

8-10-1992

Minimization of Exclusive Sum of Products Expressions for Multiple-Valued Input Incompletely Specified Functions

Ning Song
Portland State University

Follow this and additional works at: https://pdxscholar.library.pdx.edu/open_access_etds



Part of the [Electrical and Computer Engineering Commons](#)

Let us know how access to this document benefits you.

Recommended Citation

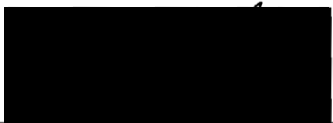
Song, Ning, "Minimization of Exclusive Sum of Products Expressions for Multiple-Valued Input Incompletely Specified Functions" (1992). *Dissertations and Theses*. Paper 4684.
<https://doi.org/10.15760/etd.6568>

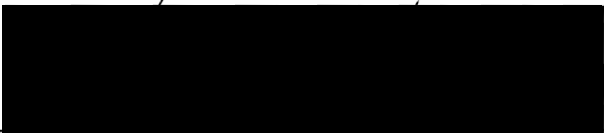
This Thesis is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: pdxscholar@pdx.edu.

AN ABSTRACT OF THE THESIS OF Ning Song for the Master of Science in Electrical and Computer Engineering presented August 10, 1992.

Title: Minimization of Exclusive Sum of Products Expressions for Multiple-Valued Input Incompletely Specified Functions

APPROVED BY THE MEMBERS OF THE THESIS COMMITTEE:


Marek A. Perkowski, Chair


Michael A. Driscoll


James L. Hein

In recent years, there is an increased interest in the design of logic circuits which use EXOR gates. Particular interest is in the minimization of arbitrary Exclusive Sums Of Products (ESOPs). Functions realized by such circuits can have fewer gates, fewer connections, and take up less area in VLSI and especially, FPGA realizations. They are also easily testable.

So far, the ESOPs are not as popular as their Sum of Products (SOP) counterparts. One of the main reasons is that the problem of the minimization of ESOP circuits was

traditionally an extremely difficult one. Since exact solutions can be practically found only for functions with not more than 5 variables the interest is in approximate solutions. Two approaches to generate sub optimal solutions can be found in the literature. One approach is to minimize sub-families of ESOPs. Another approach is to minimize ESOPs using heuristic algorithms. The method we introduced in this thesis belongs to the second approach, which normally generates better results than the first approach.

In the second approach, two general methods are used. One method is to minimize the coefficients of Reed-Muller forms. Another method is to perform a set of cube operations iteratively on a given ESOP. So far, this method has achieved better results than other methods.

In this method (we call it cube operation approach), the quality of the results depends on the quality of the cube operations. Different cube operations have been invented in the past a few years. All these cube operations can be applied only when some conditions are satisfied. This is due to the limitations of the operations. These limitations reduce the opportunity to get a high quality solution and reduce the efficiency of the algorithm as well. The efforts of removing these limitations led to the invention of our new cube operation, exorlink, which is introduced in this thesis. Exorlink can be applied on any two cubes in the array without condition. All the existing cube operations in this approach are included in it. So this is the most general operation in this approach.

Another key issue in the cube operation approach is the efficiency of the algorithm. Existing algorithms perform all possible cube operations, and give little guide to select the operations. Our new algorithm selectively performs some of the possible operations. Experimental results show that this algorithm is more efficient than existing ones. New algorithms to minimize multiple output functions and especially incompletely specified ESOPs are also presented. The algorithms are included in the program EXORCISM-MV-2, which is a new version of EXORCISM-MV.

EXORCISM-MV-2 was tested on many benchmark functions and compared to the results from literature. The program in most cases gives the same or better solutions on binary and 4-valued completely specified functions. More importantly, it is able to efficiently minimize arbitrary-valued and incompletely specified functions, while the programs from literature are either for completely specified functions, or for binary variables. Additionally, as in Espresso, the number of variables in our program is unlimited and the only constraint is the number of input cubes that are read, so very large functions can be minimized.

Based on our new cube operation and new algorithms, in the thesis, we present a solution to a problem that has not yet been practically solved in the literature: efficient minimization of arbitrary ESOP expressions for multiple-output multiple-valued input incompletely specified functions.

**MINIMIZATION OF EXCLUSIVE SUM OF PRODUCTS EXPRESSIONS FOR
MULTIPLE-VALUED INPUT INCOMPLETELY SPECIFIED FUNCTIONS**

by
NING SONG

A thesis submitted for the partial fulfillment of the
requirements for the degree of

**MASTER OF SCIENCE
in
ELECTRICAL AND COMPUTER ENGINEERING**

Portland State University
1993

TO THE OFFICE OF GRADUATE STUDIES:

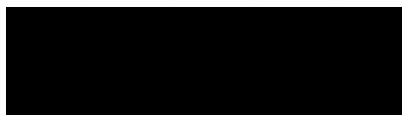
The members of the Committee approve the thesis of Ning Song presented
August 10, 1992.



Marek A. Perkowski, Chair



Michael A. Driscoll

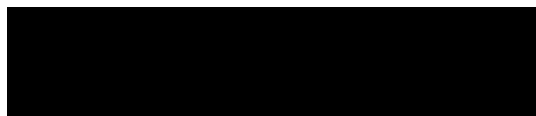


James L. Hein

APPROVED:



Rolf Schaumann, Chairman, Department of Electrical Engineering



Roy W. Koch, Vice Provost for Graduate Studies and Research

ACKNOWLEDGMENT

I would like to thank Dr. Marek A. Perkowski, my advisor, who introduced me to the area of Design Automation, guided me, and encouraged me in my research.

I would also like to thank Dr. Michael A. Driscoll, Dr. James L. Hein and Dr. W. Robert Daasch for their valuable comments which helped to improve this thesis.

TABLE OF CONTENTS

		PAGE
ACKNOWLEDGMENT		iii
LIST OF TABLES		vi
LIST OF FIGURES		viii
LIST OF SPECIAL SYMBOLS		xi
 CHAPTER		
I	INTRODUCTION	1
II	BASIC CONCEPTS AND DEFINITIONS	6
	II.1 Sets	6
	II.2 Functions	8
	II.2.1 Multiple Valued Functions	
	II.2.2 Cubical Representation	
	II.2.3 Incompletely Specified Functions	
	II.2.4 Multiple Output Functions	
	II.3 Operations	25
	II.3.1 Set Theoretic Operations	
	II.3.2 Cube Operations	
III	MULTIPLE-VALUED INPUT EXCLUSIVE SUMS OF PRODUCTS MINIMIZATION	31
	III.1 The Cost Functions	31
	III.2 The Properties of the ESOP	32
	III.3 Basic Ideas to Minimize the ESOP	33
	III.3.1 Removing Two Equal cubes	
	III.3.2 Combining Two Cubes which Differ in One Variable	
	III.3.3 Reshaping Two Cubes which Differ by 2	

	III.3.4 Increasing the Number of Cubes	
	III.4 The Operations Used in Exorcism	35
	III.4.1 Primary Xlinking	
	III.4.2 Secondary Xlinking	
	III.4.3 Unlinking	
	III.5 The Operations Used in EXMIN	40
IV	THE MULTIPLE-VALUED EXORLINKING OPERATION	45
	IV.1 The Formula	45
	IV.2 Difference 1 Exorlinking	50
	IV.2.1 Difference 1 Distance 1 Exorlinking	
	IV.2.2 Difference 1 Distance 0 Exorlinking	
	IV.3 Difference 2 Exorlinking	52
	IV.4 Difference 3 Exorlinking	61
	IV.5 Summary	62
V	ALGORITHM OF EXORCISM-MV-2 PROGRAM	64
	V.1 The Algorithm of EXORCISM	64
	V.2 The Algorithm of EXMIN	64
	V.3 The New Algorithm	66
	V.4 Minimization of Multiple Output Functions	70
	V.5 Minimization of Incompletely Specified Functions	73
	V.6 The Algorithm of EXORCISM-MV-2	81
VI	EVALUATION OF RESULTS OF EXORCISM-MV-2	85
VII	CONCLUSION	93
	REFERENCES	94
	APPENDIX	98

LIST OF TABLES

TABLE		PAGE
I	An Example of Binary Input Binary Output Function	10
II	An Example of Multiple-Valued Input Binary Output Function	11
III	Examples of Incompletely Specified Functions	20
IV	An Example of Multiple-Valued Input Binary Output Incompletely Specified Function	21
V	An Example of Multiple Output Function	23
VI	ON-Array of the Multiple Output Function	24
VII	Converting a Multiple Output Function to a Single Output Function	24
VIII	Comparison of Exorlinking with Xlinking	59
IX	Comparison of Exorlinking with Operations in EXMIN	59
X	Multiple Output Function	72
XI	Experimental Results of Functions with 1 Bit and 2 Bit Decoders	85
XII	Experimental Results of Functions with 3 Bit Decoders	86
XIII	Experimental Results of EXORCISM-MV-2	87
XIV	Comparison of EXORCISM-MV-2 with EXMIN	90
XV	Comparison of EXORCISM-MV-2 with EXMIN-2	90
XVI	Comparison of EXORCISM-MV-2 with EXMIN-2 for the Same Cost Functions	91

XVII	Comparison of EXORCISM-MV-2 with EXMIN-2 for the Same Execution Time	91
XVIII	Comparison of Our Results with ESPRESSO	92
XVII	Comparison of Minimization of ON-Cubes with Minimization of ON- and DC-Cubes	92

LIST OF FIGURES

FIGURE		PAGE
1.	ON- and OFF-arrays of function f	11
2.	Karnaugh map for a binary input function of 3 variables	13
3.	Map for a multiple-valued input function of 2 variables	13
4.	Karnaugh map for binary input cubes	17
5.	Map for multiple-valued input cubes	18
6.	Map for a multiple-valued input incompletely specified function	21
7.	ON-, OFF-, and DC-array of an incompletely specified function	21
8.	ESOPs for a multiple-valued input incompletely specified function	23
9.	Map for a multiple output function	25
10.	Map and circuit for a multiple output function	26
11.	Example of a disjoint sharp operation	30
12.	Combining two cubes into one cube	34
13.	Reshaping two cubes	34
14.	Procedure of distance 1 primary xlinking	36
15.	Procedure of distance 2 primary xlinking	37
16.	Procedure of distance 1 secondary xlinking	39
17.	Procedure of distance 2 secondary xlinking	40
18.	X-MERGE	41

19.	RESHAPE	41
20.	DUAL-COMPLEMENT	42
21.	X-EXPAND-1	43
22.	X-EXPAND-2	43
23.	X-REDUCE-1	44
24.	X-REDUCE-2	44
25.	An example of exorlinking two terms	46
26.	An example of exorlinking two cubes	48
27.	Difference 1 distance 1 exorlinking	51
28.	An example of difference 1 distance 0 exorlinking ($S_i \supset R_i$) ...	51
29.	An example of difference 1 distance 0 exorlinking ($S_i \not\bowtie R_i$)	52
30.	Difference 2 exorlinking ($T_S \otimes T_R$ and $T_R \otimes T_S$)	53
31.	Difference 2 exorlinking ($S_i \cap R_i = \emptyset, S_j \cap R_j = \emptyset$)	54
32.	Difference 2 exorlinking ($S_i \cap R_i = \emptyset, S_j \supset R_j$)	54
33.	Difference 2 exorlinking ($S_i \cap R_i = \emptyset, S_j \not\bowtie R_j$)	55
34.	Difference 2 exorlinking ($S_i \supset R_i, S_j \supset R_j$)	56
35.	Difference 2 exorlinking ($S_i \supset R_i, S_j \subset R_j$)	57
36.	Difference 2 exorlinking ($S_i \supset R_i, S_j \not\bowtie R_j$)	58
37.	Difference 2 exorlinking ($S_i \not\bowtie R_i, S_j \not\bowtie R_j$)	58
38.	An example of ESOP minimization by difference 2 exorlinking	60
39.	An example of ESOP minimization by difference 3 exorlinking	63
40.	An example of infinite loop in EXMIN	65
41.	An example of performing all possible difference 2 operations .	67
42.	Karnaugh maps for Example III.23	68
43.	Conditionally performing difference 2 exorlinking	69
44.	Karnaugh maps corresponding to Figure 43	70

45.	Example of minimizing an incompletely specified function	73
46.	Linking DC-cubes	74
47.	The position of DC-cubes	74
48.	The size of DC-cubes	75
49.	Linking DC-cubes by Saul's algorithm	76
50.	An ON-cube is equal to a DC-cube	78
51.	An ON-cube is contained by a DC-cube	78
52.	Minimization of an incompletely specified function	80
53.	Scatter plot of number of terms versus execution time	88
54.	Scatter plot of number of variables versus execution time	89

LIST OF SPECIAL SYMBOLS

Page number refers to the page on which the symbol is defined or first used.

SYMBOL		PAGE
Set Relations		
\in	Is a member of	6
\subseteq	Is a subset of	6
\subset	Is a proper subset of	7
\emptyset	Empty set	7
\times	Cartesian product	7
\bowtie	Two sets are overlapping	27
$\not\subseteq$	Is not a subset of	28
Set Operations		
\cup	Union of two sets	25
\cap	Intersection of two sets	25
$-$	Difference of two sets	26
\oplus	Exclusive-OR of two sets	27
\neg	Complement of a set	29
Logic Operations		
\oplus	Exclusive-OR operator	11
Cube Operations		
\cup	Supercube of two cubes	27
\cap	Intersection of two cubes	27

$\#d$	disjoint sharp of cubes	28
\neg	Complement of a cube	29
\oplus	Exclusive-OR of cubes	35
Φ	Primary xlinking	35
\ominus	Secondary xlinking	38
\otimes	Exorlinking	45

CHAPTER I

INTRODUCTION

In recent years, there is an increased interest in the design of logic circuits which use EXOR gates [Hell 88, Sasa 90b]. Particular interest is in the minimization of arbitrary Exclusive Sums Of Products (ESOPs) and their various canonical sub-families such as Consistent Generalized Reed-Muller canonical forms (CGRMs) [Lui 92, Sara 92], Kronecker Reed-Muller forms [Gree 91, Gill 91], Canonical Restricted Mixed Polarity forms (CRMPs) [Csan 92], and other. Functions realized by such circuits can have fewer gates, fewer connections, and take up less area in VLSI and especially, FPGA realizations. They are also easily testable [Fuji 86, Prad 87]. It was shown, both theoretically and experimentally [Sasa 90b, Sasa 91a, Sasa 91b, Sara 92, Salm 89] that ESOPs have on average smaller numbers of terms for both "worst case" and "average" Boolean functions. It was also shown that ESOPs and all their sub-families have their counterparts in logic with multiple-valued inputs: Multiple-valued Input ESOPs (MIESOPs) [Perk 89, Sasa 90a], Multiple-valued Input Generalized Reed-Muller forms (MIGRMs) [Scha 91], Multiple-valued Input Kronecker Reed-Muller forms (MIKRM)s [Scha 92], Multiple-valued Input Generalized Reed-Muller Trees (MIGRMTs) [Perk 91], and others [Perk 92]. Logic with multiple-valued inputs (mv logic, for short) generalizes the classical Boolean logic and finds many important applications in logic design [Sasa 78, Sasa 81, Sasa 87, Rude 85]. MIESOPs are never worse than ESOPs, and they were shown to be superior on several classes of functions [Sasa 90b, Sasa 91a, Sasa 91b]. Minimized ESOP and MIESOP expressions are the starting points to technology mapping for new EXOR-based low granularity FPGAs such as CLi 6000 series from Concurrent Logic

Inc. [CLi 91]. ESOPs are also a starting point to ESOP factorization procedures which produce multi-level AND-EXOR circuits [Saul 91]. Finally, ESOPs are used as the internal function representation in GATEMAP [Salm 89] and POLO [Perk 92] design automation systems.

So far, the ESOPs are not as popular as their Sum of Products (SOP) counterparts. One of the main reasons is that the problem of the minimization of ESOP circuits was traditionally an extremely difficult one. Papakonstantinou [Papa 79] gave an exact algorithm for 4 input functions. The algorithm from [Perk 90] has theoretically no limits on the number of variables but 4 is its practical limit. Since exact solutions can be practically found only for functions with not more than 5 variables the interest is in approximate solutions. Two approaches to generate sub optimal solutions can be found in the literature. One approach is to minimize sub-families of ESOPs. Another approach is to minimize ESOPs using heuristic algorithms. Efficient programs for sub-families of ESOPs were given in [Bess 83, Csan 92, Scha 91]. Heuristic computer programs have been presented in [Bess 91, Flei 83, Flei 87, Hell 88, Perk 89, Robi 82, Saul 90]. In these programs, two general methods are used. One method is to minimize the coefficients of Reed-Muller forms [Wu 82, Bess 83, Robi 82]. Another method is to perform a set of cube operations iteratively on minterms or disjoint cubes. Fleisher, et. al. [Flei 87] presented an algorithm which starts from positive Reed-Muller forms and performs three cube operations iteratively. The paper [Hell 88] introduced a new cube operation - crosslink, and presented an algorithm based on this new operations. The algorithm from [Hell 88] was next improved in [Perk 89], and also extended for the case of logic with multiple-valued inputs. The unlink operation has also been added. The unlink operation was efficiently implemented in [Saul 90]. A few more cube operations were also included in an independent realization by Sasao [Sasa 90a], which is the only other author that published on the most general ESOP minimization algorithms for the

multiple-valued input logic. Contrary to [Sasa 90a], however, the program described in [Perk 89] assumed that the number of truth values for each variable can be different. Although the theory presented by Sasao in [Sasa 90a] is general, the EXMIN algorithm implemented by him, for efficiency reasons, accepts only 2-valued and 4-valued variables. It also can not deal with incompletely specified functions.

In this thesis, we present a solution to a problem that has not yet been practically solved in the literature: efficient minimization of arbitrary ESOP expressions for multiple-output multiple-valued input incompletely specified functions. This thesis describes an approximate method that yields especially good results for the minimization of strongly unspecified multi-output logic functions with multiple-valued inputs and binary-valued outputs. Our program, EXORCISM-MV-2 is a new version of EXORCISM-MV presented in [Perk 89]. The algorithm currently used in EXORCISM-MV-2 was created after much experimentation with previous variants. A new cube operation - exorlinking is also introduced to support the new algorithm.

Our method and program are applicable to any type of multiple-valued input functions, and each input variable can have an arbitrary number of logic values. For simplicity, 4-valued logic will be presented in most of the multiple valued logic examples. This variant finds, among others, applications in the minimization of PLAs in which pairs of inputs are implemented via 2-by-4 decoders (2 inputs, 4 outputs).

AND/EXOR circuits with various kinds of function generators on inputs to the AND plane are used to realize the MIESOP expressions minimized here. Such circuits can have smaller complexity than both the circuits which implement mixed-polarity ESOP expressions [Perk 89], AND/XOR-Fields [Froe 91], PLAs with decoders [Sasa 84], and networks with two variable function generators [Sasa 86]. The most important advantage of the presented approach is that it can produce FPGA circuits that are superior (in terms of both speed and area) to those obtained using other methods for the logic

with multiple-valued inputs and are also very easily testable. There are several ways to realize MIESOP circuits in modern technologies: XOR PLAs (XPLAs, AND/XOR-Fields), standard cells, optical and Josephson junction AND/EXOR PLAs, and especially various types of FPGAs (like those from Xilinx, Actel, CLi, Algotronix, and Texas Instruments) [FPGA 92]. A particularly well-suited FPGA is the recently announced CLi 6000 series from Concurrent Logic Inc. [CLi 91], since one of the most efficient uses of its basic logic cell is the 3-input AND/EXOR function [Wu 92].

In Chapter II, the basic concepts and definitions related to this thesis are presented. The main concepts in this chapter are the multiple-valued input binary output incompletely specified functions, the exclusive-OR sum-of-products expressions (ESOPs), and the corresponding cube notations.

In Chapter III, we first introduce the basic ideas of ESOP minimization. Two major programs for ESOP minimization, EXORCISM and EXMIN are then presented. The operations used in these two programs are discussed in detail with examples.

Our new cube operation, exorlinking, is presented in Chapter IV. Exorlinking generalizes all cube operations such as xlinking, unlinking, or X-merge introduced in EXORCISM [Hell 88, Perk 89] and EXMIN [Sasa 90a]. Contrary to the operations in EXORCISM and EXMIN, which can be applied in certain conditions, we proved in this chapter that exorlinking can be applied in any two cubes in arbitrary distance. The procedure of the exorlinking operation is discussed, different examples are given, and comparisons with operations in EXORCISM and EXMIN are shown.

In Chapter V, we first introduce the algorithms used in EXORCISM and EXMIN. Then our new algorithm used in EXORCISM-MV-2 is discussed. The major advantage of our new algorithm is that it gives priority to those difference 2 operations which will directly reduce the number of cubes in the array. By this way, our program can achieve better results in shorter time as compared to the former algorithms. Our algorithms to

handle multiple output functions and incompletely specified functions are also discussed in this chapter.

Chapter VI shows the experimental results. Then in the last chapter, Chapter VII, we give the conclusion. The "man page" about how to use our program is presented in the appendix with examples.

CHAPTER II

BASIC CONCEPTS AND DEFINITIONS

In this chapter, we give some basic definitions and discuss some basic concepts. These concepts and definitions are necessary for the following chapters. More detailed discussion on these topics can be found in [Stol 79, Grät 79, Brow 90].

II.1. SETS

A *set* is a collection of objects; the objects in the collection are called *elements* of the set. Here we consider only finite sets, i.e., sets possessing a finite number of elements. We write

$$x \in P$$

to denote that x is an element of a set named P .

If all the elements of set P are also elements of set Q , then we say that P is included in (or is a subset of) Q and we represent the *relation* between P and Q by writing

$$P \subseteq Q.$$

If $P \subseteq Q$ and $Q \subseteq P$, we write

$$P = Q.$$

$P = Q$ indicates that sets P and Q contain exactly the same elements.

If $P = Q$ does not hold, we write

$$P \neq Q.$$

$P \neq Q$ indicates that sets P and Q do not contain exactly the same elements.

If P is a subset of Q , and Q contains at least one element that does not belong to P , then P is called a proper subset of Q , and we write

$$P \subset Q$$

to indicate the relation between P and Q . By definition, $P \subset Q$ means $P \subseteq Q$ and $P \neq Q$.

The empty set is denoted by \emptyset , and is the set comprising no elements at all. The empty set is included in every set, i.e.,

$$\emptyset \subseteq P$$

for any set P .

We write $P(k)$ to indicate that P contains k elements. For instance, $P(2)$ denotes that P contains two elements.

The *Cartesian* (direct, cross) product of sets P and Q , written $P \times Q$, is the set defined by

$$P \times Q = \{ (p, q) \mid p \in P \text{ and } q \in Q \}.$$

Thus

$$\{p, q\} \times \{p, q, r\} = \{(p, p), (p, q), (p, r), (q, p), (q, q), (q, r)\}.$$

We write $P^n(k)$ to signify the n -fold Cartesian product of $P(k)$ with itself, i.e.,

$$P^n(k) = P(k) \times P(k) \times \cdots \times P(k).$$

In more general cases, we write P^n to denote $P_1(k_1) \times P_2(k_2) \times \cdots \times P_n(k_n)$, where $P_1(k_1), P_2(k_2), \dots, P_n(k_n)$ are sets, and they may or may not contain the same elements.

If $P_1(k_1) = P_2(k_2) = \dots = P_n(k_n)$, then $P^n(k) = P^n$. So, $P^n(k)$ is a special case of P^n .

II.2. FUNCTIONS

II.2.1. Multiple Valued Functions

A *completely specified function* (function for short) f from set P^n to set Q is a subset of $P^n \times Q$, such that for each $p \in P^n$, there is a $q \in Q$, and $(p, q) \in f$. We write

$$f : P^n \rightarrow Q$$

to denote that f takes elements from the sets P_1, P_2, \dots, P_n to yield elements in the set Q .

Given a function $f : P^n \rightarrow Q$, P^n is called the *domain* of the function; Q is called the *co-domain* of the function. If all of the sets P_i ($i = 1, \dots, n$) in the domain P contain two elements, the function is a *binary input function*. For example, $f : P_1(2) \times P_2(2) \rightarrow Q$ is a binary input function. If some of the sets in the domain contain more than two elements, the function is a *multiple-valued input function*. For example, $f : P_1(4) \times P_2(4) \rightarrow Q$ and $f : P_1(2) \times P_2(4) \rightarrow Q$ are multiple-valued input functions. If the co-domain of the function contains two elements only, it is a *binary output function*. A binary output function is also called a *switching function*. If the co-domain contains more than two elements, it is a multiple-valued output function. Only the binary output functions will be discussed in this thesis.

A symbol that may represent any one of the elements of set P_i or Q is called a *variable*.

Example II.1: Given a set $P_i = \{0,1,2,3\}$ which contains 4 elements. If X can assume values 0, 1, 2, or 3, then X is a 4-valued variable.

A variable corresponding to a set in the domain is called an input variable. A variable corresponding to a set in the co-domain is called an output variable.

If the domain of a function is P^n , we call the function an *n-input function*. If $n = 1$, the function is a *single input function*. If the domain of the function is $P^n(2)$, it is a *n-*

binary-input function. On the other hand, if one or more sets in the domain have more than two elements, it is a *multiple-valued multiple-input function*. If the domain is $P^n(k)$, its radix is fixed. If the domain is $P_1(k_1) \times P_2(k_2) \times \dots \times P_n(k_n)$, and $\exists (i, j)$ such that $k_i \neq k_j$, the radix is mixed.

If p_i is a positive integer, $P_i = \{0, 1, \dots, p_i-1\}$ is a set in the domain, and X_i is a variable corresponding to P_i , then for any subset $S_i \subseteq P_i$, $X_i^{S_i}$ is a *literal* of variable X_i representing a function such that

$$X_i^{S_i} = \begin{cases} 1 & \text{if } X_i \in S_i \\ 0 & \text{if } X_i \notin S_i. \end{cases}$$

A literal is a function that maps from domain P_i to co-domain $Q(2)$. The variable X_i can take one of the values of 0 through p_i-1 . We write $X = 0$ to indicate that X takes value of 0, $X = 1$ to indicate that X takes value of 1, etc. S_i specifies the set of values of X_i for which $X_i^{S_i}$ is true. For instance, for a binary logic, $P_i = \{0,1\}$:

X_i^\emptyset is false for both $X_i = 0$ and $X_i = 1$,

X_i^1 is true only for $X_i = 1$,

X_i^0 is true only for $X_i = 0$,

$X_i^{0,1}$ is true for both $X_i = 0$ and $X_i = 1$.

For a binary logic, we also write X_i^1 as X_i and X_i^0 as \bar{X}_i . So, in the case of binary logic, a literal can be represented by a variable or its complement.

For a 4-valued logic, $p_i = \{0,1,2,3\}$

$X_i^{0,1,2,3}$ is true for any value of X_i ,

$X_i^{0,2}$ is true for $X_i = 0$ or $X_i = 2$,

$X_i^{0,2}$ is false for $X_i = 1$ or $X_i = 3$.

A product of literals, $X_1^{S_1} X_2^{S_2} \dots X_n^{S_n}$, is referred to as a *product term* (also called *term* or *product* for short). A product term that includes literals for all function

variables X_1, X_2, \dots, X_n is called a *full term*. A sum of products is denoted as a *sum-of-products (SOP)* expression while a product of sums is called a *product-of-sums (POS)* expression. An Exclusive-Or of products will be called an *ESOP*. The name *Multiple-Valued Input Exclusive Sum of Products Form (MIESOP)* for short) will be used if we want to emphasize that the ESOP takes multiple-valued inputs.

Example II.2: Given a function f as specified by Table I.

TABLE I
AN EXAMPLE OF BINARY INPUT BINARY OUTPUT FUNCTION

x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Table I is called a *truth table*. A truth table enumerates the values of functions for each of their possible combinations of the inputs. Each row of the truth table presents a possible input combination and an associated output value. The set of those input combinations, whose associated output values are 1, is referred to as an *ON-array* of the function f . The set of those input combinations, whose associated output values are 0, is referred to as an *OFF-array* of the function f . In Table I, x , y , and z are binary input variables, f is a binary output variable. Figure 1 shows the ON- and OFF-arrays of f . Please note that either ON-array or OFF-array of the function f contain the same information as the truth table.

The function f can also be specified by the following Boolean equation:

$$f = \bar{x} \bar{y} z \oplus \bar{x} y z \oplus x \bar{y} z \oplus x y \bar{z} \oplus x y z.$$

ON-array			OFF-array		
x	y	z	x	y	z
0	0	1	0	0	0
0	1	1	0	1	0
1	0	1	1	0	0
1	1	0			
1	1	1			

Figure 1. ON- and OFF- arrays of function f .

where $\bar{x}\bar{y}z \oplus \bar{x}yz \oplus x\bar{y}z \oplus xy\bar{z} \oplus xyz$ is an ESOP expression. Please note that x , y and z in Table I are variables, while in the ESOP expression they are used as literals. Another way to write above equation is:

$$f = x^0y^0z^1 \oplus x^0y^1z^1 \oplus x^1y^0z^1 \oplus x^1y^1z^0 \oplus x^1y^1z^1.$$

Example II.3: In Table II, X and Y are multiple-valued variables, f is a binary output variable.

TABLE II

AN EXAMPLE OF MULTIPLE-VALUED INPUT BINARY OUTPUT FUNCTION

X	Y	f
0	0	0
0	1	1
0	2	0
1	0	1
1	1	0
1	2	1
2	0	1
2	1	1
2	2	0

This function can be represented by the following equation:

$$f = X^0Y^1 \oplus X^1Y^{02} \oplus X^2Y^{01}$$

where $X^0, X^1, X^2, Y^1, Y^{02}$, and Y^{01} are literals.

The *difference* of two terms is the number of variables for which the corresponding literals have different sets of truth values. The *distance* of two terms is the number of variables for which the corresponding literals have disjoint sets of truth values.

Example II.4: Given three terms $T_1 = X^0Y^1$, $T_2 = X^1Y^{02}$, and $T_3 = X^1Y^{01}$. The difference of T_1 and T_2 is 2, because two literals have different sets of truth values:

for X: $\{0\} \neq \{1\}$,

for Y: $\{1\} \neq \{02\}$.

The distance of T_1 and T_2 is also 2, because two literals have disjoint sets of truth values:

for X: $\{0\} \cap \{1\} = \emptyset$,

for Y: $\{1\} \cap \{02\} = \emptyset$.

The difference of T_2 and T_3 is 1, because only one literal has different sets of truth values:

for X: $\{1\} = \{1\}$,

for Y: $\{02\} \neq \{01\}$.

The distance of T_2 and T_3 is 0, because none of the literals have disjoint sets of truth values:

for X: $\{1\} \cap \{1\} \neq \emptyset$,

for Y: $\{02\} \cap \{01\} \neq \emptyset$.

We write *difference* $(T_i, T_j) = d$ to indicate that the difference of two terms T_i and T_j is d . Similarly, we write *distance* $(T_i, T_j) = d$ to indicate that the distance of T_i and T_j is d .

A *map* of an n -variable, p -valued input, two-valued output function consists of p^n *cells*. Cells that contain a 1 will be called *true minterms* (*1-cells*) while cells that contain a 0 will be called *false minterms* (*0-cells of the map*). For a binary input function, our

map is a *Karnaugh map*. For the case of multiple-valued input functions, the maps that we will use in this thesis generalize the concept of the Karnaugh maps. We will simply call each of them a *map*.

In Example II.2, the function has 3 binary variables. So, a corresponding Karnaugh map has $2^3 = 8$ cells. There are 5 true minterms and 3 false minterms as shown in Figure 2.

	yz	00	01	11	10
x	0	0	1	1	0
1	0	1	1	1	

Figure 2. Karnaugh map for a binary input function of 3 variables.

In Example II.3, the function has 2 variables, and both variables have three values. So, a corresponding map has $3^2 = 9$ cells. There are 5 true minterms and 4 false minterms as shown in Figure 3.

	Y	0	1	2
X	0	0	1	0
1	1	0	1	
2	1	1	0	

Figure 3. Map for a multiple-valued input function of 2 variables.

II.2.2. Cubical Representation

A *binary vector* (*vector* for short) is a series of symbols, where each symbol is either a 0 or a 1. We call each 0 or 1 a *bit*. An m -valued switching function f of n variables X_1, X_2, \dots, X_n can be represented by binary vectors. If the variable X_i is m -valued, the literal $X_i^{S_i}$ can be represented by a binary vector of m bits:

$$c_i^0 c_i^1 \dots c_i^{m-1}$$

where $c_i^j = 0$ if $j \notin S_i$ and $c_i^j = 1$ if $j \in S_i$. For example, for a binary logic, each literal can be represented by a vector of two bits as follows:

X_i^\emptyset is represented as 00,

X_i^1 is represented as 01,

X_i^0 is represented as 10,

$X_i^{0,1}$ is represented as 11.

For a 4-valued logic, each literal is represented by a vector of 4 bits:

$X_i^{0,1,2,3}$ is represented as 1111,

$X_i^{0,2}$ is represented as 1010,

$X_i^{1,3}$ is represented as 0101.

A product term of n literals can be represented by n such vectors. A symbol "-" is used for separating each vector. For instance, $X_1^{0,1} X_2^{1,2} X_3^{1,3}$ in cubical representation is represented by three vectors:

$$[1100 - 0110 - 0101],$$

which is called a *cube*. A cube can represent:

1. a product of literals,

2. a sum of literals,
3. an exclusive sum of literals.

There is no difference in the representation of these forms as a cube. For example, both $X_1^{0,1} + X_2^{1,2} + X_3^{1,3}$ and $X_1^{0,1} \oplus X_2^{1,2} \oplus X_3^{1,3}$ are presented as

$$[1100 - 0110 - 0101].$$

When using a cube, we should specify which form the cube is used to represent. We can represent a POS, a SOP or a ESOP by an array of cubes. This way of representation is called *positional notation of cube calculus (cube notation for short)*. In this thesis, without other specification, we use a cube to represent a product term, and use an array of cubes to represent an ESOP.

Example II.5: For binary logic, an ESOP

$$\bar{x} \bar{y} \bar{z} \oplus \bar{x} z v \oplus y z \oplus x \bar{y} z \bar{v}$$

in cube notation can be represented as following 4 cubes:

$$[10 - 10 - 10 - 11]$$

$$[10 - 11 - 01 - 01]$$

$$[11 - 01 - 01 - 11]$$

$$[01 - 10 - 01 - 10].$$

For binary logic, we can simplify the cube notation as follows:

- 00 is represented as ϵ ,
- 10 is represented as 0,
- 01 is represented as 1,
- 11 is represented as x .

So, the above example can also be represented as

$$000x$$

$$0x11$$

$$x11x$$

$$1010.$$

Example II.6: For 4-valued logic, an ESOP

$$X^0_1 X^0_2 X^2_3 X^0_4 \oplus X^2_1 X^1_2 X^3_3 X^1_4 \oplus X^0_1 X^1_2 X^2_3 X^0_4 \oplus X^1_1 X^0_2 X^0_3 X^0_4^{123}$$

can be represented by an array of 4 cubes:

$$[1100 - 1000 - 0110 - 1100]$$

$$[0011 - 0100 - 0101 - 0100]$$

$$[1100 - 0100 - 0011 - 1100]$$

$$[0101 - 1001 - 1100 - 1111].$$

The ON-array of a function can be represented by the ON-array of cubes, and the OFF-array of a function can be represented by the OFF-array of cubes.

Example II.7: The function f of Example II.2 can be represented by the following ON-array of cubes:

$$001$$

$$011$$

$$101$$

$$110$$

$$111.$$

We can draw cubes on maps. On the map, each circle indicates one cube, as shown in Figure 4a. We can also use maps to minimize the number of cubes. For instance, 5 cubes in Figure 4a is reduced to 2 cubes in Figure 4b. In other words, we can use an ON-array of 2 cubes to express the function f of Example II.2:

$$xx1$$

$$110.$$

Example II.8: The function f of Example II.3 can be represented by the following ON-array of cubes:

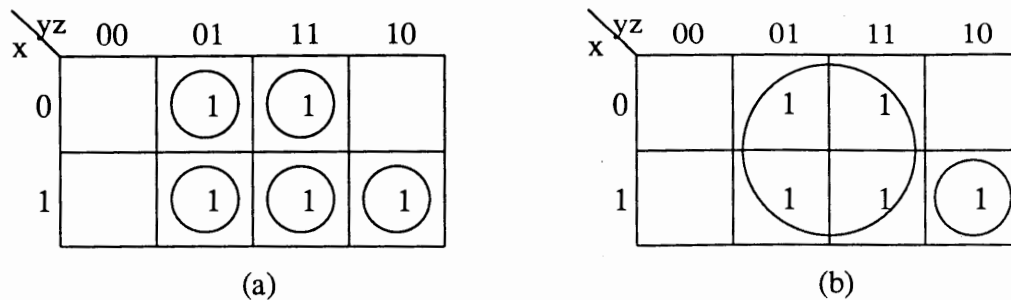


Figure 4. Karnaugh map for binary input cubes.

$$[100 - 010]$$

$$[010 - 101]$$

$$[001 - 110]$$

Figure 5a shows these three cubes on the map. Please note that two circles connected by an arc denote one cube $[010 - 101]$. In ESOPs a true minterm can be covered by cubes an odd number of times, a false minterm can be covered an even number of times. Figure 5b, 5c, and 5d show other arrays of cubes representing the same function. This means that for a specific function, there are different ESOPs to represent it.

If A is a cube, we write A_i or $A[i]$ to denote the i -th vector of A . And we write A_{ij} or $A[i,j]$ to denote the j -th bit of A_i . For instance, in Example II.8, if we denote the first cube in the array as A , then

$$A_1 = 100,$$

$$A_2 = 010,$$

$$A_{11} = 1,$$

$$A_{12} = 0,$$

$$A_{13} = 0,$$

$$A_{21} = 0,$$

$$A_{22} = 1,$$

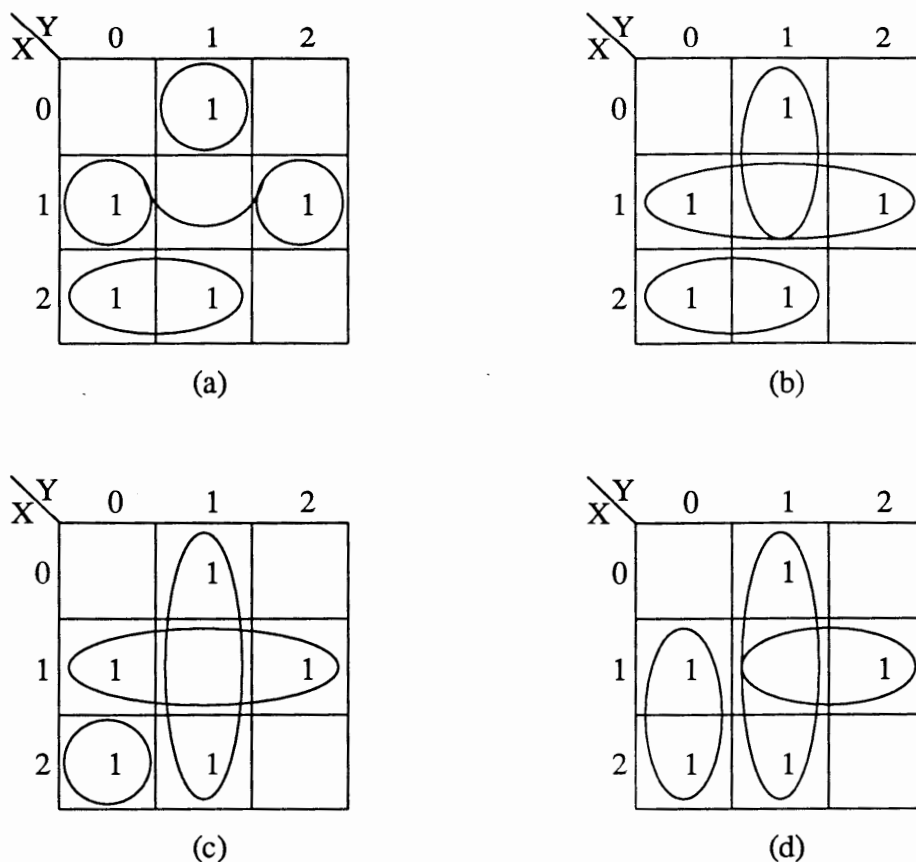


Figure 5. Map for multiple-valued input cubes.

$$A_{23} = 0.$$

Please note the difference between S_i the truth set of variable X_i and A_i the vector in a cube corresponding to literal $X_i^{S_i}$. For a 4-valued logic, for instance, if $S_i = \{0,1,2\}$, the corresponding A_i is 1110. They represent the same literal, but in two different ways.

We define the *difference* of two cubes as the difference of the corresponding two terms, and the *distance* of two cubes as the distance of the corresponding two terms. In other words, the difference of two cubes is the number of their vectors that are different; the distance of two cubes is the number of their vectors that are disjoint. For instance, given two cubes

$$A = [1100 - 1010 - 0111]$$

and

$$B = [0110 - 0100 - 0101],$$

the difference of two cubes is 3, because $A_1 \neq B_1$, $A_2 \neq B_2$, and $A_3 \neq B_3$. The distance of two cubes is 1, because only A_2 and B_2 are disjoint ($A_2 \cap B_2 = \emptyset$). If A and B are cubes, we write *difference* (A, B) = r to indicate that their difference is r , and *distance* (A, B) = r to indicate that their distance is r .

Given an array of cubes, if the distance of any two cubes in the array is greater than 0, then the array of cubes is called *an array of disjoint cubes*. Both the arrays showed in Figures 4a and 4b are arrays of disjoint cubes. The array showed in Figure 5a are also array of disjoint cubes. But the arrays showed in Figures 5b, 5c, and 5d are not arrays of disjoint cubes. If an array of cubes is disjoint, its corresponding SOP expression and ESOP expression represent the same function. For instance, the disjoint cubes showed in Figure 4b can be expressed as $z + x y \bar{z}$ or $z \oplus x y \bar{z}$.

II.2.3. Incompletely Specified Functions

Given a function $f: P^n \rightarrow Q$, if for some $p \in P^n$, there exists more than one q in Q such that (p, q) is in f , then f is an incompletely specified function.

For a binary output function, the possible output values are 0 and 1. If the function is incompletely specified, then for some input combinations, its output value can be either 0 or 1. In this case, the output value is a *don't care value*.

When we realize an incompletely specified function, we have to fix its don't care values to either 0 or 1.

In Table III, f is an incompletely specified function. f_1 and f_2 are two examples to realize this function. Function g and h in Table III are two extreme ways to realize the function f . If we fix all the don't care values in f to 0, we get g . On the other hand,

TABLE III
EXAMPLES OF INCOMPLETELY SPECIFIED FUNCTIONS

x_1	x_2	g	h	f	f_1	f_2
0	0	0	0	0	0	0
0	1	0	1	{0,1}	0	1
1	0	0	1	{0,1}	1	0
1	1	1	1	1	1	1

if we fix all the don't care values in f to 1, we get h . If g and h are so defined, then for any $p \in P_n$, the following relation is true:

$$g(p) \leq f(p) \leq h(p).$$

This relation shows us another way to define or explain an incompletely specified function:

Given $g: P^n \rightarrow Q$ and $h: P^n \rightarrow Q$ are two completely specified functions, such that $g(p) \leq h(p)$ for all $p \in P^n$. If a function f is defined by the interval of g and h ,

$$f(p) = [g(p), h(p)] \forall p \in P^n,$$

then f is an *incompletely specified function*.

A multiple-valued input, binary output, incompletely specified function f (*multiple-valued function*, for short) is a mapping $f(X_1, X_2, \dots, X_n) : P_1 \times P_2 \times \dots \times P_n \rightarrow Q$, where X_i is a *multiple-valued variable*, $P_i = \{0, 1, \dots, p_i - 1\}$ is a *set of truth values* that this variable may assume, and $Q = 0, 1, d$ (d denotes a don't care value). This is a generalization of an ordinary n -input switching function $f: P^n(2) \rightarrow P(2)$.

Example II.9: Table IV is an example of a multiple-valued input binary output incompletely specified function. Figure 6 is the corresponding map.

Given an incompletely specified function, the set of those input combinations, whose associated output values are *don't cares*, is referred to as a *DC-array* of the func-

TABLE IV
AN EXAMPLE OF MULTIPLE-VALUED INPUT BINARY OUTPUT
INCOMPLETELY SPECIFIED FUNCTION

X	Y	f
0	0	0
0	1	1
0	2	0
1	0	{0,1}
1	1	{0,1}
1	2	1
2	0	1
2	1	1
2	2	0

X \ Y	0	1	2
0	0	1	0
1	d	d	1
2	1	1	0

Figure 6. Map for a multiple-valued input incompletely specified function.

tion. The ON-array and OFF-array are defined as for completely specified functions. Figure 7 shows the ON- OFF- and DC-array of the function specified in Table IV.

ON-array		OFF-array		DC-array	
X	Y	X	Y	X	Y
0	1	0	0	1	0
1	2	0	2	1	1
2	0	2	2		
2	1				

Figure 7. ON-, OFF- and DC-array of an incompletely specified function.

The cubes in the ON-array will be called *ON-cubes*. Similarly, the cubes in the OFF-array and DC-array will be called *OFF-cubes* and *DC-cubes* respectively. We write $ON(f)$ to indicate an array of cubes for which $f = 1$, $OFF(f)$ and $DC(f)$ to indicate arrays of cubes for which $f = 0$ and $f = \text{don't care}$, respectively. Figure 8 shows the different ESOPs representing the function given in Example II.9. Please note that the don't care minterms can be covered by cubes an arbitrary number of times.

Example II.10: The corresponding ESOP expressions for Figure 8 are:

$$\text{Figure 8a. } X^0Y^1 \oplus X^1Y^2 \oplus X^2Y^{01},$$

$$\text{Figure 8b. } X^{12}Y^{01} \oplus X^{01}Y^1 \oplus X^1Y^{12},$$

$$\text{Figure 8c. } Y^{01} \oplus X^0Y^0 \oplus X^1,$$

$$\text{Figure 8d. } 1 \oplus X^{01}Y^0 \oplus X^{02}Y^2.$$

II.2.4. Multiple Output Functions

A multiple output function is a mapping $f : P^n \rightarrow Q^m$, where $Q = \{0,1,d\}$. If $m=1$, the function is a *single output function*. If $m > 1$, it is a *multiple-output function*.

One way of dealing with multiple-output functions is to represent and minimize them as a function with a single two-valued output. Let us consider a Boolean function F with multiple outputs, that is, $F(X_1, \dots, X_n) = (f_0, \dots, f_{m-1})$. We define a single-output switching function F on $n+1$ variables where the variable X_{n+1} takes the values $\{0, \dots, m-1\}$. $F(X_1, \dots, X_n, X_{n+1}) = F_{X_{n+1}}(X_1, \dots, X_n)$ where $F_i(X_1, \dots, X_n)$ denotes the i -th *projection* of $F(X_1, \dots, X_n)$, that is, f_i . Therefore, only single-output switching functions will be considered and $n+1$ will denote the number of input variables.

Example II.11: Given is a 3-input 2-output binary function $F(x, y, z) = (f_0, f_1)$ with binary inputs specified by Table V.

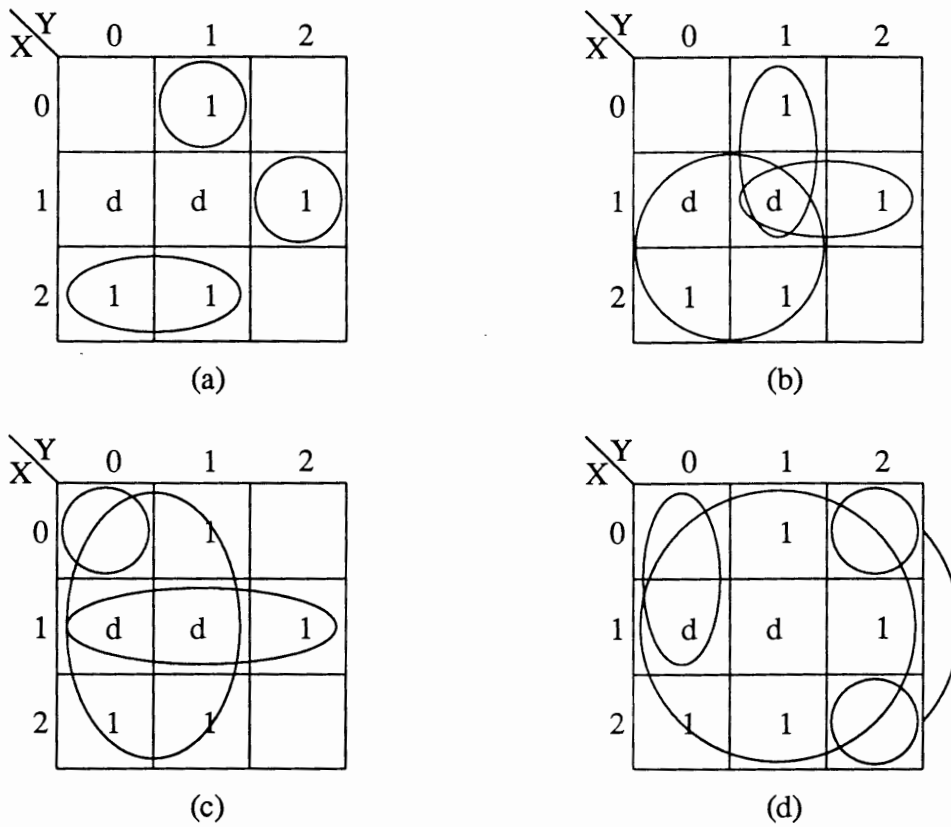


Figure 8. ESOPs for a multiple-valued input incompletely specified function.

TABLE V

AN EXAMPLE OF MULTIPLE OUTPUT FUNCTION

x	y	z	f_0	f_1
0	0	0	0	0
0	0	1	1	0
0	1	0	1	1
0	1	1	0	0
1	0	0	0	0
1	0	1	1	0
1	1	0	0	0
1	1	1	1	1

This function is described by an ON-array in Table VI.

TABLE VI
ON-ARRAY OF THE MULTIPLE OUTPUT FUNCTION

x	y	z	f_0	f_1
0	0	1	1	0
0	1	0	1	1
1	0	1	1	0
1	1	1	1	1

After transforming this ON-array of the function to the ON-array of 4-input function $F(x, y, z, v)$ with 2-valued input v , the ON-array of F is shown in Table VII.

TABLE VII
CONVERTING A MULTIPLE OUTPUT FUNCTION
TO A SINGLE OUTPUT FUNCTION

x	y	z	v
0	0	1	0
0	1	0	0
0	1	0	1
1	0	1	0
1	1	1	0
1	1	1	1

Figure 9a and 9b are maps corresponding to Table VI. Figure 9c is a map corresponding to Table VII.

In Figure 10a, the ON-array of the function F is covered by an array of three cubes. The corresponding ESOP expression of the function F is

$$\bar{x}y \oplus yz \oplus z\bar{v}.$$

We call this ESOP expression a *solution* of F . Remember that although we transform the multiple output function to a binary output function for easier handling, we have to convert it back to a multiple output function as the final result. Substituting $v = 0$ to the

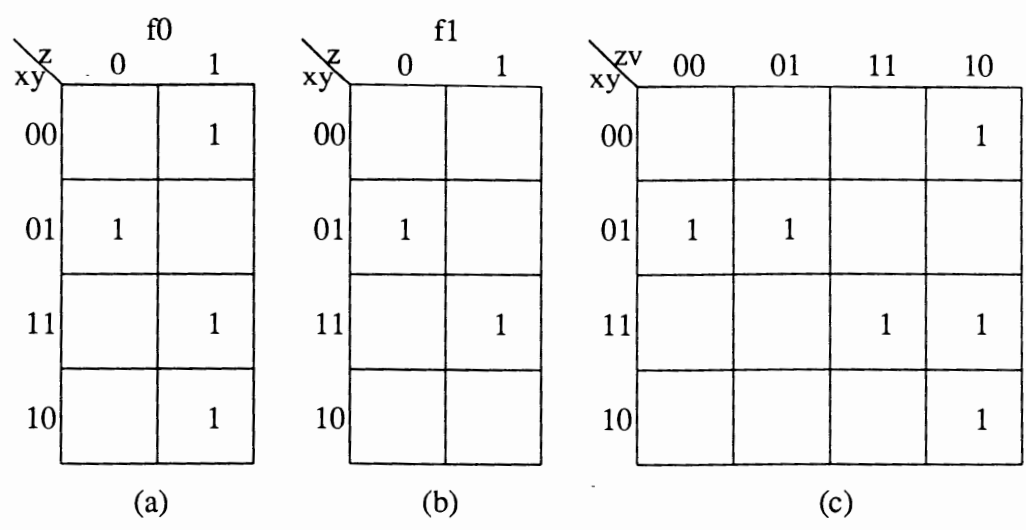


Figure 9. Map for a multiple output function.

solution of F , we obtain function $f_0(x, y, z) = \bar{x}y \oplus z$. Substituting $v = 1$ we obtain $f_1(x, y, z) = \bar{x}y \oplus yz$. The corresponding circuit is presented in Figure 10b.

II.3. OPERATIONS

An *operation* is a mapping $f : P^n \rightarrow P$. Note that an operation is a special case of a function. For $n=1,2$, the operations are called unary, and binary respectively. For example, $f : P \rightarrow P$ is a unary operation, and $f : P^2 \rightarrow P$ is a binary operation.

II.3.1. Set Theoretic Operations

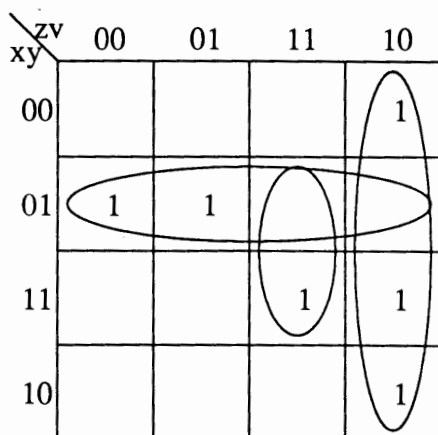
If P and Q are sets, we define the *union* of P and Q as

$$\{ x \mid x \in P \text{ or } x \in Q \text{ or both} \}.$$

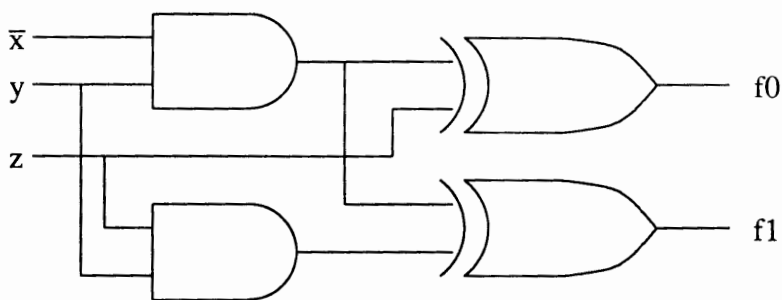
The union of P and Q is denoted by $P \cup Q$. For example: $\{0,1\} \cup \{1,2\} = \{0,1,2\}$; $\{0,1\} \cup \{2,3\} = \{0,1,2,3\}$.

If P and Q are sets, the *intersection* of P and Q , denoted by $P \cap Q$, is defined as

$$\{ x \mid x \in P \text{ and } x \in Q \}.$$



(a)



(b)

Figure 10. Map and circuit for a multiple output function.

The *difference* of sets P and Q , denoted by $P - Q$ is defined as

$$\{x \mid x \in P \text{ but } x \notin Q\}.$$

For example: $\{0,1,2\} - \{1\} = \{0,2\}$; $\{0,1\} - \{2\} = \{0,1\}$.

Please do not confuse the difference of two sets and the difference of two cubes. The difference of two sets is a set operation. The result of the operation is a set which contains the elements in set A but not in set B . The difference of two cubes (see page 18) is a function which indicates how many vectors in the cubes are different.

If $P \cap Q = \emptyset$, we say that P and Q are *disjoint*, otherwise, they are *non-disjoint*. For example: sets $\{a, b\}$ and $\{c, d\}$ are disjoint; sets $\{a, b\}$ and $\{b, c\}$ are non-disjoint. If $P \not\subset Q$, $Q \not\subset P$, and $P \cap Q \neq \emptyset$, we say that P and Q are overlapping. Note that overlapping is a special case of non-disjoint. For instance, sets $\{p, q\}$ and $\{q, r\}$ are overlapping, sets $\{p, q, r\}$ and $\{q, r\}$ are non-disjoint, but are not overlapping. We write

$$P \text{ } \bowtie \text{ } Q$$

to indicate the sets P and Q are overlapping.

The *exclusive-or* (*exor* for short) of sets P and Q , denoted as $P \oplus Q$ is defined as

$$\{x \mid \text{either } x \in P \text{ or } x \in Q \text{ but not both}\}$$

By definition, $P \oplus Q = (P - Q) \cup (Q - P)$. For example: $\{1, 2\} \oplus \{2, 3\} = \{1, 3\}$. In the case of sets P and Q being disjoint, $P \oplus Q = P \cup Q$.

II.3.2. Cube Operations

If A is a cube, then the mapping $A^n \rightarrow A$ is a cube operation. We will only discuss unary and binary cube operations in this thesis.

The *supercube operation* of cubes A and B is defined as follows:

$$A \cup B = [A_1 \cup B_1, \dots, A_n \cup B_n]$$

where $A_i \cup B_i$ is a set union.

The *intersection operation* of cubes A and B is defined as follows:

$$A \cap B = \begin{cases} [A_1 \cap B_1, \dots, A_n \cap B_n] & \text{if there is no such } i \text{ that } A_i \cap B_i = \emptyset \\ \emptyset & \text{otherwise} \end{cases}$$

where $A_i \cap B_i$ is a set intersection, and \emptyset is an empty cube.

Since we use a cube to represent a term, the operation of terms can be represented as the operation of cubes. For instance, if two cubes $A = [A_1, \dots, A_n]$ and $B = [B_1, \dots, B_n]$ represent two terms $T_S = X_1^{S_1}, \dots, X_n^{S_n}$ and $T_R = X_1^{R_1}, \dots, X_n^{R_n}$ respectively,

then the supercube operation of two cubes A and B

$$A \cup B = [A_1 \cup B_1, \dots, A_{i-1} \cup B_{i-1}, A_i \cup B_i, A_{i+1} \cup B_{i+1}, \dots, A_n \cup B_n]$$

is equivalent to the supercube operation of two terms T_S and T_R

$$T_S \cup T_R = X_1^{S_1 \cup R_1} \dots X_{i-1}^{S_{i-1} \cup R_{i-1}} X_i^{S_i \cup R_i} X_{i+1}^{S_{i+1} \cup R_{i+1}} \dots X_n^{S_n \cup R_n}$$

In cube notation, an operation between two variables is a *local operation*, and an operation between two cubes is a *global operation*. A local operation is a set operation, and a global operation is a cube operation. For example, the supercube operation $A \cup B$ is a global operation, and $A_i \cup B_i$ is a local operation.

Sometimes, different local or global operations may be performed on two cubes depending on a relation between them. In cube notation, a relation between two vectors (corresponding to a variable) is a *local relation* and a relation between two cubes (corresponding to two terms) is a *global relation*. For instance, given two cubes $A = [A_1, A_2, \dots, A_n]$ and $B = [B_1, B_2, \dots, B_n]$, then $A \subseteq B$ is a global relation, and $A_i \subseteq B_i$ is a local relation. A global relation is satisfied if all the local relations are satisfied. For example, given:

$$A = [0011 - 1100 - 1110]$$

and

$$B = [1011 - 0110 - 1111]$$

we can see that $A_1 \subseteq B_1$ and $A_3 \subseteq B_3$, but $A_2 \not\subseteq B_2$. So, $A \not\subseteq B$.

The *disjoint sharp operation* on cubes A and B is defined as follows:

$$A \#d B = \begin{cases} A & \text{when } A \cap B = \phi \\ \phi & \text{when } A \subseteq B \\ A \#d_{basic} B & \text{otherwise} \end{cases}$$

where $A \#d_{basic} B$ is defined as follows:

$$A \#d_{basic} B = \{A_1, \dots, A_{i-1}, \neg B_i \cap A_i, A_{i+1} \cap B_{i+1}, \dots, A_n \cap B_n \mid \text{for such } i = 1, \dots, n, \text{ that } A_i \not\subseteq B_i\}$$

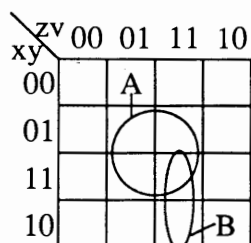
In this formula, $A \cap B$ and $A \subseteq B$ are global relations. If the relation $A \cap B = \phi$ is true (satisfied), the resultant cube is A . If the relation $A \subseteq B$ is true, an empty cube is generated. Otherwise the global operation $A \#_d B$ is performed. $A_i \not\subseteq B_i$ is a local relation, and both $A_i \cap B_i$ and $\neg B_i \cap A_i$ are local operations.

Example II.12: Given two terms $T_1 = y \vee$ and $T_2 = x z \vee$. They can be expressed as two cubes:

$$A = [11 - 01 - 11 - 01]$$

$$B = [01 - 11 - 01 - 01].$$

These two cubes are shown in Figure 11a. Figure 11b shows that $A \cap B \neq \phi$ (none of the vectors in resultant cube are empty sets). Next we check if the relation $A \subseteq B$ is true. Please note that $A \subseteq B$ is equivalent to $\neg A \cup B \neq \emptyset$. We can also check if the relation $A \subseteq B$ is false. This is equivalent to check if $\exists i$, such that $\neg B_i \cap A_i \neq \emptyset$. Figure 11c converts B to $\neg B$. Here $\neg B = [\neg B_1, \dots, \neg B_n]$ is a global operation. Figure 11d intersects A and $\neg B$ and shows that two of the local relations are true ($\neg B_1 \cap A_1 \neq \emptyset$ and $\neg B_3 \cap A_3 \neq \emptyset$). These two local relations are indicated by two arrows. Since the relation $A \subseteq B$ is false, the operation $A \#_{d_{basic}} B$ should be performed. Two resultant cubes are generated in Figure 11e and Figure 11f, respectively. Figure 11g shows the final results in the Karnaugh map. Note that the disjoint sharp operation of cubes A and B generates an array of disjoint cubes, which cover the minterms in cube A but not in cube B .



(a)

A	11	01	11	01
B	01	11	01	01
$A \cap B$	01	01	01	01

(b)

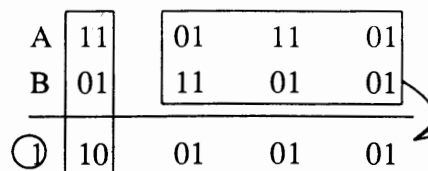
B	01	11	01	01
$\neg B$	10	00	10	10

(c)

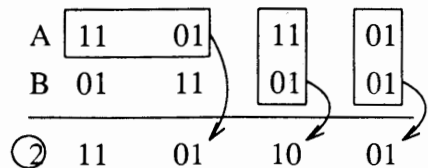
A	11	01	11	01
$\neg B$	10	00	10	10
$\neg B \cap A$	10	00	10	00



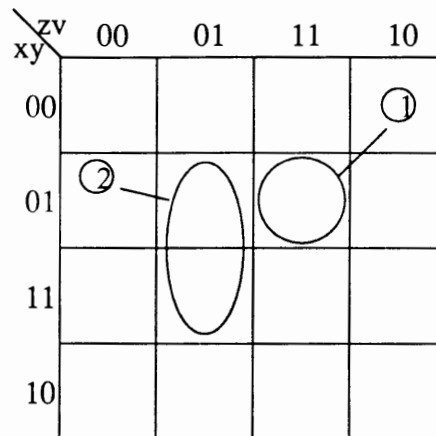
(d)



(e)



(f)



(g)

Figure 11. Example of a disjoint sharp operation.

CHAPTER III

MULTIPLE-VALUED INPUT EXCLUSIVE SUMS OF PRODUCTS MINIMIZATION

III.1. THE COST FUNCTIONS

The object of logic minimization is to reduce the cost function. Our primary goal of MIESOP synthesis is to *minimize the number of terms*. For the circuit with the minimum number of terms our secondary goal is to minimize the total number of connections (wires). To combine these two goals together, the *cost function* C to be used in our program is:

$$C = NT + \frac{NI}{NI_{in}}$$

where:

- NT is the total number of terms in the solution,
- NI is the total number of input wires to AND and EXOR gates in the solution,
- NI_{in} is the total number of input wires to AND and EXOR gates in the initial function.

After the function is minimized, the number of connections in the solution will be less or equal to the number of connections in the initial function. So,

$$0 < \frac{NI}{NI_{in}} \leq 1.$$

Since one term reduced in the solution will reduce the cost function by 1, our program will select the solution which has minimum number of terms. Among the solutions which have the same number of terms, the program will select the one which has minimum number of connections. For instance, literal X^{012} as an input to an AND gate requires a single wire for the 2-by-4 decoder realization of logic with 4-valued inputs.

X^{01} is realized as $X^{012}X^{013}$. It, therefore, requires two wires. Similarly $X^0 = X^{012}X^{013}X^{023}$ requires three wires. A product term X^0Y^1 requires six wires. An ESOP $X^0Y^1 \oplus X^1Y^0$ contains two terms and requires 12 wires to AND gates and 2 wires to an EXOR gate. If this ESOP is our initial function, then the cost function is:

$$C = NT + NI / NI_{in} = 2 + 14 / 14 = 3.$$

If the ESOP is minimized as $X^{01}Y^1 \oplus X^1Y^{01}$, then only 10 wires to AND gates are required. The corresponding cost function is:

$$C = 2 + 12 / 14 = 2.86.$$

For different technologies, different cost functions may be used. For instance, if we only concern the number of terms in the final results, then we can use the cost function $C = NT$. On the other hand, if we only concern the number of wires, then we can use the cost function $C = NI$. In chapter VI, we will try to use different cost functions to compare our results with the results from other authors.

III.2. THE PROPERTIES OF THE ESOP

Let A, B, C denote any multiple-valued input literals, or any functions on them.

The following operations hold for multiple-valued input algebra:

1. Associative laws: $A \oplus (B \oplus C) = (A \oplus B) \oplus C$
2. Commutative laws: $A \oplus B = B \oplus A$
3. Identities:
 - 3a. $A \oplus A = 0$
 - 3b. $X_i^{S_i} \oplus X_i^{R_i} = X_i^{S_i \oplus R_i}$
 - 3c. $X_i^{S_i} X_j^{S_j} \oplus X_i^{R_i} X_j^{R_j} = X_i^{S_i \oplus R_i} X_j^{S_j} \oplus X_i^{R_i} X_j^{S_j \oplus R_j}$
 $= X_i^{S_i \oplus R_i} X_j^{R_j} \oplus X_i^{S_i} X_j^{S_j \oplus R_j}$

III.3. BASIC IDEAS TO MINIMIZE THE ESOP

Given an ESOP, it can be represented as an array of cubes. We can do the following things to minimize the function:

III.3.1. Removing Two Equal Cubes

According to property 3a, if any two cubes A and B in the array are equal ($\text{difference}(A, B) = 0$), they can be removed from the array.

Example III.1: The following array of four cubes

[0101 – 1111 – 1001 – 0101]

[0101 – 0011 – 1101 – 1101]

[0101 – 1111 – 1001 – 0101]

[0111 – 1001 – 0001 – 1110]

can be reduced to an array of two cubes

[0101 – 0011 – 1101 – 1101]

[0111 – 1001 – 0001 – 1110],

because the first cube and the third cube in the array are identical.

III.3.2. Combining Two Cubes which Differ in One Variable

According to property 3b, if any two cubes in the array differ in one variable, these two cubes can be combined to one cube.

Example III.2: Two cubes $x101$ and $x111$ can be combined to one cube $x1x1$ as shown in Figure 12.

III.3.3. Reshaping Two Cubes which Differ by 2

If two cubes in the array differ by 2, these two cubes can be *reshaped*. In other words, they can be replaced by another pair of two cubes. The number of cubes in the

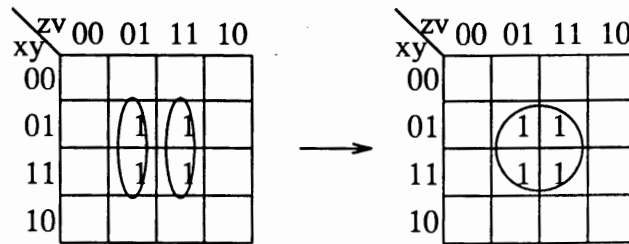


Figure 12. Combining two cubes into one cube.

array is not reduced by reshaping. However, reshaping may provide the chance to reduce the number of cubes later.

Example III.3: Given three cubes: $A = x101$, $B = 1111$ and $C = 10x1$. The number of cubes can not be reduced directly, since none of them can be removed or combined. Because that the cubes A and B differ by two, they can be reshaped to cubes A' and B' . Then cubes B' and C can be combined to cube C' . The number of cubes are reduced from 3 to 2. This process is shown in Figure 13.

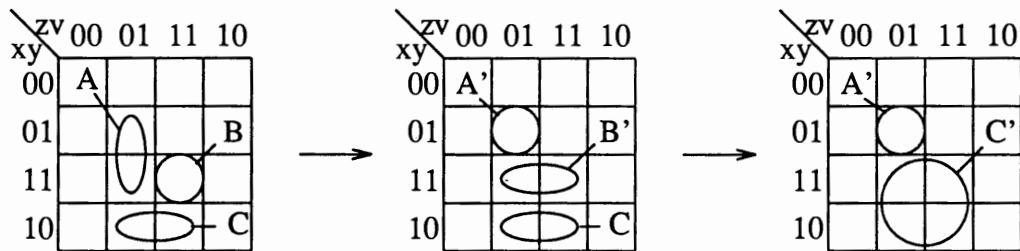


Figure 13. Reshaping two cubes.

III.3.4. Increasing the Number of Cubes

This seems contradictory to our goal: minimizing the number of cubes. However, it has been proved [Bran 91] that a set of rules (operations) can not generate a minimum form of certain ESOPs if it satisfies the following conditions:

1. Each rule changes at most two terms at a time.

2. Each rule does not increase the number of the terms.

In other words, if we only use the first three methods (remove, combine, and reshape), our solution may be a local minimum. We will discuss how to increase the number of cubes later.

III.4. THE OPERATIONS USED IN EXORCISM

Dr. Perkowski and his former students developed an algorithm for ESOP minimization called EXORCISM [Hell 88, Perk 89]. The following operations are used in EXORCISM.

III.4.1. Primary Xlinking

Given two terms, if their truth value sets for each variable are either equal or disjoint, then these two terms are primary xlinkable. For example, terms $X^{01}Y^0Z^0U^2V^{13}$ and $X^2Y^2Z^2U^2V^{13}$ are primary xlinkable, because their truth value sets for each variable are either equal (variable U and V) or disjoint (variable X, Y, Z). Terms $X^{01}Y^0Z^0U^2V^{13}$ and $X^{12}Y^2Z^2U^2V^{13}$ are not primary xlinkable, because their truth value sets for variable X are neither equal nor disjoint.

Primary xlinking is defined by the following formula:

$$T_S \oplus T_R = \bigoplus \left\{ X_1^{S_1} \cdots X_{i-1}^{S_{i-1}} X_i^{S_i \cup R_i} X_{i+1}^{R_{i+1}} \cdots X_n^{R_n} \mid \right. \\ \left. \text{for such } i = 1, \dots, n, \text{ that } S_i \cap R_i = \emptyset \right\}$$

where $T_S = X_1^{S_1} \cdots X_n^{S_n}$ and $T_R = X_1^{R_1} \cdots X_n^{R_n} \neq T_S$ are two terms, and all the variables have either equal or disjoint truth value sets. We give the following two examples to show the primary xlinking. In both examples, 4-valued variables are assumed.

Example III.4: Given two terms $T_1 = X^{01}Y^1Z^0V^1$ and $T_2 = X^{01}Y^0Z^0V^1$. These two terms are primary xlinkable, since

- for X: $\{01\} = \{01\}$.
- for Y: $\{1\} \cap \{0\} = \emptyset$.
- for Z: $\{0\} = \{0\}$.
- for V: $\{1\} = \{1\}$.

Since the distance between two terms is 1, only one resultant term is generated, which is:

$$A \oplus B = X^{01}Y^{01}Z^{00}V^1.$$

Figure 14 shows the procedure.

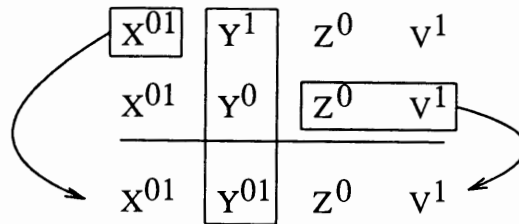


Figure 14. Procedure of distance 1 primary xlinking.

Example III.5:

Given two terms $T_1 = X^{01}Y^{01}Z^{00}V^1$ and $T_2 = X^{01}Y^{23}Z^{12}V^1$. These two terms are also primary xlinkable, since

- for X: $\{01\} = \{01\}$.
- for Y: $\{01\} \cap \{23\} = \emptyset$.
- for Z: $\{0\} \cap \{12\} = \emptyset$.
- for V: $\{1\} = \{1\}$.

Since the distance between two terms is 2, two resultant terms are generated by primary xlinking of terms T_1 and T_2 . The result is:

$$T_1 \oplus T_2 = X^{01}Z^{12}V^1 \oplus X^{01}Y^{01}Z^{012}V^1.$$

Figure 15 shows the procedure.

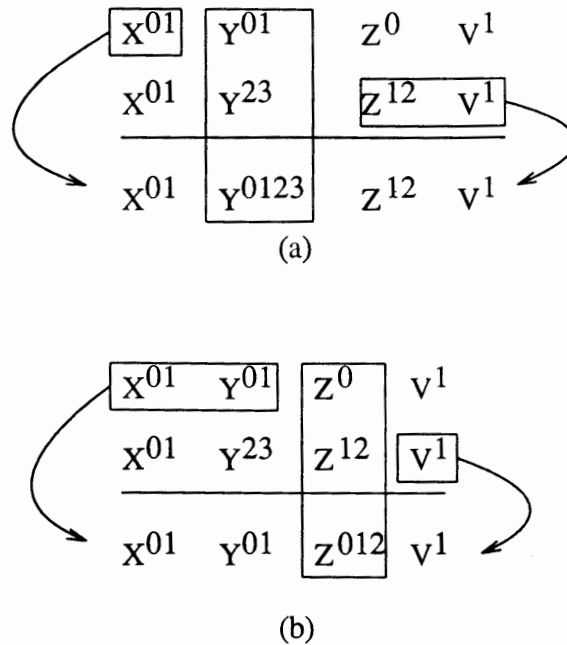


Figure 15. Procedure of distance 2 primary xlinking.

III.4.2. Secondary Xlinking

Given two terms, if there exists exactly one variable for which the truth value sets associated with one term is a sub set of the truth value set associated with the other term, and for all other variables, their truth value sets for each variable are either equal or disjoint, then these two terms are secondary xlinkable. For example, terms $X^{01}Y^0Z^0U^2V^{13}$ and $X^1Y^2Z^2U^2V^{13}$ are secondary xlinkable, because there is one variable (X) for which one truth value set ($\{1\}$) is a sub set of another ($\{01\}$), and for all other variables, their truth value sets for each variable are either equal (variable U and V) or disjoint (variable Y, Z). Terms $X^{01}Y^0Z^0U^2V^{13}$ and $X^{12}Y^2Z^2U^2V^{13}$ are not secondary xlinkable, because their truth value sets for variable X are overlapping. Terms $X^{01}Y^{01}Z^0U^2V^{13}$ and $X^1Y^0Z^2U^2V^{13}$ are also not secondary xlinkable, because there are two variables (X and Y) for which the truth value sets associated with one term are included in the truth value sets associated with the other term.

Secondary xlinking is defined by the following formula:

$$T_S \ominus T_R = T_N \oplus (T_S \Phi (T_R \Phi T_N))$$

where $T_S = X_1^{S_1} \dots X_n^{S_n}$ and $T_R = X_1^{R_1} \dots X_n^{R_n} \neq T_S$ are two terms; and there exists exactly one variable X_i such that $S_i \supset R_i$ and other variables have either disjoint or equal truth value sets, and $T_N = X_1^{R_1} \dots X_{i-1}^{R_{i-1}} X_i^{S_i - R_i} X_{i+1}^{R_{i+1}} \dots X_n^{R_n}$.

We give the following two examples to show the secondary xlinking.

Example III.6: Given two terms $T_S = X^{01}Y^1Z^0V^1$ and $T_R = X^1Y^0Z^0V^1$. These two terms are secondary xlinkable, since

$$\text{for X: } \{01\} \supset \{1\}.$$

$$\text{for Y: } \{1\} \cap \{0\} = \emptyset.$$

$$\text{for Z: } \{0\} = \{0\}.$$

$$\text{for V: } \{1\} = \{1\}.$$

The distance between two cubes is 1, but the difference between the two cubes is two, so two resultant cube are generated. The result is:

$$T_S \ominus T_R = X^{00}Y^0Z^0V^1 \oplus X^{01}Y^0Z^0V^1.$$

Figure 16 shows the procedure. In Figure 16a, T_N is generated. Figure 16b shows the result of $(T_R \Phi T_N)$. Figure 16c shows the result of $(T_S \Phi (T_R \Phi T_N))$

Example III.7: Given two terms $T_S = X^{01}Y^{01}Z^0V^1$ and $T_R = X^1Y^{23}Z^{12}V^1$. These two terms are also secondary xlinkable, since

$$\text{-for X: } \{01\} \supset \{1\},$$

$$\text{-for Y: } \{01\} \cap \{23\} = \emptyset,$$

$$\text{-for Z: } \{0\} \cap \{12\} = \emptyset,$$

$$\text{-for V: } \{1\} = \{1\}.$$

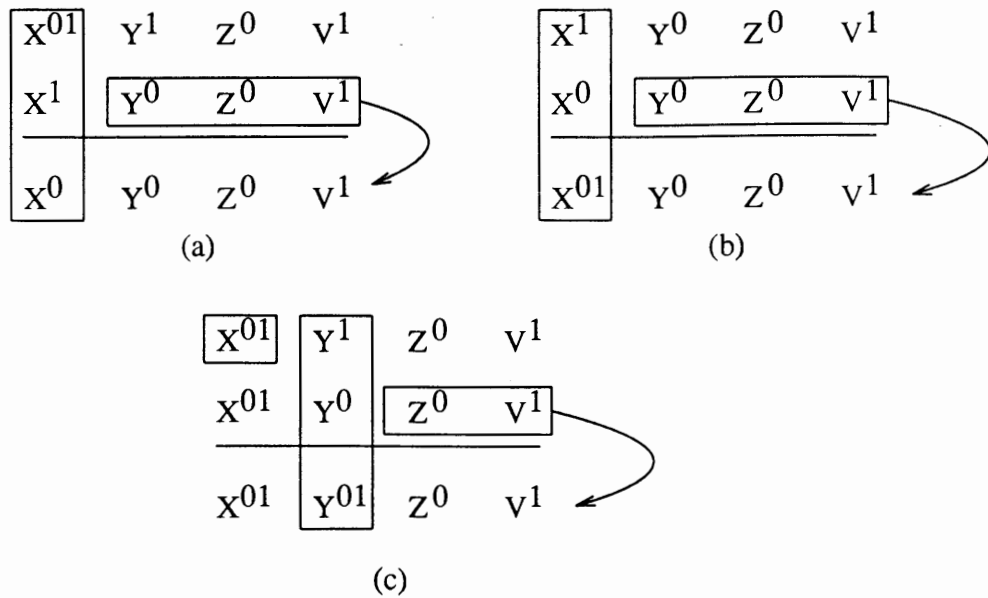


Figure 16. Procedure of distance 1 secondary xlinking.

The distance between the two terms is 2, and the difference between the two terms is three. So, three resultant terms should be generated by secondary xlinking of terms T_S and T_R . Figure 17 shows the procedure. In Figure 17a, T_N is generated. In Figure 17b, $(T_R \Phi T_N)$ is generated. In figure 17c and figure 17d, two resultant cubes are generated by $(T_S \Phi (T_R \Phi T_N))$. The result is:

$$T_S \Theta T_R = X^0Y^{23}Z^{12}V^1 \oplus X^{01}Z^{12}V^1 \oplus X^{01}Y^{01}Z^{012}V^1.$$

III.4.3. Unlinking

Unlinking operations are inverse to the xlinking operations. Two unlinking operations are used in EXORCISM:

1. primary unlinking,
2. secondary unlinking.

The primary unlinking is an inverse operation to the primary xlinking and the secondary unlinking is an inverse operation to the secondary xlinking.

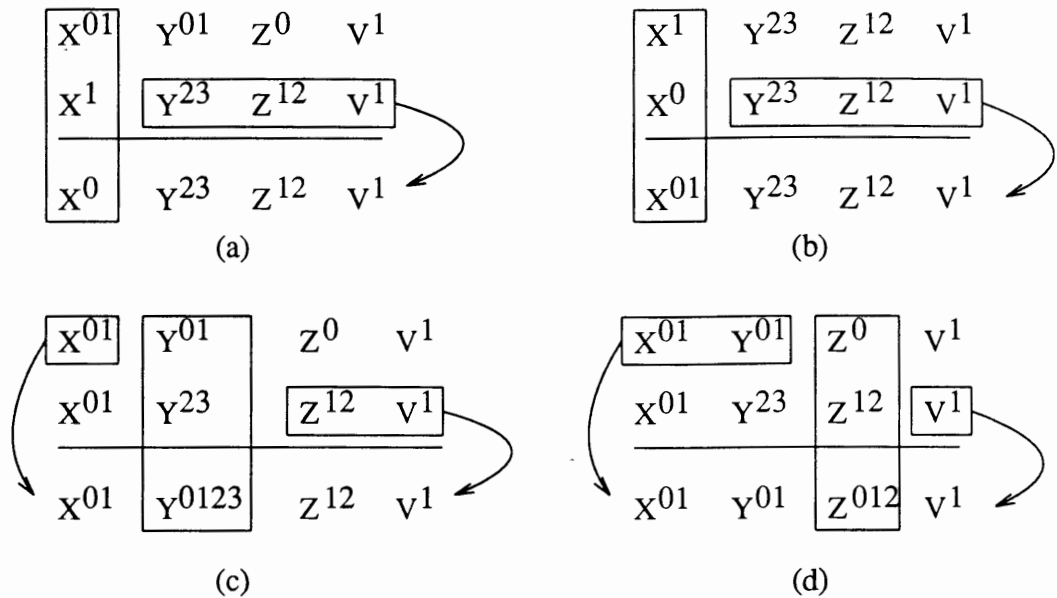


Figure 17. Procedure of distance 2 secondary xlinking.

III.5. THE OPERATIONS USED IN EXMIN

Sasao [Sasa 90a] used the following seven simplification rules in his algorithm EXMIN. The first rule, X-MERGE, can be applied if the difference of two terms is 1. The rest of the rules can be applied if the difference of two terms is 2 and certain conditions are satisfied. In this section, we are going to show all seven rule with examples. 4-valued logic is assumed for all the examples. Please note that for applying these rules, the two terms can have more than one or two literals. The number of resultant terms is equal to the difference of the two terms. Those pairs of literals which have the same truth value sets in both terms will keep the same truth value sets in the resultant terms. Those pairs of literals which have different truth value sets will generate different literals according to the corresponding rules.

(1) X-MERGE

$$X^a \oplus X^b = X^{(a \oplus b)}$$

Example III.8:

$$X^{12}Y^{01} \oplus X^{12}Y^{12} = X^{12}Y^{(01 \oplus 12)} = X^{12}Y^{02}$$

as shown in Figure 18.

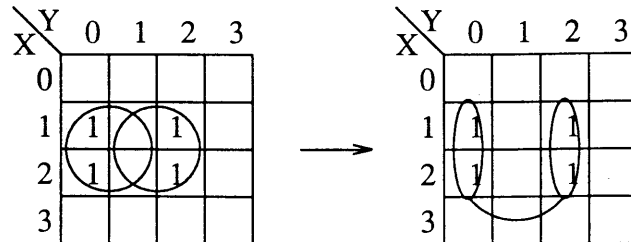


Figure 18. X-MERGE.

In example III.8, the difference of two terms is 1. So, X-MERGE can be applied and one resultant term will be generated. The literal X^{12} which is the same in both terms keeps the same truth value set in the resultant term. Another pair of literals, Y^{01} and Y^{12} generate the literal Y^{02} in the resultant term according to the rule.

(2) RESHAPE

$$X^a Y^b \oplus X^c Y^d = X^a Y^{(b \cap \bar{d})} \oplus X^{(a \cup c)} Y^d \quad \text{if } (a \cap c = \emptyset, b \supset d)$$

Example III.9:

$$X^1 Y^{12} \oplus X^{23} Y^1 = X^1 Y^2 \oplus X^{123} Y^1$$

as shown in Figure 19.

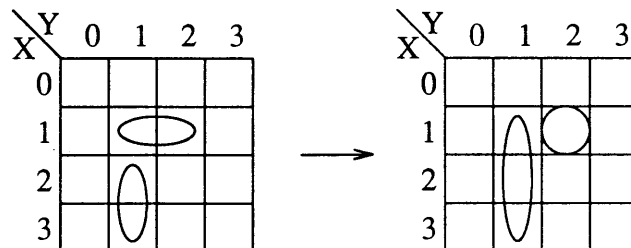


Figure 19. RESHAPE.

(3) DUAL-COMPLEMENT

$$X^a Y^b \oplus X^c Y^d = X^c Y^{(b \cap \bar{d})} \oplus X^{(\bar{a} \cap c)} Y^b \quad \text{if } (a \subset c, b \supset d)$$

Example III.10:

$$X^1 Y^{12} \oplus X^{123} Y^1 = X^{123} Y^2 \oplus X^{23} Y^{12}$$

as shown in Figure 20.

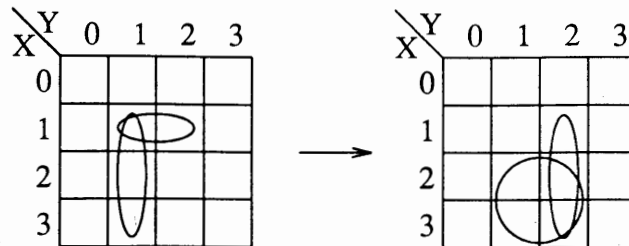


Figure 20. DUAL-COMPLEMENT.

(4) X-EXPAND-1

$$X^a Y^b \oplus X^c Y^d = X^a Y^{(b \cup d)} \oplus X^{(a \cup c)} Y^d = X^{(a \cup c)} Y^b \oplus X^c Y^{(b \cup d)} \\ \text{if } (a \cap c = \emptyset, b \cap d = \emptyset)$$

Example III.11:

$$X^{23} Y^1 \oplus X^1 Y^2 = X^{23} Y^{12} \oplus X^{123} Y^2 = X^{123} Y^1 \oplus X^1 Y^{12}$$

as shown in Figure 21.

(5) X-EXPAND-2

$$X^a Y^b \oplus X^c Y^d = X^{(a \cup c)} Y^b \oplus X^c Y^{(b \cap \bar{d})} \quad \text{if } (a \cap c = \emptyset, b \supset d)$$

Example III.12:

$$X^1 Y^{12} \oplus X^{23} Y^1 = X^{123} Y^{12} \oplus X^{23} Y^2$$

as shown in Figure 22.

(6) X-REDUCE-1

$$X^a Y^b \oplus X^c Y^d = X^{(a \cap \bar{c})} Y^b \oplus X^c Y^{(d \cap \bar{b})} \quad \text{if } (a \supset c, b \subset d)$$

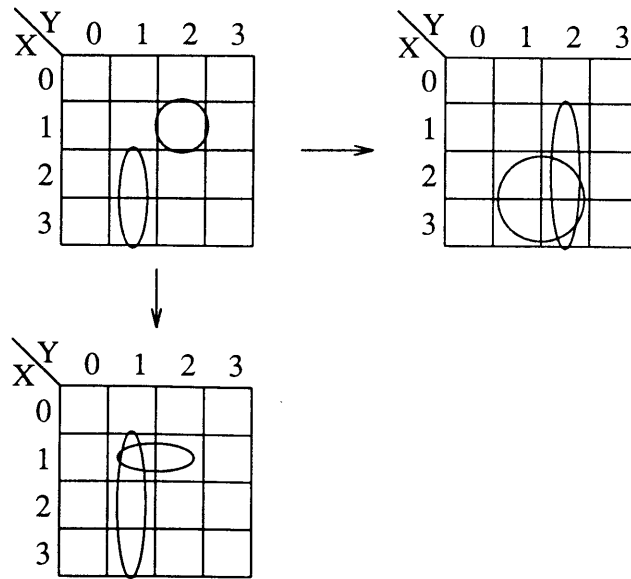


Figure 21. X-EXPAND-1.

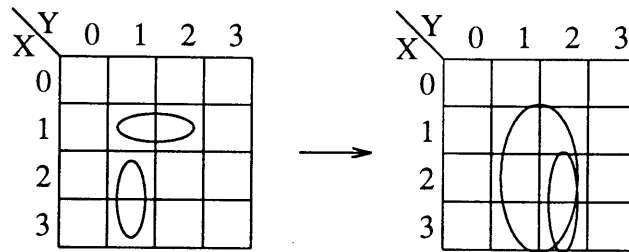


Figure 22. X-EXPAND-2.

Example III.13:

$$X^{123}Y^1 \oplus X^1Y^{12} = X^{23}Y^1 \oplus X^1Y^2$$

as shown in Figure 23.

(7) X-REDUCE-2

$$X^aY^b \oplus X^cY^d = X^{(a \cap \bar{c})}Y^b \oplus X^cY^{(b \cap \bar{d})} = X^aY^{(b \cap \bar{d})} \oplus X^{(a \cap \bar{c})}Y^d$$

if $(a \supset c, b \supset d)$

Example III.14:

$$X^{123}Y^{12} \oplus X^{23}Y^2 = X^{123}Y^1 \oplus X^1Y^2$$

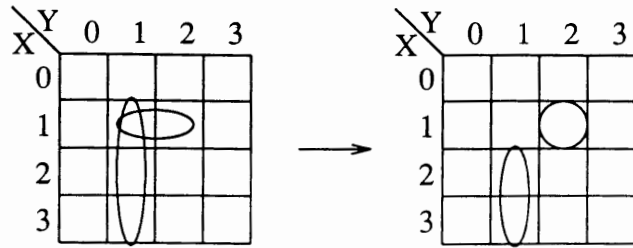


Figure 23. X-REDUCE-1.

as shown in Figure 24.

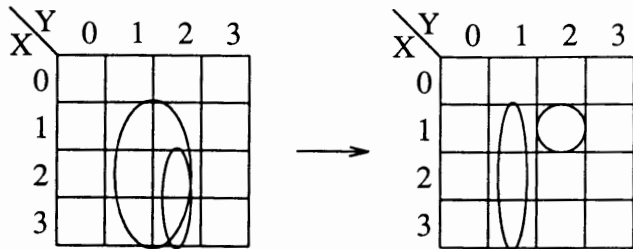


Figure 24. X-REDUCE-2.

CHAPTER IV

THE MULTIPLE-VALUED EXORLINKING OPERATION

IV.1. THE FORMULA

Let $T_S = X_1^{S_1} \dots X_n^{S_n}$ and $T_R = X_1^{R_1} \dots X_n^{R_n} \neq T_S$ be two terms. The *exorlink* of terms T_S and T_R is defined by the following formula:

$$T_S \otimes T_R = \bigoplus \{ X_1^{S_1} \dots X_{i-1}^{S_{i-1}} X_i^{(S_i \oplus R_i)} X_{i+1}^{R_{i+1}} \dots X_n^{R_n} \mid \text{for such } i = 1, \dots, n, \text{ that } S_i \neq R_i \}$$

We will give a *systematic procedure for finding exorlinks of full terms* below. The application of this procedure will be called *exorlinking*. The result of the procedure will be called the *exorlink* of the two original full terms.

Example III.15: To find the exorlink of a pair of two full terms, $T_S = X^{01} Y^{02} Z^{012} U^2 V^{13}$ and $T_R = X^{12} Y^{12} Z^2 U^2 V^{13}$, in a 4-valued function, we write them vertically as shown in Figure 25a.

Each time when the polarities of the literal are different from full term to full term in the pair it is denoted by an arrow. Each arrow will give rise to one term of the exorlink. Let us now consider each arrow separately. The above initial pair of full terms can then be expanded to three resultant terms for variables X , Y , and Z respectively, as shown in Figure 25.

1. For variable X , the resultant term $X^{02} Y^{12} Z^2 U^2 V^{13}$ is created as shown in Figure 25b. The literal X^{02} in the resultant term is generated by $X^{01} \oplus X^{12}$. Other literals in the resultant term are copied from the term T_R .
2. For variable Y , the resultant term $X^{01} Y^{01} Z^2 U^2 V^{13}$ is created as shown in Figure 25c.

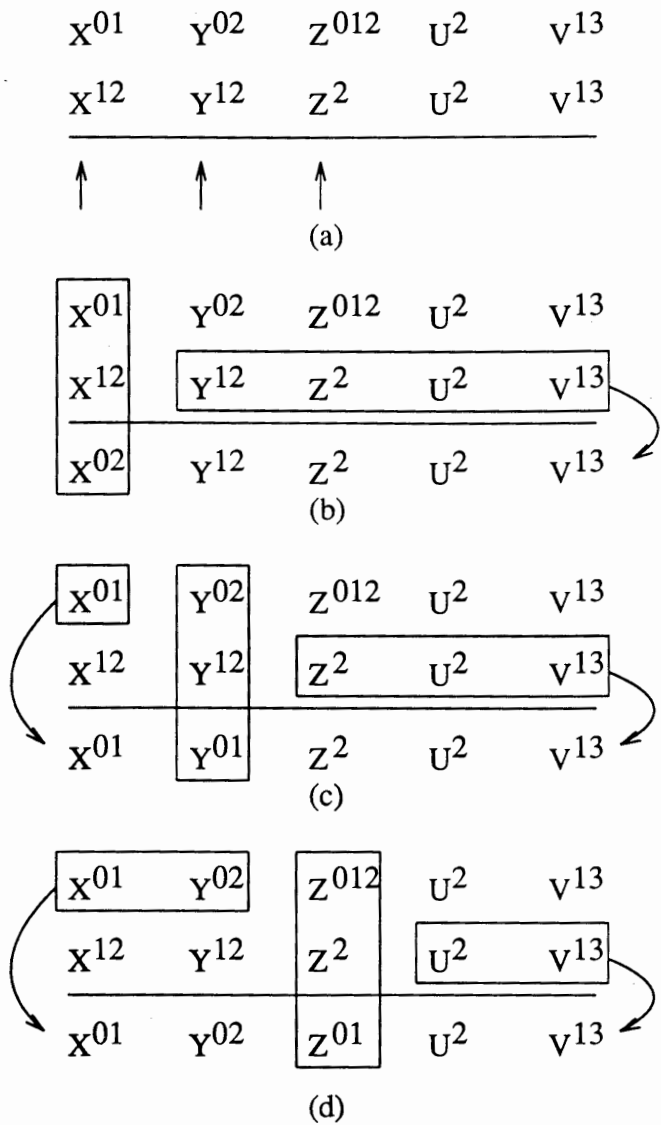


Figure 25. An example of exorlinking two terms.

3. For variable Z , the resultant term $X^{01} Y^{02} Z^{01} U^2 V^{13}$ is created as shown in Figure 25d.

Under each pair of literals of different sets of values under consideration (X in the first pair, Y in the second, Z in the third pair), we write the multiple-valued literal, with the set of truth values being the exclusive-sum of the respective sets from the literals of the terms. To create the result of exorlink for a resultant term, we copy the part of the

term to the left of the literal from the top full term, the part to the right of the literal is copied from the bottom full term as shown. The exorlink of the initial pair of full terms is an EXOR of exorlink terms of the second order pairs for each literal of different values. Therefore

$$X^{01}Y^{02}Z^{012}U^2V^{13} \otimes X^{12}Y^{12}Z^2U^2V^{13} = X^{02}Y^{12}Z^2U^2V^{13} \oplus X^{01}Y^{01}Z^2U^2V^{13} \oplus X^{01}Y^{02}Z^{01}U^2V^{13}.$$

This procedure can easily be further extended for any two terms in which for every two correspondingly different literals for the same variable, $X_i^{S_i}$ and $X_i^{R_i}$, the sets S_i and R_i are different.

Given terms T_S and T_R , if the difference of two terms is r , we call $T_S \otimes T_R$ the *difference r exorlinking*. If the distance of two terms is d , we call $T_S \otimes T_R$ the *distance d exorlinking*.

In cube notation, a term is represented by a cube, and each literal in the term is represented by a vector. We can write the formula of exorlinking in cube notation as follows:

$$A \otimes B = \bigoplus \{ [A_1, \dots, A_{i-1}, A_i \oplus B_i, B_{i+1}, \dots, B_n] \mid \text{for such } i = 1, \dots, n, \text{ that } A_i \neq B_i \}$$

The procedure of exorlinking two cubes is the same as exorlinking two terms. Figure 26 shows the exorlinking of two cubes corresponding to the two terms given in Example III.15.

Now let us prove that the exorlinking can be applied on any two cubes in the array no matter their difference.

Given two cubes $A = [A_1, A_2, \dots, A_n]$ and $B = [B_1, B_2, \dots, B_n]$ in the array:

- 1) Difference = 0 is trival. No resultant cube will be generated according to our formula. On the other hand, if the difference of two cubes is 0, they will be removed from the array. So, our formula is correct in this case.

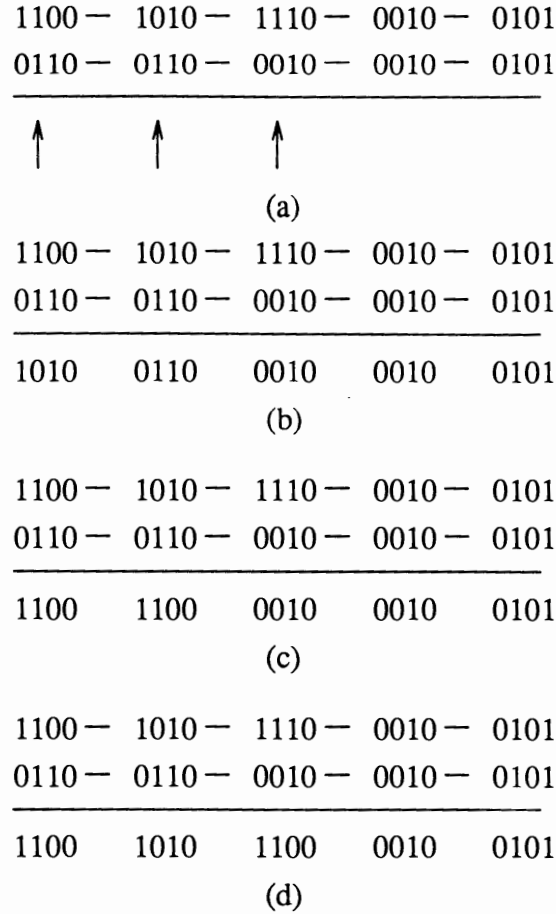


Figure 26. An example of exorlinking two cubes.

- 2) In the case difference = 1, without loss of generality, we can assume $A_i \neq B_i$.

According to the formula,

$$A \oplus B = [A_1, \dots, A_{i-1}, A_i \oplus B_i, B_{i+1}, \dots, B_n].$$

On the other hand,

$$A \otimes B = A \oplus B = [A_1, A_2, \dots, A_i, \dots, A_n] \oplus [B_1, B_2, \dots, B_i, \dots, B_n].$$

Since $A_j = B_j$ when $i \neq j$, we can rewrite A and B as

$$A = [A_1, \dots, A_i, B_{i+1}, \dots, B_n]$$

and

$$B = [A_1, \dots, A_{i-1}, B_i, \dots, B_n].$$

According to distributive law: $A B \oplus A C = A (B \oplus C)$, we have

$$A B D \oplus A C D = (A B \oplus A C) D = A (B \oplus C) D.$$

So,

$$\begin{aligned} A \oplus B &= [A_1, \dots, A_i, B_{i+1}, \dots, B_n] \oplus [A_1, \dots, A_{i-1}, B_i, \dots, B_n] \\ &= [A_1, \dots, A_{i-1}, A_i \oplus B_i, B_{i+1}, \dots, B_n]. \end{aligned}$$

- 3) Assume our formula is correct when difference = i , $i \geq 1$. Without loss of generality, let us assume $A_j \neq B_j$ when $j \leq i$ and $A_j = B_j$ when $j > i$. According to our formula, $A \otimes B$ will generate the following resultant cubes:

$$\begin{aligned} &[A_1 \oplus B_1, B_2, \dots, B_n] \\ &[A_1, A_2 \oplus B_2, B_3, \dots, B_n] \\ &\dots \\ &[A_1, \dots, A_{i-1}, A_i \oplus B_i, B_{i+1}, \dots, B_n]. \end{aligned}$$

- 4) We will prove that when difference = $i+1$, our formula is also correct. Given two cubes A' and B' . $A' = [A_1, \dots, A_i, \dots, A'_k, \dots, A_n]$ and $B' = [B_1, \dots, B_i, \dots, B'_k, \dots, B_n]$. The first i literals in the two cubes are different, as in the case of cubes A and B . Assume $A'_k \neq B'_k$ and $i < k \leq n$. So, difference(A', B') = $i+1$. We create a cube $A'' = [A_1, \dots, A_i, \dots, B'_k, \dots, A_n]$. All the literals in A'' are the same as in A' , except that A'_k is substituted by B'_k .

Since $A \oplus A = 0$, we have

$$A' \oplus B' = A' \oplus A'' \oplus A'' \oplus B'.$$

Since difference(A'', B') = i , according to 3), $A'' \oplus B' =$

$$\begin{aligned} &[A_1 \oplus B_1, B_2, \dots, B_n] \\ &[A_1, A_2 \oplus B_2, B_3, \dots, B_n] \\ &\dots \\ &[A_1, \dots, A_{i-1}, A_i \oplus B_i, B_{i+1}, \dots, B_n]. \end{aligned}$$

Since difference(A', A'') = 1, according to 2), $A' \oplus A'' =$

$$[A_1, \dots, A'_k \oplus B'_k, \dots, B_n].$$

So, $A'' \oplus B'' =$

$$[A_1 \oplus B_1, B_2, \dots, B_n]$$

$$[A_1, A_2 \oplus B_2, B_3, \dots, B_n]$$

...

$$[A_1, \dots, A_{i-1}, A_i \oplus B_i, B_{i+1}, \dots, B_n]$$

$$[A_1, \dots, A'_k \oplus B'_k, \dots, B_n].$$

These are exactly the same resultant cubes our formula will generate. So, our formula is correct no matter the difference of the two given cubes.

In the rest of this chapter, we will discuss difference 1, difference 2, and difference 3 exorlinking. These operations are used in our EXORCISM-MV-2 algorithm. We will also compare exorlinking with xlinking [Hell 88, Perk 89] and the operations in EXMIN [Sasa 90a]. For convenience, we will use terms in equations, and use cube notation in maps.

IV.2. DIFFERENCE 1 EXORLINKING

Given two terms T_S and T_R , assume X^{S_i} and X^{R_i} are a pair of literals in terms T_S and T_R , respectively. Assume $X^{S_i} \neq X^{R_i}$ and other pairs of literals in the two terms are equal. Then these two terms are difference 1 exorlinkable. Difference 1 exorlinking of two terms generates one resultant term. When the difference of two terms is 1, the distance of the two terms can be 0 or 1.

IV.2.1. Difference 1 Distance 1 Exorlinking

Example III.16: Let $T_S = X^{23} Y^{23}$, and $T_R = X^1 Y^{23}$.

$$T_S \otimes T_R = X^{(23) \oplus (1)} Y^{23} = X^{123} Y^{23}$$

This operation is equivalent to distance 1 primary xlinking of EXORCISM [Hell 88, Perk 89], and X-MERGE of EXMIN [Sasa 90a].

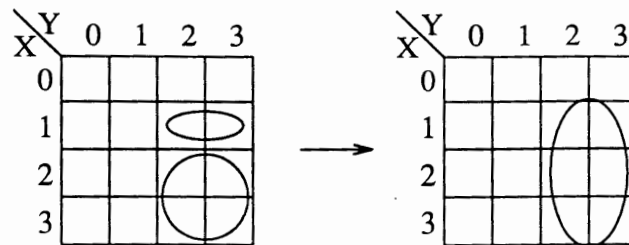


Figure 27. Difference 1 distance 1 exorlinking.

IV.2.2. Difference 1 Distance 0 Exorlinking

When the difference of two terms is 1 and the distance of two terms is 0, there are two possible cases: the two truth sets are overlapped, or one of the truth sets is contained in the other.

Example III.17: Let $T_S = X^{123} Y^{23}$ and $T_R = X^1 Y^{23}$.

$$T_S \otimes T_R = X^{\{123\} \oplus \{1\}} Y^{23} = X^{23} Y^{23}$$

The truth set of X in term T_R is contained by the truth set of X in term T_S . The operation is shown in Figure 28.

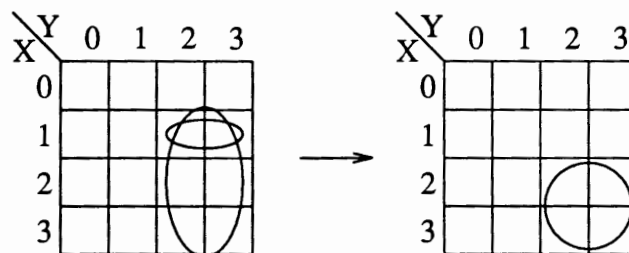


Figure 28. An example of difference 1 distance 0 exorlinking ($S_i \supset R_i$).

Example III.18: Let $T_S = X^{123} Y^{23}$ and $T_R = X^{01} Y^{23}$.

$$T_S \otimes T_R = X^{\{123\} \oplus \{01\}} Y^{23} = X^{023} Y^{23}$$

The truth sets of X in terms T_S and T_R are overlapped (denoted as $S_i \bowtie R_i$). The opera-

tion is shown in Figure 29.

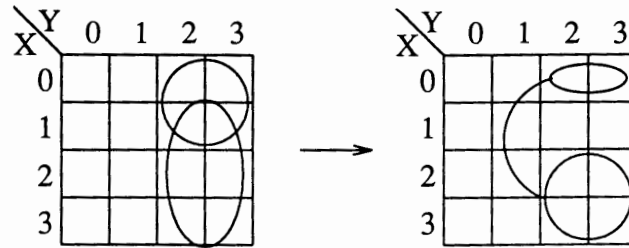


Figure 29. An example of difference 1 distance 0 exorlinking ($S_i \bowtie R_i$).

This operation is equivalent to X-MERGE of EXMIN [Sasa 90a].

IV.3. DIFFERENCE 2 EXORLINKING

Given two terms, if two pairs of their truth sets are different (assume $X^{S_i} \neq X^{R_i}$, and $Y^{S_j} \neq Y^{R_j}$), then difference 2 exorlink can be performed, and two resultant terms will be generated.

Let us observe that difference 2 exorlink of two terms T_S and T_R is different from the difference 2 exorlink of two terms T_R and T_S .

Example III.19: Given two terms $T_S = X^{013} Y^{13}$ and $T_R = X^{23} Y^{01}$:

$$T_S \otimes T_R = X^{013} Y^{13} \otimes X^{23} Y^{01} = X^{012} Y^{01} \oplus X^{013} Y^{03}$$

$$T_R \otimes T_S = X^{23} Y^{01} \otimes X^{013} Y^{13} = X^{012} Y^{13} \oplus X^{23} Y^{03}.$$

The procedures of these two operations are shown in Figure 30a and 30b.

As we discussed in section II.3.1 (see page 27), if the truth sets S_i and R_i are different, there are three possibilities:

- $S_i \cap R_i = \emptyset$,
- $S_i \bowtie R_i$,
- $S_i \supset R_i$ or $R_i \supset S_i$.

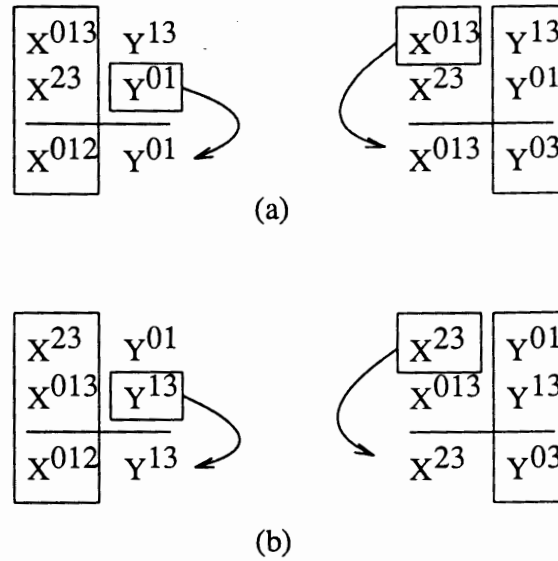


Figure 30. Difference 2 exorlinking ($T_S \otimes T_R$ and $T_R \otimes T_S$).

In the case of difference 2, there are many different combinations for relations S_i, R_i and S_j, R_j . Next, we are going to discuss each of these cases by showing examples.

$$1. \quad S_i \cap R_i = \emptyset, S_j \cap R_j = \emptyset$$

Given $T_S = X^{23}Y^1$, and $T_R = X^1Y^2$

$$T_S \otimes T_R = X^{23}Y^1 \otimes X^1Y^2 = X^{123}Y^2 \oplus X^{23}Y^{12}$$

as shown in Figure 31a.

$$T_R \otimes T_S = X^1Y^2 \otimes X^{23}Y^1 = X^{123}Y^1 \oplus X^1Y^{12}$$

as shown in Figure 31b.

This operation is equivalent to distance 2 primary xlinking of EXORCISM [Perk 89], and X-EXPAND-1 of EXMIN [Sasa 90a].

$$2. \quad S_i \cap R_i = \emptyset, S_j \supset R_j$$

Given $T_S = X^{23}Y^{12}$, and $T_R = X^1Y^2$

$$T_S \otimes T_R = X^{23}Y^{12} \otimes X^1Y^2 = X^{123}Y^2 \oplus X^{23}Y^1$$

as shown in Figure 32a.

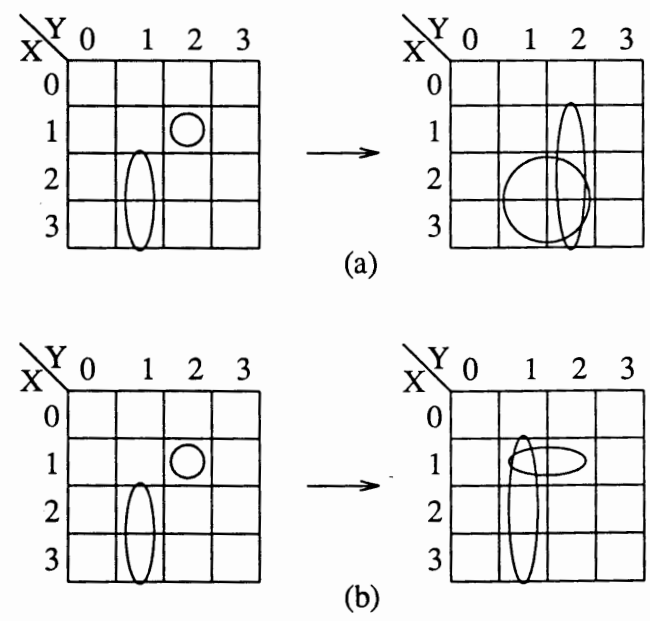


Figure 31. Difference 2 exorlinking ($S_i \cap R_i = \emptyset, S_j \cap R_j = \emptyset$).

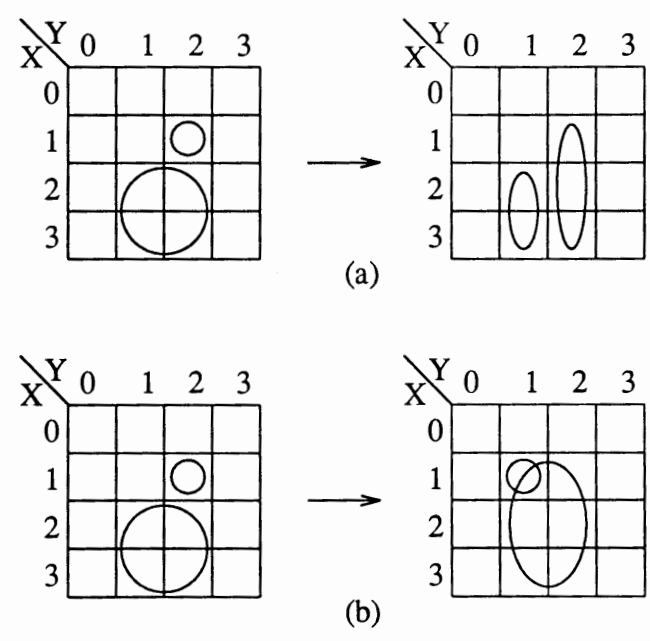


Figure 32. Difference 2 exorlinking ($S_i \cap R_i = \emptyset, S_j \supset R_j$).

$$T_R \otimes T_S = X^1Y^2 \otimes X^{23}Y^{12} = X^{123}Y^{12} \oplus X^1Y^1$$

as shown in Figure 32b.

The operation RESHAPE of EXMIN [Sasa 90a] generates the same result as Figure 32a.

The operation distance 1 secondary xlinking of EXORCISM [Perk 89], and X-EXPAND-2 of EXMIN [Sasa 90a] generate the same result as Figure 32b.

$$3. \quad S_i \cap R_i = \emptyset, S_j \not\ll R_j$$

Given $T_S = X^{23}Y^{12}$, and $T_R = X^1Y^{23}$

$$T_S \otimes T_R = X^{23}Y^{12} \otimes X^1Y^{23} = X^{123}Y^{23} \oplus X^{23}Y^{13}$$

as shown in Figure 33a.

$$T_R \otimes T_S = X^1Y^{23} \otimes X^{23}Y^{12} = X^{123}Y^{12} \oplus X^1Y^{13}$$

as shown in Figure 33b.

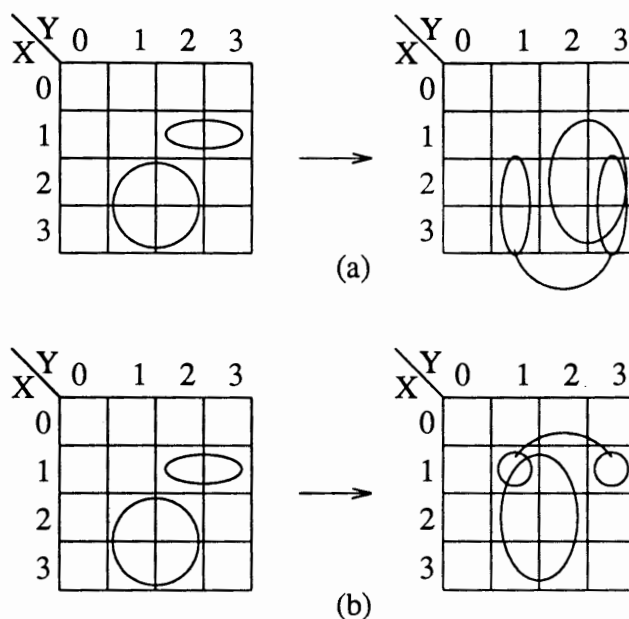


Figure 33. Difference 2 exorlinking ($S_i \cap R_i = \emptyset, S_j \not\ll R_j$).

No compatible operation can be found in the previous literature.

$$4. \quad S_i \supset R_i, S_j \supset R_j$$

Given $T_S = X^{123}Y^{12}$, and $T_R = X^{23}Y^2$

$$T_S \otimes T_R = X^{123}Y^{12} \otimes X^{23}Y^2 = X^1Y^2 \oplus X^{123}Y^1$$

as shown in Figure 34a.

$$T_R \otimes T_S = X^{23}Y^2 \otimes X^{123}Y^{12} = X^1Y^{12} \oplus X^{23}Y^1$$

as shown in Figure 34b.

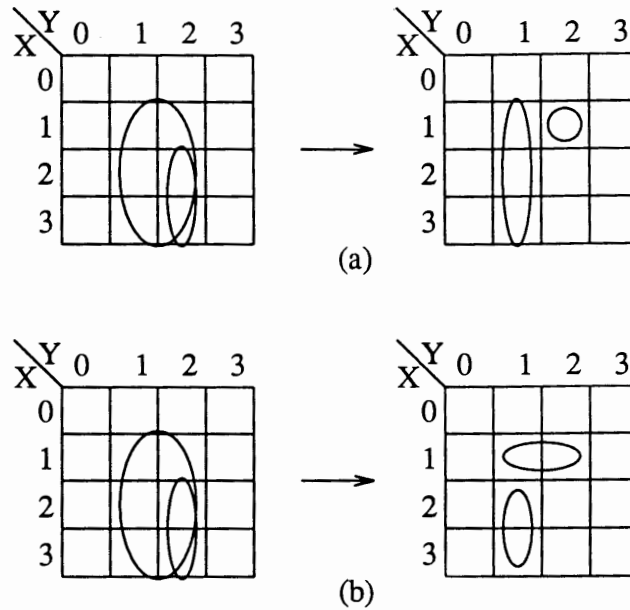


Figure 34. Difference 2 exorlinking ($S_i \supset R_i, S_j \supset R_j$).

This operation is equivalent to the secondary unlink of EXORCISM [Perk 89] and X-REDUCE-2 of EXMIN [Sasa 90a].

$$5. \quad S_i \supset R_i, S_j \subset R_j$$

Given $T_S = X^{123}Y^1$, and $T_R = X^1Y^{12}$

$$T_S \otimes T_R = X^{123}Y^1 \otimes X^1Y^{12} = X^{23}Y^{12} \oplus X^{123}Y^2$$

as shown in Figure 35a.

This operation is equivalent to DUAL-COMPLEMENT of EXMIN [Sasa 90a].

$$T_R \otimes T_S = X^1Y^{12} \otimes X^{123}Y^1 = X^{23}Y^1 \oplus X^1Y^2$$

as shown in Figure 35b.

This operation is equivalent to primary unlink of EXORCISM [Perk 89], and X-

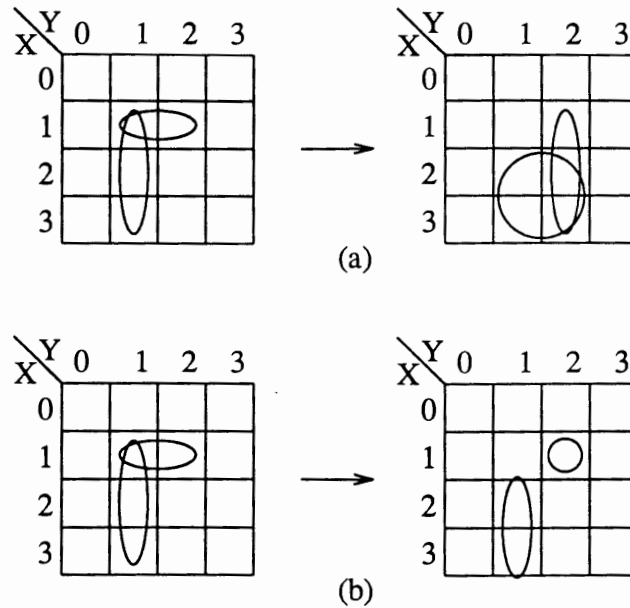


Figure 35. Difference 2 exorlinking ($S_i \supset R_i, S_j \subset R_j$).

REDUCE-1 of EXMIN [Sasa 90a].

$$6. \quad S_i \supset R_i, S_j \not\ll R_j$$

Given $T_S = X^{123}Y^{12}$, and $T_R = X^1Y^{23}$

$$T_S \otimes T_R = X^{123}Y^{12} \otimes X^1Y^{23} = X^{23}Y^{23} \oplus X^{123}Y^{13}$$

as shown in Figure 36a.

$$T_R \otimes T_S = X^1Y^{23} \otimes X^{123}Y^{12} = X^{23}Y^{12} \oplus X^1Y^{13}$$

as shown in Figure 36b.

No compatible operation can be found in the previous literature.

$$7. \quad S_i \not\ll R_i, S_j \not\ll R_j$$

Given $T_S = X^{23}Y^{12}$, and $T_R = X^{12}Y^{23}$

$$X^{23}Y^{12} \oplus X^{12}Y^{23} = X^{13}Y^{23} \oplus X^{23}Y^{13}$$

as shown in Figure 37a.

$$X^{12}Y^{23} \oplus X^{23}Y^{12} = X^{13}Y^{12} \oplus X^{12}Y^{13}$$

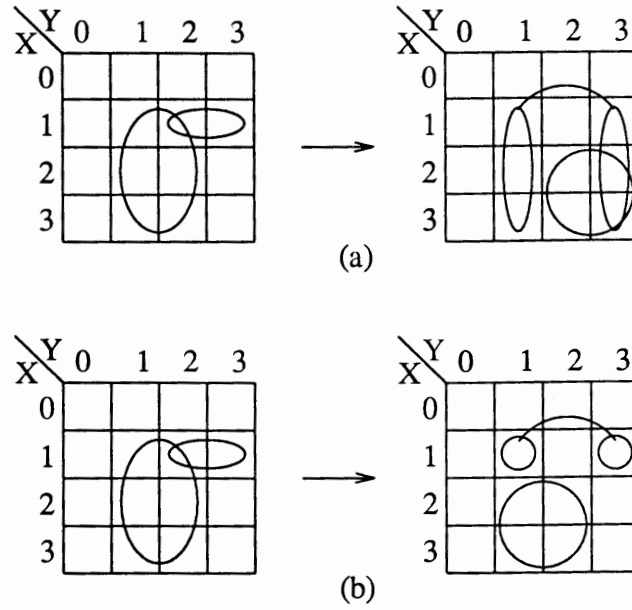


Figure 36. Difference 2 exorlinking $(S_i \supset R_i, S_j \not\ll R_j)$.

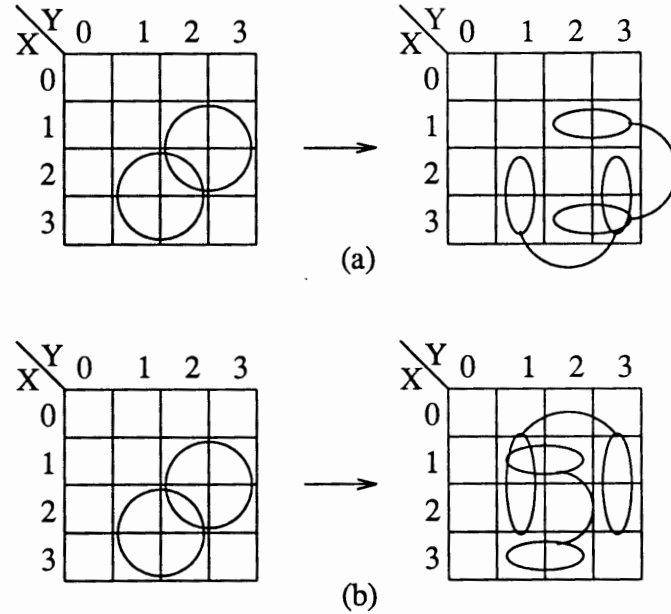


Figure 37. Difference 2 exorlinking $(S_i \not\ll R_i, S_j \not\ll R_j)$.

as shown in Figure 37b.

No compatible operation can be found in the previous literature.

The following two tables comparing the difference 2 operations of exorlinking with xlinking of EXORCISM [Perk 89] and operations of EXMIN [Sasa 90a].

TABLE VIII
COMPARISON OF EXORLINKING WITH XLINKING

	$S_i \cap R_i = \emptyset$ $S_j \cap R_j = \emptyset$	$S_i \cap R_i = \emptyset$ $S_j \supset R_j$	$S_i \cap R_i = \emptyset$ $S_j \not\supset R_j$	$S_i \supset R_i$ $S_j \supset R_j$	$S_i \supset R_i$ $S_j \subset R_j$	$S_i \supset R_i$ $S_j \not\supset R_j$	$S_i \not\supset R_i$ $S_j \not\supset R_j$
$T_S \otimes T_R$	distance 2 primary xlinking	distance 1 secondary xlinking		distance 1 secondary unlinking			
$T_R \otimes T_S$	distance 2 primary xlinking			distance 1 secondary unlinking	distance 2 primary unlinking		

TABLE IX
COMPARISON OF EXORLINKING WITH OPERATIONS IN EXMIN

	$S_i \cap R_i = \emptyset$ $S_j \cap R_j = \emptyset$	$S_i \cap R_i = \emptyset$ $S_j \supset R_j$	$S_i \cap R_i = \emptyset$ $S_j \not\supset R_j$	$S_i \supset R_i$ $S_j \supset R_j$
$T_S \otimes T_R$	X-EXPAND-1	RESHAPE		X-REDUCE-2
$T_R \otimes T_S$	X-EXPAND-1	X-EXPAND-2		X-REDUCE-2

	$S_i \supset R_i$ $S_j \subset R_j$	$S_i \supset R_i$ $S_j \not\supset R_j$	$S_i \not\supset R_i$ $S_j \not\supset R_j$
$T_S \otimes T_R$	DUAL-COMPLEMENT		
$T_R \otimes T_S$	X-REDUCE-1		

As we discussed in section III.3.3. (see page 33), difference 2 operations do not directly reduce the number of cubes in an ESOP. However, these operations provide opportunities for reducing the cost of ESOPs at later stages. Exorlinking has more difference 2 operations than approaches of EXMIN [Sasa 90a] and EXORCISM [Hell 88, Perk 89], which means it provides more opportunities for reducing the cost of ESOPs than

EXMIN or EXORCISM.

Example III.20: Given is an ESOP with three terms: $T_1 = X^{12}Y^{23}$, $T_2 = X^{23}Y^{12}$ and $T_3 = X^0Y^{13}$. Any pairs of these three terms are different by two, but no operations in EXORCISM or EXMIN can be applied to these terms. We can, however, apply exorlinking to these terms. In Figure 38, three terms T_1, T_2 and T_3 are represented by three cubes A, B and C , respectively. $A \otimes B$ generates A' and B' ; $A' \otimes C$ generates A'' . The ESOP with three cubes is minimized to an ESOP with two cubes.

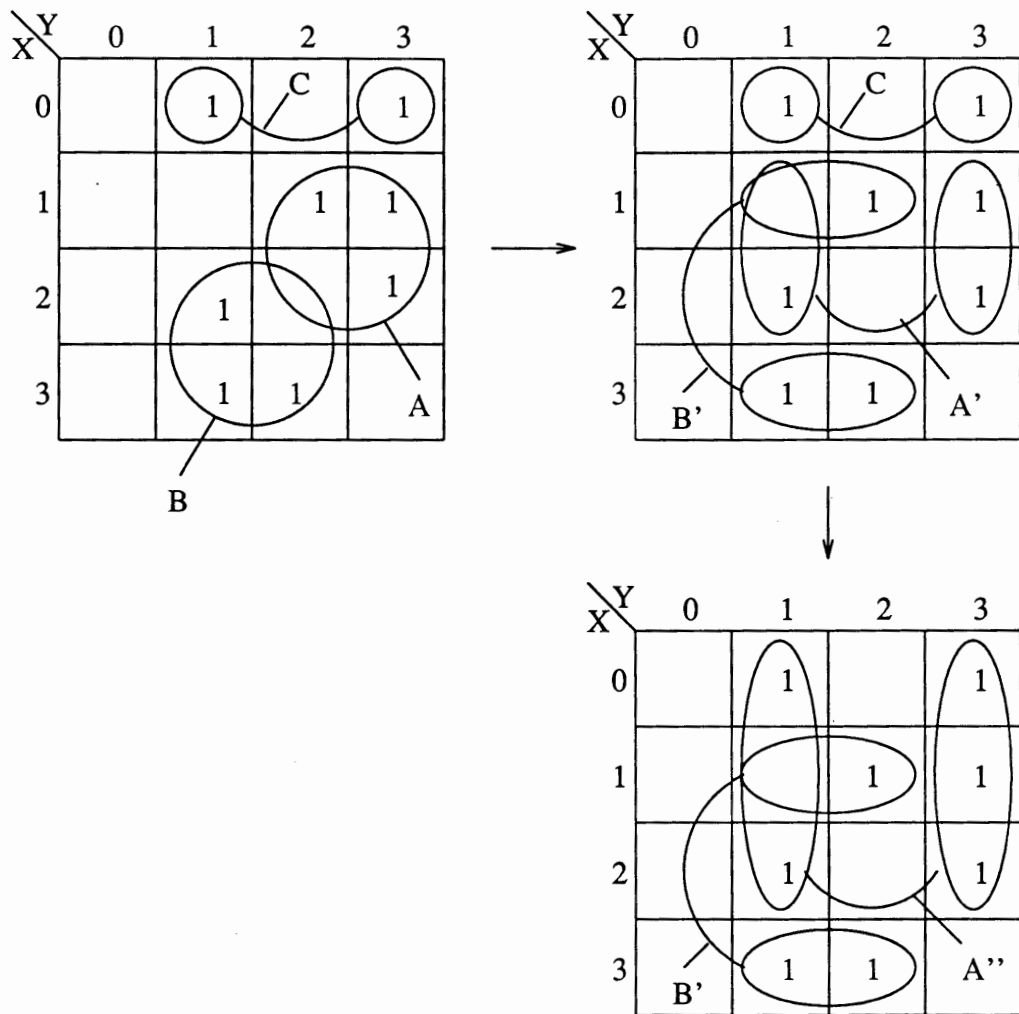


Figure 38. An example of ESOP minimization by difference 2 exorlinking.

IV.4. DIFFERENCE 3 EXORLINKING

Difference 3 exorlink generates three resultant terms from two given terms. If $S_i \supset R_i$, $S_j \cap R_j = \emptyset$, and $S_k \cap R_k = \emptyset$, the operation is equivalent to distance 2 secondary xlinking of EXORCISM [Perk 89].

Difference 3 exorlinking increases the number of terms in the ESOP. However, as we discussed in section III.3.4. (see page 34), increasing the number of terms may help to reduce the number of terms at the later stage and get better results.

Example III.21: In binary logic, a given ESOP with 4 cubes is as follows:

$$000x, 0x11, x11x, 1010.$$

The ESOP is shown in Figure 39a. Any pair of these cubes differ by 3. So, there are no difference 1 or difference 2 operations that can be performed. Performing difference 3 exorlinking on the first two cubes, we get:

$$000x \otimes 0x11 = 0111 \oplus 00x1 \oplus 0000.$$

Replacing the first two cubes by these three cubes, we get a new ESOP with five cubes:

$$0111, 00x1, 0000, x11x, 1010,$$

which is shown in Figure 39b. Two of the above cubes differ by 2: 0000 and 1010. We can perform difference 2 exorlinking on them:

$$0000 \otimes 1010 = x010 \oplus 00x0.$$

After this operation, the ESOP contains five cubes:

$$0111, 00x1, x010, x11x, 00x0,$$

which is shown in Figure 39c.

Now, the cubes 00x1 and 00x0 differ by 1, we can perform difference 1 exorlinking on them:

$$00x1 \otimes 00x0 = 00xx.$$

The ESOP now contains four cubes:

$$0111, 00xx, x010, x11x,$$

which is shown in Figure 39d.

Performing difference 2 exor linking on cubes $x010$ and $x11x$, we obtain:

$$x010 \otimes x11x = xx10 \oplus x111.$$

The ESOP is

$$0111, 00xx, xx10, x111,$$

which is shown in Figure 39e.

Cubes 0111 and $x111$ can be combined to one cube:

$$0111 \otimes x111 = 1111.$$

The final result is an ESOP with three cubes as shown in Figure 39f. By using difference 3 exorlinking, the number of the ESOP is temporarily increased, but it helps to jump out of the local minimum and achieve a better result.

IV.5. SUMMARY

Xlinking uses one formula to present the primary xlinking for any distances. Another formula is used for distance 2 secondary xlinking. Separate rules are used for unlinking. There are no general formulas in EXMIN. Each operation is presented by a separate formula. However, there are more operations in difference 1 and difference 2 in EXMIN than in EXORCISM. That is one of the reasons that EXMIN generates better results than EXORCISM [Sasa 90a]. Exorlinking is described by one formula. It contains all the possible operations in xlinking approach. In other words, it can link any two terms in a given ESOP without condition. All the operations used in EXORCISM and EXMIN are included in exorlinking as special cases. In xlinking approach, the basic method is to perform different cube operations iteratively to combine or reshape cubes. More operations will provide more opportunity to minimize the ESOPs. This is the reason that exorlinking is superior than xlinking and the operations in EXMIN.

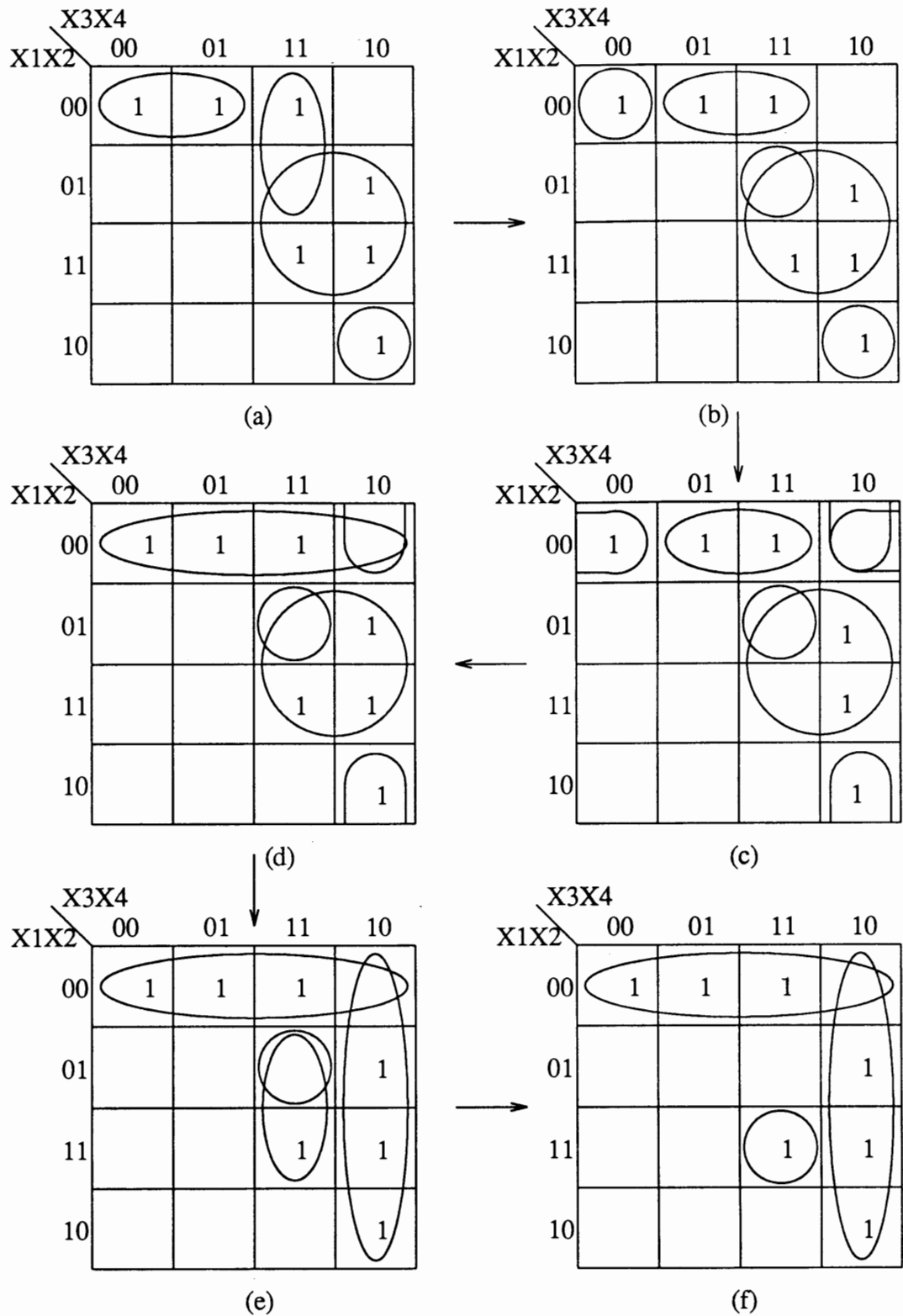


Figure 39. An example of ESOP minimization by difference 3 exorlinking.

CHAPTER V

ALGORITHM OF EXORCISM-MV-2 PROGRAM

V.1. THE ALGORITHM OF EXORCISM

EXORCISM uses the following algorithm to minimize an ESOP:

1. For each pair of cubes in the ESOP, do distance 1 primary xlinking.
2. For each pair of cubes in the ESOP, do distance 2 primary xlinking.
3. For each pair of cubes in the ESOP, do distance 1 secondary xlinking.
4. For each pair of cubes in the ESOP, do distance 2 secondary xlinking.
5. If the cost is reduced, go to step 1, else go to step 6.
6. Do unlinking.
7. If time has not exceeded the time limit, then go to step 1, else stop.

V.2. THE ALGORITHM OF EXMIN

1. For each pair of cubes in the ESOP, do X-MERGE if possible.
2. For each pair of cubes in the ESOP, do the following operations if possible:
RESHAPE, DUAL-COMPLEMENT, X-EXPAND-1, AND X-EXPAND-2.
3. For the cubes modified by the above operations, do X-MERGE if possible.
4. In step 3, if X-MERGE is performed, go to step 2, else go to step 5.
5. In step 3, if X-EXPAND-1 or X-EXPAND-2 is performed, go to step 2, else go to step 6.

6. For each pair of cubes in the ESOP, do X-REDUCE-1 and X-REDUCE-2 if possible.
7. If the number of cubes is reduced, go to step 1, else stop.

While EXMIN has more operations for a difference 2 pair of cubes than EXOR-CISM has, it also introduces a new problem: it may fall into an infinite loop. Sasao gave the following example to show such a case.

Example III.22:

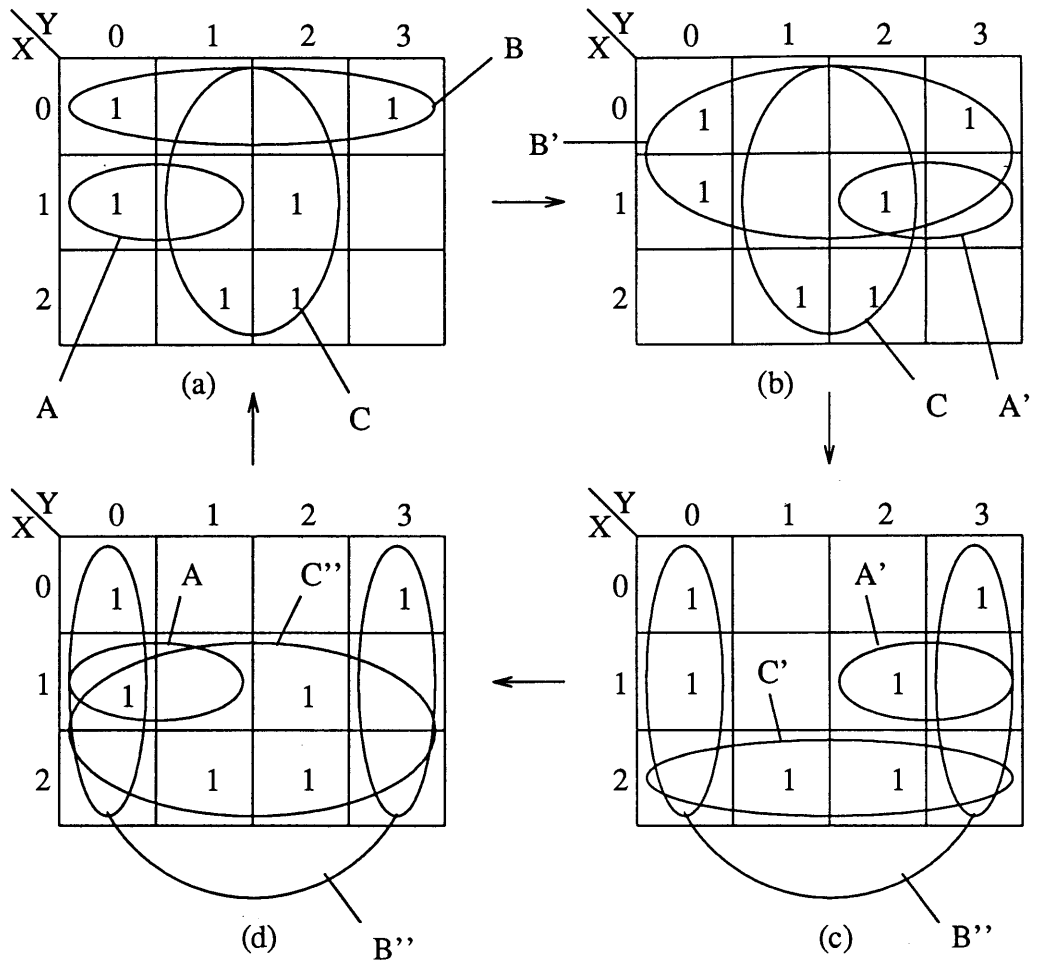


Figure 40. An example of infinite loop in EXMIN.

In Example III.22, an ESOP with three cubes $A = [010 - 1100]$, $B = [100 - 1111]$ and C

$= [111 - 0110]$ is shown in Figure 40a. Performing X-EXPAND-2 on cubes A and B , we get A' and B' in Figure 40b. Performing DUAL-COMPLEMENT on B' and C , we get B'' and C' as shown in Figure 40c. Performing X-EXPAND-2 on cubes A' and C' , we get cube A and C'' as shown in Figure 40d. Performing DUAL-COMPLEMENT on B'' and C'' , we get cubes B and C , which is the original ESOP as shown in Figure 40a.

For avoiding the problem of infinite loop, Sasao separates the difference 2 operations to two groups: X-REDUCE-1 and X-REDUCE-2 form one group, and the rest of the operations form another group. Two groups of operations are performed separately (step 2 and step 6 in above).

V.3. THE NEW ALGORITHM

As we discussed in section 4.3, the main purpose of difference 2 operations is to provide opportunities for difference 1 or difference 0 operations. So, the new algorithm is: instead of doing all possible difference 2 operations, perform only those difference 2 operations which will lead to difference 0 or difference 1 operations. More specifically, if two cubes, A and B , are different by 2, then $A \otimes B$ generates resultant cubes C_1 and C_2 , $B \otimes A$ generates resultant cubes D_1 and D_2 . We check the cubes C_1 , C_2 , D_1 , and D_2 with all the cubes in the ESOP except cubes A and B . If difference 0 or difference 1 operations are possible, we perform the difference 2 operation $A \otimes B$ or $B \otimes A$ followed by difference 0 or difference 1 operations. Otherwise, this difference 2 operation is not performed.

Example III.23: Given an ESOP with 5 cubes:

$$0011 \ 0110 \ 1111 \ 010x \ 10x1.$$

If we perform all the possible difference 2 operations on these cubes, the procedure is shown in Figure 41.

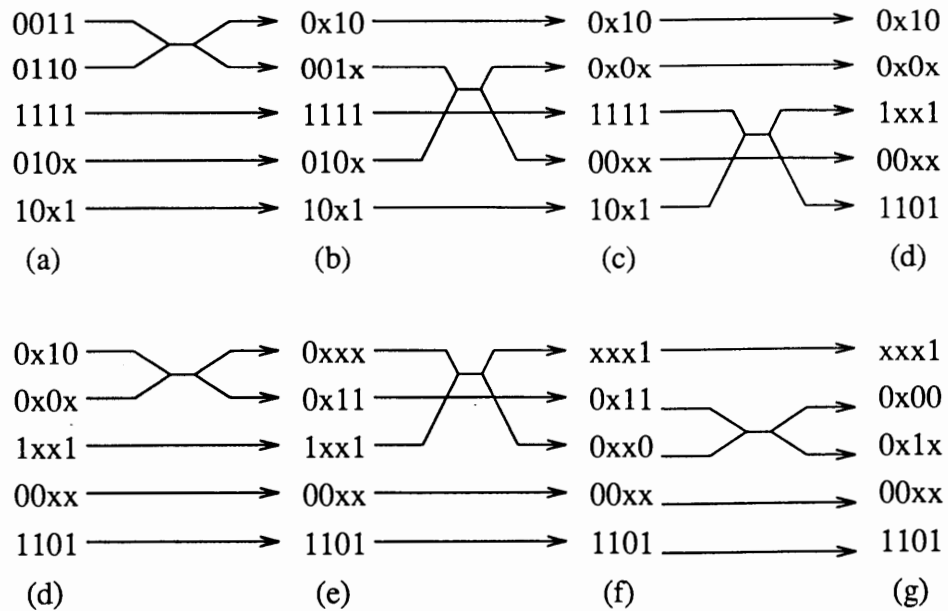


Figure 41. An example of performing all possible difference 2 operations.

Figure 42 shows the procedure using Karnaugh maps. From one map to the next map, two cubes are reshaped. Figure 42a shows the original ESOP with 5 cubes. After six operations, the same function is represented by another ESOP with 5 cubes as shown in Figure 42g. The number of cubes has not been reduced so far.

Our new approach is shown in Figure 43:

1. Perform all possible remove_equal and difference 1 exorlinking operations. In this example, none of these operations are possible now (see Figure 43a).
2. Check if two cubes differ by 2. For instance, cube 0011 and 0110 differ by 2 as shown in Figure 43a.
3. Check the two pairs of resultant cubes with other cubes in the array to see if we can find two cubes that differ by 0 or by 1. In the example, cube 0011 and 0110 generate two pairs of resultant cubes:

$$0110 \otimes 0011 = 0x11 \oplus 011x$$

as shown in Figure 43b1, and

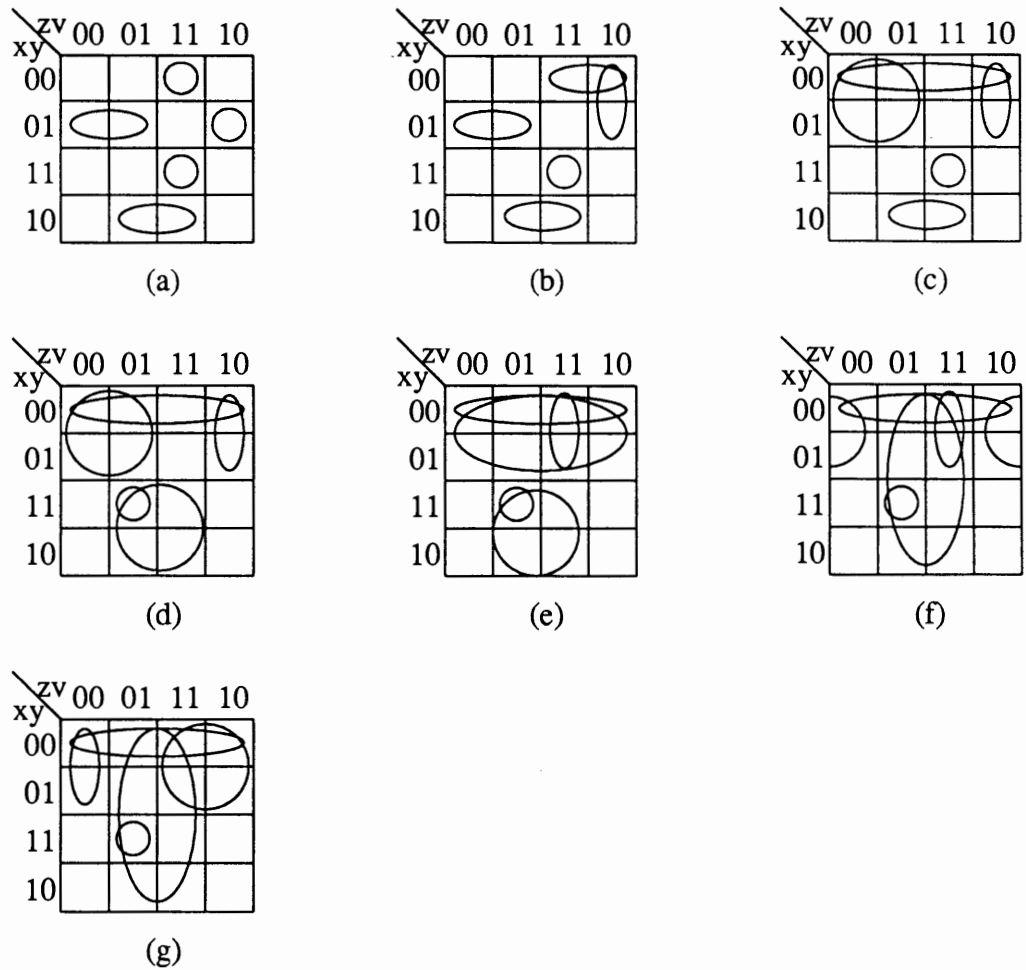


Figure 42. Karnaugh maps for Example III.23.

$$0011 \otimes 0110 = 0x10 \oplus 001x$$

as shown in Figure 43b2.

We check these four resultant cubes with the rest of the cubes in the array:

$$1111 \ 010x \ 10x1$$

and we can find that cubes $011x$ and $010x$ are different by 1 as shown in Figure 43b1.

4. In step 3, if we can find two cubes that differ by 0 or different by 1, we perform the difference 2 exorlinking and then perform the remove_equal or difference 1 exorlinking operation. For instance, in step 3 we found two cubes $011x$ and $010x$ that

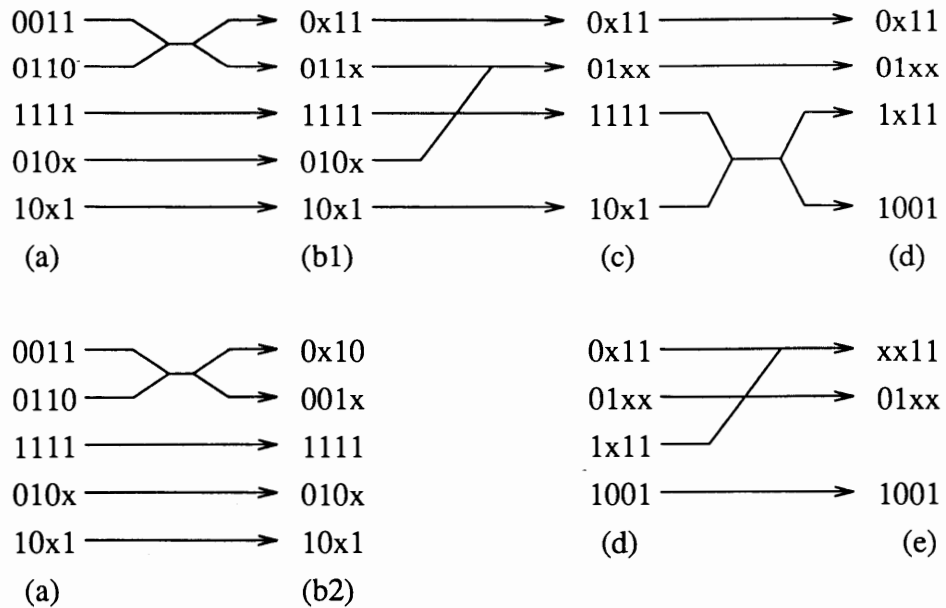


Figure 43. Conditionally performing difference 2 exorlinking.

differ by 1, we perform

$$0110 \otimes 0011 = 0x11 \oplus 011x$$

and then we perform

$$011x \otimes 010x = 01xx$$

as shown in Figure 43c.

After performing remove_equal or difference 1 exorlinking, we go back to step 1. If we can not find any two cubes that differ by 0 or by 1 in step 3, we do not perform difference 2 operation, and go back to step 2 to check other two cubes.

We continue this loop as shown in Figure 43d and 43e. If the number of cubes has not been reduced for certain number of iterations, we go to next stage of minimization. Figure 44 shows this approach in the Karnaugh maps. Comparing the Figure 41 with the Figure 43, we can see that our new approach is more efficient.

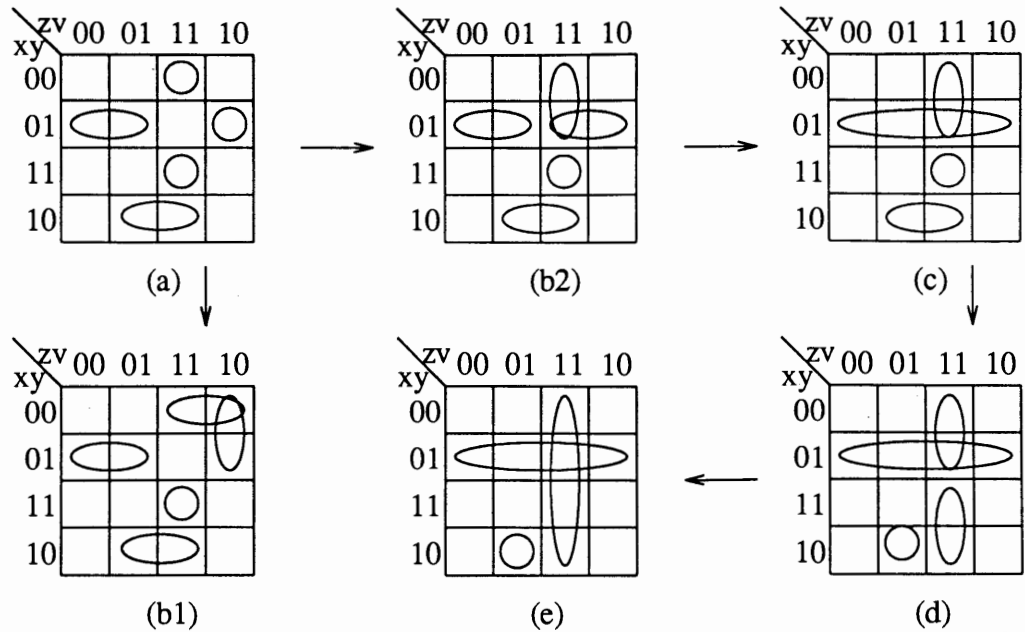


Figure 44. Karnaugh maps corresponding to Figure 43.

V.4. MINIMIZATION OF MULTIPLE OUTPUT FUNCTIONS

There are two ways to minimize a multiple output function:

1. decompose the multiple output functions to single output functions, minimize each single output function separately, and then minimize the set of functions again;
2. minimize the multiple output function directly.

Both the EXORCISM and the EXMIN use the first method. In our program, we let the user to select which method to use. The following procedure shows our algorithm.

1. If the function is a multiple output one and the option "decomposition to single output" is selected, then go to step 2; otherwise, go to step 5.
2. Decompose the function to a set of single output functions.
3. Minimize each single output function separately.

4. Combine the minimized single output functions to a multiple output function.
5. Minimize the function.

Example III.24 shows this procedure.

Example III.24: In Example II.11 (page 22), the multiple-output function $F(x, y, z)$ can be represented by the following array of cubes:

001 10

010 11

101 10

111 11.

The first three symbols in each cube represent input variables, the last two symbols represent output variables. For instance, the first cube in the array, 001 10, means that when input combination is $x = 0$, $y = 0$, and $z = 1$, the output variables $f_0 = 1$ and $f_1 = 0$. Since this is a two output function, we can decompose it to two single output functions which is represented by the following array of 6 cubes:

001 10

010 10

101 10

111 10

010 01

111 01.

By minimizing the first 4 cubes, we get

01x 10

xx1 10.

By minimizing the last two cubes, we get

010 01

111 01.

Put these 4 cubes together, we get a solution, which is

01x 10

xx 1 10

010 01

111 01.

Minimize these 4 cubes again, we get the final result:

x 11 01

xx 1 10

01x 11

which is the same result we showed in Example II.11.

By experimentation, we found that this method is on average better than the method without decomposition. Table X shows the comparison. In our algorithm, by default, the decomposition to a set of single output functions is used.

TABLE X
MULTIPLE OUTPUT FUNCTION

	Decomposition to single output functions				Without decomposition			
	cube	AND	EXOR	Time	cube	AND	EXOR	Time
ADR4	31	114	40	7.5	31	106	40	5.4
LOG8	95	508	192	266.8	96	515	226	207.4
MLP4	62	301	86	38.2	62	311	96	43.8
NRM4	67	375	131	51.1	72	404	136	65.0
RDM8	31	110	42	8.5	31	111	44	10.7
ROT8	37	200	62	6.4	36	193	64	8.2
SQR8	114	533	202	264.4	126	622	240	352.4
WGT8	58	261	64	60.5	60	260	79	250.2

In Table X, cube indicates how many cubes (terms) in the solution. AND indicates how many wires to AND gates. EXOR indicates how many wires to the EXOR gates. Time indicates the user time on a SPARC II station.

V.5. MINIMIZATION OF INCOMPLETELY SPECIFIED FUNCTIONS

An incompletely specified function can be represented by an ON-array of cubes and a DC-array of cubes. We can minimize an incompletely specified function by minimizing its ON-array of cubes only. However, linking the ON-array of cubes with the DC-array of cubes may generate better results.

Example III.25: Given is an ESOP with one ON-cube, $01x1$, and two DC-cubes, $11x1$ and $1x10$, as shown in Figure 45a. The function can be represented by the ON-cube only. By linking the ON-cube with one of the DC-cubes, we get the cube $x1x1$ as shown in Figure 45b, which is a better result than ON-cube $01x1$.

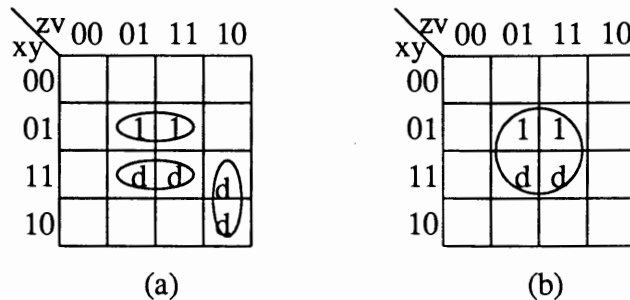


Figure 45. Example of minimizing an incompletely specified function.

Saul [Saul 90] pointed out that minimization of incompletely specified functions in ESOP form is difficult, because:

1. The DC-cubes may cover some minterms which are not in the DC-array, as shown in Figure 46.
2. The DC-array may not contain a cube that can be directly linked with a cube in ON-array because of positions or sizes of the DC-cubes, as shown in Figure 47a and 48a, respectively.

In Figure 46, cubes $x101$ and $01x1$ are in the DC-array, and cube $x100$ is an ON-cube. We can not link the DC-cube $x101$ with the ON-cube $x100$, because the DC-cube

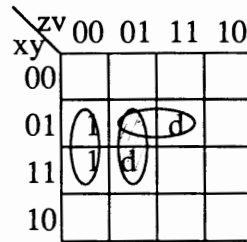


Figure 46. Linking DC-cubes.

$x 101$ contains a minterm 0101, which is a false minterm. This problem can be solved by making the DC-array disjoint. In a disjoint DC-array, each DC-minterm is covered by a cube once, and a false minterm is not covered by any cubes.

In Figure 47a, the ON-cube can not be linked with any one of the DC-cubes. If the DC-cubes are in the right position, however, they can be linked as shown in Figure 47b and 47c.

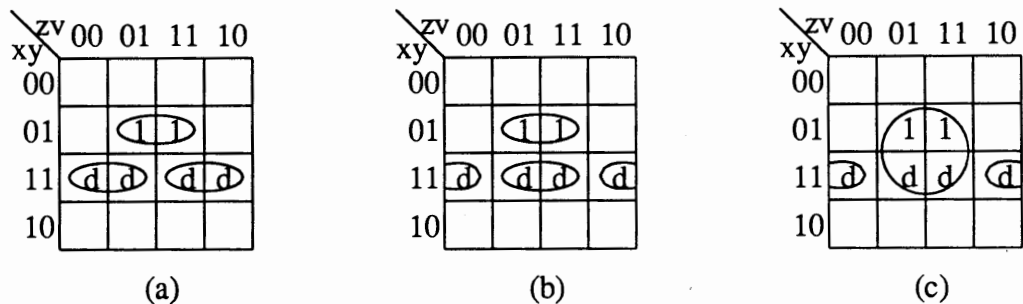


Figure 47. The position of DC-cubes.

In Figure 48a, the ON-cube can not be linked with the DC-cube, because the size of the DC-cube is larger than the size of the ON-cube. If we can separate the DC-cube properly, as shown in Figure 48b, then the ON-cube can be linked with one of the DC-cubes, as shown in Figure 48c.

Saul [Saul 90] gave the following algorithm to link the ON-cubes with the DC-cubes:

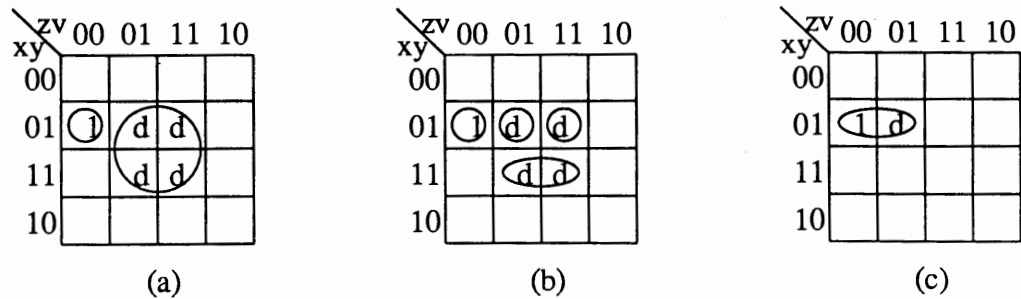


Figure 48. The size of DC-cubes.

```

function don't care minimize ( $F$ , inverse_ $D$ )
  for (each cube) {
    blocking_cover ← sharp (inverse_ $D$ , cube)
    cube ← expand (blocking_cover, cube)
  }
  return  $F$ .

```

Here, F is the ON-array of cubes, D is the DC-array of cubes, and inverse_ D is the complemented DC-array of cubes. Before calling this algorithm, the DC-array is converted to the disjoint sum of products representation, and then complemented. In the algorithm, "for each cube" means for each cube in the ON-array F . Inverse_ D covers all the minterms that are not covered by the DC-array. Please note that inverse_ D is an array of cubes. So, sharp(inverse_ D , cube) means

```

for each cube  $c_d$  in the inverse_ $D$  {
  do  $c_d \#$  cube
}

```

By using the sharp operation, block_cover contains the minterms that are not covered either by DC-array or by the cube. The operation expand is carried out by expanding each cube in such a way that it does not intersect the cube with the blocking_cover. Let us look at the following example to make this procedure clear.

Example III.26: Given an ON-array with one cube, and a DC-array with two cubes as shown in Figure 49a.

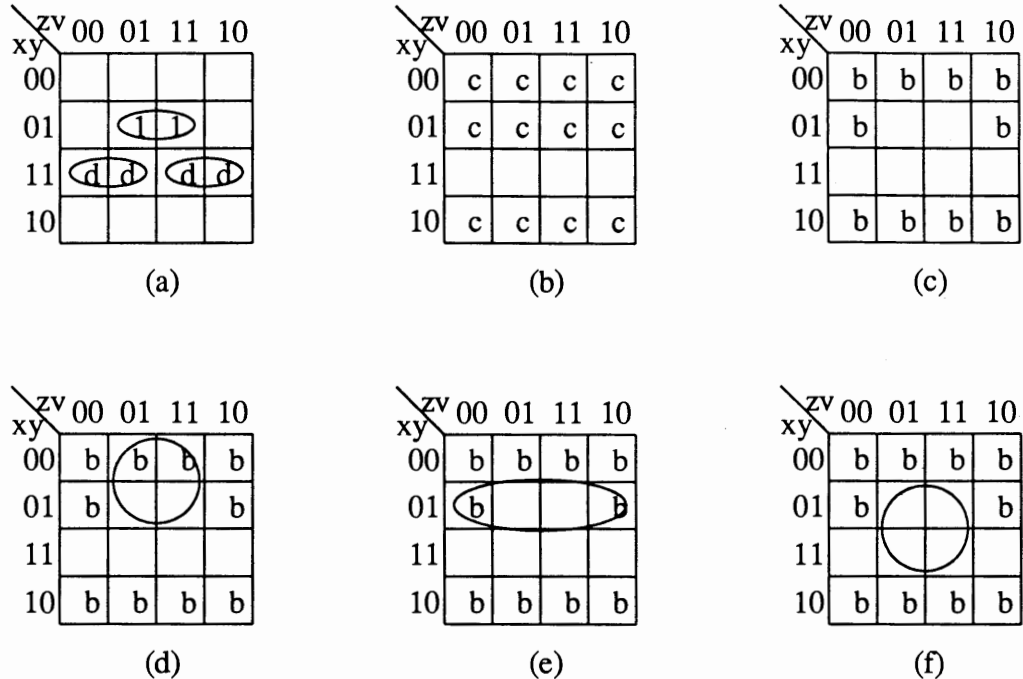


Figure 49. Linking DC-cubes by Saul's algorithm.

The two DC-cubes can not be linked with the ON-cube because of their positions. According to Saul's algorithm, the following operations will be performed:

1. Generate $inverse_D$: $inverse_D$ is an array of cubes which should cover all the minterms marked as "c" in Figure 49b. $inverse_D$ can be in the form of disjoint SOPs or non-disjoint SOPs.
2. Generate $block_cover$: Performing the sharp operation on $inverse_D$ and the ON-cube, we can get an array of cubes called $block_cover$. $block_cover$ contains all the minterms marked as "b" in Figure 49c. These minterms are not covered by DC-cubes and are not covered by the ON-cube.
3. Expanding the ON-cube: There are three ways to expand the ON-cube $01x1$ as shown in Figure 49d, 49e and 49f, respectively. In Figure 49d and 49e, the

expanded ON-cube intersects the `block_cover`, which means such an expanding is impossible. So, the only possible way to expand the ON-cube is shown in Figure 49f.

Saul's algorithm has an obvious limitation: it tries to reduce the number of connections but does not consider the possibility of reducing the number of cubes. There are two possible ways to reduce the number of cubes:

1. Remove an ON-cube if it is equal to a DC-cube.
2. Delete an ON-cube if it is contained by a DC-cube.

The following two examples show the above two cases respectively.

Example III.27: Given is an ON-array of three cubes

0101

0111

1101

and a DC-array of one cube

1111

as shown in Figure 50a. After minimizing the ON-array, we get two ON-cubes $x1x1$ and 1111 (Figure 50b). By comparing the ON-cubes with the DC-cube (Figure 50c), we find out that the ON-cube 1111 is equal to the DC-cube 1111. We remove these two cubes, and get the result $x1x1$ which is one cube less than the minimized ON-array (Figure 50d).

Example III.28: Given is the same ON-array as in Example III.27, and a DC-array of one cube $111x$ as shown in Figure 51a. After minimizing the ON-cubes as shown in Figure 51b, we can find out that the DC-cube $111x$ contains the ON-cube 1111 as shown in Figure 51c. We delete the ON-cube 1111. The result is shown in Figure 51d, which is the same as Example III.27:

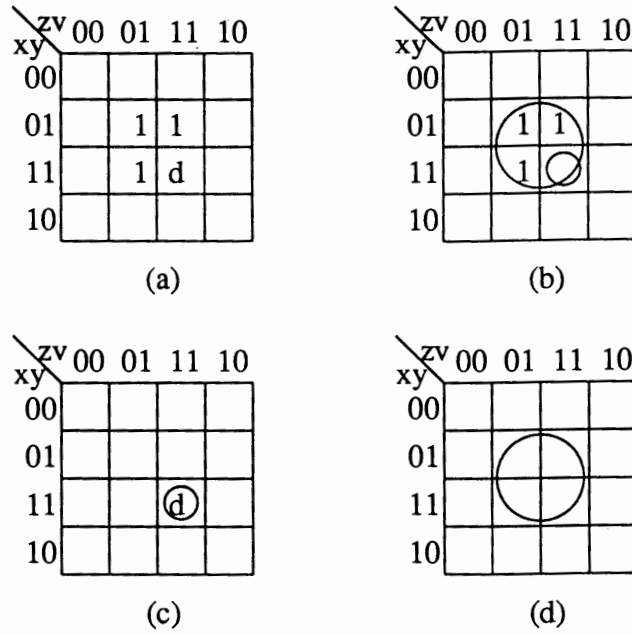


Figure 50. An ON-cube is equal to a DC-cube.

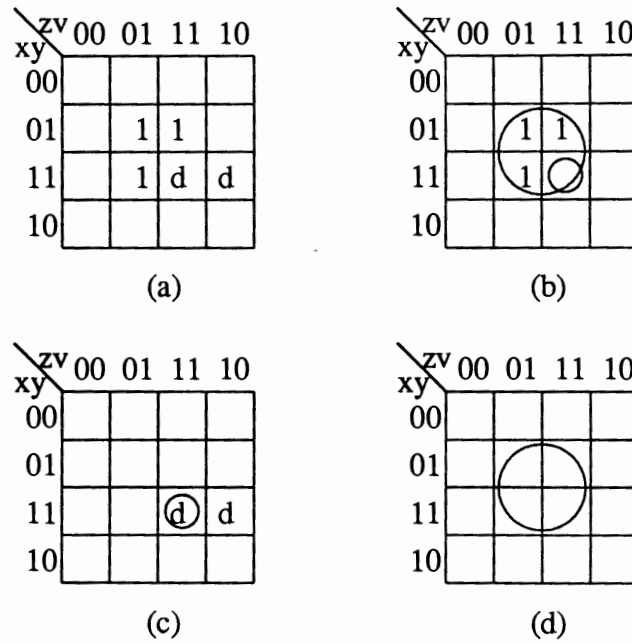


Figure 51. An ON-cube is contained by a DC-cube.

To check whether a DC-cube contains or is equal to an ON-cube, we use the disjoint sharp operation:

$$R \leftarrow \text{ON-cube} \#_d \text{DC-cube.}$$

If the sharp operation returns an empty cube, this means that the DC-cube contains the ON-cube, or is equal to it. The ON-cube can then be removed.

Based on the above discussion, our approach to minimize incompletely specified functions is described by the following procedure:

```
function don't care minimize (ON-array, DC-array)
  for (each ON-cube) {
    for (each DC-cube) {
      R ← ON-cube #d DC-cube
    }
    if (R = φ) remove ON-cube from ON-array
  }
  return ON-array.
```

If no more ON-cubes can be removed by this method, we can perform difference 2 exor-linking on the ON-array in order to reshape the ON-cubes. Example III.29 shows that reshape may help to reduce the ON-cubes.

Example III.29: Given ON-cubes

110x

0x 11

1110

and DC-cubes

0x 10

10x 1

as shown in Figure 52a. The minimization is carried out by the following steps:

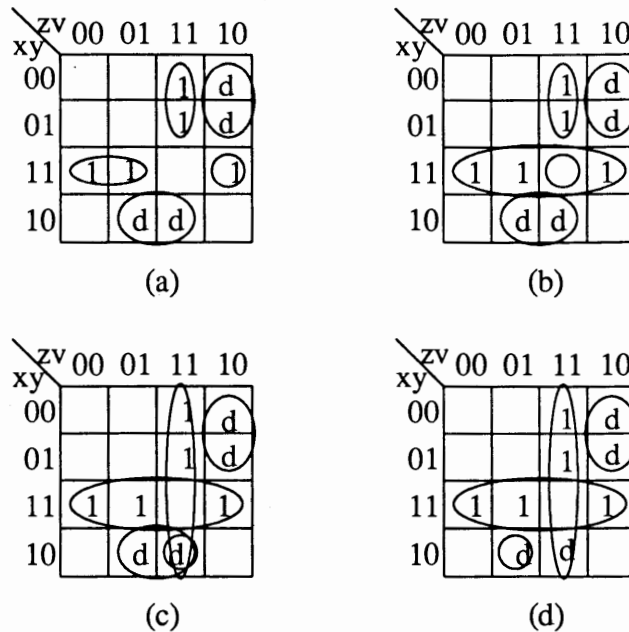


Figure 52. Minimization of an incompletely specified function.

1. Perform the don't care minimization procedure. None of the ON-cubes can be removed.
2. Reshape the ON-cubes as shown in Figure 52b. Again, none of the ON-cubes can be removed.
3. Reshape the ON-cubes as shown in Figure 52c, we get three ON-cubes:

11xx

xx 11

1011.

Since the DC-cube $10x 1$ contains the ON-cube 1011, the operation

$$1011 \#_d 10x 1$$

generates an empty cube. So, the ON-cube 1011 can be removed. The final result is an array of two cubes: 11xx and xx 11.

By performing sharp and difference 2 exorlinking iteratively, the number of ON-

cubes can be reduced, which serves our primary goal: minimizing the number of terms in the ESOPs. After we tried all possible reshapes, the next step is to achieve our secondary goal: minimizing the number of connections. This is done by trying all possible expanding operations of ON-cubes, as shown in Figure 49.

The next section presents our whole algorithm.

V.6. THE ALGORITHM OF EXORCISM-MV-2

We take the disjoint cubes as our starting point. The pairs of equal cubes are removed and difference 1 exorlinking operations are performed iteratively. Then difference 2 exorlinking operations are executed which may provide opportunities for difference 1 exorlinking. If difference 2 exorlinking can not further improve the cost function, difference 3 exorlinking is performed. After certain loops, if no further difference 1 exorlinking operations are possible, difference 2 exorlinking operations are performed to minimize the number of connections.

In the case of a multi-output function, by default, the function is first transformed from a multiple-output array to single-output arrays. We minimize each single-output array, and then minimize the whole function, by using the methods discussed in section V.4.

For incompletely specified functions, the ON-array is minimized first. Then the disjoint sharp operation is executed between each cube in the ON-array and the cubes in the DC-array. If the disjoint sharp operation generates an empty cube, the ON-cube will be removed from the ON-array. If no more cubes in the ON-array can be sharpened out, difference 2 exorlinking operations are performed in order to provide further opportunities. Next, we try to expand each ON-cube into DC-cubes. Successful application of these operations decreases the numbers of connections.

Since our algorithm is a heuristic one, we do not know whether or not we have got the minimum results. So, we need some criteria to stop the program. The following methods can be used as termination criteria:

1. Cost functions. We can use a cost function as a termination criterion. For instance, we can stop the program if the number of terms in the current solution meets a preset number.
2. Number of loops. We can stop the program after a certain number of loops. This is a simple method. But the quality of the results is not guaranteed.
3. Execution time. We can stop the program if time limit has been exceeded. This is the method used in EXORCISM.
4. Improvements of the current solution. By this method, the program is controlled by comparing the current result with the previous result. If no improvement for a certain number of loops, the program goes to the next step. This is the method used in ESPRESSO and EXMIN. In our program this is the method used by default. The user can also select other methods as options.

Algorithm to Minimize MIESOP Expressions for Incompletely Specified Functions.

Input: ON-array and DC-array of disjoint cubes for a multi-valued input function.

1. $F := ON; D := DC.$
2. $SOLUTION := F, MIN_COST := COST(F).$ (MIN_COST will be updated in the steps below to reflect always the lowest cost of solutions obtained until now. This solution is also stored).
3. If the option "do not decompose the function to single output function" is selected, go to step 5; else go to step 4.
4. Decompose the function to a set of single output functions. For each single output function, perform the steps 5 to 8.

5. Perform all possible remove_equal operations.
6. Perform all possible difference 1 exorlinking operations.
7. For each pair of cubes in ESOP, check if a difference 2 exorlink is possible. If it is possible, further check if it makes a remove_equal operation or a difference 1 exorlink operation possible. If it is possible, perform the difference 2 exorlink operation and then perform remove_equal or difference 1 exorlink operation. Otherwise, do nothing.
8. Check the number of cubes. If the number of cubes has not been reduced for certain number of iterations go to 9, else go to 5.
9. If the option "do not decompose the function to single output function" is selected, go to step 11.
If the single functions have been combined to a multiple output function, go to step 11.
If all the single output functions are minimized, go to step 10;
else go to step 5 to minimize the next single output function.
10. Combine the single output functions to a multiple output function.
11. Perform difference 3 exorlink operation iteratively.
12. Check the number of cubes. If the number of cubes has not been reduced for certain number of iterations go to 13, else go to 5.
13. Check all possible difference 2 exorlinking operations. For each difference 2 exorlink operation, if it reduces the cost, perform the difference 2 exorlink operation, otherwise, do nothing.
14. If the option "don't care" is not selected, go to 21, else go to 15.
15. If D is empty (no dc cubes) go to 21, else go to 16.

16. Perform disjoint sharp between each cube in F and all cubes in D.
17. Minimize F again (perform the steps from 5 to 11)
18. If the number of cubes has not been reduced for certain number of iterations, go to 19, else goto 15.
19. Expand each cube in F into D if possible.
20. Check the cost. If the cost has no improvement for certain number of iterations go to 21, else go to 19.
21. Print the output and stop the program.

CHAPTER VI

EVALUATION OF RESULTS OF EXORCISM-MV-2

Exorcism-mv-2 was tested on a set of MCNC benchmarks, as illustrated in Tables XI, XII, XIII. All the benchmarks are run on the Sparc II station. The time is the user time in seconds.

TABLE XI

EXPERIMENTAL RESULTS OF FUNCTIONS WITH 1 BIT AND 2 BIT DECODERS

	1 bit decoder				2 bit decoder			
	cube	AND	EXOR	Time	cube	AND	EXOR	Time
ADR4	31	114	40	4.5	11	46	14	4.7
LOG8	95	508	192	266.8	85	663	163	500.3
MLP4	62	301	86	38.2	51	380	75	89.6
NRM4	67	375	131	51.1	53	429	86	66.8
RDM8	31	110	42	8.5	26	152	38	12.7
ROT8	37	200	62	6.4	26	201	51	9.1
SQR8	114	533	202	264.4	96	692	161	407.7
WGT8	58	261	64	60.5	22	120	29	64.3

Table XI presents, for each function, the total number of cubes, the number of inputs to AND gates, the number of inputs to EXOR gates, and the user time. All these data are presented first for EXOR PLAs with 1-bit and next for 2-bit decoders. Table XII does the same for 3-bit decoders.

While Table XI and XII present the results on arithmetic functions, Table XIII shows the results on different kinds of functions from MCNC benchmarks. Three different cost functions are used for evaluating the results. C_T measures the number of terms in the results. C_L measures the number of literals in the results. C is the cost function we introduced at page 31.

TABLE XII
EXPERIMENTAL RESULTS OF FUNCTIONS WITH 3 BIT DECODERS

	3 bit decoder			
	cube	AND	EXOR	Time
ADR4	10	50	15	5.0
LOG8	41	307	112	544.2
MLP4	29	174	54	94.5
NRM4	26	170	46	73.3
RDM8	19	102	37	14.8
ROT8	16	102	26	11.6
SQR8	66	447	267	495.8
WGT8	10	78	17	65.9

Figure 53 and 54 are scatter plots of number of terms versus execution time and number of variables versus execution time respectively. Both figures show weak correlations between the independent variables and dependent variables. From these two figures we can see that the execution time depends more on the number of terms than on the number of variables. Since our basic algorithm is to perform cube operations iteratively, the execution time needed is mainly determined by the number of loops the program runs and the number of cubes in the array. In each loop, we check the difference for each pair of cubes. There are $(n \times (n-1))/2$ different pair of cubes in an array of n cubes. So, if we go through one loop only, the time complexity is $O(n^2)$. Since we run the program for many loops, we can predict intuitively that the time complexity is higher than $O(n^2)$.

For analysis time complexity, we used multiple regression technique provide by a statistical package -- SPSS. The number of terms in the initial form and the number of variables in the form are taken as independent variables. The execution time is used as dependent variable. The regression generates the following results:

$$Execution\ Time = -3.96 + 1.20 \times VARS + 2.32 \times \left(\frac{TERM}{100}\right)^3 - 0.11 \times e^{\left(\frac{TERM}{100}\right)}$$

where $VARS$ is the number of variables in the form which equals to number of input vari-

TABLE XIII
EXPERIMENTAL RESULTS OF EXORCISM-MV-2

	input	output	INITIAL FORM		MINIMIZED FORM			
	var.	var.	C_T	C_L	C_T	C_L	C	Time
b12	15	9	57	358	28	163	28.46	2.0
bw	5	28	26	369	22	320	22.87	2.1
clip	9	5	163	1340	63	491	63.35	55.8
con1	7	2	10	44	9	37	9.84	0.1
duke2	22	29	103	1170	79	918	79.78	91.7
ex5	8	63	183	2739	78	860	78.31	61.1
inc	7	9	34	229	28	174	28.76	3.5
misex1	8	7	14	106	12	85	106.8	0.6
misex2	25	18	28	217	27	210	27.97	3.5
rd53	5	3	31	183	15	68	15.37	0.3
rd73	7	3	127	999	39	192	39.19	9.8
rd84	8	4	255	2253	58	325	58.14	46.4
sao2	10	4	93	893	28	277	28.31	13.0
squar5	5	8	26	127	19	78	19.61	0.9
table3	14	14	182	2767	166	2506	166.9	173.3
table5	17	15	167	2687	156	2462	156.92	60.4
t481	16	1	980	13705	23	197	23.01	186.2
vg2	25	8	219	2570	184	2000	184.78	115.3
5xpl	7	10	71	437	32	170	32.39	4.8
9sym	9	1	145	1294	51	425	51.33	67.6
xor5	5	1	16	96	5	10	5.625	0.1
Adr4	8	5	155	1215	31	154	31.13	4.5
Log8	8	8	408	3395	95	700	95.21	266.8
Mlp4	8	8	234	1909	62	387	62.20	38.2
Nrm4	8	5	313	2620	67	506	67.19	51.1
Rdm8	8	8	174	1295	31	152	31.12	8.5
Rot8	8	5	280	2355	37	262	37.11	6.4
Sqr8	8	16	505	4146	114	735	114.18	264.4
Wgt8	8	4	314	2757	58	325	58.12	60.5

ables + number of output variables, and *TERM* is the number of terms in the initial form.

The above equation shows that the time complexity is $O(n^3)$ which is polynomial. Since we use the improvements of the current solution as termination criteria, if the current solution has been improved, the program will keep looping, otherwise it will stop. So, the execution time is not a deterministic one. The R square of the regression is 0.72, which gives us a measurement of the randomness in the execution time. Detailed

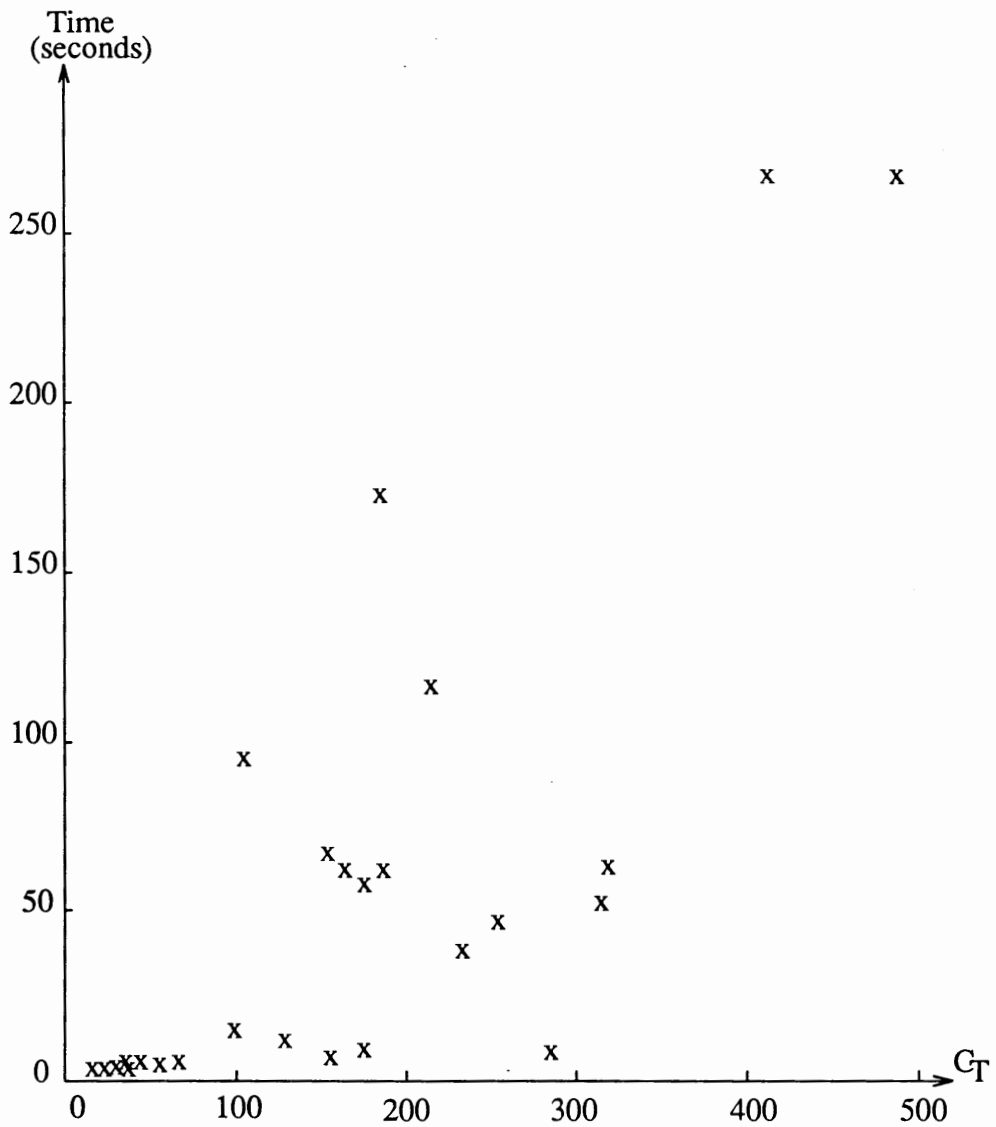


Figure 53. Scatter plot of number of terms versus execution time.

discussion on multiple regression can be found in any standard statistics books, for instance [Mend 87].

Table XIV and XV compare our results with those of EXMIN [Sasa 90a] and EXMIN-2 [Sasa 92] respectively. Since no timing information are provided in [Sasa 90a] and [Sasa 92] for these benchmarks, we can compare only cost functions in these two tables. While Table XIV compares C_T only, Table XV compares all three cost functions.

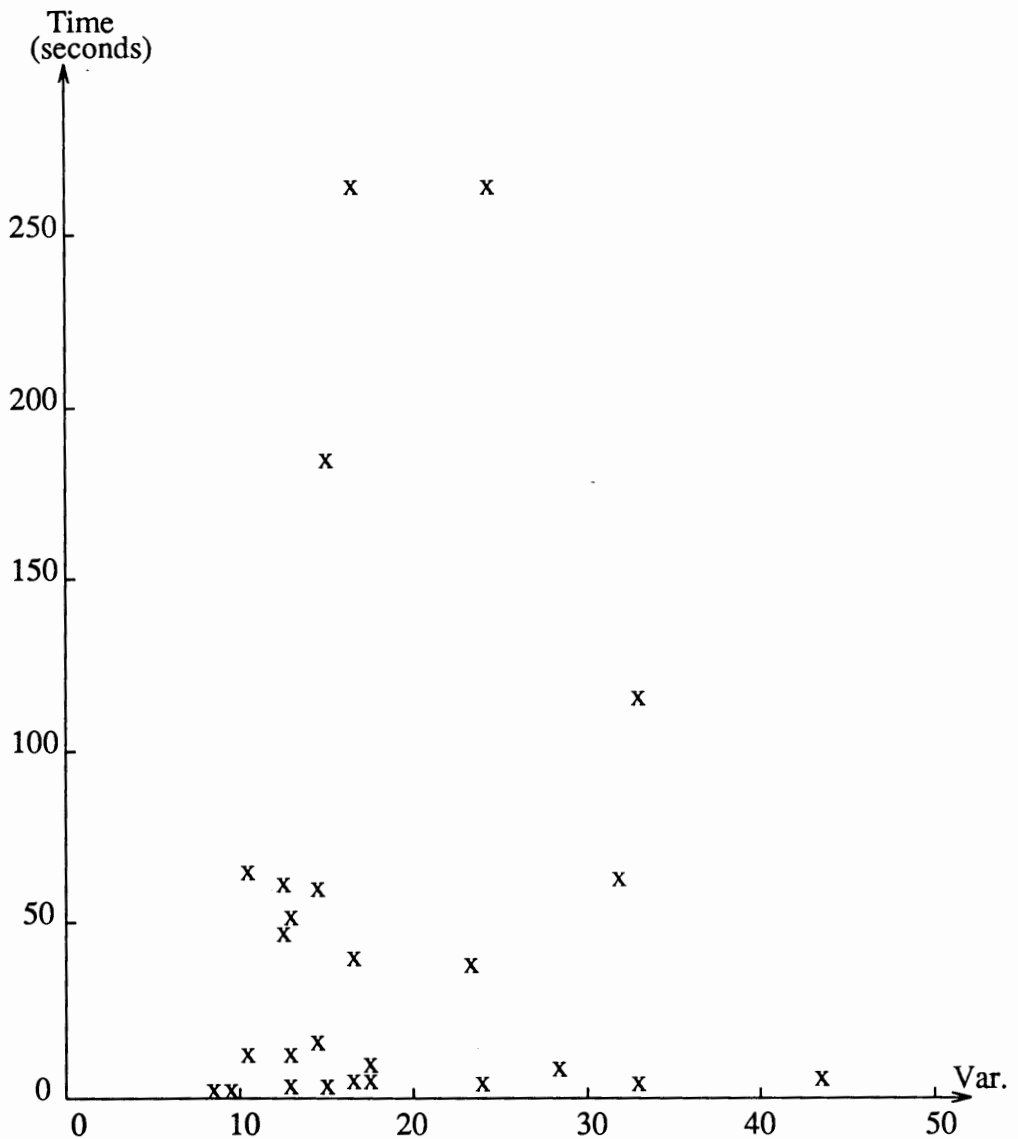


Figure 54. Scatter plot of number of variables versus execution time.

Table XVI and XVII compare other benchmarks with EXMIN-2. Both time and cost comparisons are shown in these two tables. Table XVI compares the execution time needed for achieving approximately the same cost functions (C_T). At the end of each loop, the cost function is checked. The program will stop if the cost of the current solution is less than or equal to the target. We can see that in most cases our program generates the same or better results in shorter time. Table XVII compares the cost functions when the same time limits are given. Again, in most cases our program generates better

TABLE XIV
COMPARISON OF EXORCISM-MV-2 WITH EXMIN

	EXMIN		EXORCISM-MV-2	
	1 bit	2 bit	1 bit	2 bit
ADR4	32	11	31	11
LOG8	105	105	95	85
MLP4	66	56	62	51
NRM4	76	62	67	53
RDM8	31	28	31	26
ROT8	37	32	37	26
SQR8	121	121	114	96
WGT8	66	26	58	22

TABLE XV
COMPARISON OF EXORCISM-MV-2 WITH EXMIN-2

	EXMIN-2			EXORCISM-MV-2			
	C_T	C_L	C	C_T	C_L	C	Time
Adr4	31	162	31.13	31	154	31.13	4.5
Log8	99	690	99.20	95	700	95.21	266.8
Mlp4	63	395	63.21	62	387	62.20	38.2
Nrm4	71	533	71.20	67	506	67.19	51.1
Rdm8	32	161	32.12	31	152	31.12	8.5
Rot8	37	257	37.11	37	262	37.11	6.4
Sqr8	112	747	112.18	114	735	114.18	264.4
Wgt8	59	330	59.12	58	325	58.12	60.5
Adr4*	11	60	11.05	11	60	11.05	4.7
Log8*	92	944	92.28	85	826	85.24	500.3
Mlp4*	50	412	50.22	51	455	51.24	89.6
Nrm4*	52	521	52.20	53	515	53.20	66.8
Rdm8*	26	181	26.14	26	190	26.15	12.7
Rot8*	26	242	26.10	26	252	26.11	9.1
Sqr8*	108	1078	108.26	96	853	96.21	407.7
Wgt8*	25	159	25.06	22	149	22.05	64.3

* with 2 bit decoders

results.

Table XVIII compares numbers of resultant terms for Espresso and Exorcism-mv-2. Table XIX shows the experimental results for the minimization of incompletely

TABLE XVI
COMPARISON OF EXORCISM-MV-2 WITH EXMIN-2
FOR THE SAME COST FUNCTIONS

	EXMIN-2				EXORCISM-MV-2			
	C_T	C_L	C	Time	C_T	C_L	C	Time
b12	28	164	28.46	4	28	163	28.46	3.5
clip	68	517	68.39	55	66	490	66.37	49.2
rd53	15	60	15.33	2	15	60	15.33	1.2
rd73	42	221	42.22	20	40	207	40.21	8
rd84	59	330	59.15	45	59	338	59.15	69.3
sao2	29	308	29.35	8	29	302	29.34	6.6
t481	13	53	13.00	677	23	197	23.01	379.1
vg2	184	1992	184.78	163	184	2000	184.78	241.3
5xp1	34	186	34.43	13	33	190	33.43	1.7
9sym	53	433	53.32	27.7	52	431	52.32	27.4

TABLE XVII
COMPARISON OF EXORCISM-MV-2 WITH EXMIN-2
FOR THE SAME EXECUTION TIME

	EXMIN-2				EXORCISM-MV-2			
	C_T	C_L	C	Time	C_T	C_L	C	Time
b12	28	164	28.46	4	28	163	28.46	3.5
clip	68	517	68.39	55	64	485	64.36	55.2
rd53	15	60	15.33	2	15	60	15.33	1.2
rd73	42	221	42.22	20	39	192	39.19	19.8
rd84	59	330	59.15	45	62	351	62.16	45.5
sao2	29	308	29.35	8	29	294	29.33	7.4
t481	13	53	13.00	677	23	197	23.01	415
vg2	184	1992	184.78	163	184	2000	184.78	242.6
5xp1	34	186	34.43	13	32	171	32.39	12.4
9sym	53	433	53.33	27.7	52	431	52.32	27.4

specified functions. The results of minimizing ON-cubes only are compared with the results of minimizing ON- and DC-cubes. The results show that better results are achieved when DC-cubes are taken into account.

TABLE XVIII
COMPARISON OF OUR RESULTS WITH ESPRESSO

	input	output	ESPRESSO	EXORCISM-MV-2			
	var.	var.		cube	cube	AND	EXOR
b12	15	9	43	28	123	40	2.0
bw	5	28	22	22	81	239	2.1
clip	9	5	120	66	380	111	32.8
con1	7	2	9	9	28	9	0.1
ex5	8	63	74	78	456	404	61.1
inc	7	9	30	28	117	57	3.5
misex1	8	7	12	12	48	37	0.6
misex2	25	18	28	27	172	38	3.5
rd53	5	3	31	15	47	21	0.3
rd73	7	3	127	41	154	51	11.2
rd84	8	4	255	58	259	66	46.4
sao2	10	4	58	28	225	52	13.0
squar5	5	8	25	19	50	28	0.9
table3	14	14	175	166	1848	658	173.3
table5	17	15	158	156	1860	604	228.7
t481	16	1	481	23	174	23	201.0
vg2	25	8	110	184	1807	193	115.3
5xp1	7	10	65	32	117	53	4.8
9sym	9	1	86	51	374	51	67.6
xor5	5	1	16	5	5	5	0.1

TABLE XIX
COMPARISON OF MINIMIZATION OF ON-CUBES
WITH MINIMIZATION OF ON- AND DC-CUBES

	ON-cubes				ON- and DC- cubes			
	cube	AND	EXOR	Time	cube	AND	EXOR	Time
bench	28	149	47	2.5	24	121	54	6.1
exam	183	1467	353	284.9	160	1263	362	1504.1
p1	64	411	235	21.8	62	400	338	98.7
p3	40	262	119	10.6	39	252	143	51.5
test1	166	1056	448	330.6	164	1037	493	476.6

CHAPTER VII

CONCLUSION

A new cube operation and algorithm for MIESOP minimization along with the efficient program to minimize such forms have been introduced. No algorithms have been published so far for MIESOP solutions to multi-output and incompletely specified functions.

One approach to minimize ESOPs is to apply a set of cube operations iteratively on each pair of cubes in the array. Our new operation -- exorlinking is the most general operation in this approach which can link any two cubes in the array in arbitrary distance. All the cube operations introduced previously in the literature in this approach are the special cases of this operation.

Exorcism-mv-2 was tested on many benchmark functions and compared to EXMIN [Sasa 90a] and others. The program in most cases gives the same or better solutions on binary and 4-valued completely specified functions. More importantly, it is able to minimize efficiently arbitrary-valued and incompletely specified functions, while the programs from literature are only for completely specified functions: those from [Saul 90] and [Bess 91] only for binary variables, and the program from [Sasa 90a] for 2-valued and 4-valued variables. Additionally, as in Espresso, the number of variables in our program is unlimited and the only constraint is the number of input cubes that are read, so very large functions can be minimized.

REFERENCES

- [Bess 83] Besslich, Ph.W., "Efficient Computer Method for EXOR Logic Design", *Proc. IEE*, Vol. 130, Part E, CDT, No. 6., pp. 203-206, 1983.
- [Bess 91] Besslich, Ph. W., M.W. Riege, "An Efficient Program for Logic Synthesis of Mod-2 Sum Expressions", *Euro ASIC'91*, pp. 136-141, Paris, France, 1991.
- [Bran 91] Brand, D., and T. Sasaso, "On the minimization of AND-EXOR expressions", *23rd IEEE Conference on Fault Tolerant Computing*, pp. 1-9, 1990.
- [Brow 90] Brown, F. M., "Boolean Reasoning", Kluwer Academic Publishers, 1990.
- [CLi 91] Concurrent Logic, Inc., "CLi 6000 Series Field Programmable Gate Arrays. Preliminary Information", *Dec. 1991, Rev. 1.3*.
- [Csan 92] Csanky, L., Perkowski, M., and I. Schaefer, "Canonical Restricted Mix-Polarity Exclusive Sums of Products and the Efficient Algorithm for Their Minimization", accepted for the publication in *IEE proc. Pt. E., May, 1992*.
- [Flei 83] Fleisher, H., Tavel, M., and J. Yeager, "Exclusive-OR representations of Boolean functions", *IBM J. Res. Develop.*, Vol. 27, pp. 412-416, July 1983.
- [Flei 87] Fleisher, H., Tavel, M., and J. Yeager, "A Computer Algorithm for Minimizing Reed-Muller Canonical Forms", *IEEE Trans. on Computers*, Vol. C-36, No. 2, pp. 247 - 250, February 1987.
- [FPGA 92] Proceedings of FPGA'92, 1992 ACM First International Workshop on Field-Programmable Gate Arrays, Hotel Durant, Berkeley, February 16-18, 1992.
- [Froe 91] Froessl, J., and B. Eschermann, "Module Generation for AND/XOR-Fields (XPLAs)", *Proc. IEEE ICCD-91 Conference*, pp. 26-29, 1991.
- [Fuji 86] Fujiwara, H., "*Logic Testing and Design for Testability*", Computer System Series, The MIT Press, 1986.
- [Gr] Grätzer, G., "*Universal Algebra*", 2nd edition,
- [Gill 91] Gilliam, P., "A Practical Parallel Algorithm for the Minimization of Kroenecker Reed-Muller Expansions", *M.S. Thesis*, PSU EE Dept., 1991.

- [Gree 90] Green, D. H., "Reed-Mueller Canonical Forms with Mixed Polarity and Their Manipulations", *IEE Proceedings*, Vol. 137, Part E, No. 1, pp. 103-113, January 1990.
- [Gree 91] Green, D. H., "Families of Reed-Mueller canonical forms", *International Journal of Electronics*, pp. 259-280, February 1991, No. 2.
- [Hell 88] Helliwell, M., and M.A. Perkowski, "A Fast Algorithm to Minimize Multi-Output Mixed-Polarity Generalized Reed-Muller Forms", *Proc. 25-th ACM/IEEE Design Automation Conference*, pp. 427-432, June 12- June 15, 1988.
- [Lui 92] Lui, P.K., and J.C. Muzio, "Boolean Matrix Transforms for the Minimization of Modulo-2 Canonical Expansions", *IEEE Trans. on Computers*, Vol. 41, No. 3, pp. 342-347, March 1992.
- [Mend 87] Mendenhall, W., "Introduction to Probability and Statistics", 7th edition, 1987, PWS Publishers.
- [Papa 79] Papakonstantinou, G., "Minimization of modulo-2 sum of products", *IEEE Trans. on Computers.*, Vol. C-28, pp. 163-167, February 1979.
- [Perk 89] Perkowski, M., Helliwell, M., and P. Wu, "Minimization of multiple-valued input multi-output mixed-radix exclusive sums of products for incompletely specified boolean functions", *Proc. of the 19th International Symposium on Multiple-valued Logic*, pp. 256-263, May 1989.
- [Perk 90] Perkowski, M., and M. Chrzanowska-Jeske, "An Exact Algorithm to Minimize Mixed-Radix Exclusive Sums of Products for Incompletely Specified Boolean Functions", *Proc. of the ISCAS'90, International Symposium on Circuits and Systems*, pp. 1652-1655, New Orleans, May 1990.
- [Perk 91] Perkowski, M., and P. Johnson, "Canonical Multi-Valued Input Reed-Mueller Trees and Forms", *Proc. of 3rd NASA Symposium on VLSI Design*, pp. 11.3.1 - 11.3.13, University of Idaho, Oct 30-31, 1991.
- [Perk 92] Perkowski, M., "Generalized Orthonormal Expansion and Some of Its Applications", *Proc. Intern. Symp. on Multiple-Valued Logic*, pp. 442 - 450, Sendai, Japan, May 1992.
- [Prad 87] Pradhan, D.K., "*Fault-Tolerant Computing. Theory and Techniques. Vol. I.*" Prentice-Hall, 1987.
- [Robi 82] Robinson, J.P., and C.L. Yeh, "A Method for modulo-2 Minimization", *IEEE Trans. on Computers*, Vol. C-31, pp. 800-801, August 1982.
- [Rude 85] Rudell, R.L., and A.L. Sangiovanni-Vincentelli, "ESPRESSO-MV: algorithms for multiple-valued logic minimization, *Proc. IEEE Custom Integrated Circuits Conf.*, 1985.

- [Salm 89] Salmon, J. V., E.P. Pitty, E.P., and Abramson, M. S., "Syntactic Translation and Logic Synthesis in Gatemap", *IEE Proceedings*, Vol. 136, Part E., No. 4, pp. 321-328, July 1989.
- [Sara 92] Sarabi, A., and M.A. Perkowski, "Fast Exact and Quasi-Minimal Minimization of Highly Testable Fixed-Polarity AND/XOR Canonical Networks", *Proc. 1992 IEEE Design Automation Conference*, pp. 30 - 35, June 1992.
- [Sasa 78] T. Sasao, "An application of multiple-valued logic to a design of Programmable Logic Arrays", *Proc. 8th Intern. Symp. on Multiple-Valued Logic (ISMVL)*, pp. 65 - 72, May 1978.
- [Sasa 81] Sasao, T., "Multiple-valued decomposition of generalized boolean functions and the complexity of programmable logic arrays," *IEEE Trans. Comput.*, Vol. C-30, pp. 635-643, Sept. 1981.
- [Sasa 84] Sasao, T., "Input Variable Assignment and Output Phase Optimization of PLA's", *IEEE Trans. on Comp.*, Vol. C-33, No. 10, pp. 879-894, October 1984.
- [Sasa 86] Sasao, T., "MACDAS: Multi-level AND-OR circuit synthesis using two-variable generators", *Proc. of 23-rd Design Automation Conference*, Las Vegas, pp. 86-93, June 1986.
- [Sasa 90a] Sasao, T., "EXMIN: A simplification algorithm for Exclusive-OR-Sum-of-Products expressions for multiple-valued input two-valued output functions", *Proc. ISMVL-90*, May 1990, pp.128-135.
- [Sasa 90b] Sasao, T., and Besslich, Ph., "On the Complexity of MOD-2 Sum PLAs", *IEEE Transactions on Computers*, Vol. 39., No. 2, pp. 262-266, February 1990.
- [Sasa 91a] Sasao, T., "A transformation of multiple-valued input two-valued output functions and its application to simplification of exclusive-or sum-of-products expressions", *Proc. ISMVL-91, May 1991*, pp. 270-279.
- [Sasa 91b] Sasao, T., "On the complexity of some classes of AND-EXOR expressions", *IEICE Technical Report FTS 91-35, October 1991*.
- [Sasa 92] Sasao, T., private communication between Dr. Sasao and Dr. Perkowski.
- [Saul 90] Saul, J. M., "An Improved Algorithm for the Minimization of Mixed Polarity Reed-Mueller Representations", *Proc. Intern. Conf. on Computer Design: VLSI in Computers and Processors*, pp. 372 - 375, September 17-19, 1990,
- [Saul 91] Saul, J. M., "An Algorithm for the Multi-level Minimization of Reed-Muller Representations", *Proc. Intern. Conf. on Computer Design: VLSI in Computers and Processors*, 1991, pp. 634-637.

- [Scha 91] Schaefer, I., and M. Perkowski, "Multiple-Valued Input Generalized Reed-Muller Forms", *Proc. of ISMVL'91*, pp. 40-48, May 1991.
- [Scha 92] Schaefer, I., and M. Perkowski, "Exact Minimization of Multiple-Valued Input Kronecker Reed-Muller Forms for Incompletely Specified Functions", *Submitted to IEEE Transactions on Computers*, April 1992.
- [Stol 79] stoll, R. R., "Set Theory and Logic", Dover Publications, Inc. New York, 1979.
- [Wu 82] Wu, X., Chen, X., and Hurst, S.L., "Mapping of Reed-Muller coefficients and the minimisation of exclusive OR-switching functions", *IEE proc. E, Comput. & Digital Tech.*, 1982, pp. 15-20.
- [Wu 92] Wu, L-F., Perkowski, M., "Exact and Approximate Minimization of Reed-Muller Trees for Cellular Logic with Application to CLI6000 Field Programmable Gate Arrays", *Proc. Second International Workshop on Field-Programmable Logic and Applications*, Vienna, Austria, Aug. 31 - Sept. 2, 1992.

APPENDIX

MAN PAGE OF EXORCISM-MV-2

NAME

EXORCISM-MV-2 -- EXOR Circuit Speedy Minimizer, for Multiple-valued logic, Version II.

SYNOPSIS

`exorcism [inputfile][options]`.

The *inputfile* is an ascii file containing data of a given logic function. The *inputfile* should be prepared before hand. To minimize the logic function, type

`exorcism inputfile` .

The optimized solution will be shown on the screen. For saving the solution in a file, type

`exorcism inputfile > outputfile`

where *outputfile* is the output file name given by the user.

DESCRIPTION

Exorcism-MV-2 takes an ESOP or a disjoint SOP as input. The function can be binary input or multiple-valued input, single output or multiple output, completely or incompletely specified. If the function is a multiple output function, the algorithm will translate it to a multiple-valued input description. If the function is incompletely specified, the program will separate it to an array of ON-cubes and an array of DC-cubes. The final result will be a minimal or near minimal logically equivalent set of product terms to represent the ON-array and optionally terms which lies in the DC-array.

Comments are allowed within the input by placing a pound sign "#" as the first character on a line. Comments and unrecognized keywords are passed directly from the input file to standard output. Any white space (blanks, tabs, etc), except when used as a delimiter in an embedded command, is ignored.

KEYWORDS

The following keywords can be shown in the *inputfile*. [d] denotes a decimal number and [s] denotes a text string.

.i [d]

Specifies the number of input variables, if the function is binary input and binary output.

.o [d]

Specifies the number of output variables, if the function is binary input and binary output.

.pair [d]

Specifies the number of pairs of variables which will be paired together by using two-bit decoders. The rest of the line contains pairs of numbers which specify the binary variables of the XPLA which will be paired together. The XPLA will be reshaped so that any unpaired binary variables occupy the leftmost part of the array, then the paired multiple-valued columns, and finally any multiple-valued variables.

.mv [d] [d] [d] ... [d]

The first *d* specifies the number of variables for multiple-valued functions. The second *d* shows the number of binary input variables. The rest of the line gives information of each multiple-valued variables.

.p [d]

Specifies the number of product terms.

.e (.end)

Marks the end of the input description.

EXAMPLE 1:

The first example is shown a description of a function with 4 binary input variables and 1 binary output variable. a - indicates the output is don't care corresponding to the input combination.

```
.i 4
.o 1
.p 7
0001 1
0101 1
1100 1
1111 1
1110 -
1001 -
0011 -
.e
```

EXAMPLE 2:

This example shows a description of a multiple-valued function with no binary variable and 3 multiple-valued variables, where the multiple-valued variables have sizes of 4, 4, and 3, and the last multiple-valued variable is the "output".

```
.mv 3 0 4 4 3
.p 3
1101 1101 101
0101 0010 110
0111 0101 111
```

.e

OPTIONS

The command line options can be specified anywhere on the command line and must be separated by spaces.

-help

Provides a quick summary of the available command line options.

-s

provides a short summary of the execution of the program including the initial cost of the function, the final cost, and the computer resources used.

EXAMPLE 3:

Given an input file **test6**, type

```
exorcism test6 -s
```

the output will be:

```
# Portland State University, EXORCISM-MV Version #2, Release date 5/18/92
```

```
# XPLA is test6 with 5 inputs and 1 outputs
```

```
# ON-set cost is c=31(31) in=139 out=31 tot=170
```

```
# OFF-set cost is c=1(1) in=5 out=1 tot=6
```

```
# DC-set cost is c=0(0) in=0 out=0 tot=0
```

```
# 0K bytes active out of 1K bytes allocated
```

```
# EXORCISM Time was 0.1 sec, cost is c=8(8) in=22 out=8 tot=30
```

```
.i 5
```

```
.o 1
```

```
.p 8
```

```
—000 1
```

```
11111 1
```

```
—0— 1
```

01—0 1

1—1 1

001— 1

10-0- 1

-1-1- 1

.e

-t

Provides a trace showing the execution of the program. After each step of the algorithm a single line is printed which reports the processor time used, and the current cost of the function.

-x

Suppresses printing of the solution.

EXAMPLE 4:

If we type

exorcism test6 -x -s

the output will be:

Portland State University, EXORCISM-MV Version #2, Release date 5/18/92

XPLA is test6 with 5 inputs and 1 outputs

ON-set cost is c=31(31) in=139 out=31 tot=170

OFF-set cost is c=1(1) in=5 out=1 tot=6

DC-set cost is c=0(0) in=0 out=0 tot=0

0K bytes active out of 1K bytes allocated

EXORCISM Time was 0.1 sec, cost is c=8(8) in=22 out=8 tot=30

-mo

Select the option "minimize the multiple output directly". The multiple output function will not be decomposed to a set of single output functions

if this option is selected.

-dc

Select the option "don't care". If this option is selected, the program will minimize the ON-array with the DC-array. Otherwise, only the ON-array will be minimized.

-do [s]

This option executes subprogram. [s] denotes a name of subprogram. Some useful subprograms are listed separately below:

-do echo

This echoes "-out fdr" to standard output, and to compute the complement of a function.

-do stats

Provide simple statistics on the size of the function.

EXAMPLE 5:

If we type

exorcism test6 -do stats

we will get the following output:

Portland State University, EXORCISM-MV Version #2, Release date 5/18/92

XPLA is test6 with 5 inputs and 1 outputs

ON-set cost is c=31(31) in=139 out=31 tot=170

OFF-set cost is c=0(0) in=0 out=0 tot=0

DC-set cost is c=0(0) in=0 out=0 tot=0

If we use this option with the option -s, this option will be ignored.