} \cap \{0, 2\}\{1, 2\}\{0, 1, 2\} = \{0, 2\}\{2\}\{0, 1\},$$

while

$$\{0,2\}\{2\}\{0,1\} \cap \{1\}\{1,2\}\{0,1,2\} = \emptyset.$$

A subset T of D^k is said to be *compatible* if $\bigcap_{t \in T} t \neq \emptyset$. If T is compatible then the greatest lower bound of T exists and is equal to $\bigcap_{t \in T} t$. We denote the *glb* of T as shown below to stress the fact that the *glb* is equal to the intersection only if the intersection is nonempty.

$$glb\, T = \bigsqcap_{t \in T} t.$$

A *set function* is a function $f : \Sigma^n \rightarrow D^m$. The set of input variables of f is denoted by X while the set of output variables of f is denoted by Y.

The set functions we consider are represented by *set matrices.* A *set matrix* is a matrix with entries from D. Let M be a $\rho \times (n+m)$ set matrix. The rows of M are numbered $1, \ldots, \rho$, and the set of rows of M is denoted by F. An element t of F is referred to as a *cube.* The vector consisting of the first n components of t, denoted by t^X, is called an *input cube*; while the vector consists of the last m components of t, denoted by t^Y, is called an *output cube.*

A set matrix M is said to be *consistent*, if, for any set T of rows from F,

$$\text{if } \{t^X \mid t \in T\} \text{ is compatible, then } \{t^Y \mid t \in T\} \text{ is compatible.}$$

From now on we only consider consistent set matrices.

The interpretation of a consistent set matrix M is as follows: For $e \in \Sigma^n$, define

$$f(e) = \bigsqcap_{F_{X \supseteq e}} t^Y,$$

where $F_{X \supseteq e} = \{t \in F \mid t^X \supseteq e\}$, and, if $F_{X \supseteq e}$ is empty, $f(e)$ is defined to be $(E, \ldots, E)$ (m Es). In other words, the *glb* of the empty set of vectors is interpreted as the complete don't care. Function f is well defined because of the consistent condition on M.

By convention, to simplify matrix M we leave out any row which has the complete don't care as its output.

A minterm $e \in \Sigma^n$ is said to be *involved* in a function cube $t \in F$, if $t^X \supseteq e$. We say e is *relevant to* f if e is involved in some vector $t \in F$. Thus, for our simplified matrix M, if e is relevant to f, then $f(e)$ is not the complete don't care.

Table 2.1: Set Matrix M

Row	x_1	x_2	x_3	y_1	y_2
1	0	0	0, 2	0, 1	0
2	0, 1	1, 2	1	0, 2	1
3	2	0, 1, 2	0	1, 2	2
4	1, 2	0, 2	0, 2	0, 1	1, 2

Example 2.1.1 *In Table 2.1 we illustrate a set matrix M with the underlying set $E = \{0, 1, 2\}$. For clarity, we omit $\{\}$ for each entry of the matrix. A numbering of the rows is shown, while the columns are labeled by the input variable x_1, x_2, x_3 and output variables y_1, y_2. The set function f defined by M is listed in Table 2.2. Note that we can consider this table to be another set matrix, although the input cubes of this matrix are minterms. To illustrate the construction of Table 2.2, we show the calculations of three rows. The minterm $e = \{0\}\{0\}\{0\}$ is involved only in Row 1; hence $f(e)$ has the value $\{0, 1\}\{0\}$ as specified in Row 1. The minterm $e' = \{0\}\{0\}\{1\}$ is not involved in any row; hence it is absent from the table, meaning that the value of $f(e')$ is the complete don't care $\{0, 1, 2\}\{0, 1, 2\}$.*

The minterm $e'' = \{2\}\{2\}\{0\}$ *is involved in Rows 3 and 4; hence the value of* $f(e'')$ *is the glb of the values of* f *specified in Rows 3 and 4, i.e.,* $f(e'') = \{1\}\{2\}$.

□

Table 2.2: Set Function f

x_1	x_2	x_3	y_1	y_2
0	0	0	0, 1	0
0	0	2	0, 1	0
0	1	1	0, 2	1
0	2	1	0, 2	1
1	0	0	0, 1	1, 2
1	0	2	0, 1	1, 2
1	1	1	0, 2	1
1	2	0	0, 1	1, 2
1	2	1	0, 2	1
1	2	2	0, 1	1, 2
2	0	0	1	2
2	0	2	0, 1	1, 2
2	1	0	1, 2	2
2	2	0	1	2
2	2	2	0, 1	1, 2

2.2 Blankets and Set Systems

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

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

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

The *product* $\beta{*}\beta'$ of two blankets is

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

For example,

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

this shows that blanket product is not idempotent. From the definition of product, we can see that this operation is both commutative and associative.

If β and β' are blankets on S, we write $\beta \leq \beta'$ if for each B_i in β there exists a B_j in β' such that $B_i \subseteq B_j$. For example, $\{\overline{1};\ \overline{2};\ \overline{3}\} \leq \{\overline{1,2};\ \overline{2,3}\}$. The relation $\leq$ is reflexive and transitive, but not antisymmetric, since $\{\overline{1};\ \overline{1,2}\} \leq \{\overline{1,2}\}$ and $\{\overline{1,2}\} \leq \{\overline{1};\ \overline{1,2}\}$, but $\{\overline{1,2}\} \neq \{\overline{1};\ \overline{1,2}\}$. Also, we can deduce the following properties from the definitions:

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

A *set system* [10] 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 $i = j$. Blankets can be mapped to set systems by using the *max* operator defined by

$$max\,\beta = \beta - \{B \in \beta \mid B \subset B' \text{ for some } B' \in \beta\}$$

We can extend the $*$ operation defined on blankets to set systems. However, as we have seen above, $\{\overline{1,2,3};\ \overline{3,4,5}\}*\{\overline{1,2,3};\ \overline{3,4,5}\}$ is not a set system. So set systems are not closed under $*$ operation. The *product* $\sigma \circ \sigma'$ of two set systems is defined as follows:

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

This product operation is idempotent.

We can define $\leq$ relation on set systems similar to that on blankets. In this case, $\leq$ is a partial order.

2.3 Blankets on Set Functions

Let f be a set function represented by a set matrix M. We will be dealing with blankets on F, the set of rows of M. However, for simplicity, we represent each element of F by its corresponding row number $t \in \{1, \ldots, \rho\}$.

Suppose $W = \{x_{i_1}, \ldots, x_{i_r}\}$ is a subset of the set X of input variables, and t is a set vector in F. We denote by t^W the r-tuple $(t_{i_1}, \ldots, t_{i_r})$. Similarly, for a minterm $e \in \Sigma^n$ relevant to f, we denote by e^W the r-tuple $(e_{i_1}, \ldots, e_{i_r})$. Define an input blanket β_W as follows: $\beta_W = ne\{F_{W \supseteq e^W} \mid e \in \Sigma^n\}$,, and

$$F_{W \supseteq e^W} = \{t \in F \mid t^W \supseteq e^W\}.$$

We also define the output blanket β_Y of f as $\beta_Y = ne\{F_{Y \supseteq e}\}$, where $e \in \Sigma^m$ and

$$F_{Y \supseteq e} = \{t \in F \mid t^Y \supseteq e\}.$$

For the example shown in Table 2.1, suppose $W = \{x_2, x_3\}$. Then we have, for example, $F_{W \supseteq 00} = \overline{1,3,4}$ and $F_{W \supseteq 01} = \emptyset$. Complete this calculation and we get

$$\beta_W = \{\overset{B_1}{\overline{1,3,4}};\ \overset{B_2}{\overline{1,4}};\ \overset{B_3}{\overline{3}};\ \overset{B_4}{\overline{2}};\ \overset{B_5}{\overline{3,4}};\ \overset{B_6}{\overline{4}}\},$$

where the blocks have been numbered as shown.

We calculate the output blanket β_Y similarly and we get

$$\beta_Y = \{\overset{B_1'}{\overline{1}};\ \overset{B_2'}{\overline{2,4}};\ \overset{B_3'}{\overline{4}};\ \overset{B_4'}{\overline{3,4}};\ \overset{B_5'}{\overline{2}};\ \overset{B_6'}{\overline{3}}\}.$$

Suppose B is a block of β_X and $B = \{t_1, \ldots, t_k\}$. Then by definition of input blankets, $\{t_i^X\}$ is compatible. Thus, by the consistency condition, we have $\{t_i^Y\}$ is compatible. This means $\{t_i^Y\}$ is contained in some block B' of β_Y. Therefore, we have $\beta_X \leq \beta_Y$.

The *representative* r_B of block B of an input blanket β_W is the *glb* of the cubes in B. We say a minterm $b \in \Sigma^r$ *exemplifies* block B of an input blanket β_W if $B = F_{W \supseteq b}$; the set of all such minterms is denoted $\chi(B)$. By definition, $B = F_{W \supseteq e^W}$ for some $e \in \Sigma^n$, hence $e^W \in \chi(B)$ and $\chi(B)$ is non-empty. For the example above, we have $r_{B_2} = \{0\}\{0,2\}$ and $r_{B_4} = \{1,2\}\{1\}$, while $\chi(B_2) = \{02\}$ and $\chi(B_4) = \{11, 21\}$.

Similarly, we can also define the representative for a block B_j' of the output blanket β_Y and $\chi(B_j')$, the set of minterms that exemplify B_j'. For the above example, we have $r_{B_3'} = \{0,1\}\{1,2\}$, while $\chi(B_3') = \{02, 11\}$.

We now prove several properties of input and output blankets of a set function.

Proposition 2.3.1 *Let W and W' be two subsets of the input variable set X. Then we have the following:*

a) If $W \supseteq W'$, then $\beta_W \leq \beta_{W'}$;

b) If $V = W \cup W'$, then $\beta_W * \beta_{W'} \leq \beta_V$;

c) If $W \cap W' = \emptyset$ and $V = W \cup W'$, then $\beta_W * \beta_{W'} = \beta_V$;

d) If $W \cup W' = X$, then $\beta_W * \beta_{W'} \leq \beta_Y$.

Proof. Let B be a block of β_W, and let e be a minterm relevant to f such that e^W exemplifies B. Let $\hat{B} = F_{W' \supseteq e^{W'}}$, then $\hat{B}$ is a block of $\beta_{W'}$. Suppose $t \in B$, then $t^W \supseteq e^W$. Since $W \supseteq W'$, we get $t^{W'} \supseteq e^{W'}$. Thus, we have $t \in \hat{B}$. This proves $B \subseteq \hat{B}$, which, in turn, implies a).

Let $U = W' - W$, then $W' \supseteq U$, and $V = W \cup U$. By part a), we get $\beta_{W'} \leq \beta_U$. Thus, to prove b), it is sufficient to show $\beta_W * \beta_U \leq \beta_V$. Suppose $B \in \beta_W * \beta_U$. Then we have $B = B_W \cap B_U$ for some blocks $B_W \in \beta_W$ and $B_U \in \beta_U$. Suppose $B_W = F_{W \supseteq b}$ and $B_U = F_{U \supseteq c}$. Take any cube $t \in B$. We have $t^W \supseteq b$ and $t^U \supseteq c$. Since W and U are disjoint, we get $t^V \supseteq bc$. Thus $F_{V \supseteq bc}$ is not empty and hence is a block of β_V. Since $t \in F_{V \supseteq bc}$, we get $B \subseteq F_{V \supseteq bc}$ and our result follows.

Now suppose W and W' are disjoint. Consider $B \in \beta_W * \beta_{W'}$. As in the proof of part b), we assume $B = B_W \cap B_{W'}$, where $B_W = F_{W \supseteq b} \in \beta_W$, and $B_{W'} = F_{W \supseteq c} \in \beta_{W'}$. Then $F_{V \supseteq bc}$ is a block of β_V. We claim that $B \leftrightarrow F_{V \supseteq bc}$ is a one-to-one correspondence between the blocks of $\beta_W \times \beta_{W'}$ and β_V. This is true because each block B_V of β_V can be written uniquely in the form $F_{V \supseteq bc}$, and hence corresponds to a unique block B of $\beta_W \times \beta_{W'}$. It remains to show that $B = F_{V \supseteq bc}$. We know that $B \subseteq F_{V \supseteq bc}$ from Part b). Now suppose $t \in F_{V \supseteq bc}$. Then we have $t \in F_{V \supseteq b}$ and $t \in F_{V \supseteq c}$, which, in turn, implies $t \in B$. Therefore, we get $B \supseteq F_{V \supseteq bc}$ as required.

Suppose $W \cup W' = X$. Then, by Part b), we have $\beta_W * \beta_{W'} \leq \beta_X$. We noted earlier that we have $\beta_X \leq \beta_Y$. Thus Part d) follows by the transitivity of $\leq$.

$\square$

Let x_i be an input variable of a set function f, and let $W = X - \{x_i\}$. Variable x_i is said to be *non-essential*, if, for every set of minterms $S \subseteq \Sigma^n$ such that $s^W = s'^W$ for all $s, s' \in S$, $\bigsqcap_{s \in S} f(s)$ is defined. Otherwise, variable x_i is said to be *essential*. A non-essential input variable x_i can be "removed" in the sense that set matrix M', which is matrix M with the ith column removed, defines a set function f_{x_i}, with input variables $\{x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n\}$ where

$$f_{x_i}(e_1, \ldots, e_{i-1}, e_{i+1}, \ldots, e_n) = \bigsqcap_{e_i \in \Sigma} f(e_1, \ldots, e_{i-1}, e_i, e_{i+1}, \ldots, e_n).$$

The fact that x_i is non-essential guarantees that f_{x_i} is a set function as the above *glb* is defined.

Consider set function f defined by Table 2.1. We can see that variable x_1 is essential since minterms 000 and 100 have the same x_2 and x_3 components, but $f(000)$ and $f(100)$ are not compatible. On the other hand, both x_2 and x_3 are non-essential. We can check if an input variable is essential using blankets as shown in following proposition.

Proposition 2.3.2 *Let $x_i \in X$ be an input variable, and let $W = X - \{x_i\}$. Then x_i is non-essential if and only if $\beta_W \leq \beta_Y$.*

Proof. Suppose $\beta_W \leq \beta_Y$. Let $S \in \Sigma^n$ be a set of minterms such that $s^W = s'^W$ for all $s, s' \in S$. Let $B = F_{W \supseteq s^W}$, where $s \in S$; then B is a block of β_W. By our assumption, we have $B \subseteq B_Y$, where $B_Y = F_{Y \supseteq e}$ is a block of β_Y for some minterm e. Since $F_{X \supseteq s} \subseteq B$, we get

$$f(s) = \bigsqcap_{F_{X \supseteq s}} t^Y \supseteq \bigsqcap_{t \in B} t^Y \supseteq \bigsqcap_{t \in B_Y} t^Y \supseteq e.$$

Thus, $\bigsqcap_{s \in S} f(s)$ is defined as s is any element of S. By definition, x_i is non-essential.

Now suppose x_i is non-essential. We want to show $\beta_W \leq \beta_Y$. Let $B = F_{W \supseteq e^W} \in \beta_W$, where e is a minterm relevant to f. Take $t \in B$. Then there exists a minterm b involved in t such that $b^W = e^W$. Also, by definition of $f(b)$, we see that $t^Y \supseteq f(b)$. Let $y = \bigsqcap_{c^W = e^W} f(c)$; then we have $f(b) \supseteq y$. Thus, $t \in F_{Y \supseteq y}$, where $F_{Y \supseteq y}$ is a block of β_Y. Since this is true for all $t \in B$, we get $B \subseteq F_{Y \supseteq y}$ and our result follows.

□

As the definition of non-essential variable indicates, for any minterm e, we have $f(e) \supseteq f_{x_i}(e^W)$. This means f and f_{x_i} do not give exactly the same output with the corresponding inputs. As we shall see later, this is acceptable in our definition of decomposition. In the case where the outputs of f do not contain don't cares, equality is achieved.

A subset W' of X is said to be *non-essential* if, for every set of minterms $S \subseteq \Sigma^n$ such that $s^W = s'^W$ where $s, s' \in S$ and $W = X - W'$, $\bigsqcap_{s \in S} f(s)$ is defined. Clearly, a set containing one non-essential variable is a non-essential subset. However, the union of such sets are not necessarily non-essential. For example, we know that x_2 and x_3 are non-essential inputs of function defined by Table 2.2. But the subset $\{x_2, x_3\}$ of X is not non-essential as minterms 000 and 011 fail our condition. We can use the following proposition to find non-essential subsets. The proof is similar to that of Proposition 2.3.2.

Proposition 2.3.3 *Let W' be a subset of X, and let $W = X - W'$. Then W' is non-essential if and only if $\beta_W \leq \beta_Y$.*

2.4 Serial Separations of Set Functions

Let $X = \{x_1, \ldots, x_n\}$ be the set of input variables of a set function $f : \Sigma^n \to D^m$. Let U and V be two subsets of X such that $U \cup V = X$. Without loss of generality, we relabel the variables $x_1, ..., x_n$ so that $U = \{x_1, ..., x_r\}$ and $V = \{x_{n-s+1}, ..., x_n\}$. Consequently, for an n-tuple x, the first r components are denoted by x^U, and the last s components, by x^V. The output variables of f constitutes the set $Y = \{y_1, \ldots, y_m\}$. The set of rows of f is denoted by F.

Definition 2.4.1 *Let g be a set function with $s \geq n - r$ inputs and p outputs, and let h be a set function with $r + p$ inputs and $m > 0$ outputs. The pair (g, h) is a* serial separation *of f* with respect to *(U, V), if, for every minterm $e \in \Sigma^n$ relevant to f, $g(e^V)$ is a minterm in Σ^p, and*

$$f(e) \supseteq h(e^U, g(e^V)). \tag{2.1}$$

A serial separation is called disjunctive *if $U \cap V = \emptyset$, and* non-disjunctive *otherwise.*

In the following, we will denote the set of outputs of g by Z. Note that there is no restriction on the domain of each output z_i of g. Thus, given a serial separation (g, h), we can find a corresponding separation (g', h'), where g' has only one output by encoding the outputs of g with a single variable over a large enough domain. It may be desirable, however, to limit the size of the domain for z_i. We will assume that the domain of each z_i is D.

Example 2.4.2 *Consider set function f specified by Table 2.3. This function has five inputs and two outputs, and the underlying set is $E = \{0, 1, 2\}$. Let $U = \{x_1, x_2, x_3\}$ and $V = \{x_3, x_4, x_5\}$. Function g of Table 2.4 and function h*

Table 2.3: Set function f

Row	x_1	x_2	x_3	x_4	x_5	y_1	y_2
1	1	0, 1	0	0	0, 1	1	0
2	1, 2	0, 1	0	0, 1, 2	2	0	1, 2
3	0, 1	0	0	0, 1	0	1, 2	0
4	2	0, 2	1, 2	2	2	0, 1	2
5	0, 1	0	0, 2	1	1	2	1, 2
6	0	0	0, 1	0	2	1	1
7	0, 1, 2	1	0, 2	0	2	0	0, 1
8	0	2	0	0	2	1	2

of Table 2.5 constitute a non-disjunctive serial separation of f. To verify this, first we check that g and h are set functions, i.e., the two set matrices are consistent. Next, for each minterm $e \in \Sigma^n$, $g(e^V)$ is a minterm. Thus, it remains to verify (2.1). One relevant input vector is $(x_1, x_2, x_3, x_4, x_5) = 00000$, and we have $f(00000) = \{1,2\}\{0\}$. We find $g(000) = 0$ from Table 2.4, and $h(0000) = \{1\}\{0\}$ from Table 2.5. Here (2.1) is satisfied with the inclusion being proper. For $(x_1, x_2, x_3, x_4, x_5) = 00002$, we find $f(00002) = \{1\}\{1\}$, $g(002) = 1$, and $h(0001) = \{1\}\{1\}$; thus (2.1) is again satisfied, and this time we have equality. The rest of the entries are verified similarly.

$\square$

Definition 2.4.3 *Under the conditions stated earlier, let W be a subset of X with k elements. Let g be a set function with k inputs. Define $\gamma_g = ne\{F_{g=a}\}$, where*

$$F_{g=a} = \{t \in F \mid \exists b \in \Sigma^k \text{ such that } t^W \supseteq b, \text{ and } g(b) = a\}.$$

Table 2.4: Function g

x_3	x_4	x_5	z_1
0	0	$0,1$	0
0	$0,1,2$	2	1
0	$0,1$	0	0
$1,2$	2	2	0
$0,2$	1	1	2
$0,1,2$	0	2	1

Table 2.5: Function h

x_1	x_2	x_3	z_1	y_1	y_2
1	$0,1$	0	0	1	0
$1,2$	$0,1$	0	1	0	$1,2$
$0,1$	0	0	0	1	0
2	$0,2$	$1,2$	0	$0,1$	2
$0,1$	0	$0,2$	2	2	$1,2$
0	0	$0,1$	1	1	1
$0,1,2$	1	$0,2$	1	0	$0,1$
0	2	0	1	1	2

We say a blanket β on F and g correspond with respect to W *if* $\gamma_g \leq \beta$.

Each block of the blanket γ_g consists of rows in F that contain minterms which are mapped to the same value by function g. For example, for function g specified by Table 2.4 and $W = \{x_3, x_4, x_5\}$, we have

$$\gamma_g = \{F_{g=0}, F_{g=1}, F_{g=2}\} = \{\overline{1,3,4};\ \overline{2,6,7,8};\ \overline{5}\}.$$

Now we prove a sufficient condition for the existence of a serial separation. The converse of Theorem 2.4.4 is not always true. We shall prove a restricted form of the converse in Theorem 2.6.1.

Theorem 2.4.4 *Let $f(x)$ be a set function specified by a set F of set vectors, and let (U, V) be a pair of subsets of X satisfying $U \cup V = X$. For every blanket β_g satisfying the conditions stated below, there is a serial separation (g, h) of f such that β_g and g correspond with respect to V.*

- $\beta_V \leq \beta_g$, *and*
- $\beta_U * \beta_g \leq \beta_Y$.

Proof. Suppose β_g satisfying the conditions of the theorem is given and has q blocks. We encode the blocks of β_g uniquely by assigning to each block B a minterm $\alpha(B)$ in Σ^p, where $p = \lceil \log_k q \rceil$, $k = |E|$. Note α is a one-to-one function.

Let e be a minterm in Σ^n relevant to f, and suppose e^V exemplifies block B of β_V. By the first condition of the theorem, block B is contained in some block B_g of β_g. Assign to $g(e^V)$ the value $\alpha(B_g)$. In this manner we define function g.

Next we construct function h. For each minterm $e \in \Sigma^n$ relevant to f, we find $z = g(e^V)$. Minterm $a = (e^U, g(e^V))$ is a possible input vector for h. The output

$h(e^U, g(e^V))$ is defined as follows. Suppose e^U exemplifies block B' of β_U, and e^V exemplifies block B of β_V. Let $B'' = \alpha^{-1}(g(e^V))$; we know that B'' is a block of β_g since α is one-to-one. Also suppose e is involved in some $t \in F$. Then $t \in B'$ and $t \in B$ by definitions of B' and B, respectively. By the construction of g, we see that $B \subseteq B''$. Thus $B' \cap B''$ is not empty. By the second condition of the theorem, $B' \cap B''$ is contained in some block B_h of β_Y. We assign to $h(e^U, g(e^V))$ the value $\bigsqcap_{t \in B_h} t^Y$.

Take any minterm $e \in \Sigma^n$ relevant to f. We know that by the definition of g, $g(e^V)$ is a minterm in Σ^p. Let B', B'', and B_h be as defined in the construction of h. We have shown that if e is involved in some $t \in F$, then $t \in B_h$. This implies $\hat{B} = F_{X \supseteq e} \subseteq B_h$. Thus, we get

$$f(e) = \bigsqcap_{t \in \hat{B}} t^Y \supseteq \bigsqcap_{t \in B_h} t^Y = h(e^U, g(e^V)).$$

It remains to show that β_g and g correspond with respect to V. Take a block $F_{g=a}$ of γ_g and suppose $t \in F_{g=a}$. By definition of block $F_{g=a}$, we see that $t \in \alpha^{-1}(a)$ which is a block of β_g. Thus $F_{g=a} \subseteq \alpha^{-1}(a)$ which implies $\gamma_g \leq \beta_g$.

$\square$

2.5 Examples of Serial Separation

Example 2.5.1 *Consider function f specified by Table 2.3, with $U = \{x_1, x_2, x_3\}$ and $V = \{x_3, x_4, x_5\}$. Also suppose that we are given blanket*

$$\beta_{g'} = \{\overline{1,3,4};\ \overline{2,6,7,8};\ \overline{4,5}\}.$$

Note that for this function we have

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

Table 2.6: Finding function g'

x_3	x_4	x_5	β_V	$\beta_{g'}$	options	z_1
0	0	0	$\overline{1,3}$	$\overline{1,3,4}$	0	0
0	0	1	$\overline{1}$	$\overline{1,3,4}$	0	0
0	0	2	$\overline{2,6,7,8}$	$\overline{2,6,7,8}$	1	1
0	1	0	$\overline{3}$	$\overline{1,3,4}$	0	0
0	1	1	$\overline{5}$	$\overline{4,5}$	2	2
0	1	2	$\overline{2}$	$\overline{2,6,7,8}$	1	1
0	2	2	$\overline{2}$	$\overline{2,6,7,8}$	1	1
1	0	2	$\overline{6}$	$\overline{2,6,7,8}$	1	1
1	2	2	$\overline{4}$	$\overline{1,3,4}$; $\overline{4,5}$	0, 2	0
2	0	2	$\overline{7}$	$\overline{2,6,7,8}$	1	1
2	1	1	$\overline{5}$	$\overline{4,5}$	2	2
2	2	2	$\overline{4}$	$\overline{1,3,4}$; $\overline{4,5}$	0, 2	0

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

and

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

We can check that $\beta_{g'}$ satisfies the conditions of the theorem. The blocks of $\beta_{g'}$ can be encoded by a single 3-valued variable. One such encoding is shown below.

$$\beta_{g'} = \{\overset{0}{\overline{1,3,4}};\ \overset{1}{\overline{2,6,7,8}};\ \overset{2}{\overline{4,5}}\}.$$

The construction of g' is shown in Table 2.6. For example, minterm $e = 00000$ is relevant to f, hence $e^V = 000$ is a possible input to g. This minterm e^V exemplifies block $\overline{1,3}$ of blanket β_V. This is listed in the fourth column. The next column lists the block of β_g containing the block in the previous column. In this case, it is $\overline{1,3,4}$. Block $\overline{1,3,4}$ is assigned the value 0. In this case there is no option and the final output value $g(000)$ is 0. For minterms $\{1\}\{2\}\{2\}$ and $\{2\}\{2\}\{2\}$, we have a choice for the output because row 4 appears in two different blocks. We randomly pick the value 0 for both minterms. The rest of the table is constructed similarly. One verifies that this represents the same set function as that of Table 2.4.

$\square$

Example 2.5.2 *The construction of function h' is illustrated in Table 2.7. For minterm 10000 relevant to f, and the output value of $g'(000)$ from Table 2.6 is 0; hence 1000 is a relevant input vector for h'. Minterm 100 exemplifies block $\overline{1,2,3,5}$ of β_U, and the block of $\beta_{g'}$ assigned the output value 0 is $\overline{1,3,4}$. Thus, the intersection of the two blocks is $\overline{1,3}$, which is a block of $\beta_U {*} \beta_{g'}$. This block is contained only in block $\overline{1,3}$ of β_Y. Hence the value of h' is $\{1\}\{0\}$, the glb of the output values from Rows 1 and 3 of Table 2.3.*

Table 2.7: Finding function h'

x_1	x_2	x_3	z_1	β_U	$\beta_{g'}$	$\beta_U{*}\beta_{g'}$	β_Y	y_1	y_2
0	0	0	0	$\overline{3,5,6}$	$\overline{1,3,4}$	$\overline{3}$	$\overline{3}$, $\overline{1,3}$	1, 2	0
0	0	0	1	$\overline{3,5,6}$	$\overline{2,6,7,8}$	$\overline{6}$	$\overline{6}$	1	1
0	0	0	2	$\overline{3,5,6}$	$\overline{4,5}$	$\overline{5}$	$\overline{5}$	2	1, 2
0	0	1	1	$\overline{6}$	$\overline{2,6,7,8}$	$\overline{6}$	$\overline{6}$	1	1
0	0	2	2	$\overline{5}$	$\overline{4,5}$	$\overline{5}$	$\overline{5}$	2	1, 2
0	1	0	1	$\overline{7}$	$\overline{2,6,7,8}$	$\overline{7}$	$\overline{7}$, $\overline{2,7}$	0	0, 1
0	1	2	1	$\overline{7}$	$\overline{2,6,7,8}$	$\overline{7}$	$\overline{7}$, $\overline{2,7}$	0	0, 1
0	2	0	1	$\overline{8}$	$\overline{2,6,7,8}$	$\overline{8}$	$\overline{4,8}$	1	2
1	0	0	0	$\overline{1,2,3,5}$	$\overline{1,3,4}$	$\overline{1,3}$	$\overline{1,3}$	1	0
1	0	0	1	$\overline{1,2,3,5}$	$\overline{2,6,7,8}$	$\overline{2}$	$\overline{2,7}$, $\overline{2,4}$	0	1, 2
1	0	0	2	$\overline{1,2,3,5}$	$\overline{4,5}$	$\overline{5}$	$\overline{5}$	2	1, 2
1	0	2	2	$\overline{5}$	$\overline{4,5}$	$\overline{5}$	$\overline{5}$	2	1, 2
1	1	0	0	$\overline{1,2,7}$	$\overline{1,3,4}$	$\overline{1}$	$\overline{1,3}$	1	0
1	1	0	1	$\overline{1,2,7}$	$\overline{2,6,7,8}$	$\overline{2,7}$	$\overline{2,7}$	0	1
1	1	2	1	$\overline{7}$	$\overline{2,6,7,8}$	$\overline{7}$	$\overline{7}$, $\overline{2,7}$	0	0, 1
2	0	0	1	$\overline{2}$	$\overline{2,6,7,8}$	$\overline{2}$	$\overline{2,7}$, $\overline{2,4}$	0	1, 2
2	0	1	0	$\overline{4}$	$\overline{1,3,4}$	$\overline{4}$	$\overline{2,4}$, $\overline{4,8}$	0	2
2	0	2	0	$\overline{4}$	$\overline{1,3,4}$	$\overline{4}$	$\overline{2,4}$, $\overline{4,8}$	0	2
2	1	0	1	$\overline{2,7}$	$\overline{2,6,7,8}$	$\overline{2,7}$	$\overline{2,7}$	0	1
2	1	2	1	$\overline{7}$	$\overline{2,6,7,8}$	$\overline{7}$	$\overline{7}$, $\overline{2,7}$	0	0, 1
2	2	1	0	$\overline{4}$	$\overline{1,3,4}$	$\overline{4}$	$\overline{2,4}$, $\overline{4,8}$	0	2
2	2	2	0	$\overline{4}$	$\overline{1,3,4}$	$\overline{4}$	$\overline{2,4}$, $\overline{4,8}$	0	2

For minterm 00000*, the value of* g' *is* 0*; hence* 0000 *is relevant to* h'*. The corresponding block of* $\beta_U * \beta_{g'}$ *is* $\overline{3}$*, which is contained in two blocks of* β_Y*. Block* $\overline{1,3}$ *corresponds to output* $\{1\}\{0\}$*, and block* $\overline{3}$ *corresponds to output* $\{1,2\}\{0\}$*. We choose the last block, since it maximizes the don't cares.*

For minterm 20122*, the value of* g' *is* 0*; hence* 2010 *is relevant to* h'*. The corresponding block of* $\beta_U * \beta_{g'}$ *is* $\overline{4}$*, which is contained in two blocks of* β_Y*. Block* $\overline{2,4}$ *corresponds to output* $\{0\}\{2\}$*, and block* $\overline{4,8}$ *corresponds to output* $\{1\}\{2\}$*. Arbitrarily, we choose the first block. As another option, we could also assign* $\{0,1\}\{2\}$ *as the output.*

The rest of Table 2.7 is computed similarly. One can see that this is not *the same function as that of Table 2.5.*

□

2.6 Disjunctive Serial Separations

In the case of disjunctive separations we can prove the converse of Theorem 2.4.4.

Theorem 2.6.1 *For every disjunctive serial separation* (g,h) *of* f*, there is a blanket* β_g *such that* β_g *and* g *correspond with respect to* V*, the set of input variables of* g*, and* β_g *satisfies the conditions of Theorem 2.4.4.*

Proof. Suppose first that a disjunctive serial separation (g,h) exists. Then, for every minterm $e \in \Sigma^n$ relevant to f, $g(e^V)$ is a minterm in Σ^p. Define blanket $\beta_g = \gamma_g$; then clearly β_g and g correspond.

Let B be a block of β_V and let d be a minterm in Σ^s that exemplifies B. Then by definition of β_g, we have $B \subseteq F_{g=g(d)}$. This implies $\beta_V \leq \beta_g$.

We now show β_g also satisfies $\beta_U * \beta_g \leq \beta_Y$. Suppose B is a block of $\beta_U * \beta_g$. Then $B = B_U \cap B_g$ for some $B_U \in \beta_U$ and $B_g \in \beta_g$. Suppose $B_U = F_{U \supseteq c}$ and $B_g = F_{g=a}$ for some $c \in \Sigma^r$ and $a \in \Sigma^p$. Take $t \in B$. Then we have $t^U \supseteq c$. Also there exist minterm $d \in \Sigma^s$ such that $t^V \supseteq d$ and $g(d) = a$. Let $e = cd$, then e is relevant to f since $t^X \supseteq e$. By definition of output blankets, we get $t \in F_{Y \supseteq f(e)}$. Thus, $B \subseteq F_{Y \supseteq f(e)}$. Since $F_{Y \supseteq f(e)}$ is nonempty, it is a block of β_Y. This proves our result.

□

Table 2.8: Function g

x_3	x_4	x_5	z_1
0	0	0, 1	0
0	0, 1, 2	2	1
0	0, 1	0	0
1	2	2	0
2	2	2	2
0, 2	1	1	2
0, 1, 2	0	2	1

Example 2.6.2 *Consider function f from Table 2.3. Suppose that we are given the separation consisting of function g of Table 2.8 and function h of Table 2.9. We construct blanket β_g as follows. The value* 0 *is produced by g if the inputs are one of four the minterms* 000, 001, 010, *and* 122. *These four minterms are involved in rows 1, 3 and 4. Thus $\overline{1,3,4}$ is a block of β_g. Similarly, the blocks corresponding to the values 1 and 2 for g are $\overline{2,6,7,8}$ and $\overline{4,5}$, respectively. Hence*

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

One verifies that β_g satisfies the conditions of Theorem 2.6.1.

□

Table 2.9: Function h

x_1	x_2	z_1	y_1	y_2
1	0, 1	0	1	0
1, 2	0, 1	1	0	1, 2
0, 1	0	0	1, 2	0
2	0, 2	0, 2	0, 1	2
0, 1	0	2	2	1, 2
0	0	1	1	1
0, 1, 2	1	1	0	0, 1
0	2	1	1	2

Table 2.10: Function f for Example 2.6.3

Row	x_1	x_2	x_3	x_4	y_1
1	0	0, 1	0	1	0
2	0	0, 1	1	1	1

Example 2.6.3 *The following example, taken from [5], shows that Theorem 2.6.1 is not true if we remove the disjunctive requirement. Consider functions f of Table 2.10, g of Table 2.11 and h of Table 2.12. Let $U = \{x_1, x_2\}$ and $V = \{x_2, x_3, x_4\}$.*

One can verify that (g, h) is a separation of f with respect to (U, V). For this function, we have $\beta_U = \{\overline{1,2}\}$, and $\beta_V = \beta_f = \{\overline{1}; \overline{2}\}$. Let $d = \{0\}\{0\}\{1\}$, $e = \{1\}\{1\}\{1\}$. Suppose there is a β_g that satisfies all the conditions of the theorem. By Definition 2.4.3, we see that block $\{1, 2\}$ must be contained in some block of β_g. But then this β_g does not satisfy the condition $\beta_U * \beta_g \leq \beta_f$. Therefore, no such β_g exists.

$\square$

Table 2.11: Function g for Example 2.6.3

x_2	x_3	x_4	z_1
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	0

Table 2.12: Function h for Example 2.6.3

x_1	x_2	z_1	h
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0

2.7 Serial Decompositions

Let the sets U and V be as before. Suppose (g, h) is a serial separation with respect to (U, V) of set function f, where g has s inputs and p outputs, and h has $r + p$ inputs and m outputs. Also, suppose the cardinality of E is k. In order for (g, h) to be considered a decomposition of f, we require that both g and h have strictly fewer inputs than f.

Definition 2.7.1 *A* serial decomposition *of f with respect to (U, V) is a serial separation (g, h) of f with respect to (U, V), where $s < n$ and $r + p < n$.*

Let $\beta_g = ne\{F_{g=a}\}$. By definition, β_g and g correspond with respect to V. Then the number q of blocks of β_g is equal to the number of different outputs function g produces. Thus, we have $p = \lceil \log_k q \rceil$. Since $r + p < n$ by definition of a serial decomposition, we deduce that q satisfies

$$r + \lceil \log_k q \rceil < n.$$

As in the Boolean case [5], we now develop a useful necessary condition for the existence of a decomposition of a set function f.

Let $\beta = \{B_i\}$ and $\beta' = \{B'_i\}$ be two blankets on a set S. The *quotient* B/β' 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* $c(B, \beta')$ is the minimum number of blocks of B/β' required to cover B, in the sense that the union of these blocks is B. The *cover cost* of β using β', $c(\beta, \beta')$, is now defined as the largest block-cover cost, that is,

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

where *max* here denotes the largest integer in a set.

Theorem 2.7.2 *Let f be a set function, and let U and V be as before. If a serial decomposition (g, h) of f with respect to (U, V) exists, then*

$$n - r > \lceil \log_k c(\beta_U, \beta_Y) \rceil.$$

Proof. To prove the above inequality, it is sufficient to show

$$n - r - 1 \geq \log_k c(\beta_U, \beta_Y).$$

Let $q = k^{n-r-1}$. Then this is equivalent to showing $q \geq c(\beta_U, \beta_Y)$. By definition of serial decomposition, we know that g has at most $n - r - 1$ outputs, and thus g has at most q distinct output combinations. This implies that, given a minterm e relevant to f, there are at most q different input combinations to h of the form $e^U z$, where $z = g(v)$ for some $v \in \Sigma^s$. Thus, given e, there are at most q blocks of the form $F_{Y \supseteq h(e^U z)}$ contained in the output blanket β_Y. We shall prove that these blocks together covers block $F_{U \supseteq e^U}$ of β_U. Suppose $t \in F_{U \supseteq e^U}$, then there exists a minterm $b \subseteq t^X$ such that $b^U = e^U$. We know $z = g(b^V)$ is defined since b is relevant to f. By consistency condition of set matrices and the definition of serial separations, we see that $t^Y \supseteq f(b) \supseteq h(e^U z)$. Thus, we get $t \in F_{Y \supseteq h(e^U z)}$. This shows that any block of β_U is covered by at most q blocks of β_Y. Therefore, by definition of block-cover cost, our result follows.

$\square$

Example 2.7.3 *Let us find all the sets U for which a decomposition of the set function f of Table 2.3 is not ruled out by the condition of Theorem 2.7.2. For convenience, we shall use l_0 as a shorthand for $\lceil \log_k c(\beta_U, \beta_Y) \rceil$. Recall that*

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

For one-variable sets, that is, for $U = \{x_i\}$, *we have* $n - r = 4$. *The condition of the theorem is satisfied, because the number of blocks in* β_Y *is* 8*; consequently, the cost* $c(B, \beta_Y)$ *cannot be greater than* 8 *for any block* B *of* β_U. *Therefore,* $l_0 \leq \lceil \log_3 8 \rceil = 2 < n - r$. *Similarly, for two-variable sets, the condition of the theorem is also satisfied, since* $l_0 \leq \lceil \log_3 8 \rceil = 2 < 3 = n - r$.

For $U = \{x_1, x_2, x_3\}$, *we have:*

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

For the first block, $B_1/\beta_Y = max\{\overline{3};\ \overline{6};\ \overline{5}\}$. *Similarly,* $B_2/\beta_Y = \{\overline{6}\}$, $B_3/\beta_Y = \{\overline{5}\}$, $B_4/\beta_Y = \{\overline{7}\}$, $B_5/\beta_Y = \{\overline{8}\}$, $B_6/\beta_Y = \{\overline{1,3};\ \overline{2};\ \overline{5}\}$, $B_7/\beta_Y = \{\overline{2}\}$, $B_8/\beta_Y = \{\overline{4}\}$, *and* $B_9/\beta_Y = \{\overline{2,7}\}$. *The corresponding costs are* $c(B_1, \beta_Y) = 3$, $c(B_2, \beta_Y) = 1$, *etc. Hence,* $l_0 = 1 < n - r = 2$, *and the condition is satisfied.*

For $U = \{x_1, x_3, x_4\}$,

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

Here U *fails to satisfy the condition, because* $B_1/\beta_Y = \{\overline{3};\ \overline{6};\ \overline{7};\ \overline{8}\}$. *Hence* $l_0 = 2 \not< 2 = n - r$.

We can check that the condition is satisfied for all three-element set except $U = \{x_1, x_3, x_4\}$.

For another example, let $U = \{x_2, x_3, x_4\}$ *and* $V = \{x_1, x_5\}$*; then*

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

$$\beta_V = \{\overset{B'_1}{\overline{3}};\ \overset{B'_2}{\overline{5}};\ \overset{B'_3}{\overline{6,7,8}};\ \overset{B'_4}{\overline{1,3}};\ \overset{B'_5}{\overline{1,5}};\ \overset{B'_6}{\overline{2,7}};\ \overset{B'_7}{\overline{2,4,7}}\}.$$

Any β_g *satisfying the conditions of Theorem 2.4.4 must have at least 4 blocks, since the blocks* B'_3, B'_4, B'_5 *and* B'_7 *are pairwise incompatible. Thus, function* g

must have at least two outputs. Since g has only two inputs, according to our definition, the pair (g, h) does not constitute a decomposition of f. Therefore, no disjunctive decomposition of f exists for this pair of subsets (U, V). However, if we let $V = \{x_1, x_4, x_5\}$, then

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

We can check that $\beta_g = \{\overline{1,3};\ \overline{5,6,7,8};\ \overline{2,4,7}\}$ satisfies the conditions from Theorem 2.4.4 and thus leads to a non-disjunctive decomposition of f.

After the verification of the necessary conditions from Theorem 2.4.4, we find all three-element subsets except $\{x_1, x_3, x_4\}$ and $\{x_2, x_3, x_4\}$ lead to disjunctive decompositions, and $\{x_2, x_3, x_4\}$ leads to non-disjunctive decomposition only.

□

2.8 Decomposition Method Using Cubes

In the proof of Theorem 2.4.4 we described a method of constructing functions g and h for the decomposition of f. However, while function f is represented by a general set matrix, both intermediate functions g and h are in their respective base representation, i.e., all input vectors are minterms. In this section, we describe a new way to construct functions g and h. The functions g and h resulting from this construction are not necessarily in their base representations. This gives more compact set matrices for functions g and h. We refer to this new method as the *cube method.* In this section, we require the given blanket β_g to satisfy the two conditions of Theorem 2.4.4. In Section 3.4, we will show an example of a non-disjunctive decomposition where β_g does not satisfy the second condition.

We shall use the set function in Table 2.13 as our example. This example is the same as the function specified by Table 2.3 except for the last row. Let $U = \{x_1, x_2\}$ and $V = \{x_3, x_4, x_5\}$. Note that, because of row 8, all 27 minterms are relevant to g; thus g will have 27 rows if we use our old construction.

Table 2.13: Set Function f

Row	x_1	x_2	x_3	x_4	x_5	y_1	y_2
1	1	0, 1	0	0	0, 1	1	0
2	1, 2	0, 1	0	0, 1, 2	2	0	1, 2
3	0, 1	0	0	0, 1	0	1, 2	0
4	2	0, 2	1, 2	2	2	0, 1	2
5	0, 1	0	0, 2	1	1	2	1, 2
6	0	0	0, 1	0	2	1	1
7	0, 1, 2	1	0, 2	0	2	0	0, 1
8	0	2	0, 1, 2	0, 1, 2	0, 1, 2	1	2

The blankets β_U, β_V and β_g are as follows:

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

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

and

$$\beta_Y = \{\overset{00}{\overline{7}};\ \overset{01}{\overline{2,7}};\ \overset{02}{\overline{2,4}};\ \overset{10}{\overline{1,3}};\ \overset{11}{\overline{6}};\ \overset{12}{\overline{4,8}};\ \overset{20}{\overline{3}};\ \overset{21,22}{\overline{5}}\}.$$

Note that we show a numbering of the blocks of β_V while the minterms that exemplify each block of β_Y are also shown.

We can check that the following blanket β_g satisfies the conditions of Theorem 2.4.4, and thus leads to a separation of function f. A possible encoding of the blocks is also shown.

$$\beta_g = \{\overline{1,3,4,8}^{\,0};\ \overline{2,6,7,8}^{\,1};\ \overline{4,5,8}^{\,2}\}.$$

We now construct a function g such that β_g and g correspond. Once this is done, the construction of h follows easily.

In Table 2.14 we list the blocks of β_V and minterms that exemplify each block. Note that all 15 minterms that do not exemplify the other blocks exemplify block B_7; and, for clarity, only three of the 15 are shown here.

Table 2.14: Correspondence between blocks and minterms

Blocks of β_V	Cubes of F	$\chi(B_i)$
B_1	$\{1,3,8\}$	$\{000\}$
B_2	$\{1,8\}$	$\{001\}$
B_3	$\{2,6,7,8\}$	$\{002\}$
B_4	$\{3,8\}$	$\{010\}$
B_5	$\{5,8\}$	$\{011,211\}$
B_6	$\{2,8\}$	$\{012,022\}$
B_7	$\{8\}$	$\{020,021,...,221\}$
B_8	$\{6,8\}$	$\{102\}$
B_9	$\{4,8\}$	$\{122,222\}$
B_{10}	$\{7,8\}$	$\{202\}$

Table 2.15 shows the construction of g. For example, consider $B_5 = \{5,8\}$. Two minterms $\{0\}\{1\}\{1\}$ and $\{2\}\{1\}\{1\}$ exemplify this block. Function g assigns 2 to

both these minterms. Consequently we can assign the value 2 to the representative of block B_5 as we know it covers both minterms. Hence we derive the output column of the table.

Note that $r_{B_2} = \{0\}\{0\}\{0,1\}$ contains $r_{B_1} = \{0\}\{0\}\{0\}$, i.e. the two blocks overlap. This does not matter since both blocks are assigned the value 0. Block B_7, however, is a different case. It is contained in all three blocks of β_g and hence can be assigned to any value. We shall randomly pick value 2. Then its representative has a non-empty intersection with representative $\{0\}\{0\}\{0\}$ of B_1 which is assigned a different value. We say that the pair of blocks (B_1, B_7) creates a *conflict*, that is, the representatives of the two blocks have non-empty intersection but the two blocks are assigned different values by g. To make the resulting matrix for function g consistent, we must resolve this conflict. To do this, we subtract from the representative of B_1 all minterms exemplifying B_7, and vice versa. In our example, block B_1 is not affected by this subtraction, while we must subtract minterm 000 from representative of B_7. There are also conflicts between B_7 and all other blocks assigned the values 0 or 1 by g. Thus we must subtract from B_7 all minterms exemplifying those blocks. The result is four cubes $\{1,2\}\{1,2\}\{0,1,2\}$, $\{0\}\{1\}\{1\}$, $\{0\}\{2\}\{0,1\}$ and $\{1,2\}\{0\}\{0,1\}$. These cubes can be assigned the value 2. We also have to subtract the 15 minterms exemplifying B_7 from the blocks conflicting with B_7. In this case, just as with B_3, none of the blocks is affected.

We also have a choice to assign values 0 or 2 to block B_9; we choose the value 2, and this creates no conflict. After resolving all the conflicts, the resulting set matrix is consistent.

To obtain the table for function h, we use a similar method with blocks of $\beta_U * \beta_g$. In our example, we have

Table 2.15: Cubes for g

Block	Representative			Output	Final Cubes		
B_1	0	0	0	0	0	0	0
B_2	0	0	0, 1	0	0	0	0, 1
B_3	0	0	2	1	0	0	2
B_4	0	0, 1	0	0	0	0, 1	0
B_5	0, 2	1	1	2	0, 2	1	1
B_6	0	0, 1, 2	2	1	0	0, 1, 2	2
B_7	0, 1, 2	0, 1, 2	0, 1, 2	2	1, 2	1, 2	0, 1, 2
				2	0	1	1
				2	0	2	0, 1
				2	1, 2	0	0, 1
B_8	0, 1	0	2	1	0, 1	0	2
B_9	1, 2	2	2	2	1, 2	2	2
B_{10}	0, 2	0	2	1	0, 2	0	2

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

To compute the table for h, we consider each block B_i' of $\beta_U * \beta_g$ in turn. For each block B_i', we first find $\bigsqcap_{t \in B_i'} t^U$. This is the first part of the input cube to h. Then we find the blocks of β_g that contain B_i', pick one of these blocks randomly, and the outputs of g assigned to this block become the second part of the input cube. The output assigned to this cube is then $\bigsqcap_{t \in B_i'} t^Y$. For example, consider B_1'. The first part of the input cube is $\{0,1\}\{0\}$; the only block of β_g containing B_1' is $\{1,3,4,8\}$, so the second part of the input cube is 0. Finally, the output is $\{1,2\}\{0\}$. As another example, block B_5' is contained in all three blocks of β_g, thus the second part of the input cube is $\{0,1,2\}$. The remaining rows are computed similarly. The complete table for h is shown in Table 2.16.

Table 2.16: Cubes for h

Block	x_1	x_2	z	h_1	h_2
B_1'	0, 1	0	0	1, 2	0
B_2'	0	0	1	1	1
B_3'	0, 1	0	2	2	1, 2
B_4'	0, 1, 2	1	1	0	0, 1
B_5'	0	2	0, 1, 2	1	2
B_6'	1	0	0	1	0
B_7'	1, 2	0, 1	1	0	1, 2
B_8'	1	0, 1	0	1	0
B_9'	1, 2	1	1	0	1
B_{10}'	2	0, 2	0, 2	0, 1	2

Proposition 2.8.1 *The pair (g, h) of functions constructed using the above procedure constitutes a serial separation of f, and β_g and g correspond with respect to V.*

Proof. We already know that g is a set function. We now show that h is also a set function, i.e., set matrix M_h for function h is consistent. Suppose the set of rows of M_h is denoted by H. Let $e \in \Sigma^{r+p}$ be a minterm relevant to h. Suppose e^U exemplifies block B_U of β_U, and block B_g of β_g is assigned the value e^Z. Suppose e is involved in some row $t_h \in H$, where t_h corresponds to block B'_i of $\beta_U \times \beta_g$. Since $e^U \subseteq t_h^U \subseteq t^U$ for all $t \in B'_i$ and $B_U = F_{U \supseteq e^U}$, we have $B'_i \subseteq B_U$. Also, by our construction, $B'_i \subseteq B_g$. Thus, $B'_i \subseteq B_U \cap B_g$. By the condition $\beta_U \times \beta_g \leq \beta_Y$, we know that $B_U \cap B_g$ is contained in some block B_Y of β_Y. This implies $B'_i \subseteq B_Y$. Since t_h^Y is defined as $\bigsqcap_{t \in B'_i} t^Y$, we get $t_h^Y \supseteq \bigsqcap_{t \in B_Y} t^Y$. Since this is true for all rows $t_h \in H$ in which e is involved, we see that the outputs of all these rows are compatible. This shows that M_h is consistent.

Let $e \in \Sigma^n$ be a minterm relevant to f. It is clear from the construction that $g(e^V)$ is a minterm in Σ^p. Suppose e^U exemplifies block $\hat{B}$ of β_U and block $\hat{B}'$ of β_g is assigned the value $g(e^V)$. Then $\hat{B} \cap \hat{B}' = B'_i$ is a block of $\beta_U * \beta_g$. Thus, $(e^U, g(e^V))$ is relevant to h as it is involved in the row corresponding to B'_i. Suppose e is involved in some $t \in F$. Then we have $t \in \hat{B}$. Suppose e^V exemplifies block B_j of β_V. Then by the construction of g, we know $B_j \subseteq \hat{B}'$. Also, since $t \in B_j$, we get $t \in \hat{B}'$. Thus, we have $t \in \hat{B} \cap \hat{B}' = B'_i$. Consider the row corresponding to B'_i in the matrix M_h. The input part of this row contains the minterm $(e^U, g(e^V))$, while the output part is $\bigsqcap_{t' \in B'_i} t'^Y$. This implies

$$h(e^U, g(e^V)) \subseteq \bigsqcap_{t' \in B'_i} t'^Y \subseteq t^Y \subseteq f(e).$$

It remains to show that β_g and g correspond with respect to V. Take a block

$F_{g=a}$ of γ_g, and suppose $t \in F_{g=a}$. Then there exists a minterm $d \subseteq t^V$ such that $g(d) = a$. Let B_g be the block of β_g assigned the value a. By our construction, we see that B_g must contain block $F_{V \supseteq d}$ of β_V. Thus we have $t \in B_g$ which implies $F_{g=a} \subseteq B_g$. Therefore, $\gamma_g \leq \beta_g$.

□

2.9 Finding Blanket β_g

So far, we have not consider the problem of finding a blanket β_g given sets U and V. One way to solve this problem is by finding maximal mergeable classes as described in [5]. We illustrate this method with the following two examples.

Example 2.9.1 *We use set function f specified by Table 2.13 as an example. Let $U = \{x_1, x_2\}$, $V = \{x_3, x_4, x_5\}$. Recall that we have*

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

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

and

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

A subset $\{B_{i_1}, \ldots, B_{i_k}\}$ of β_V is said to be mergeable, *if blanket β, obtained from β_V by merging blocks $B_{i_1}, \ldots, B_{i_k}$, satisfies $\beta_U * \beta \leq \beta_Y$. Otherwise the subset is* unmergeable. *From this definition, we see that the unmergeable pairs in our example are $\{B_1, B_3\}$, $\{B_1, B_5\}$, $\{B_1, B_6\}$, $\{B_1, B_8\}$, $\{B_1, B_{10}\}$, $\{B_2, B_3\}$, $\{B_2, B_5\}$,*

$\{B_2, B_6\}$, $\{B_2, B_{10}\}$, $\{B_3, B_4\}$, $\{B_3, B_5\}$, $\{B_4, B_5\}$, $\{B_4, B_6\}$, $\{B_4, B_8\}$, $\{B_5, B_6\}$, $\{B_5, B_8\}$. *Thus, any subset of* β_V *which can be merged to form a block of* β_g *must not contain both* B_1 *and* B_3*, or both* B_1 *and* B_5*, etc. This means we can get the minimal set of blocks to be omitted in such a subset by expanding the following product:*

$$(B_1 + B_3)(B_1 + B_5) \ldots (B_5 + B_6)(B_5 + B_8).$$

We multiply out this expression and then omit terms whose corresponding set of blocks is a superset of the corresponding set of blocks of another term. The resulting sum-of-product expression is

$$B_1B_2B_3B_5B_6B_8 + B_1B_2B_4B_5 + B_1B_2B_3B_4B_6B_8 + B_1B_3B_4B_5B_6B_{10} + B_3B_5B_6B_8B_{10}.$$

Each product describes a minimal set of blocks to be omitted. Thus, there are five possible maximal compatible subsets corresponding to the five terms. For example $\{B_4, B_7, B_9, B_{10}\}$ *corresponds to the first term. Note that unlike in the Boolean case, these subsets are not necessarily mergeable. In fact we will show a case where some of the subsets are not mergeable in the next example. The reason for this difference is pairwise mergeability does not imply mergeability for the subset when we are dealing with set functions. In this example, however, we see that these five subsets are indeed mergeable and hence are also maximal. They correspond to the following blocks:*

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

The problem now reduces to finding a minimal cover for β_V *using these maximal mergeable subsets. This is equivalent to the covering problem encountered in modern logic synthesis, and several algorithm have been developed for efficient calculation of minimal covers [2, 19]. In our example, there is only one minimal cover with* *subsets* $\{B_3, B_6, B_7, B_8, B_9, B_{10}\}$, $\{B_5, B_7, B_9, B_{10}\}$, *and* $\{B_1, B_2, B_4, B_7, B_9\}$. *The corresponding blanket is*

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

□

Table 2.17: Set Function f for Example 2.9.2

Row	x_1	x_2	x_3	y_1
1	0, 2	1, 2	1	0, 1
2	0	0	2	0, 2
3	0, 1	2	0, 2	1, 2
4	2	0	1	2

Example 2.9.2 *Consider the set function* f *specified by Table 2.17. For* $U = \{x_1\}$ *and* $V = \{x_2, x_3\}$, *we have*

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

$$\beta_V = \{\overset{B_1}{\overline{1}};\ \overset{B_2}{\overline{2}};\ \overset{B_3}{\overline{3}};\ \overset{B_4}{\overline{4}}\},$$

and

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

Using the method described in the last example, we find that the only unmergeable pair is $\{B_1, B_4\}$. *Thus, we have* $\{B_2, B_3, B_4\}$ *and* $\{B_1, B_2, B_3\}$ *as possible maximal compatible subsets. However, it is clear that* $S = \{B_1, B_2, B_3\}$ *is not a compatible subset. In order to find the maximal compatible subsets, we need to calculate* *the product* $(B_1 + B_4)(B_1 + B_2 + B_3)$. *Expanding this product, we see the new candidates are* $\{B_2, B_3, B_4\}$, $\{B_1, B_3\}$ *and* $\{B_1, B_2\}$. *All these subsets are compatible. There are two possible minimal covers, and they correspond to the blankets* $\beta_g = \{\overline{2,3,4};\ \overline{1,3}\}$, *and* $\beta_{g'} = \{\overline{2,3,4};\ \overline{1,2}\}$.

□

2.10 Decomposition of Decision Trees

In this section, we look at the decomposition of decision trees, where set functions arise naturally. We illustrate this with an example taken from [11].

Example 2.10.1 *Table 2.18 shows some data describing weather conditions. Each row has four input attributes and then a decision value* Class *to indicate whether such an input combination occurred (P for Positive) or not (N for Negative).*

Figure 2.1 shows a decision tree constructed from this table, with the variable order $(Outlook, Humidity, Windy, Temperature)$. *For a method to construct such a tree, see [6]. We can get a set matrix directed from the decision tree as follows:*

- *Each path from the root to a terminal node in general corresponds to a row for the set matrix, where input values are as indicated by the labels with missing variables assigned complete don't cares;*
- *If two paths end in terminal nodes with the same value and they differ by only one label, they can be combined to form one row in the set matrix, for example,* $(Outlook = rain, Windy = medium)$ *and* $(Outlook = rain, Windy = very)$ *are combined.*

Table 2.18: Sample Data Set for Weather Conditions

Row	Outlook	Humidity	Windy	Temperature	Class
1	overcast	high	not	hot	N
2	overcast	high	very	hot	N
3	overcast	high	medium	hot	N
4	sunny	high	not	hot	P
5	sunny	high	medium	hot	P
6	rain	high	not	mild	N
7	rain	high	medium	mild	N
8	rain	normal	not	hot	P
9	rain	normal	medium	cool	N
10	rain	normal	very	hot	N
11	sunny	normal	very	cool	P
12	sunny	normal	medium	cool	P
13	overcast	high	not	mild	N
14	overcast	high	medium	mild	N
15	overcast	normal	not	cool	P
16	overcast	normal	medium	cool	P
17	rain	normal	not	mild	N
18	rain	normal	medium	mild	N
19	overcast	normal	medium	mild	P
20	overcast	normal	very	mild	P
21	sunny	high	very	mild	P
22	sunny	high	medium	mild	P
23	sunny	normal	not	hot	P
24	rain	high	very	mild	N

Figure 2.1: Decision Tree for Weather Conditions

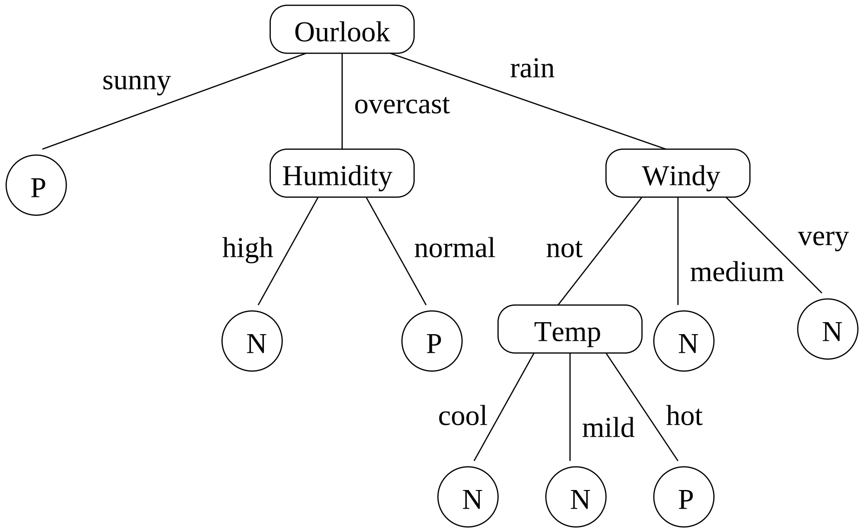


Table 2.19: Set Matrix for Weather Example

Row	*Outlook*	*Humidity*	*Windy*	*Temperature*	*Class*
1	*sunny*	–	–	–	*P*
2	*overcast*	*high*	–	–	*N*
3	*overcast*	*normal*	–	–	*P*
4	*rain*	–	*not*	*cool,mild*	*N*
5	*rain*	–	*not*	*hot*	*N*
6	*rain*	–	*medium,very*	–	*N*

The resulting set matrix is shown in Table 2.19, for clarity, complete don't cares are represented as $-$. *Now we can decompose this set function using procedures described earlier. One possible decomposition is with* $U = \{Humidity\}$, *and* $V = \{Outlook, Windy, Temperature\}$. *Functions* g *and* h, *shown in Tables 2.20 and 2.21, give a decomposition with respect to this pair of* (U, V).

Table 2.20: Function g for Weather Example

Outlook	*Windy*	*Temperature*	*Type*
sunny	–	–	0
overcast	–	–	1
rain	*not*	*cool,mild*	2
rain	*not*	*hot*	0
rain	*medium,very*	–	2

Table 2.21: Function h for Weather Example

Humidity	*Type*	*Class*
–	0	*P*
high	1	*N*
normal	1	*P*
–	2	*N*

□

Chapter 3

Shestakov's Decomposition

In 1994, Shestakov described a method of decomposition of Boolean functions in his papers [21, 22]. He expresses a Boolean function $f(x)$ in the form $h(g_1(u), g_2(v))$, where as before, $U \cup V = X$, and the sets corresponding to u, v, and x satisfy $U \cup V = X$. His methods have been formalized using blankets and extended to the multi-valued case by Lou and Brzozowski [14]. Here we describe the formalization in that paper.

We will call the separation described in Chapter 2 *single separation* and the separation described by Shestakov *double separation.* In the following, the conventions we established earlier are used.

3.1 Double Separation of Set Functions

Definition 3.1.1 *Let f be a set function, with $n > 0$ inputs and $m > 0$ outputs, and let (U, V) be as above. Assume that f is specified by a set F of function cubes. Let g_1 be a set function with r inputs and p outputs, let g_2 be a set function with s*

inputs and q outputs, and let h be a set function with $p+q$ inputs and m outputs. The triple (g_1, g_2, h) is a double separation of f with respect to (U, V)*, if, for every minterm $b \in \Sigma^n$ relevant to f, $g_1(b^U) \in \Sigma^p$, $g_2(b^V) \in \Sigma^q$, and*

$$f(b) \supseteq h(g_1(b^U), g_2(b^V)). \tag{3.1}$$

Note that the single separation is a special case of this double separation where function g_1 is the identity function. The following theorem gives a necessary condition, similar to that in Theorem 2.4.4, for the existence of a double separation.

Theorem 3.1.2 *Let $f(x)$ be a set function with the set F of rows and let (U, V) be a pair of subsets of X satisfying $U \cup V = X$. For every pair $(\beta_{g_1}, \beta_{g_2})$ of blankets satisfying the conditions stated below, there is a double separation (g_1, g_2, h) such that β_{g_1} and g_1 correspond with respect to U, and β_{g_2} and g_2 correspond with respect to V.*

- $\beta_U \leq \beta_{g_1}$,
- $\beta_V \leq \beta_{g_2}$, *and*
- $\beta_{g_1} * \beta_{g_2} \leq \beta_Y$.

Proof. Suppose β_{g_1} and β_{g_2} satisfying the conditions of the theorem are given. Following the method in the proof of Theorem 2.4.4, we can construct g_1 and g_2 from β_{g_1} and β_{g_2}, respectively. See the examples in Section 3.2 for more details.

Next we construct the truth table for a function h with $p+q$ inputs and m outputs. Consider $x = x_1 x_2$ with $x_1 \in \Sigma^p$ and $x_2 \in \Sigma^q$. Define $F_{g_1 = x_1}$ to be the block of β_{g_1} that is assigned the value x_1, and $F_{g_2 = x_2}$ to be the block of β_{g_2} that is assigned the value x_2. Let $F_{g_1=x_1} \cap F_{g_2=x_2} = B$. If $B = \emptyset$, then x is omitted from the table for h. Otherwise, if $B \neq \emptyset$, by the condition $\beta_{g_1} * \beta_{g_2} \leq \beta_Y$, we know that B is a nonempty subset of some block $\hat{B}$ of the output blanket β_Y. Thus we know that $\bigcap_{t \in B} t^Y$ exists. Assign this value to $h(x)$.

The proof that the above construction is correct closely follows the proof of Theorem 2.4.4 and is omitted.

□

Just as for Theorem 2.4.4, the converse of the above theorem is true if U and V are disjoint. The proof is similar to Theorem 2.6.1 and is omitted. Example 2.6.3 also serves as a counterexample here when U and V are not disjoint.

Now we prove that double separation is equivalent to two single separations performed in sequence.

Proposition 3.1.3 *Let f be a set function and let (g_1, g_2, h) be a double separation of f. Let r, q and m be as before. Then there exists a function h' with $r + q$ inputs and m outputs such that (g_2, h') is a single separation of f and (g_1, h) is a single separation of h'.*

Proof. We define h' as follows. For $x = cx'$ where $c \in \Sigma^r$ and $x' \in \Sigma^q$, if there exists a minterm b relevant to f such that $b^U = c$ and $g_2(b^V) = x'$, then we know that $x_1 = g_1(c)$ exists and $h(x_1, x')$ also exists. In this case, we define $h'(x) = h(x_1, x')$. Otherwise x is not present in the table for h'. Now for every minterm b relevant to f,

$$f(b) \supseteq h(g_1(b^U), g_2(b^V)) = h'(b^U, g_2(b^V)).$$

Thus (g_2, h') is a single separation of f.

Let the set of outputs of g_2 be V'. Then the input set of h' is $U \cup V'$, and for every minterm e relevant to h',

$$h'(e) = h'(e^U, e^{V'}) = h(g_1(e^U), e^{V'}).$$

Thus (g_1, h) is a single separation of h'. In fact the $\supseteq$ in the definition is replaced by strict equality here.

$\square$

3.2 Examples of Separations

Example 3.2.1 *In this example, we derive a double separation of function f specified by Table 2.3. Let $U = \{x_1, x_2, x_3\}$, and $V = \{x_3, x_4, x_5\}$; recall that*

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

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

and

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

Now let $\beta_{g_1} = \{\overline{3,5,6};\ \overline{1,2,3,5,7};\ \overline{4,8}\}$, $\beta_{g_2} = \{\overline{1,3};\ \overline{2,6,7,8};\ \overline{4,5}\}$. We can check that these two blankets satisfy the conditions of Theorem 3.1.2. We encode both β_{g_1} and β_{g_2} with one variable. One such encoding is as follows:

$$\beta_{g_1} = \{\overset{0}{\overline{3,5,6}};\ \overset{1}{\overline{1,2,3,5,7}};\ \overset{2}{\overline{4,8}}\},$$

$$\beta_{g_2} = \{\overset{0}{\overline{1,3}};\ \overset{1}{\overline{2,6,7,8}};\ \overset{2}{\overline{4,5}}\}.$$

Table 3.1: Finding function g_1

x_1	x_2	x_3	β_U	β_{g_1}	*options*	z_1
0	0	0	$\overline{3,5,6}$	$\overline{3,5,6}$	0	0
0	0	1	$\overline{6}$	$\overline{3,5,6}$	0	0
0	0	2	$\overline{5}$	$\overline{3,5,6}$,$\overline{1,2,3,5,7}$	$0,1$	0
0	1	0	$\overline{7}$	$\overline{1,2,3,5,7}$	1	1
0	1	2	$\overline{7}$	$\overline{1,2,3,5,7}$	1	1
0	2	0	$\overline{8}$	$\overline{4,8}$	2	2
1	0	0	$\overline{1,2,3,5}$	$\overline{1,2,3,5,7}$	1	1
1	0	2	$\overline{5}$	$\overline{3,5,6}$,$\overline{1,2,3,5,7}$	$0,1$	0
1	1	0	$\overline{1,2,7}$	$\overline{1,2,3,5,7}$	1	1
1	1	2	$\overline{7}$	$\overline{1,2,3,5,7}$	1	1
2	0	0	$\overline{2}$	$\overline{1,2,3,5,7}$	1	1
2	0	1	$\overline{4}$	$\overline{4,8}$	2	2
2	0	2	$\overline{4}$	$\overline{4,8}$	2	2
2	1	0	$\overline{2,7}$	$\overline{1,2,3,5,7}$	1	1
2	1	2	$\overline{7}$	$\overline{1,2,3,5,7}$	1	1
2	2	1	$\overline{4}$	$\overline{4,8}$	2	2
2	2	2	$\overline{4}$	$\overline{4,8}$	2	2

Using the method described in Chapter 2, we construct tables for g_1 *and* g_2*. They are shown in Tables 3.1 and 3.2, respectively. In the table for* g_2*, we omit the options columns since* β_{g_2} *is a partition and there is only one option for all minterms.*

Table 3.2: Finding function g_2

x_3	x_4	x_5	β_V	β_{g_2}	z_2
0	0	0	$\overline{1,3}$	$\overline{1,3}$	0
0	0	1	$\overline{1}$	$\overline{1,3}$	0
0	0	2	$\overline{2,6,7,8}$	$\overline{2,6,7,8}$	1
0	1	0	$\overline{3}$	$\overline{1,3}$	0
0	1	1	$\overline{5}$	$\overline{4,5}$	2
0	1	2	$\overline{2}$	$\overline{2,6,7,8}$	1
0	2	2	$\overline{2}$	$\overline{2,6,7,8}$	1
1	0	2	$\overline{6}$	$\overline{2,6,7,8}$	1
1	2	2	$\overline{4}$	$\overline{4,5}$	2
2	0	2	$\overline{7}$	$\overline{2,6,7,8}$	1
2	1	1	$\overline{5}$	$\overline{4,5}$	2
2	2	2	$\overline{4}$	$\overline{4,5}$	2

To get h*, we first find block* B_1 *of* β_{g_1} *corresponding to input* z_1 *and block* B_2 *of* β_{g_2} *corresponding to input* z_2*. Next we calculate* $B_1 \cap B_2$*. The output is then the greatest lower bound of the outputs of the rows contained in* $B_1 \cap B_2$*. The resulting function is shown in Table 3.3. For example, for* $z_1 z_2 = 10$*, we have* $B_1 = \overline{1,2,3,5,7}$*, and* $B_2 = \overline{1,3}$*. The intersection of the two blocks is* $\overline{1,3}$*. Hence the output is* $glb\{\{1\}\{0\},\{1,2\}\{0\}\} = \{1\}\{0\}$*. The rest of the table follows simi-*

larly. Note that if the intersection $B_1 \cap B_2$ is empty, then the corresponding input combination $z_1 z_2$ is omitted from the table for h. For example, input combination $z_1 z_2 = 20$ is omitted here.

Table 3.3: Finding function h

z_1	z_2	$B_1 \cap B_2$	y_1	y_2
0	0	$\overline{3}$	$1,2$	0
0	1	$\overline{6}$	1	1
0	2	$\overline{5}$	2	$1,2$
1	0	$\overline{1,3}$	1	0
1	1	$\overline{2,7}$	0	1
1	2	$\overline{5}$	2	$1,2$
2	1	$\overline{8}$	1	2
2	2	$\overline{4}$	$0,1$	2

□

Example 3.2.2 *In this example, we use the functions g_2 and h derived in the last example to construct function h' as described in the proof of Proposition 3.1.3. In Table 3.4, we list the possible input combinations (x_1, x_2, x_3, z_2) to function h'. For each such combination, there exists a minterm b relevant to f satisfying $b^U = (x_1, x_2, x_3)$ and $g_2(b^V) = z_2$. We then list $z_1 = g_1(x_1, x_2, x_3)$, and finally the two outputs $h(z_1, z_2)$ which are the outputs of h'.*

Note that those input combinations for h' which do not have a corresponding minterm b are omitted. For example, the input combination 0010 does not corre-

Table 3.4: Matrix defining h'

x_1	x_2	x_3	z_2	z_1	y_1	y_2
0	0	0	0	0	1, 2	0
0	0	0	1	0	1	1
0	0	0	2	0	2	1, 2
0	0	1	1	0	1	1
0	0	2	2	0	2	1, 2
0	1	0	1	1	0	1
0	1	2	1	1	0	1
0	2	0	1	2	1	2
1	0	0	0	1	1	0
1	0	0	1	1	0	1
1	0	0	2	1	2	1, 2
1	0	2	2	0	2	1, 2
1	1	0	0	1	1	0
1	1	0	1	1	0	1
1	1	2	1	1	0	1
2	0	0	1	1	0	1
2	0	1	2	2	0, 1	2
2	0	2	2	2	0, 1	2
2	1	0	1	1	0	1
2	1	2	1	1	0	1
2	2	1	2	2	0, 1	2
2	2	2	2	2	0, 1	2

spond to any minterm, while input combination 0012 *corresponds to minterm* 00122 *which is not relevant to* f. *Both of these are omitted from our table.*

We can check that the pair (g_2, h') *is a single separation of* f, *and that the pair* (g_1, h) *is a single separation of* h'.

□

3.3 Decomposition Method Using Tables

We now describe the tabular method used in [22]. This method can be used to do both single and double separation. We shall use it to derive a single separation first, and then proceed to derive a double separation. Note that our single separation will be of the form $h(g_1(u), v)$, so the usual roles of U and V are reversed.

We again use the function specified by Table 2.3 as our example. The blocks of β_U and β_V are labeled as follows.

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

$$\beta_V = \{\overline{1,3}^{B'_1};\ \overline{1}^{B'_2};\ \overline{2,6,7,8}^{B'_3};\ \overline{3}^{B'_4};\ \overline{5}^{B'_5};\ \overline{2}^{B'_6};\ \overline{6}^{B'_7};\ \overline{4}^{B'_8};\ \overline{7}^{B'_9}\}.$$

We construct a *block multiplication table* where the rows are labeled B_i and columns are labeled B'_j. The entry (i, j) of this table is $B_i \wedge B'_j$, where $\wedge$ is defined as follows:

Definition 3.3.1 *Given a block* B_i *from* β_U *and a block* B'_j *from* β_V, *we define* $B_i \wedge B'_j = B_i \cap B'_j$ *if*

- $B_i \cap B'_j \neq \emptyset$,

- *there exists a minterm b relevant to f such that b^U exemplifies B_i and b^V exemplifies B'_j.*

Otherwise, we define $B_i \wedge B'_j = \emptyset$.

Intuitively, the operation $\wedge$ is the block intersection if there is a common minterm which exemplifies both blocks, and empty otherwise. In fact, we can show that in the case $U \cap V = \emptyset$, this operation is equivalent to intersection.

The block multiplication table T for f is shown in Table 3.5. For clarity, empty sets are represented by $-$ and we omit $\{\}$ for all other entries.

Table 3.5: Block Multiplication Table

	B'_1	B'_2	B'_3	B'_4	B'_5	B'_6	B'_7	B'_8	B'_9
B_1	3	–	6	3	5	–	–	–	–
B_2	–	–	–	–	–	–	6	–	–
B_3	–	–	–	–	5	–	–	–	–
B_4	–	–	7	–	–	–	–	–	7
B_5	–	–	8	–	–	–	–	–	–
B_6	1, 3	1	2	3	5	2	–	–	–
B_7	1	1	2, 7	–	–	2	–	–	–
B_8	–	–	2	–	–	2	–	–	–
B_9	–	–	–	–	–	–	–	4	–
B_{10}	–	–	2, 7	–	–	2	–	–	–

Note that $T_{2,3}$ is empty instead of $B_2 \cap B'_3 = \{6\}$. This is because the only minterm that exemplifies B_2 is 001, while the only minterm that exemplifies B'_3 is

002; thus the minterm b in our definition of $\wedge$ does not exist and we have $B_2 \wedge B'_3 = \emptyset$. All entries where $B_i \wedge B'_j \neq B_i \cap B'_j$ are denoted with $-^*$.

Now we replace $T_{i,j}$ with $\bigcap_{t \in T_{i,j}} t^Y$ if $T_{i,j}$ is not empty. If $T_{i,j}$ is empty, it is replaced by the complete don't care $\{0,1,2\}\{0,1,2\}$. For clarity, we use $-$ to denote the complete don't care. This results in Table 3.6. We call this table the *compatibility table.*

Table 3.6: Compatibility Table

	B'_1	B'_2	B'_3	B'_4	B'_5	B'_6	B'_7	B'_8	B'_9
B_1	$\{1,2\}\{0\}$	$-$	$\{1\}\{1\}$	$\{1,2\}\{0\}$	$\{2\}\{1,2\}$	$-$	$-^*$	$-$	$-$
B_2	$-$	$-$	$-^*$	$-$	$-$	$-$	$\{1\}\{1\}$	$-$	$-$
B_3	$-$	$-$	$-$	$-$	$\{2\}\{1,2\}$	$-$	$-$	$-$	$-$
B_4	$-$	$-$	$\{0\}\{0,1\}$	$-$	$-$	$-$	$-$	$-$	$\{0\}\{0,1\}$
B_5	$-$	$-$	$\{1\}\{2\}$	$-$	$-$	$-$	$-$	$-$	$-$
B_6	$\{1\}\{0\}$	$\{1\}\{0\}$	$\{0\}\{1,2\}$	$\{1,2\}\{0\}$	$\{2\}\{1,2\}$	$\{0\}\{1,2\}$	$-$	$-$	$-$
B_7	$\{1\}\{0\}$	$\{1\}\{0\}$	$\{0\}\{1\}$	$-$	$-$	$\{0\}\{1,2\}$	$-$	$-$	$-^*$
B_8	$-$	$-$	$\{0\}\{1,2\}$	$-$	$-$	$\{0\}\{1,2\}$	$-$	$-$	$-$
B_9	$-$	$-$	$-$	$-$	$-$	$-$	$-$	$\{0,1\}\{2\}$	$-$
B_{10}	$-$	$-$	$\{0\}\{1\}$	$-$	$-$	$\{0\}\{1,2\}$	$-$	$-$	$-^*$

Definition 3.3.2 *A subset $S = \{B_{i_1}, \ldots, B_{i_k}\}$ of the blanket $\{B_i\}$ is called* compatible *if, in each column of the row compatibility table, the entries in rows $i_1, \ldots, i_k$ are compatible, i.e., they have a greatest lower bound.*

We now find a blanket β_{B_i} on the set $\{B_i\}$ where the blocks are compatible sets. Each block of this blanket is a set of blocks in β_U. We obtain a blanket β

on F by replacing each block by the union of its elements. We then use blanket β as β_{g_1} to obtain a single separation. Note this is similar to Łuba and Selvaraj's method [16], where blocks are merged in β_U to get β such that $\beta_V * \beta \leq \beta_Y$. The table reduction method is, however, different from the method in [16], because of the $\wedge$ operation we defined. In the next section, we will show that it is possible to get blankets which do not satisfy the conditions of Theorem 3.1.2.

One possible blanket β_{B_i} with one possible encoding is

$$\beta_{B_i} = \{\overline{B_1, B_2, B_3}^{0};\ \overline{B_4, B_6, B_7, B_8, B_9, B_{10}}^{1};\ \overline{B_5}^{2}\}.$$

The corresponding blanket β'_{g_1} with the encoding is

$$\beta'_{g_1} = \{\overline{3, 5, 6}^{0};\ \overline{1, 2, 3, 4, 5, 7}^{1};\ \overline{8}^{2}\}.$$

Note that this is not the same as β_{g_1} used in the last section. We can check that $\beta'_{g_1} * \beta_V \leq \beta_Y$. Thus this blanket corresponds to a single separation of f'.

Denote the block encoded l by δ_l where $l = 0, 1, 2$. We construct the *column compatibility table* as follows. The rows of the table are labeled δ_l and the columns labeled B'_j. The entry (l, j) of the table is the greatest lower bound of entries (i_k, j) in the row compatibility table where $\delta_l = \{B_{i_k}\}$. This table is shown in Table 3.7.

Table 3.7: Column Compatibility Table

	B'_1	B'_2	B'_3	B'_4	B'_5	B'_6	B'_7	B'_8	B'_9
δ_0	$\{1,2\}\{0\}$	–	$\{1\}\{1\}$	$\{1,2\}\{0\}$	$\{2\}\{1,2\}$	–	$\{1\}\{1\}$	–	–
δ_1	$\{1\}\{0\}$	$\{1\}\{0\}$	$\{0\}\{1\}$	$\{1,2\}\{0\}$	$\{2\}\{1,2\}$	$\{0\}\{1,2\}$	–	$\{0,1\}\{2\}$	$\{0\}\{0,1\}$
δ_2	–	–	$\{1\}\{2\}$	–	–	–	–	–	–

Now we use similar procedure as before to find compatible columns.

Definition 3.3.3 *A subset $S' = \{B'_{i_1}, \ldots, B'_{i_k}\}$ of blanket $\{B'_i\}$ is called* compatible *if in each row of the column compatibility table, the entries in columns $i_1, \ldots, i_k$ are compatible, i.e., they have a greatest lower bound.*

Merging compatible blocks, we obtain the following blanket, with a possible encoding, on the set $\{B'_j\}$:

$$\beta_{B'_j} = \{\overbrace{B'_1, B'_2, B'_4}^{00};\ \overbrace{B'_3, B'_6, B'_7, B'_9}^{01};\ \overbrace{B'_5}^{10};\ \overbrace{B'_8}^{11}\}.$$

The corresponding blanket β'_{g_2} with the encoding is

$$\beta'_{g_2} = \{\overbrace{1, 3}^{00};\ \overbrace{2, 6, 7, 8}^{01};\ \overbrace{5}^{10};\ \overbrace{4}^{11}\}.$$

Table 3.8: Final Table

	δ'_0	δ'_1	δ'_2	δ'_3
δ_0	$\{1,2\}\{0\}$	$\{1\}\{1\}$	$\{2\}\{1,2\}$	$-$
δ_1	$\{1\}\{0\}$	$\{0\}\{1\}$	$\{2\}\{1,2\}$	$\{0,1\}\{2\}$
δ_2	$-$	$\{1\}\{2\}$	$-$	$-$

As we can see, β'_{g_2} has four blocks as opposed to three blocks for β_{g_2}. With the domain of all variables assumed to be $\{0, 1, 2\}$, this means we will have one more output for g_2, and hence one more input to h. Using similar methods as we used in the construction of the column compatibility table, we merge columns this time and the result is shown in Table 3.8. We can get function h directly from this table: each input combination of h is formed by concatenating the encodings of δ_i and δ'_j while the output tuple is the entry (i, j) of the table. An input combination is omitted from the table for h if the output tuple is a complete don't care (denoted by $-$ in our tables).

3.4 Tabular Method Using Cubes

As we noted earlier, merging columns or rows in our tables is not quite the same as merging blocks. When using the table method, the blankets β_{g_1} and β_{g_2} we produce may not satisfy the conditions of Theorem 3.1.2. This situation arises because of the $\wedge$ operation we defined, as this allows us to merge blocks of β_V which are considered unmergeable using method introduced in Section 2.9. In fact, we now show an example where the blankets produced do not satisfy the conditions of Theorem 3.1.2.

Table 3.9: Row Compatibility Table

	δ'_0	δ'_1	δ'_2
B_1	$\{1,2\}\{0\}$	$\{1\}\{1\}$	$\{2\}\{1,2\}$
B_2	$\{1\}\{1\}$	$-$	$-$
B_3	$-$	$-$	$\{2\}\{1,2\}$
B_4	$-$	$\{0\}\{0,1\}$	$-$
B_5	$-$	$\{1\}\{2\}$	$-$
B_6	$\{1\}\{0\}$	$\{0\}\{1,2\}$	$\{2\}\{1,2\}$
B_7	$\{1\}\{0\}$	$\{0\}\{1\}$	$-$
B_8	$-$	$\{0\}\{1,2\}$	$-$
B_9	$\{0,1\}\{2\}$	$-$	$-$
B_{10}	$-$	$\{0\}\{1\}$	$-$

We still use Table 3.6, but this time we merge columns first and then rows. This is equivalent to exchanging the sets U and V. The following is a possible blanket

$\beta_{B'_j}$ with a possible encoding:

$$\beta_{B'_j} = \{\overset{0}{\overline{B'_1, B'_2, B'_4, B'_7, B'_8}};\ \overset{1}{\overline{B'_3, B'_6, B'_9}};\ \overset{2}{\overline{B'_5}}\}.$$

This gives

$$\beta_{g_2} = \{\overset{0}{\overline{1, 3, 4, 6}};\ \overset{1}{\overline{2, 6, 7, 8}};\ \overset{2}{\overline{5}}\}.$$

Note that this blanket does not satisfy $\beta_U * \beta_{g_2} \leq \beta_Y$. It is possible for this blanket to lead to a separation since we are dealing with non-disjunctive separations. However, the method we used earlier to get function g_2 becomes inadequate. Later we will show how that method is refined to handle this situation.

Following the method described in Section 3.3, we get a row compatibility table. It is shown in Table 3.9. One possible β_{B_i} and the corresponding β_{g_1} with possible encodings are as follows:

$$\beta_{B_i} = \{\overset{00}{\overline{B_1, B_3}};\ \overset{01}{\overline{B_2, B_4}};\ \overset{10}{\overline{B_5, B_9}};\ \overset{11}{\overline{B_6, B_7, B_8, B_{10}}}\}.$$

$$\beta_{g_1} = \{\overset{00}{\overline{3, 5, 6}};\ \overset{01}{\overline{6, 7}};\ \overset{10}{\overline{4, 8}};\ \overset{11}{\overline{1, 2, 3, 5, 7}}\}.$$

Table 3.10: New Final Table

	δ'_0	δ'_1	δ'_2
δ_0	$\{1,2\}\{0\}$	$\{1\}\{1\}$	$\{2\}\{1,2\}$
δ_1	$\{1\}\{1\}$	$\{0\}\{0,1\}$	$-$
δ_2	$\{0,1\}\{2\}$	$\{1\}\{2\}$	$-$
δ_3	$\{1\}\{0\}$	$\{0\}\{1\}$	$\{2\}\{1,2\}$

Table 3.11: Cubes for g_2

Block	Representative			z_1	Final Cubes		
B'_1	0	0	0	0	0	0	0
B'_2	0	0	0, 1	0	0	0	0, 1
B'_3	0	0	2	1	0	0	2
B'_4	0	0, 1	0	0	0	0, 1	0
B'_5	0, 2	1	1	2	0, 2	1	1
B'_6	0	0, 1, 2	2	1	0	0, 1, 2	2
B'_7	0, 1	0	2	0	1	0	2
B'_8	1, 2	2	2	0	1, 2	2	2
B'_9	0, 2	0	2	1	0, 2	0	2

We can now deduce our new final table from β_{g_1}. The result is shown in Table 3.10.

We now derive functions g_1, g_2 and h and show how consistency of the functions are achieved. We will be using the cube method described earlier, although with a slight modification.

The table for g_2 is shown in Table 3.11. As before, we consider the blocks of β_V. For each block B'_j, we first list its representative, and then the output corresponding to block B'_j. Finally, we list the final cubes after we resolve all the conflicts. The difference here is how we assign the output value to block B'_j. Note that before, the output is the value assigned to the block of β_{g_2} that contains B'_j. We have a choice for the output if more than one block of β_{g_2} contain B'_j. Using our current method, however, the possible outputs are the encodings of the blocks of $\beta_{B'_j}$ that contains B'_j. With the blanket $\beta_{B'_j}$ shown above, there is no option. For example, block

Table 3.12: Cubes for g_1

Block	Representative			z_2	z_3	Final Cubes		
B_1	0	0	0	0	0	0	0	0
B_2	0	0	0, 1	0	1	0	0	0, 1
B_3	0, 1	0	0, 2	0	0	0, 1	0	2
B_4	0, 1, 2	1	0, 2	0	1	0	1	0
				0	1	0, 1, 2	1	2
B_5	0	2	0	1	0	0	2	0
B_6	1	0	0	1	1	1	0	0
B_7	1	1	0	1	1	1	1	0
B_8	1, 2	0, 1	0	1	1	1, 2	0, 1	0
B_9	2	0, 2	1, 2	1	0	2	0, 2	1, 2
B_{10}	1, 2	1	0	1	1	1, 2	1	0

B_7' is contained in two blocks of β_{g_2}, so using the old method, we have a choice of assigning the output to be 0 or 1. Using our new definition of the output, only 0 is a possible output, since B_7' is only contained in block δ_0 of $\beta_{B_j'}$.

We can construct the cubes for g_1 similarly. The result is shown in Table 3.12. Function h is shown in Table 3.13.

Table 3.13: Matrix defining h

z_1	z_2	z_3	y_1	y_2
0	0	0	$1,2$	0
0	0	1	1	1
0	1	0	$0,1$	2
0	1	1	1	0
1	0	0	1	1
1	0	1	0	$0,1$
1	1	0	1	2
1	1	1	0	1
2	0	0	2	$1,2$
2	1	1	2	$1,2$

Proposition 3.4.1 *The triple (g_1, g_2, h) obtained using the above procedure is a double separation of f.*

Proof. By our construction, we can see that g_1, g_2 and h are all consistent, and $g_1(b^U)$ and $g_2(b^V)$ are minterms for any minterm b relevant to f.

Consider minterm $b \in \Sigma^n$ relevant to f. Suppose b^U exemplifies block B_i of β_U, and b^V exemplifies block B_j' of β_V. Entry (i, j) of the block multiplication table is then by definition $B_i \wedge B'_j = B_i \cap B'_j$. Further, since $B_i = F_{U \supseteq b^U}$ and $B'_j = F_{V \supseteq b^V}$, we get $B_i \cap B'_j = F_{X \supseteq b}$. Thus, the corresponding entry $T_{i,j}$ of the compatibility table is equal to $f(b)$.

Suppose that B_i is assigned the minterm e_1 and B'_j is assigned the minterm e_2. The value of $h(e_1 e_2)$ is an entry $\hat{y}$ in the final table. We know that $\hat{y}$ is the *glb* of some entries in the first compatibility table, one of which is entry $T_{i,j}$. Therefore,

$$f(b) \supseteq \hat{y} = h(e_1 e_2) = h(g_1(b^U), g_2(b^V)).$$

By our definition, we see that (g_1, g_2, h) is a double separation of f.

$\square$

Chapter 4

Survey of Decomposition Methods

Since the early 1950's, the problem of decomposition of functions has been studied extensively. The functions considered range from completely-specified, single-output Boolean functions to set functions discussed earlier.

4.1 Ashenhurst's Decomposition Method

In the early 1950's, Ashenhurst studied the problem of decomposing switching functions. His results were summarized in his paper [1] published in 1959. A *switching function* is a completely-specified, single-output Boolean function. Ashenhurst decomposes a switching function $f(x)$ into the form $h(u, g(v))$, where $U \cup V = X$ and U, V are disjoint. Set V is called the *bound set*, while set U is called the *free set*. Also, a complete theory of how complex disjunctive decompositions (defined later) are built up from simple ones is developed in [1]. Later, Curtis included Ashenhurst's work in his book [7], and extended the theory to non-disjunctive decompositions.

What Ashenhurst referred to as "decomposition" is really a separation in our terminology, as either U or V could be empty, or V could be a singleton set. We will not consider these trivial cases. A disjunctive decomposition of a switching function $f(x)$ is defined as follows:

Definition 4.1.1 *Let f, g and h be switching functions with n inputs, s inputs, and $n-s+1$ inputs, respectively, where $1 < s < n$. Let X be the set of inputs of f, and V be an s-element subset of X. We say (g, h) is a* disjunctive decomposition of f with respect to (U, V), *where $U = X - V$, if, for all minterms e,*

$$f(e) = h(e^U, g(e^V)). \tag{4.1}$$

Example 4.1.2 *We now demonstrate Ashenhurst's decomposition method. The function we consider, shown in Table 4.1, is taken from [1]. Note that the row numbers correspond to the decimal value of $x_1x_2x_3x_4$.*

Let $U = \{x_2, x_3\}$ and $V = \{x_1, x_4\}$. Table 4.1 is then transformed into Table 4.2. In this new table, the rows are labeled with values of x_2x_3, the columns are labeled with values of x_1x_4, while each entry is the corresponding output value. This table is equivalent to the compatibility table we considered earlier.

The number of different columns (rows) in the compatibility table is called the column (row) multiplicity. *Since function g can have at most two different outputs, we know that for a decomposition to exist, the column multiplicity can be no more than 2. In our case, the column multiplicity is 2, so a decomposition exists. Columns* 00 *and* 11 *are arbitrarily assigned the value* 0, *while the other two columns* 01 *and* 10 *are assigned the value* 1. *Functions g and h are shown in Tables 4.3 and 4.4.*

□

Table 4.1: Switching Function f

Row	x_1	x_2	x_3	x_4	y_1
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	0
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	1
15	1	1	1	1	0

Table 4.2: Compatibility Table for f

	x_1x_4			
x_2x_3	00	01	10	11
00	1	0	0	1
01	0	0	0	0
10	0	0	0	0
11	0	1	1	0

Table 4.3: Switching Function g

x_1	x_4	z
0	0	0
0	1	1
1	0	1
1	1	0

Table 4.4: Switching Function h

x_2	x_3	z	y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

For small n, decomposition charts can be used to find all possible sets U and V. A decomposition chart consists of a set of tables. Each table is associated with a partition (U, V) of X and is similar to a compatibility table. However, since these charts are generated without knowledge of f, each entry of the table is the row number of the output instead of the output itself. Table 4.5 shows one table in the four-variable decomposition charts. The underlined entries are those which correspond to output 1 for function f from Table 4.1. Using this table, we can check that the column multiplicity is 2 and hence a decomposition exists.

For a table in the decomposition chart, looking at it sideways is the same as exchanging the sets U and V. Thus, including the $2^n \times 1$ table, we only need to have 2^{n-1} tables for an n-variable decomposition chart. It is unrealistic to pre-generate the decomposition charts for large n, as the number of tables grows exponentially.

Suppose the pair (g, h) is a decomposition of f. There are two ways to further decompose function f. We can decompose function h, which is called *multiple*

Table 4.5: One Table in a Decomposition Chart

	x_1x_4			
	$\underline{0}$	1	8	$\underline{9}$
x_2x_3	2	3	10	11
	4	5	12	13
	6	$\underline{7}$	$\underline{14}$	15

decomposition; or, we can decompose function g, which is called *iterative decomposition*. Any combination of multiple and iterative decompositions is called *complex*.

For a switching function f with no non-essential variables, the following five theorems hold. These theorems give some possible bound sets when more than one decompositions of f are known. Together they characterize the structure of the complex decomposition of a function in terms of the simple ones.

Theorem 4.1.3 *Let $A \cup B \cup C = X$ where A, B and C are disjoint. If*

$$f(e) = h_1(e^A, g_1(e^{B \cup C})) = h_2(e^{A \cup B}, g_2(e^C)),$$

then

$$f(e) = h_1(e^A, g_3(e^B, g_2(e^C))),$$

where g_3 is a uniquely determined switching function with $1 + |B|$ inputs and for which

$$g_3(e^B, g_2(e^C)) = g_1(e^{B \cup C}).$$

That is, if one bound set is contained in another bound set, then we can find an iterative decomposition using these two bound sets.

Theorem 4.1.4 *Let $A \cup B = X$, where A and B are disjoint. If*

$$f(e) = h_1(e^A, g_1(e^B)) = h_2(e^B, g_2(e^A)),$$

then

$$f(e) = h_3(g_2(e^A), g_1(e^B)),$$

where h_3 is a uniquely determined switching function with 2 inputs. That is, if the union of two disjoint bound sets is the input set X, then we can find a multiple decomposition with the two bound sets.

Theorem 4.1.5 *Let $A \cup B \cup C = X$ where A, B and C are disjoint. If*

$$f(e) = h_1(e^{A\cup B}, g_1(e^C)) = h_2(e^{A\cup C}, g_2(e^B)),$$

then

$$f(e) = h_3(e^A, g_2(e^B), g_1(e^C)),$$

where h_3 is a uniquely determined switching function with $2 + |A|$ inputs. This is similar to the last theorem except that the union of the bound sets is not the input set X. The result is then the multiple decomposition has a non-empty free set.

Theorem 4.1.6 *Let $A \cup B \cup C = X$ where A, B and C are disjoint. If*

$$f(e) = h_1(e^A, g_1(e^{B\cup C})) = h_2(e^B, g_2(e^{A\cup C})),$$

then

$$f(e) = h_3(g_1'(e^A), g_2'(e^B), g_3'(e^C)),$$

where $h_3(g_1', g_2', g_3') = g_1' {*} g_2' {*} g_3'$, the operation $*$ is uniquely determined as AND, OR, or XOR, and the functions g_1', g_2', and g_3' are uniquely determined if $*$ is AND or OR, and uniquely determined up to complementation if $*$ is XOR. That is, if the two bound sets have non-empty intersection but the two free sets are disjoint, then function f has a complex decomposition of a special form.

Theorem 4.1.7 *Let $A \cup B \cup C \cup D = X$ where A, B, C and D are disjoint. If*

$$f(e) = h_1(e^{A\cup B}, g_1(e^{C\cup D})) = h_2(e^{A\cup C}, g_2(e^{B\cup D})),$$

then

$$f(e) = h_3(e^A, h'(g_1'(e^B), g_2'(e^C), g_3'(e^D))),$$

where h_3 is determined to within complementation of the variable h', and h' satisfies the same conditions as function h_3 in the previous theorem. This covers the most general case where neither the bound sets and the free sets are disjoint. The result is similar to the last theorem except there is a free set in the final decomposition.

Suppose (g, h) is a decomposition of f with respect to (U, V), and (g', h') is a decomposition of f with respect to (U', V'). Define $(g, h) \leq (g', h')$ if $V \subseteq V'$. This relation $\leq$ defines a partial order. Using the five theorems above, we can show that the set of simple decompositions of f forms a lattice with this partial order.

The decomposition chart method can be extended to incompletely specified, single-output Boolean functions. In this case, our definition of a decomposition has to be modified so that $h(e^U, g(e^V))$ may be either 0 or 1 if $f(e)$ is unspecified. In a table from the decomposition chart, we first underline those entries with output 1. Then we pick some entries which correspond to don't care output, so that by underlining those, we get a table with column (row) multiplicity at most 2.

4.2 Roth-Karp Minimization Algorithm

In 1962, Roth and Karp described an algorithm for designing gate-type combinational switching circuits without loops [18]. They use Ashenhurst's decomposition for minimization of such circuits.

The functions being considered are two-valued, and incompletely-specified. In fact, they may also have multiple outputs, although such a function is treated as several single-output functions. Functions are represented by sets of input cubes. Each set of cubes corresponds to a different output combination, and each cube is a 3-valued tuple, where the three values represent 0, 1 and don't care (denoted Φ), respectively. Note that the output corresponding to each set of cubes does not contain don't cares. In the case of single-output functions, there are only two sets of cubes, the ON-array, denoted C_1, and the OFF-array, denoted C_0.

Let f be a Boolean function with n inputs. Let $C_1 = \{a_1, \ldots, a_k\}$ be the ON-array and $C_0 = \{b_1, \ldots, b_l\}$ be the OFF-array. In this case, we have

$$F = \{a_1, \ldots, a_k, 1\} \cup \{b_1, \ldots, b_l, 0\},$$

where F is the set of all cubes for f. Note that there may be don't care minterms which are not contained in any cube in F. In Roth and Karp's method, the domain of f is the set of minterms contained in some cube in F. Thus, the don't care minterms are ignored. Suppose X is the input set of f, and $U \cup V = X$, where U and V are disjoint. Let d and e be such that $d \subseteq t^V$ and $e \subseteq t'^V$ where $t, t' \in F$. We say that d and e are *compatible* if for all $c \in \{0,1\}^r$, $f(cd) = f(ce)$. This definition can be extended to the case where $U \cap V$ is not empty, although in that case (c, d) and (c, e) correspond to valid inputs for f only if the overlapping variables agree. The decomposition method is based on checking compatibility of the minterms in $\{0,1\}^s$. This is similar to checking the compatibility of columns of a table in the decomposition chart. Since f is incompletely specified, this compatibility relation is no longer transitive. This is illustrated in Example 4.2.1.

Example 4.2.1 *Consider function f where*

$$C_1 = \{1\Phi10, 101\Phi, \Phi\Phi01\},$$

$$C_0 = \{\Phi 111, 0\Phi 1\Phi, 0\Phi\Phi 0\}.$$

Let $U = \{x_1, x_2\}$ and $V = \{x_3, x_4\}$. As we can see, a_1^U and b_1^U have nonempty intersection. Thus $a_1^V = 10$ and $b_1^V = 11$ are not compatible, as $f(1110) \neq f(1111)$. We can find all incompatible pairs by inspecting each pair (a_i, b_j). In fact, the only compatible pairs are $(00, 10)$ and $(00, 11)$. The compatibility relation is not transitive as 10 and 11 are not compatible.

□

For each $i = 1, \ldots, k$, define B_i^V to be the set of b_j^V such that b_j^U intersect a_i^U. Observe that if d is contained in a_i^V and e is contained in some b_j^V in B_i^V, then d and e are incompatible. For example, for function f specified by Table 4.6, we have $a_1^U = \Phi 01\Phi 0$. This cube has non-empty intersections with $b_1^U = 1\Phi\Phi 0\Phi$ and $b_5^U = \Phi\Phi\Phi\Phi 0$; hence $B_1^V = \{0\Phi, 00\}$.

We represent a_i^V by a binary vector $\mathbf{A}_i$ of length 2^s; the positions of $\mathbf{A}_i$ are numbered $0, 1, \ldots, s-1$. The kth position of $\mathbf{A}_i$ is 0 if the binary representation of k is contained in a_i^V. For example, since $a_1^V = 11$, the first three positions of $\mathbf{A}_1$ are 1s, while the fourth position, representing 11, is 0. Similarly, we represent B_i^V by a binary vector $\mathbf{B}_i$. For example, since $B_1^V = \{0\Phi, 00\}$, the positions of $\mathbf{B}_1$ corresponding to 00 and 01 are 0s, while the other positions are 1s. Hence, we have $\mathbf{B}_1 = 0011$.

A binary vector s is said to *cover* another binary vector s' if s has a 1 in each position where s' has a 1. Maximal compatible sets are derived with the following procedure:

1. Let S^0 be the set consisting of a single vector where every element is a 1.

2. For $i = 1, \ldots, k$, if B_i^V is non-empty, do the following: for each $s \in S^{i-1}$, form $s \cdot \mathbf{A}_i$ and $s \cdot \mathbf{B}_i$, where $\cdot$ denotes logical intersection. From this set of vectors, delete all vectors covered by other vectors; the resulting set is S_i. If $B_i^V = \emptyset$, then $S^i = S^{i-1}$.

3. The sets represented by elements of S^n are the maximal compatible sets.

We now give an informal explanation of this algorithm. As we know, minterms contained in a_i^V are incompatible with minterms contained in B_i^V. Thus, a maximal compatible set cannot have both minterms contained in a_i^V and minterms contained in B_i^V. That is, we must omit one of the two sets of minterms. The vector $\mathbf{A}_i$ represents the set where minterms in a_i^V are omitted, while the vector $\mathbf{B}_i$ represents the set where minterms in B_i^V are omitted. By multiplying these vectors $\mathbf{A}_i$ and $\mathbf{B}_i$ together, we are in fact taking the intersection of their corresponding sets. After repeated multiplication, we end up with sets that omit at least one minterm from each incompatible pair. Such sets are compatible sets. Also, only maximal sets remain as we remove the non-maximal sets during each iteration. This procedure is analogous to the method described in Section 2.9, except that we are finding compatible sets directly this time, instead of finding incompatible sets first. This procedure is demonstrated with the example below.

Example 4.2.2 *Consider the function specified by the ON- and OFF-arrays shown in Table 4.6. Let $U = \{x_1, \ldots, x_5\}$ and $V = \{x_6, x_7\}$. The calculation of the maximal compatible sets is shown in Table 4.7. Note that no calculation is required for $i = 4$ as B_4^V is empty.*

Thus, the maximal compatible sets are $\{00\}$, $\{01\}$, and $\{10, 11\}$. From the set of maximal compatible sets, we select a minimal cover for the set of minterms

Table 4.6: ON- and OFF-arrays for function f

C_1							C_0						
Φ	0	1	Φ	0	1	1	1	Φ	Φ	0	Φ	0	Φ
0	Φ	Φ	1	Φ	0	1	0	1	Φ	Φ	0	1	0
Φ	0	Φ	0	Φ	1	1	Φ	1	1	0	1	1	1
Φ	Φ	Φ	1	1	Φ	1	1	1	1	Φ	0	0	Φ
Φ	Φ	0	Φ	1	1	Φ	Φ	Φ	Φ	Φ	0	0	0
Φ	0	Φ	Φ	Φ	1	1	1	1	1	0	0	1	0
							1	0	0	0	Φ	0	0

contained in a_i^V *for some* $i = 1, \ldots, k$. *Function* g *is then obtained by assigning different outputs to the members of that cover.*

The above calculation can be simplified if, instead of considering each cube in C_1, *we consider a blanket on* C_1. *Then, instead of consider the minterm* 11 *three times when* $i = 1, 3, 6$, *we only need to consider it once with block* $\{t \in C_1 \mid t^V = 11\}$.

□

Suppose we have a collection of basic elements to use in the circuit, and each is associated with a cost. Then there is a realization of the function which has the minimum cost. This is because we can find a circuit which realizes our function, and there are only a finite number of circuits with a lower cost. Given an alphabetical ordering of all possible decomposition sequences, Roth-Karp algorithm does an exhaustive search to determine the sequence with the least cost. We also keep track of the minimum cost found so far and if a subsequence has a higher cost, all sequences containing that subsequence are discarded.

Table 4.7: Maximal Compatible Sets

i	a_i^V	B_i^V	$\mathbf{A}_i$	$\mathbf{B}_i$	S^i
1	11	0Φ	1110	0011	1110
		00			0011
2	01	10	1011	0101	1010
		00			0011
					0100
3	11	0Φ	1110	0011	1010
		00			0100
					0011
4	Φ1				1010
					0100
					0011
5	1Φ	0Φ	1100	0011	1000
		00			0100
					0011
6	11	0Φ	1110	0011	1000
		00			0100
					0011

This algorithm is indeed laborious. For small n (4 or 5), the solution can be reached within reasonable amount of time. When n is larger, we will have to accept approximate solutions. Heuristics can be used so that good solutions come up early in the calculation. For one such heuristic, see [23].

4.3 Decomposition Using BDDs

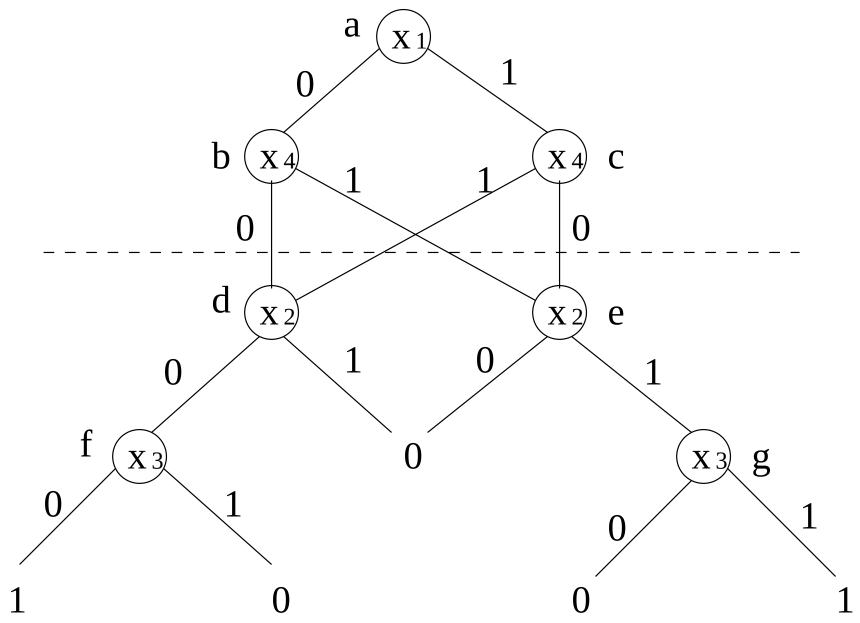
In [13], Lai, Pedram and Vrudhula described a method to decompose Boolean functions using binary decision diagrams (BDDs). A BDD is a directed graph with a single source node called the *root*. Each directed path from the root ends in a *terminal node* labeled either 0 or 1, while all other nodes, including the root, are called *non-terminal nodes*. The edges in a BDD are usually labeled 0 or 1. If edges are not labeled, then, by convention, the left edge is the 0-edge while the right edge is the 1-edge.

Denote by $\mathbf{f}$ the BDD representing a switching function $f(x_1, x_2, \ldots, x_n)$ with the variable order $x_1, \ldots, x_n$. Let $U = \{x_1, \ldots, x_r\}$, and $V = \{x_{r+1}, \ldots, x_n\}$. For clarity of notation, we assume the decomposition is of the form $h(g(u), v)$. We define a *cut set* as follows:

$$cut\_set(\mathbf{f}, U) = \{u \mid u = eval(\mathbf{f}, i), 0 \leq i < 2^r\},$$

where $eval(\mathbf{f}, i)$ is the node we reach when we start from the root and follow the path corresponding to the binary value of i. Intuitively, we get the cut set by "cutting" the BDD after the first r variables and collecting the nodes just below that cut. The size of $cut\_set(\mathbf{f}, U)$ corresponds to the column multiplicity in Ashenhurst's method.

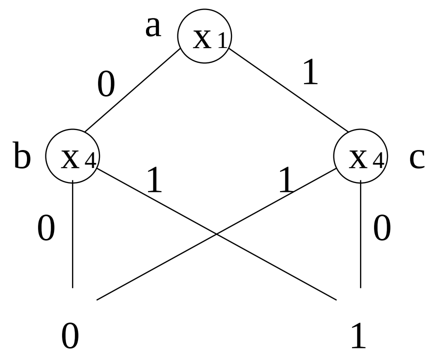
Figure 4.1: BDD for function f



Example 4.3.1 *We use the Boolean function specified by Table 4.1. As we have shown, there is a decomposition with* $U = \{x_1, x_4\}$ *and* $V = \{x_2, x_3\}$. *We use the variable order* $\{x_1, x_4, x_2, x_3\}$ *to construct the BDD, the result is Figure 4.1. For methods to construct such a BDD, see [6]. Note that each non-terminal node is labeled with a lower case letter. In this case, we have* $cut\_set(\mathbf{f}, U) = \{d, e\}$. *Since the size of the cut set is* 2*, we know there is a decomposition.*

To get the decomposition, we "cut" the BDD in half, indicated by a horizontal line in Figure 4.1. The top half include nodes for variables in $U = \{x_1, x_4\}$*, while the bottom half includes nodes for variables in* $V = \{x_2, x_3\}$. *To get function* g*, we assign values* 0 *and* 1 *to nodes* d *and* e*, and complete the top half of the BDD by connecting the edges coming out of the last variable (*x_4 *in this case) to the appropriate value. Suppose* d *is assigned* 0 *and* e *is assigned* 1*; the BDD for* g *is*

Figure 4.2: BDD for function g



shown in Figure 4.2.

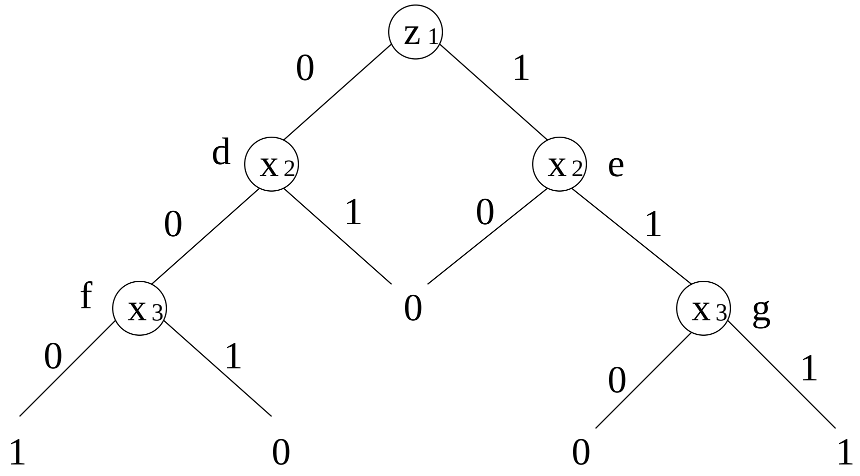
To get function h, we take the bottom half of the BDD and create a new node to be the root, the edges coming from this root are connected to the nodes assigned the corresponding value. The BDD for h is shown in Figure 4.3. Note that these two BDDs represent the same functions as those in Tables 4.3 and 4.4.

□

In the above example, we are given the sets U and V. Usually, this is not the case. If we have to make a new BDD for each possible pair (U, V), it would be too expensive. As variable order affects the size of the BDD, we may end up with a very large BDD. We can avoid this problem by performing operations on the BDD [13].

We can modify this method to handle incompletely specified functions. First we need to add a third terminal node to represent don't cares; next, after we find the cut set as before, we need to check the compatibility of elements of the cut set. This is because with the presence of don't cares, nodes may appear to be different yet compatible.

Figure 4.3: BDD for function h



This method can also be extended to handle non-disjunctive decompositions or multiple-output functions with different definitions of the cut set. These extensions are described by Lai, Pan, and Pedram in [12].

C. Files et al. described how to extend this to multi-valued functions in [8].

Chapter 5

Concluding Remarks

One big open problem in our decomposition method is how to choose sets U and V. Theorem 2.7.2 helps us with this problem, but as we can see, it eliminates very few possible pairs. As the size of the problem increases, many pairs (U, V) give us a decomposition. The question then becomes which decomposition put us in a better position when we want to further decompose the function. Roth and Karp's algorithm solves this problem by using exhaustive search. This is obviously not possible when the problem size increases. So heuristic methods must be employed.

As we have seen, using Shestakov's table method, we can get some decompositions which are cannot be derived with Łuba and Selvaraj's method. However, even the table method does not enable us to derive all possible decompositions. Maybe a different definition of the $\wedge$ operation will accomplish that.

Łuba and Selvaraj's method has been included in the logic synthesis tool DEMAIN dedicated to FPGA-based logic synthesis [5]. There is also a functional decomposer being developed by Perkowski et al. at Portland University.

Bibliography

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

[2] R. K. Brayton, et al., *Logic Minimization Algorithms for VLSI Synthesis,* Kluwer Academic Publishers, Boston, MA, 1984.

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

[4] J. A. Brzozowski and J. J. Lou, Blanket algebra for multiple-valued function decomposition, *Proceedings of the International Workshop on Formal Languages and Computer Systems,* Kyoto, Japan, March 18-21, 1997. (invited paper, to be published by World Scientific)

[5] J. A. Brzozowski and T. Łuba, Decomposition of Boolean Functions Specified by Cubes, *Research Report CS–97–01,* Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, January 1997.

[6] R. Bryant, Graph Based Algorithm for Boolean Function Manipulation, *IEEE Transaction on Computers, C-35(8),* pp. 677–691, August 1986.

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

[8] C. Files, R. Drechsler, M. A. Perkowski, Functional Decomposition of MVL Functions using Multi-Valued Decision Diagrams, *IEEE International Symposium on Multi-Valued Logic,* pp. 27–32, May 28-30, 1997.

[9] S. Grygiel, M. Perkowski, M. Marek-Sadowska, T. Łuba, and L. Jóźwiak, Cube Diagram Bundles: A New Representation of Strongly Unspecified Multiple-Valued Functions and Relations, *Proc. IEEE International Symposium on Multiple-Valued Logic,* Antigonish, Nova Scotia, Canada,

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

[11] M. Li and P. Vitanyi, *An Introduction to Kolmogorov Complexity and Its Applications,* Springer-Verlag, New York, NY, 1997.

[12] Y.-T. Lai, K.-R. Pan, and M. Pedram, OBDD Based Function Decomposition: Algorithms and Implementation, *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems, Vol. 15, No. 8,* pp. 977–990, August 1996.

[13] Y.-T. Lai, M. Pedram and S. B.K. Vrudhula, BDD Based Decomposition of Logic Functions with Application to FPGA Synthesis, *Proc. 30th ACM/IEEE DAC,* pp. 642–647, June 1993.

[14] J. J. Lou and J. A. Brzozowski, A Formalization of Shestakov's Decomposition, *Research Report CS–98–03,* Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, February 1998.

[15] T. Łuba, Decomposition of Multiple-Valued Functions, *Proc. 25th International Symposium on Multiple-Valued Logic,* Bloomington, Indiana, pp. 256–261, May 1995.

[16] 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.

[17] M. Perkowski et al., Decomposition of Multiple-Valued Relations, *Proc. IEEE International Symposium on Multiple-Valued Logic,* Antigonish, Nova Scotia, Canada, pp. 13–18, May 1997.

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

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

[20] C. Scholl and P. Molitor, Communication Based FPGA Synthesis for Multi-Output Boolean Functions, *Proceedings of the Asia and South Pacific Design Automation Conference,* Makuhari, Chiba, Japan, pp.279–287, August 1995.

[21] E. Shestakov, Decomposition of Systems of Completely Defined Boolean Functions by Argument Covering, *Automatic Control and Computer Sciences,* Vol. 28, No. 1, pp. 12–20, 1994.

[22] E. Shestakov, Decomposition of Systems of Incompletely Defined Boolean Functions by Argument Cover, *Automatic Control and Computer Sciences,* Vol. 28, No. 6, pp. 4–15, 1994.

[23] 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,* San Francisco, pp. 65–69, 1995.

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