ALGORITHMS FOR THE MINIMIZATION OF BINARY AND MULTIPLE-VALUED LOGIC FUNCTIONS

by

Gerhard W. Dueck

A thesis
presented to the University of Manitoba
in fulfillment of the
thesis requirements for the degree of
Doctor of Philosophy
in
Computer Science

Winnipeg, Manitoba

ISBN 0-315-48034-3

ALGORITHMS FOR THE MINIMIZATION OF BINARY AND MULTIPLE-VALUED LOGIC FUNCTIONS

by

Gerhard W. Dueck

A thesis submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements of the degree of

DOCTOR OF PHILOSOPHY

ABSTRACT

The objective of logic minimization is to find a representation which lends itself to cost effective implementation. In this thesis new algorithms for the minimization of multiple-valued logic functions are presented.

Any binary multiple-output problem can be transformed into a multiple-valued function. Therefore, the binary multiple-output problem can be solved by the techniques described in this thesis. Similarly, any multiple-valued multiple-output can be transformed into a single multiple-valued logic function. The thesis thus covers the complete spectrum of logic minimization.

In some technologies, the truncated SUM operator is easier to implement than the more commonly used MAX operator. Due to the increased complexity associated with the truncated SUM operator, exact minimization is not feasible. A new direct cover algorithm for minimization with the truncated SUM is presented. Two heuristics are used by the proposed algorithm. First, the most isolated minterm is selected to be initially covered. Second, for each implicant which contains the chosen minterm the break count reduction is calculated. The implicant with the best break count reduction is chosen to be part of the solution.

Directed search minimization integrates the choice of the minimal cover into the prime implicant generation process. An extension of the directed search algorithm to accommodate multiple-valued logic function is described. The major drawback of the directed search algorithm is that it starts with a list of minterms. Often a sum-of-products expression, where the product terms are not necessarily minterms, is available.

A new binary recursive consensus algorithm which starts with a sum-of-products expression is presented. The order in which product terms are generated is different from the traditional iterated consensus. In addition, information on the intersection between product terms is kept. These two changes facilitate the early detection of essential and pseudo-essential prime implicants. Moreover, the algorithm is adapted to handle multiple-valued logic functions. The algorithm combines the advantage of starting from a list of terms and detection of essential prime implicants while generating prime implicants.

Finally, it is shown how the algorithms can be adapted to minimization with window literals. Window literals appear more frequently in the literature and are often easier to implement.

ACKNOWLEDGEMENTS

I would like to express my gratitude to Dr. Mike Miller who first introduced me to multiple-valued logic. He has been an invaluable resource person throughout the preparation of this thesis. During the first year of my studies he was my supervisor and later he continued to be my unofficial supervisor after he left the University of Manitoba.

I would like to thank Dr. John Bate who was willing to become my supervisor for the last year of my studies. I would also like to thank the other members of the examining committee, Dr. Stan Hurst, Dr. John van Rees, and Dr. Bob McLeod for their effort.

The work required in the preparation of this thesis could not have been done without the support of my family. The encouragement of my wife Elfriede was instrumental in the speedy completion of this thesis. Thanks to my three boys, Oliver, Lars, and Niels, who gave up playing games on the Macintosh, so that I could type my thesis.

Financial assistance was provided by the Natural Sciences and Engineering Research Council of Canada through a postgraduate scholarship.

to my parents

Concern for man himself and his fate must always form the chief interest of all technical endeavors, concern for the great unsolved problems of the organization of labor and the distribution of goods — in order that the creations of our mind shall be a blessing and not a curse to mankind. Never forget this in the midst of your diagrams and equations.

Albert Einstein

CONTENTS

Chapter 1

INTRODUCTION

1.1 MOTIVATION

The applications of digital systems are evident in our everyday lives, from digital alarm clocks to satellite communications — life without digital systems is difficult to imagine. As the list of possible applications grows daily, design methodologies become more important.

Digital systems accept input signals and produce output signals according to some functional specification. The outputs can be used to control other systems. They are classified under two headings: combinational systems and sequential systems. The outputs of a combinational system are a function of its inputs, *i.e.*, the outputs are at any time uniquely determined by the current inputs. The outputs of a sequential system depend on the current inputs as well as previous inputs. The system then has the capability of remembering previous inputs. Any sequential system can be represented as a combinational system plus a memory.

The top-down design methodology is widely used in the design of software systems. The main thrust of top-down design is to break a large problem into several smaller problems. Successive refinement continues until all remaining tasks are well understood and easily implemented. Comer [COM84] describes how this design methodology can be applied to the design of digital systems. The overall system, described by a set of specifications, is decomposed into modules. These modules should be as independent as possible.

Control modules can be realized as state machines. A state machine is a sequential system which can be described in terms of a set of states that the system may enter. A state machine consists of a memory which remembers the current state and two combinational

systems. The combinational systems are the input forming logic, which also determines the next state, and the output forming logic. The general model of a state machine is shown in Figure 1.1 [COM84]. The design of combinational systems is an integral facet of the design of sequential systems.

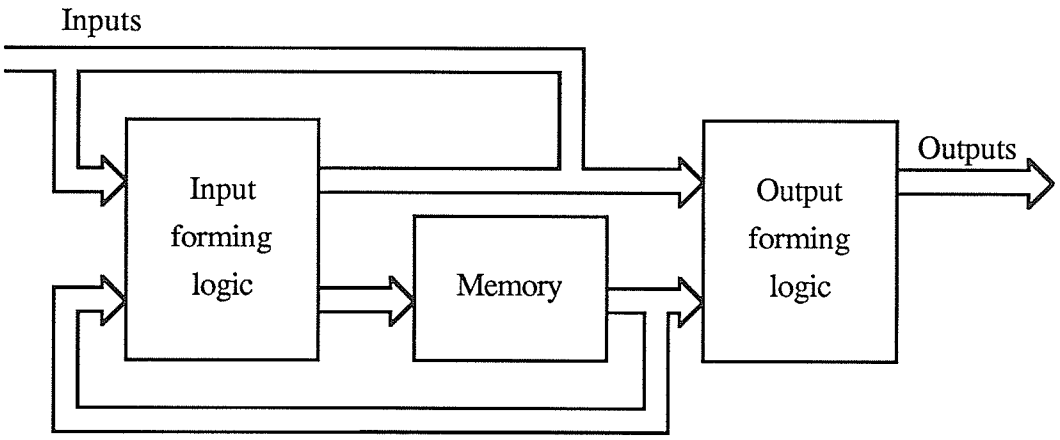


Figure 1.1 General model of a state machine.

The behaviour of a combinational system can be described by a truth table where each input combination is listed with the corresponding outputs. The table for an n-input binary function consists of 2^n entries. For functions with large number of inputs, such a table is not practical. A more concise representation must be found.

Boolean algebra can be used to represent logic functions. The axioms and theorems of Boolean algebra can be used to manipulate logic functions, with the objective of finding a better representation. Applying Boolean algebra in an *ad-hoc* fashion is not effective. Much experience in needed, and it is not clear when a minimal expression has been reached. A systematic approach is required.

The objective of logic minimization is to find a concise representation which lends itself to the most cost-effective implementation. Minimization of logic functions is thus an important step in the design of integrated circuits. Traditionally, cost has been measured in

terms of the number of discrete components, but chip area and speed performance are now the dominant factors. Minimization is a classic problem in logic design. Numerous algorithms have been proposed for the minimization of binary logic function.

It is known that binary minimization belongs to the class on NP-complete problems. Therefore, in general, the exact minimization of logic functions is not feasible. Nevertheless, exact minimization algorithms may work very well for specific functions. A number of heuristic minimizations have been proposed. These algorithms produce minimal or near minimal results.

The ever increasing demand on the implementation of complex systems on a single chip is pushing very-large-scale-integration (VLSI) to its physical limits. About 70 percent of a typical VLSI chip is devoted to the connection among devices. Therefore, any reduction in the interconnection area will result in a significant compaction of the chip area. With the use of multiple logic levels (multiple-valued logic), in contrast to the traditional two levels (binary logic), a substantial reduction in the interconnection area can be achieved.

Moreover, multiple-valued logic (MVL) offers a solution to the pin-out problem since more information can be carried on each pin of the chip. For example, a four-valued signal carries twice the information carried by a binary signal. Also, the MVL realization of a function often produces a more compact circuit.

Currently, the application of MVL is limited to very few industrial situations. Several implementations of multiple-valued memories have been used in industrial chips. A fast, concise multiplier chip using multiple-valued logic has been implemented [KAM88]. Continued research in optoelectronics may produce new device structures which employ multiple-valued logic [HUR86].

The minimization of binary logic functions has been extensively studied and the problem is well understood. Multiple-valued logic offers a rich set of operators, this in turn makes the minimization process more difficult. A subset of the possible operators

must be chosen carefully. The algebra used to represent multiple-valued logic functions must meet two criteria. First, the implementation of the operators in the target technology must be economical. Second, the expressions in the given algebra must be easy to manipulate.

Structured implementations of complex multiple-valued circuits will likely replace random logic. This is evident in the binary domain where programmable logic arrays (PLAs), read only memories (ROMs), and multiplexors (MUXs) are now widely used. The use of regular structures simplifies the design process. The minimization algorithms presented in this thesis are suitable for PLA implementation, *i.e.*, they produce a minimal or near minimal sum-of-products expression.

It has been shown that binary PLA optimization is a special case of multiple-valued logic minimization [SAS78]. Therefore, the algorithms presented in this thesis can also be applied to binary multiple-output problems.

A standard representation of MVL functions has not yet evolved. Therefore, several operators and literals can be used in the representation of a multiple-valued function.

The aim of this thesis is to provide a comprehensive treatment of multiple-valued logic minimization. Different literals and operators will be considered. A number of new algorithms are presented.

The proposed algorithms have the following characteristics:

1) They are suitable for computer implementation. Computer aided design (CAD) is becoming increasingly important in the manufacturing of integrated circuits. A minimization program performs one task of the overall design which is integrated into the CAD software. The reliability is increased by reducing the human factors.
2) They employ simple heuristics. As mentioned before, exact minimization is not feasible. The heuristics are easily understood and simple to program. Some assurance of the quality of the results (*i.e.* the degree of

minimality) of the heuristics is provided.

3) Results are given in sum-of-products form. Minimal sum-of-products expressions are needed in binary PLA optimization. Random logic, which can be used successfully with a small number of inputs, will eventually be replaced by structured implementations (PLA-like structures). At the moment, this seems to be the most promising approach.

4) The algorithms are extendible to accommodate window literals. It is not yet clear which literals are the most suitable for MVL implementations. The possibility of extending an algorithm from one literal operation to another one, kindles the hope, should a new literal operation come into favour, the algorithm can be adapted accordingly.

1.2 ORGANIZATION OF THE THESIS

In Chapter 2, binary and multiple-valued logic minimization algorithms are reviewed. Examples are given to illustrate those algorithms which form the basis for the research presented in Chapters 3 to 6.

In some technologies, the SUM operator is easier to implement than the more common MAX operator. In Chapter 3, a new algorithm for truncated SUM minimization is introduced. The algorithm makes use of break counts to measure the relative complexity of a function.

Directed search minimization [RHY77] integrates the selection of a minimal cover into the prime implicant generation process. In Chapter 4, the binary directed search minimization algorithm is extended to handle multiple-valued functions.

A new consensus algorithm for the minimization of binary logic functions is presented in Chapter 5. Further, it is shown how the algorithm can be extended to accommodate multiple-valued logic functions.

All algorithms presented in Chapters 3 to 5 make use of the generalized literal function. However, in some technologies the window literal is easier to implement. In Chapter 6, the minimization algorithms introduced in Chapters 3 to 5 are extended to handle window literals.

All the algorithms have been implemented (in APL or Pascal). Empirical results concerning the performance of the implementation are presented in the corresponding Chapters.

1.3 NOTATION AND DEFINITIONS

Definition 1.1. Let p_i, i = 1, 2, . . ., n, be positive integers representing the number of values for each of n variables. Define the set $P_i = \{0, 1, . . , p_i - 1\}$, i = 1, 2, . . ., n, to be the p_i values that the i^{th} variable may assume. Define the set B = {0, 1, *} to be the possible values of a binary valued function (* denotes a don't-care condition). A multiple-valued input, single binary-valued output function *f* is a mapping

$$f: P_1 \times P_2 \times \ldots \times P_n \rightarrow B$$

The function *f* has n multiple-valued input variables. The i^{th} variable can take one of p_i possible values.

Definition 1.2. The *radix* of a function is defined to be the maximum value of p_i, i = 1, 2, . . ,n, and is denoted R. A function with radix equal to two is said to be *binary*.

Definition 1.3. Each element in the domain of *f* is a *minterm* of the function.

Definition 1.4. An enumeration of all minterms with the corresponding value of the function is a *truth table*.

Definition 1.5. A *don't care* minterm (represented by * $\in$ B) is one for which the

function value is allowed to be either 0 or 1. Hence, functions may be incompletely specified.

Definition 1.6. The set of all minterms which evaluate to one is called the *ON-set*. Similarly, the *OFF-set* contains all minterms which evaluate to zero. The set of all minterms which are don't-cares is called the *DC-set*.

Definition 1.7. Let x_i be a variable which can take values from the set P_i, and let S_i be a subset of P_i. The literal operation is defined as follows:

$$x_i^{S_i} = \begin{cases} r - 1 & \text{if } x_i \in S_i \\ 0 & \text{if } x_i \notin S_i \end{cases}$$

r is the size of the range of the literal operator. Note for the binary output case r = 2 and the literal operator becomes:

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

For a binary function, x_i^1 will be written as x_i and x_i^0 will be written as $\bar{x}_i$.

Definition 1.8. A *product term* is the Boolean product (AND) of literals.

If a product term evaluates to 1 for a given minterm, then the product term is said to contain the minterm.

Definition 1.9. A product term is an *implicant* of the function *f* if *f* is nonzero for all minterms contained in the product term.

Definition 1.10. A *prime implicant* of the function *f* is an implicant which is not itself

contained in any other implicant of *f*.

Definition 1.11. An *essential prime implicant* is a prime implicant which contains at least one minterm which is not contained in any other prime implicant.

Definition 1.12. Prime implicant PI_1 is said to be *dominated* by prime implicant PI_2 if all the ON-set minterms included in PI_1 are also included in PI_2.

Definition 1.13. A prime implicant PI is termed *pseudo-essential* if it contains a minterm which is not contained in any other prime implicant which is not dominated by PI.

Definition 1.14. A function is said to have a *cycle* if there exists a minterm which is not included in an essential or pseudo-essential prime implicant. A function which contains a cycle is said to be *cyclic*.

A multiple-valued output function can be transformed to a set of binary-valued output functions as shown in Appendix A.

Chapter 2

PREVIOUS WORK

2.1 INTRODUCTION

A logic function maps a combination of input values to one or more output values. This mapping can be represented in a variety of ways. A truth table is the most straightforward representation for a function. However, it is only feasible for functions with a small number of inputs. The truth table for a binary function with n inputs has 2^n rows. For functions with a large number of inputs a more compact representation must be found. The truth table representation is inadequate for a direct implementation, unless the function is to be implemented by a memory or a multiplexor.

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Figure 2.1 Truth Table of a Binary Function.

The function F whose truth table is shown in Figure 2.1 can also be represented as a sum-of-products where each product term covers exactly one minterm:

$$F = \overline{A}\overline{B}C + A\overline{B}C + ABC$$

If one implements the above function without further analysis, two inverters, three 3-input AND gates and one 3-input OR gate are required. By applying the laws of Boolean Algebra to the function F we obtain the following simpler expressions

$$F = \overline{B}C + AC \tag{2.1}$$

or

$$F = C(\overline{B} + A) \tag{2.2}$$

The direct implementation of (2.1) requires two AND gates, one OR gate, and one inverter, whereas (2.2) can be implemented using one AND gate, one OR gate, and one inverter. Both expressions yield a two-level network, but the second expression is in the product-of-sums form. This simple example shows that the implementation cost of a logic function depends on its representation.

The objective of logic minimization is to find a representation which lends itself to the most cost effective implementation of the logic function. In addition, different constraints such as a limit to the number of levels, a restriction to certain types of gates, the number of fan-out lines of a gate, *etc.*, may be imposed on the final expression. Each additional constraint will increase the complexity of the minimization process.

When a function is implemented using discrete gates, the cost of realizing the function is directly related to the number of gates and gate inputs used. A sum-of-products expression can be easily implemented in a two-level AND-OR network. Programmable logic arrays (PLAs) are widely used in VLSI design [FLE75]. A PLA consists of an AND array which is used to realize the product terms of the function and an OR array which combines the product terms. The cost of a PLA is directly related to the number of inputs, outputs, and product terms. Since the number of inputs and outputs for a given function are fixed, the cost is minimized by minimizing the number of product terms. Throughout this thesis, only minimization methods suitable for PLA implementation will be considered.

2.2 BINARY MINIMIZATION

Functions with a small number of inputs can be minimized by applying the laws of Boolean Algebra in some *ad hoc* fashion. Experience is needed even to simplify functions with only four variables. It is not always apparent that a minimal expression has been reached. In general, functions with more than five variables are very difficult to minimize without a systematic procedure.

Quine [QUI52] has proven that a minimal sum-of-products Boolean expression involves only prime implicants. This simplifies the minimization process since not all implicants need be considered.

Karnaugh [KAR53] introduced a pictorial representation of a Boolean function, now known as a Karnaugh map. Karnaugh maps aid the detection of prime implicants, as well as the selection of a minimal sum-of-products expression. This method is suitable for the minimization of logic functions with up to five or six input variables. Some 4-input functions will be used to illustrate the use of Karnaugh maps.

A function can also be expressed by listing the ON-set. For a more concise notation, each minterm in the ON-set is interpreted as the binary representation of an integer. For example, the ON-set of the function shown in Figure 2.1 is {(0,0,1), (1,0,1),(1,1,1)} which can be written as {1,5,7}. This is frequently written as $F(x_1,x_2,x_3) = \Sigma m(1,5,7)$.

Example 2.1. Consider the function $F(x_1,x_2,x_3,x_4) = \Sigma m(1,3,5,6,7,8,9,10,13,14)$. The corresponding Karnaugh map[1] with its prime implicants is shown in Figure 2.2 (a). The minimal sum-of-products expression is shown in Figure 2.2 (b). The prime implicants of F are:

$$\bar{x}_1 x_4,\ \bar{x}_3 x_4,\ \bar{x}_1 x_2 x_3,\ x_2 x_3 \bar{x}_4,\ x_1 x_3 \bar{x}_4,\ x_1 \bar{x}_2 \bar{x}_3,\ x_1 \bar{x}_2 \bar{x}_4$$

The minimal sum-of-products expression is:

$$F(x_1, x_2, x_3, x_4) = \bar{x}_1 x_4 + \bar{x}_3 x_4 + x_2 x_3 \bar{x}_4 + x_1 \bar{x}_2 \bar{x}_4$$

[1] For clarity, the zero values of the function will be left as blanks on the Karnaugh map.

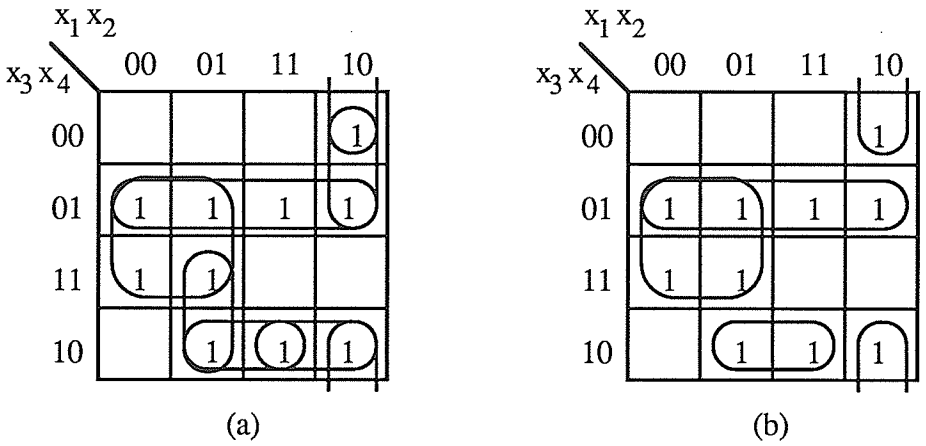


Figure 2.2 Karnaugh Maps for $F(x_1,x_2,x_3,x_4) = \sum m(1,3,5,6,7,8,9,10,13,14)$.
(a) all prime implicants, (b) the minimal cover.

With experience it is possible to find a minimal solution without considering all prime implicants. Unfortunately, the minimal sum-of-products expression of a function is not always unique. This is illustrated by the next example.

Example 2.2. Consider the function $F(x_1,x_2,x_3,x_4) = \sum m(0,1,3,4,5,6,11,14,15)$ (Figure 2.3). The four minimal sum-of-products expressions are:

(i) $F = \bar{x}_1\bar{x}_3 + x_1x_2x_4 + x_2x_3\bar{x}_4 + \bar{x}_1\bar{x}_2x_4$

(ii) $F = \bar{x}_1\bar{x}_3 + x_1x_2x_4 + x_2x_3\bar{x}_4 + \bar{x}_1x_3x_4$

(iii) $F = \bar{x}_1\bar{x}_3 + \bar{x}_1x_3x_4 + x_1x_2x_3 + x_2x_3\bar{x}_4$

(iv) $F = \bar{x}_1\bar{x}_3 + \bar{x}_1x_3x_4 + x_1x_2x_3 + \bar{x}_1x_2\bar{x}_4$

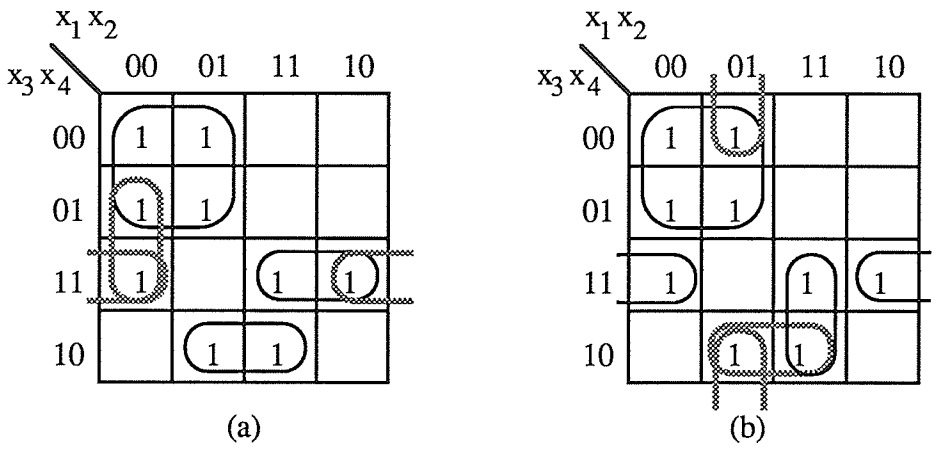


Figure 2.3 A function with four minimal sum-of-products expressions.

(The shaded terms show alternatives.)

2.2.1 Traditional Minimization Techniques

Classical minimization algorithms start with the generation of all prime implicants, followed by the selection of a minimal cover. Quine [QUI52,QUI55] introduced a systematic procedure to minimize a Boolean function. McCluskey [MCC56] refined the method described by Quine. This algorithm is commonly known as the Quine-McCluskey procedure.

In this procedure the generation of all prime implicants is based on the equality

$$Yx_i + Y\bar{x}_i = Y$$

where Y is an implicant of the function which does not involve x_i. This relation is systematically applied to a list of implicants until all prime implicants are generated. Initially, the list of terms consists of all minterms.

To aid in prime implicant generation a positional notation is used for the product terms. Each variable is denoted by 1, the complement of a variable is denoted by 0, and any missing variable is denoted by a dash. Examples of this notation are given below:

algebraic notation	positional equivalent
$\bar{x}_1 x_2 x_3 \bar{x}_4$	0 1 1 0
$\bar{x}_1 \bar{x}_2 x_3 \bar{x}_4$	0 0 1 0
$\bar{x}_1 x_3 x_4$	0 – 1 1

All possible pairs of terms must be compared to determine if a new implicant can be generated from them. Two terms can be combined if they differ in exactly one position which is not a dash. Clearly, the number of ones in both terms must differ by one. By classifying the terms by the number of ones, the number of comparisons can be cut down considerably. Initially, all minterms are grouped according to the number of ones in their positional notation. Groups are separated by a horizontal line. See the example in Table 2.1.

$F = \sum m(1, 2, 3, 4, 5, 9, 11, 13, 15)$

		$x_1 x_2 x_3 x_4$
group 1	1	0 0 0 1
	2	0 0 1 0
	4	0 1 0 0
group2	3	0 0 1 1
	5	0 1 0 1
	9	1 0 0 1
group3	11	1 0 1 1
	13	1 1 0 1
group4	15	1 1 1 1

Table 2.1 Initial ordering of minterms.

A term from any group need only be compared with all minterms from the two adjacent groups. If two terms combine, they are checked (√) to indicate they are covered

by the generated term, and the new term is written in the next column. Note that a checked term is still used to form new terms. The terms from two adjacent groups with k and k+1 ones, respectively, will generate terms with k ones in their positional notation. After all terms have been compared, the procedure is repeated on the new column, until no new terms are generated.

The second column of implicants in Table 2.2 was generated by combining minterms from the first column. Group one in the second column was obtained by combining terms from groups one and two from the first column. After all combinations of terms from groups one and two from the first column have been tried, a horizontal line is placed below the last term in column two, since all terms with a single one in their positional notation have been produced. The same procedure is now performed with the terms from groups two and three. Table 2.3 shows column three, which was obtained in a similar manner. Since no pair of terms in column three may be combined, the procedure stops.

$$F = \sum m(1, 2, 3, 4, 5, 9, 11, 13, 15)$$

	I		II		
		$x_1x_2x_3x_4$		$x_1x_2x_3x_4$	
	1	0 0 0 1 √	1,3	0 0 – 1	
group 1	2	0 0 1 0 √	1,5	0 – 0 1	
	4	0 1 0 0 √	1,9	– 0 0 1	group 1
	3	0 0 1 1 √	2,3	0 0 1 –	
group 2	5	0 1 0 1 √	4,5	0 1 0 –	
	9	1 0 0 1 √	3,11	– 0 1 1	
group 3	11	1 0 1 1 √	5,13	– 1 0 1	
	13	1 1 0 1 √	9,11	1 0 – 1	group 2
group 4	15	1 1 1 1 √	9,13	1 – 0 1	
			11,15	1 – 1 1	group 3
			13,15	1 1 – 1	

Table 2.2 List of implicants generated after pass 1.

All terms that have not been checked are prime implicants. This is easy to see, since they do not combine with any term of the same size to form a bigger term.

It is important to note that terms are not uniquely generated. For example, the prime implicant (1,3,9,11) in column III of Table 2.3 is obtained in 2 ways, by combining (1,3) and (9,11) and by combining (1,9) and (3,11). All four terms in column II must be checked. In general, a term in the i^{th} column covers 2^{i-1} minterms and is generated i - 1 times. This problem cannot be avoided due to the exhaustive nature of the procedure.

$$F = \sum m(1, 2, 3, 4, 5, 9, 11, 13, 15)$$

I		II		III	
	$x_1x_2x_3x_4$		$x_1x_2x_3x_4$		$x_1x_2x_3x_4$
1	0 0 0 1 √	1,3	0 0 - 1 √	1,3,9,11	- 0 - 1
2	0 0 1 0 √	1,5	0 - 0 1 √	1,5,9,13	- - 0 1
4	0 1 0 0 √	1,9	- 0 0 1 √	9,11,13,15	1 - - 1
3	0 0 1 1 √	2,3	0 0 1 -		
5	0 1 0 1 √	4,5	0 1 0 -		
9	1 0 0 1 √	3,11	- 0 1 1 √		
11	1 0 1 1 √	5,13	- 1 0 1 √		
13	1 1 0 1 √	9,11	1 0 - 1 √		
15	1 1 1 1 √	9,13	1 - 0 1 √		
		11,15	1 - 1 1 √		
		13,15	1 1 - 1 √		

Table 2.3 Generation of all implicants of F.

The Quine-McCluskey procedure is guaranteed to generate all prime implicants of the function. The second step in the Quine-McCluskey minimization procedure is to find a minimal cover from the generated list of prime implicants. The minimal sum-of-products expression consists of the fewest prime implicants which cover all the minterms originally

specified. To facilitate the choice of the minimal cover a *prime implicant table* is created (e.g. Table 2.4). Each row of the table corresponds to a prime implicant and each column to a minterm. A single × in any column identifies an essential prime implicant. Essential prime implicants are marked with a star in the table. Since all essential prime implicants must be part of the minimal cover, they are added to the solution and removed from the prime implicant table. All minterms which are covered by the essential prime implicants are also removed from the table, since they need not be covered again (Table 2.5). The table is now reduced to those minterms which have not yet been covered. In the example shown in Table 2.5, only minterm 1 needs to be covered. Either term (1,3,9,11) or (1,5,9,11) can be used to complete the cover.

	1	2	3	4	5	9	11	13	15	
2,3		⊗	x							*
4,5				⊗	x					*
1,3,9,11	x		x			x	x			
1,5,9,11	x				x	x	x			
9,11,13,15						x	x	⊗	⊗	*

Table 2.4 Prime implicant table for F = ∑m(1,2,3,4,5,9,11,13,15).

	1	2	3	4	5	9	11	13	15	
2,3		⊗	x							*
4,5				⊗	x					*
1,3,9,11	x		x			x	x			
1,5,9,11	x				x	x	x			
9,11,13,15						x	x	⊗	⊗	*

Minimal sum F = ∑(2,3),(4,5),(9,11,13,15),(1,3,9,11) or

$$F = \bar{x}_1 \bar{x}_2 x_3 + \bar{x}_1 x_2 \bar{x}_3 + x_1 x_4 + \bar{x}_2 x_4$$

Table 2.5 Prime implicant table after essential prime implicants are removed.

The selection of a minimal cover is not always as straightforward as in the example above. A logic function may have no essential prime implicants. This makes the selection of a minimal cover more difficult, since it is not obvious which prime implicants are part of a minimal solution. This problem can be solved by two means: algebraic or heuristic. With the use of the heuristic approach, a minimal cover is no longer guaranteed. The algebraic solution, on the other hand, will often require a great deal of computation, but it guarantees a minimal cover. McCluskey evaluates both approaches in [MCC56].

Example 2.3. Consider the function $F(x_1,x_2,x_3) = \sum m(1,2,3,4,5,6)$. The prime implicant generation is shown in Table 2.6. The prime implicant table (Table 2.7) has more than one cross in each column. Such a prime implicant table is said to be cyclic (Definition 1.13). The cycle can be broken by selecting one prime implicant to be part of the solution (Table 2.8). Prime implicant (2,6) dominates (2,3) and prime implicant (4,5) dominates (1,5). After removing the dominated prime implicants (Table 2.9) only three prime implicants remain — two of them are essential. Finally, the solution of the given function is

$$F = \bar{x}_1 x_3 + x_1 \bar{x}_2 + x_2 \bar{x}_3$$

$$F = \sum m(1, 2, 3, 4, 5, 6)$$

	I		II	
	$x_1x_2x_3$			$x_1x_2x_3$
1	0 0 1 √		1,3	0 – 1
2	0 1 0 √		1,5	– 0 1
4	1 0 0 √		2,3	0 1 –
3	0 1 1 √		2,6	– 1 0
5	1 0 1 √		4,5	1 0 –
6	1 1 0 √		4,6	1 – 0

Table 2.6 Prime implicant generation for the function used in Example 2.3.

	1	2	3	4	5	6
1,3	x		x			
1,5	x				x	
2,3		x	x			
2,6		x				x
4,5				x	x	
4,6				x		x

Table 2.7 Prime implicant table for $F(x_1,x_2,x_3) = \sum m(1,2,3,4,5,6)$.

	1	2	3	4	5	6
1,3	x		x			
1,5	x				x	
2,3		x	x			
2,6		x				x
4,5				x	x	
4,6				x		x

Table 2.8 Prime implicant table after selecting the product term 1,3.

	2	4	5	6	
2,6	⊗			x	*
4,5		x	⊗		*
4,6		x		x	

Table 2.9 Prime implicant table after removing dominated prime implicants.

The Quine-McCluskey procedure can easily be automated. The generation of prime implicants can be efficiently implemented. By selecting the correct representation, the combining of terms requires only a few simple operations. Nevertheless, there are some drawbacks.

First, the starting point must be a list of minterms. This is particularly inconvenient if the function to be minimized is given as a sum-of-products expression where some terms are actually prime implicants. The procedure must expand the terms into minterms, and then build or perhaps rebuild the prime implicants. Second, if there are don't-care conditions in the original specification, these must be treated as true minterms during the prime implicant generation (see Example 2.4). This may cause some don't-care prime implicants to be generated, *i.e.* all minterms covered by a prime implicant may be don't-care minterms. The don't-care minterms will not appear in the prime implicant table, and this may result in some empty rows.

Example 2.4. Consider the function $F(x_1,x_2,x_3,x_4)$ with ON-set = {5,6,7} and DC-set = {1,3,10,13,14}. The minterms in the DC-set must be considered for prime implicant generation (Table 2.10). Prime implicant (10,14) only covers don't care minterms, and this results in an empty row in the prime implicant table (Table 2.11 (a)). After removing all dominated prime implicants only two essential prime implicants remain. The minimal sum-of-products is

$$F = \bar{x}_1 x_4 + \bar{x}_1 x_2 x_3$$

I		II		III	
	$x_1x_2x_3x_4$		$x_1x_2x_3x_4$		$x_1x_2x_3x_4$
1	0 0 0 1 √	1,3	0 0 – 1 √	1,3,5,7	0 – – 1
3	0 0 1 1 √	1,5	0 – 0 1 √		
5	0 1 0 1 √	3,7	0 – 1 1 √		
6	0 1 1 0 √	5,7	0 1 – 1 √		
10	1 0 1 0 √	5,13	– 1 0 1		
7	0 1 1 1 √	6,7	0 1 1 –		
13	1 1 0 1 √	6,14	– 1 1 0		
14	1 1 1 0 √	10,14	1 – 1 0		

Table 2.10 Prime implicant generation for Example 2.4.

	5	6	7
1,3,5,7	x		x
5,13	x		
6,7		x	x
6,14		x	
10,14			

(a)

	5	6	7	
1,3,5,7	⊗		x	*
6,7		⊗	x	*

(b)

Table 2.11 Prime implicant tables for Example 2.4.

(a) all prime implicants, (b) without dominated prime implicants.

Quine [QUI55] developed a second method to obtain all prime implicants of a function. This method is known as *iterative consensus*. Several authors [MOT60] [TIS67] improved this method. A major difference between the iterative consensus and the Quine-McCluskey procedure is the starting point. The iterative consensus starts with a list

of terms which cover the function. The terms in the sum-of-products expression are not necessarily minterms or prime implicants.

The method is an iterative application of the consensus of two terms:

$$Px_i + Q\bar{x}_i = Px_i + Q\bar{x}_i + PQ$$

where P and Q are product terms which do not involve the literal x_i. PQ is the consensus of the two other terms.

Definition 2.1 The consensus of two terms is said to be *empty* if they differ by more than one literal.

Definition 2.2 The consensus of two terms is said to be *degenerate* if it is covered by one of the terms.

Algorithm: *Iterated consensus to generate all prime implicants of the function F.*

1) remove any term which is covered by another term;
2) find the first pair of terms which produces a nondegenerate nonempty consensus Q;
3) remove all terms in the list which are covered by Q;
4) if Q is not covered by any term in the list, add it to the end of the list;
5) find the next pair of terms whose consensus is nondegenerate and nonempty, and go back to step 3; if no such pair can be found the procedure terminates.

Once the set of prime implicants has been found, a minimal cover must be selected. The selection process is identical to the one used in the Quine-McCluskey procedure.

Generating the list of all prime implicants by iterated consensus offers significant advantage, since not all implicants of the function are necessarily generated. In fact, if all

of the prime implicants are initially given, only a single pass is needed to recognize this. If the iterated consensus is applied to a list of minterms, it proceeds as the Quine-McCluskey method. In this case, it may actually be slower, since the terms are not ordered and pairs which obviously do not combine must be checked.

More powerful algorithms for the generation of prime implicants have been developed. Morreale [MOR70] has presented recursive operators for prime implicant generation. Brayton *et al.* [BRA82] describe recursive paradigms for manipulating Boolean functions. With the use of heuristics they developed an efficient algorithm for the generation of prime implicants. These algorithms are used in the minimization procedures of Espresso and McBoole (see Section 2.2.3 and 2.2.4).

However, there exist functions which have a large number of prime implicants. Dunham and Fridshal [DUN59] have shown that the number of prime implicants of a Boolean function can be prohibitive. Igarashi [IGA79] presented an improved lower bound on the maximum number of prime implicants. For example, there exists a function with 10 input variables with 792 minterms in the ON-set and 4,200 prime implicants. The corresponding prime implicant table has 4,200 rows and 792 columns. Storage and manipulation of such a table becomes a serious problem, even on powerful computers. Functions with 20 input variables can have as many as 133,334,440 prime implicants [IGA79].

Essential prime implicants are readily detected in the prime implicant table. Unfortunately, some functions have no essential prime implicants. In fact, functions with a large number of prime implicants tend to have very few essential prime implicants. The functions mentioned above [DUN56] [IGA79] have no essential prime implicants. This makes the manipulation of the prime implicant table even more difficult, particularly if a minimal cover is to be found, as opposed to a near-minimal solution.

2.2.2 Directed Search Minimization

In 1977, Rhyne, Noe, McKinney, and Pooch [RHY77] proposed the directed search algorithm (DSA) for the minimization of single-output binary functions. The cover selection is integrated into the prime implicant generation process. Ideally, only those prime implicants should be generated which are part of the final cover. During the prime implicant generating process, the DSA recognizes essential and pseudo-essential prime implicants. Not all prime implicants are necessarily generated and, typically, the actual number is a small fraction of the total number of prime implicants. The DSA is able to detect cycles but, unfortunately, it does not resolve them. Standard cycle resolution methods or heuristics must be used.

Definition 2.3. Two minterms are said to be *adjacent* if they differ in exactly one literal.

For convenience minterms are represented by a binary number with x_1 being the most significant bit. The binary representations of two adjacent minterms differ in exactly one bit. Two minterms m_i and m_j are adjacent if the following two conditions are satisfied:

1) $ABS(m_i - m_j) = 2^k$ for some integer k;

2) $AND(m_i, m_j) = MIN(m_i, m_j)$.

where AND is the bitwise Boolean and function, ABS is the absolute value function, and MIN is the minimum function.

Definition 2.4. The set of *directions of adjacency* from the minterm m_i is defined as the signed integers $\{\pm 2^k \mid k = 0, 1, 2 \ldots n-1\}$ which, when added to the decimal value of m_i give all adjacent minterms of m_i.

Definition 2.5. A *required adjacency direction,* RAD, is an adjacency direction which leads a minterm in the ON-set to either a minterm in the ON-set or the DC-set.

Example 2.5. Consider the following 4 variable function with ON-set = {2,5,6,7,9,13} and DC-set = {0,8,11,14,15}. The list of RADs is:

minterm	RADs
2	-2, +4
5	+2, +8
6	+1, -4, +8
7	-1, -2, +8
9	-1, +2, +4
13	+2, -4, -8

RADs are used as a starting point for the generation of prime implicants. The strategy of the DSA is to select a minterm and expand it into larger implicants using its corresponding RADs. Consider the expansion of minterm 7, shown in Figure 2.4. Unsuccessful expansions, *i.e.* those which include a false minterm, are depicted with a dashed arrow. All other leaf nodes contain prime implicants. In the above example minterm 7 is covered by 2 prime implicants, namely (6,7,14,15) and (5,7,13,15).

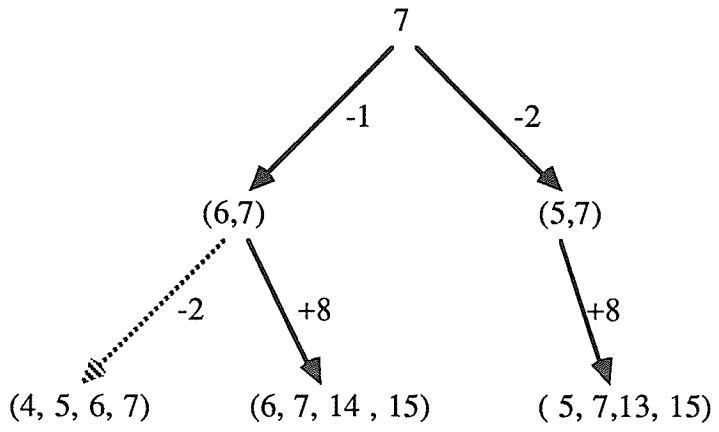


Figure 2.4 A pruned RAD tree.

The tree is created using a depth-first search. The tree can be pruned by ignoring

every path whose RADs are a subset of the RADs on a path which has led to a prime implicant. The RAD +8 is ignored at the root level, since +8 is part of a path which has led to a prime implicant already.

A detailed description of the directed search algorithm follows.

Algorithm: *Directed search minimization of a single-output binary function.*

STEP 1 Compute the RADs for all minterms in the ON-set.

STEP 2 Select the minterm with the fewest number of RADs from the ON-set and construct a pruned RAD tree. Remove any dominated prime implicants. The minterm with the fewest RADs is selected for two reasons:

1) the size of the RAD tree is kept small;
2) there is a higher probability of detecting an essential or pseudo-essential prime implicant.

The algorithm terminates if there are no more minterms in the ON-set.

STEP 3 If the current RAD tree contains an essential or pseudo-essential prime implicant (*i.e.* one that is the only cover of an expanded minterm), add it to the solution; remove all minterms covered by the added term from the ON-set, add them to the DC-set and remove any dominated prime implicant.

Iterate step 3 until either the current RAD tree has no more prime implicants, in which case go to step 2, or no more essential prime implicants can be found, in which case go to step 4.

STEP 4 Select the minterm with the fewest RADs which meets the following criteria:

- it is in the ON-set;
- it has not been expanded;
- it is covered by some prime implicant in the current RAD tree.

Construct the pruned RAD tree for the selected minterm and go to step 3. If no

such minterm exists go to step 5.

STEP 5 The current RAD tree contains a cycle. The cycle problem can be solved by traditional techniques.

Example 2.6. Consider the four-variable function with ON-set = {1,4,5,9,10,11,12}, DC-set = {7,8,14,15}, and OFF-set = {0,2,3,6,13}. The list of RADs is as follows:

Minterm	1	4	5	9	10	11	12
RADs	+4	+1	-1	-1	+1	-1	+2
	+8	+8	+2	+2	-2	-2	-4
			-4	-8	+4	+4	-8

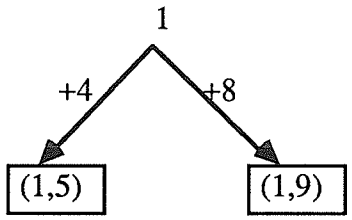


Figure 2.5 Pruned RAD tree of minterm 1.

Minterms 1 and 4 have the lowest number of RADs, and so one of them must be expanded first. Minterm 1 is chosen arbitrarily and its expansion is shown in Figure 2.5. Two prime implicants cover minterm 1 (prime implicants will be identified by enclosing them in rectangles). According to the criteria in step 4 minterms 5 and 9 are candidates for the next expansion. The expansion of minterm 9 is shown in Figure 2.6. The RAD -8 is not used in the expansion of 9 since it would regenerate the prime implicant (1,9). Don't-care minterms are underlined. The expansion of minterm 11 (Figure 2.7) yields two prime implicants (8,9,10,11) and (10,11,14,15). The prime implicant (10,11,14,15) is dominated by (8,9,10,11), therefore (8,9,10,11) is pseudo-essential (indicated by a light arrow ⟿). S_1 is added to the solution and all minterms covered by S_1 become

don't-cares. This in turn makes the prime implicant (1,5) pseudo-essential, since it now dominates the prime implicant (1,9) (Figure 2.8).

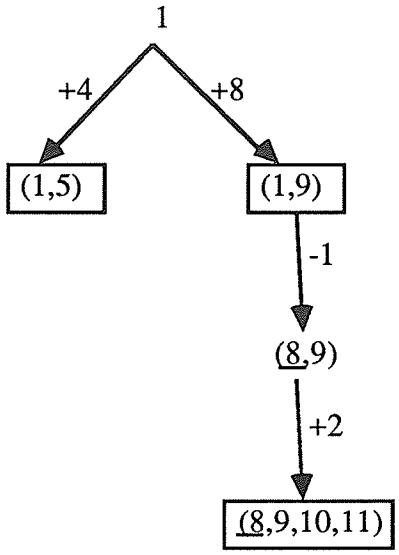


Figure 2.6 RAD trees for minterms 1 and 9.

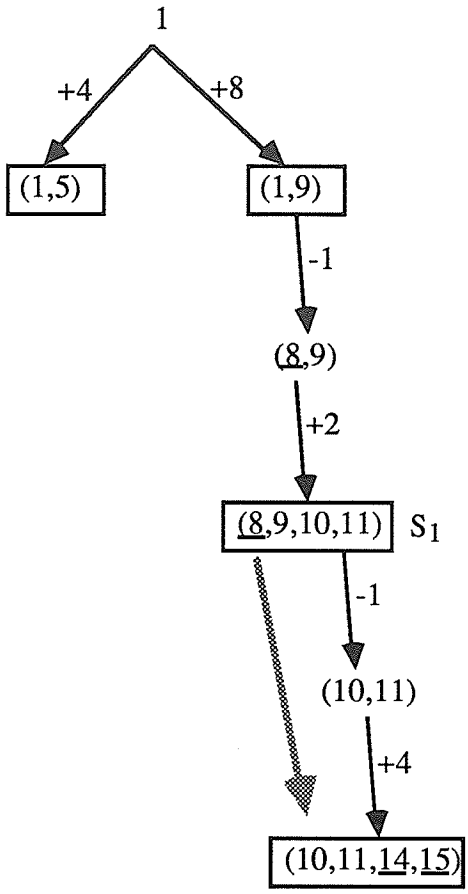


Figure 2.7 RAD trees for minterms 1, 9, and 11.

Finally, the expansion of minterm 4 (Figure 2.9) yields the pseudo-essential prime implicant (4,12). All minterms are now covered. $F = \Sigma(8,9,10,11)(1,5)(4,12)$. As mentioned earlier not all prime implicants are necessarily generated by the DSA. In this example, the following prime implicants were not generated: (5,7), (8,10,12,14), and (7,15).

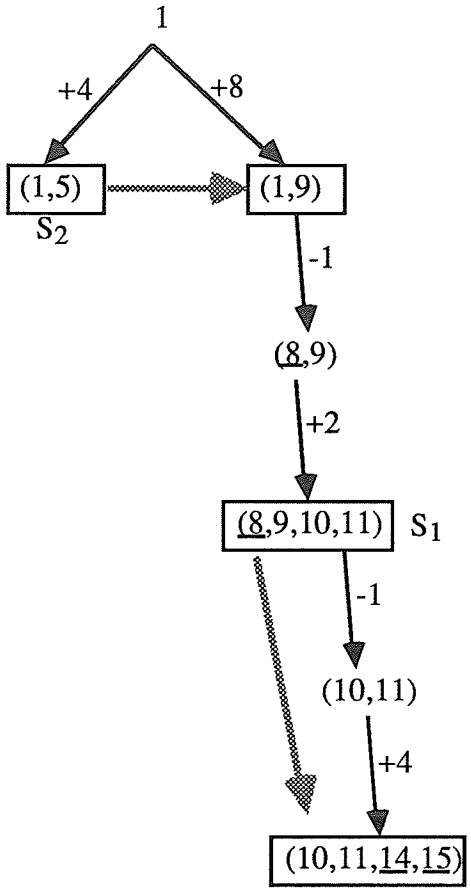


Figure 2.8 RAD trees with two pseudo-essential prime implicants.

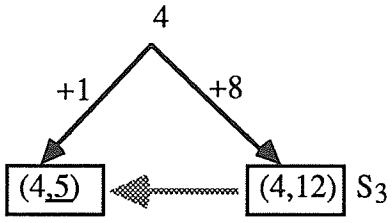


Figure 2.9 RAD tree for minterm 4.

The directed search algorithm can be applied manually or it can be programmed. The manual procedure is suitable for the minimization of functions with 5 to 8 input variables [RHY77]. The strength of DSA lies in its ability to detect essential and pseudo-essential prime implicants. Don't-cares are well integrated into the minimization process, and there

is no need to move them into the ON-set as in the Quine-McCluskey or iterated consensus procedures.

2.2.3 Heuristic Minimization: MINI and ESPRESSO

Hong, Cain, and Ostapko [HON74] presented a heuristic logic minimization technique (MINI) which addresses some of the shortcomings of traditional minimization algorithms. The cost of a function is taken to be the number of product terms, regardless of their size. Thus it is possible to reduce the size of a product term without increasing the cost of the function. The function is specified as a list of implicants. Don't-care implicants can also be specified.

The goal of MINI is to merge implicants towards a minimal cover. Since heuristics are used during the minimization procedure, a minimal cover cannot be guaranteed. A brief description of the four steps in the minimization procedure follows:

1) *Transform.* Find a cover for the function which consists of mutually disjoint implicants. This step will often increase the number of implicants. At the same time, the size of some implicants will become smaller. A list of small implicants will have more possibilities of merging two implicants into a single one. MINI does not go as far as to break the implicants into minterms.
2) *Merge*[2]. If two implicants can be covered by a single implicant, then they are replaced by that implicant. This process continues until no more implicants can be merged.
3) *Reduce.* Reduce the size of each implicant to the smallest possible one. The trimming of implicants facilitates further merging. All redundant implicants are also removed at this point.
4) *Reshape.* Find all pairs of implicants that can be transformed into another pair of

[2] This step was called *expand* in the original work, but *merge* is a more appropriate name for this process.

implicants covering the same minterms. The implicants are now ready for another merging process.

The last three steps are iterated until there is no further decrease in the solution size.

The MINI minimization technique works very well for "shallow" functions, *i.e.* a function where the minimal cover consists of a relatively small number of prime implicants. The execution time of MINI depends mainly on the number of implicants in the final solution. MINI has been tested with a variety of functions and it produced minimal or near minimal results in all cases. Problems with multiple outputs are handled as well.

ESPRESSO-II [BRA84] basically follows the philosophy of MINI. The operations: transform, merge, and reduce are iterated until no further reduction of the solution size is achieved. In addition, all essential prime implicants are extracted after the initial merging (this step is not done in MINI). The algorithms for merging and reduction are quite different from the algorithms used by MINI. Finally, the procedure LASTGASP ensures that no single prime implicant can be added such that two prime implicants become redundant. All ESPRESSO versions make use of fast recursive Boolean function manipulation.

ESPRESSO-MV [RUD87] is an extension of ESPRESSO-II which allows the minimization of multiple-valued functions. The ESPRESSO-EXACT [RUD87] algorithm allows exact minimization of multiple-valued functions.

2.2.4 McBoole

McBOOLE [DAG86] is a procedure for exact logic minimization of multiple-output functions. Its authors claim the procedure is suitable for minimization of functions with up to 20 input variables and 20 outputs. Surprisingly, all prime implicants are generated. The implicants which represent the function are recursively partitioned along the input variables, until the subfunction can be represented by a single term. The terms from both

subfunctions are merged. The procedure is roughly the same as presented in [BRA84] with some improvement which allows it to avoid some unnecessary trials.

During the prime implicant generation all prime implicants are linked into a directed graph. This way information on how a particular prime implicant was obtained is remembered and essential prime implicants are marked. Because of the information retained in the covering graph, the covering problem can be solved locally. Cycles in the function are resolved by branching.

The amount of CPU time used depends mainly on the number of prime implicants and the number of nested cycles in the function. On average, the execution time of McBOOLE is similar to ESPRESSO-II. However, for certain functions McBoole is superior to ESPRESSO-II and vice versa.

2.2.5 Multiple-Output Minimization

In the design of digital system, it is often necessary to minimize several functions which share the same input variables. The minimization of the individual functions will not result in an overall minimal solution. If a term is part of more than one function, its cost appears only once in the overall cost calculation. All of the minimization procedures reviewed in the previous sections have been extended to handle multiple-output problems.

The Quine-McCluskey procedure can be extended to solve the multiple output problem as follows [BAR61]:

1) Each minterm is labeled with symbols which indicate its association with one or more functions.
2) Terms can only be combined if their labels intersect, *i.e.* they belong to at least one common output. The resulting term is labeled with the symbols which appear with both terms.
3) A term is checked off only if all of its output symbols are part of the resulting

term.

Example 2.7. Consider the two functions $F_a(x_1,x_2,x_3,x_4) = \sum m(4,8,9,12,13,14,15)$ and $F_b(x_1,x_2,x_3,x_4) = \sum m(4,6,14,15)$. The prime implicant generation is shown in Table 2.12. The prime implicant table is shown in Table 2.13. After removing the two essential prime implicants (14,15) and (8,9,12,13) the prime implicant table is reduced to four rows and three columns (Table 2.14). Row (6,14) is dominated by row (4,6), *i.e.* all minterms covered by (6,14) are also covered by (4,6). Therefore, row (6,14) can be removed. This in turn makes the prime implicant (4,6) pseudo-essential. Finally, (4,12) is chosen to cover minterm 4 of F_a since it has fewer literals than (4). The resulting minimal solutions are:

$$F_a = x_1x_2x_3 + x_1\bar{x}_3 + x_2\bar{x}_3\bar{x}_4$$

$$F_b = x_1x_2x_3 + \bar{x}_1x_2\bar{x}_4$$

Note that the product term $x_1x_2x_3$ appears in both solutions. Four product terms are needed to realize F_a and F_b. A separate minimization of the two function produces the following solutions:

$$F_a = x_1x_2 + x_1\bar{x}_3 + x_2\bar{x}_3\bar{x}_4$$

$$F_b = x_1x_2x_3 + \bar{x}_1x_2\bar{x}_4$$

where five product terms are needed to realize F_a and F_b.

I			II			III	
	$x_1x_2x_3x_4$			$x_1x_2x_3x_4$			$x_1x_2x_3x_4$
4	0 1 0 0 [a,b]		4,6	0 1 – 0 [b]		8,9,12,13	1 – 0 – [a]
8	1 0 0 0 [a]	√	4,12	– 1 0 0 [a]		12,13,14,15	1 1 – – [a]
6	0 1 1 0 [b]	√	8,9	1 0 0 – [a]	√		
9	1 0 0 1 [a]	√	8,12	1 – 0 0 [a]	√		
12	1 1 0 0 [a]	√	6,14	– 1 1 0 [b]			
13	1 1 0 1 [a]	√	9,13	1 – 0 1 [a]	√		
14	1 1 1 0 [a,b]	√	12,13	1 1 0 – [a]	√		
15	1 1 1 1 [a,b]	√	12,14	1 1 – 0 [a]	√		
			13,15	1 1 – 1 [a]	√		
			14,15	1 1 1 – [a,b]			

Table 2.12 Prime implicant generation for Example 2.7.

	F_a							F_b				
	4	8	9	12	13	14	15	4	6	14	15	
4	x							x				
4,6								x	x			
4,12	x			x								
6,14									x	x		
14,15						x	x			x	(x)	*
8,9,12,13		(x)	(x)	x	x							*
12,13,14,15				x	x	x	x					

Table 2.13 Prime implicant table for Example 2.7.

	F_a	F_b	
	4	4	6
4	x	x	
4,6		x	(x)
4,12	x		
~~6,14~~			x

Table 2.14 Reduced prime implicant table.

Serra [SER84] proposed an extension of the DSA algorithm to minimize multiple-output networks. Each minterm may have different RADs in each of the output functions. This fact is considered when ranking the minterms in the ON-set. Essential and pseudo-essential implicants are detected. Some heuristics were added to permit the pruning of expansion trees. The algorithm was implemented in APL, but no indication of the performance of the computer program, in terms of execution time, was given.

2.2.6 Reed-Muller Expansion

Reed-Muller expansions offer an alternative representation of logic functions to the traditional sum-of-product expressions. In the Reed-Muller expansion the operators AND, NOT, and EXOR (exclusive-or) are used. The EXOR operation, denoted by the symbol $\oplus$, also known as the Sum-Modulo-Two. Table 2.15 shows the truth table of the EXOR operation.

P	Q	P ⊕ Q
0	0	0
0	1	1
1	0	1
1	1	0

Table 2.15 Truth table for the EXOR operator.

The EXOR operator is useful since it arises naturally in the representation of arithmetic functions. For example, the least significant bit S of an addition, with inputs x_1, x_2, and a carry C, can be expressed as

$$S = x_1 \oplus x_2 \oplus C$$

which is clearly a simpler representation than the sum-of-products expression

$$S = \bar{x}_1 \bar{x}_2 C + \bar{x}_1 x_2 \bar{C} + x_1 x_2 C + x_1 \bar{x}_2 \bar{C}$$

Reed-Muller expansions are also used in the design of easily testable realizations of logic functions [RED72].

The canonic Reed-Muller expansion of an n-variable Boolean function takes the following form [MUL54] [REE54]

$$F(x_1, x_2, \ldots, x_n) = b_0 \oplus b_1 x_1 \oplus b_2 x_2 \oplus \ldots \oplus b_{2^n-1} x_1 x_2 \ldots x_n$$

where $b_i \in \{0,1\}$, $i = 0, 1, \ldots, 2^n - 1$.

Wu, Chen, and Hurst [WU82] proposed a geometric representation of Reed-Muller coefficients to aid the manual minimization process. The proposed representation is similar to Karnaugh maps. Besslich [BES83] developed a computer method based on the approach described in [WU82]. Most minimization methods [BES83, ZHA84, SAL79] are exhaustive in nature. Exhaustive minimization methods are not practical for any function with a reasonable number of inputs. More recently, heuristic minimization

procedures have been proposed [SAS86b, BES87].

2.3 MVL MINIMIZATION: PREVIOUS WORK

The goal of most binary minimization is a minimal sum-of-products expression. The "product" is the AND of one or more literals. A literal is either an input variable or its complement. The product terms are "summed" using the OR operator, and less frequently the EXOR operator. All operations can be implemented quite naturally using integrated circuits. The minimization goal (a sum-of-products expression with the minimum number of product terms) is easily understood and reasonable if the function is implemented using a PLA.

MVL functions can also be expressed as sum-of-products expressions [MUZ86]. Unlike their binary counterparts, there are a wide variety of choices for the "product", "sum", and unary operators. An operator is said to be unary if it operates on a single operand. Since Post [POS21] generalized Boolean algebra in 1921, the MIN and MAX operators have been widely used as "product" and "sum" operators [HUR84]. However, the use of multi-valued integrated injection logic (I^2L) circuits [MCC79] and charge coupled devices (CCD) [KER84], has led to the introduction of the SUM operator, since in both technologies the SUM operator can be implemented more efficiently. The SUM is defined to be the arithmetic sum, truncated at R - 1. R is the radix and R - 1 is the highest value that any variable can assume.

A wide variety of unary operations have been proposed [HUR84, VRA70, ALL84]:

- complementation

$$x' = \{(R - 1) - x\}$$

- successor

$$x_i^{\rightarrow} = \{(x_i + 1) \bmod R\}$$

• predecessor

$$x_i^{\leftarrow} = \{(x_i - 1) \bmod R\}$$

• literal operator (window)

$$a_x b = \begin{cases} (r - 1) & \text{if } a \le x \le b \\ 0 & \text{otherwise} \end{cases}$$

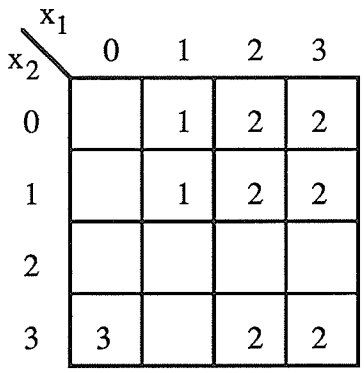
• generalized literal operator

$$x^S = \begin{cases} (r - 1) & \text{if } x \in S \\ 0 & \text{otherwise} \end{cases}$$

• staircase operator

$$x^{a\dagger b} = \begin{cases} x & \text{if } a \le x \le b \\ 0 & \text{otherwise} \end{cases}$$

A set from the unary and binary operators must be selected to give functional completeness. A set of operators Ψ is said to be functionally complete if a representation for any function can be found using only operators from Ψ. Functional completeness can be achieved in more than one way. The example in Figure 2.10 shows how a function can be represented in several ways making use of the different unary operators shown above and the binary functions MIN and MAX. The choice of a functionally complete set should reflect the efficiency of the underlying implementation.



$$F(x_1,x_2) = 2x_1^{1\ddagger3}\ {}^{0}_{2}x^{1}\ + x_1^{\leftarrow\ 3}\ {}^{3}_{2}x + 2\ {}^{2}_{1}x^{2}\ {}^{3}_{2}x^{3}$$

$$F(x_1,x_2) = x_1^{1\ddagger2}\ {}^{0}_{2}x^{1} + {}^{3}_{1}x^{3}\ x_1^{\leftarrow\ 0}\ {}^{1}_{2}x + x_1^{\leftarrow\ 3}\ {}^{3}_{2}x + x_1^{2\ddagger2}\ {}^{3}_{2}x^{3}$$

$$F(x_1,x_2) = 1\ {}^{1}_{1}x^{3}\ {}^{0}_{2}x^{1} + 2\ {}^{2}_{1}x^{3}\ {}^{0}_{2}x^{1} + 2\ {}^{2}_{1}x^{3}\ {}^{3}_{2}x^{3} + {}^{0}_{1}x^{0}\ {}^{3}_{2}x^{3}$$

$$F(x_1,x_2) = 1\ x_1^{1,2,3}\ {}^{0,1}_{2}x + 2\ x_1^{2,3}\ {}^{0,1,3}_{2}x + x_1^{0}\ {}^{3}_{2}x$$

Figure 2.10 Alternate implementations using different unary operators.

Traditional minimization techniques have been extended to MVL [SMI84, SU84, ALL84]. Miller & Muzio [MIL79] have shown that a minimal sum-of-product expression does not necessarily consist solely of prime implicants. Some literals are more costly to implement. If all implicants must be considered, the size of the implicant table becomes unmanageable, even for functions with a small number of input variables.

2.3.1 Direct Cover Minimization

The inefficiency of traditional minimization techniques led Pomper and Armstrong [POM81] to develop a direct cover minimization algorithm. A prime implicant cover is generated directly, in one pass. The algorithm proceeds as follows:

Algorithm: *Pomper-Armstrong direct cover minimization.*

1) a non-zero minterm is selected at random;
2) all prime implicants which cover the selected minterm are generated;
3) the largest prime implicant is added to the cover;
4) all covered minterms are changed to don't-cares.

These four steps are iterated until all minterms are covered. With slight modifications the algorithm can be used to minimize sum-of-products expressions using the SUM operator.

The Pomper-Armstrong minimization procedure can easily be programmed. The algorithm is efficient in terms of memory requirements as well as execution time. However, no claim of a minimal or near minimal solution is made. The randomness in the selection of the minterms to be covered makes an analysis of the algorithm extremely difficult. The minimization result of a given function is not unique. In fact, the minimization is likely to produce different results for the same function. This is due to the fact that a different minterm will be selected which in turn may lead to the choice of a different prime implicant.

Besslich [BES86] has presented a very general direct cover minimization which can be readily adapted to any algebra. Each minterm is assigned a weight that measures the degree to which minterms are clustered around it. The minterm with the minimum weight is chosen to be covered first. Justification for this choice of minterm is similar to the directed search algorithm: a minterm with low weight will have fewer possibilities for expansion and the probability of finding an essential prime implicant is higher. An efficiency coefficient is calculated for all implicants that contain the minterm to be covered. The efficiency coefficient is obtained by dividing the number of minterms covered by the cost of the corresponding implicant. This heuristic takes into consideration that a minimal sum-of-products expression may contain implicants which are not prime. The most

efficient implicant is chosen. The above steps are iterated until all minterms are covered.

Direct cover minimization limits the number of prime implicants which must be considered at any given point during the minimization process. The number of prime implicants which cover a given minterm is considerably less than the number of prime implicants of the function. As with all heuristics, a minimal sum-of-products expression is no longer guaranteed.

2.3.2 Cost Table Approach

Cost table minimization makes use of a table where each function is associated with a cost factor. To realize a function, selections from the table are made which combine to realize the target function at the lowest possible cost. Since most functions can be realized in more than one way, an exhaustive enumeration is needed to guarantee a minimal result.

A tabular-cost approach to minimize CCD circuits was first introduced by Kerkhoff and Robroek [KER82]. Lee and Butler [LEE83] improved this approach by reducing the size of the cost table. Both techniques are restricted to one variable functions. Abd-El Barr *et al.* [ABD86] developed two algorithms for the synthesis of 4-valued one and two variable functions for CCD implementation. Their results are superior to previous ones. The main drawback to their approach is its restriction to the two variable case. Since the number of functions grows from 4^2 to 4^3, a enumeration based on cost table is computationally not feasible.

In some technologies the standard sum-of-product expressions result in a very inefficient implementation. Kerkhoff [KER84] describes two CCD implementations of a two-input quaternary full-product circuit. Using the Vranesic algebra (with the unary successor and inverters) the overall cost was 1052. Kerkhoff's tabular-cost approach led to a total cost of 191. This simple example clearly shows that the algebra cannot be treated independently of the target technology.

2.4 REMARKS

Binary minimization is well understood. Over the last four decades, numerous algorithms have been presented, and some of them have been reviewed in this chapter. In general, exact minimization is not feasible. First, the number of prime implicants can be very large. If a cyclic function is to be minimized by an exact minimization procedure, all prime implicants must be considered. Second, the covering problem is known to belong to the class of NP-complete problems [BRA84]. It is worth noting that for some functions, even with a large number of input variables, a minimal cover can be extracted with no difficulty.

The intractability of exact minimization becomes evident with McBoole which claims to minimize functions with up to 20 input variables and 20 outputs. Yet for the function MULT4 [DAG86] which has only 8 input variables and 8 outputs, the minimal solution was not found, because branching was abandoned after 6 nested cycles (see Table 4.6). Nevertheless, McBoole gave minimal results in reasonable time for a large number of logic functions used in industrial PLAs.

Heuristic approaches are used in the design of practical PLAs. In the binary domain, near minimal solutions are acceptable. Minimization of multiple-valued logic functions is more complex than the minimization of their binary counterparts. Therefore, the need for heuristics is more evident in the MVL case.

Chapter 3

A DIRECT COVER ALGORITHM FOR TRUNCATED SUM MINIMIZATION

3.1 OVERVIEW

Over the past several years a number of different technologies have been considered for the implementation of MVL circuits [HUR84]. Each technology is well suited to a particular set of algebraic operators. This in turn affects the design methods used.

In the CCD and I^2L technologies for example, the realization of SUM operators is more economical than the realization of the MAX operator. Hence design methods which employ MIN, MAX and complement operators are not suited to I^2L and CCD design.

McCluskey [MCC79] has presented an algebraic system for designing multiple-valued I^2L circuits. While McCluskey's work is applicable solely to I^2L, Kerkhoff [KER84] has shown that it can be extended to the CCD case.

The tabular-cost minimization results in efficient CCD implementations (see Section 2.3.2). Unfortunately, there are two major disadvantages associated with this approach. First, tabular-cost minimization is restricted to one and two-variable functions. This seriously limits its applicability. Second, the implementation of the resulting representation is unstructured.

Structured implementations of complex multiple-valued circuits will likely replace random logic. This is evident in the binary domain where PLAs, ROMs, MUX-based designs, etc., are now widely used. Structured logic has three principal advantages over random logic:

- the use of regular structures aids the design process;
- regular structures can usually achieve a higher degree of compaction;

- regular structures are more easily tested than random ones.

PLAs seem to be promising for multiple-valued circuits. Several multiple-valued PLAs have been proposed [SAS86a, KER86, TIR84]. If the PLA is implemented using I^2L or CCD technology, a SUM operator is more suitable than the MAX operator for the 'OR' part of the PLA. The SUM operator is both easier to implement and frequently results in simpler realizations as shown in [BEN85].

The complexity of minimizing sum-of-product expressions using the SUM operator is similar to the minimization of Reed-Muller expansions — the best solution may not consist solely of prime implicants. The need for heuristic minimization is evident for exclusive-or sum-of-products expressions. In Papakonstantiou's opinion [PAP79] absolute minimization is only feasible for functions with $n \leq 4$. Even with extensive computational resources, absolute minimization is limited to small n since the number of product terms to be considered is exponential.

Design of PLAs requires a minimization procedure suited to the operators implemented in the PLA. In a PLA, each product term is realized as a single column in the 'AND' part. Hence, all product terms require the same area and can be considered to have the same cost. Realizations which minimize the number of product terms thus have minimal total cost. The lack of a suitable minimization procedure for PLA's where the 'OR' part employs the SUM operation motivated the development of the algorithm presented in this chapter.

3.2 PRELIMINARIES

The definitions of literal, product term, and implicant used in this chapter are, of necessity, different from those given in Chapter 1. In addition, the truncated sum operator (TSUM) is introduced.

Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of n input variables. Define the set $P = \{1, 2, \ldots,$

R - 1} which represents the values that the variable x_i can assume. An R-valued function *f* is a mapping

$$f: P \times P \times \ldots \times P \rightarrow P$$

Each element in the domain of *f* is a *minterm* of the function.

Let x_i be an input variable and let S_i be a subset of P. The literal function is defined as follows:

$$x_i^{S_i} = \begin{cases} R - 1 & \text{if } x_i \in S_i \\ 0 & \text{if } x_i \notin S_i \end{cases}$$

A *product term*

$$Q = c\, x_1^{S_1} x_2^{S_2} \ldots x_n^{S_n}$$

is defined to be the minimum of the literals and the constant $c \in \{1, 2 \ldots R - 1\}$. If a product term, Q, contains a literal for which $S_i = P$, Q is said to be independent of x_i. An independent variable may be omitted from the term. c is said to be the *value* of Q and is denoted by «Q». A product term contains all minterms for which it evaluates to c.

A product term Q is an *implicant* of the function *f* if the value of the function for each minterm contained in Q is greater than or equal to «Q». An implicant is termed *prime* if it is itself not contained in any other implicant of the function.

The *truncated sum* (TSUM) of two product terms is denoted by ◊ and defined as follows:

$$Q_1 \lozenge Q_2 = \text{MIN}(\langle\langle Q_1 \rangle\rangle + \langle\langle Q_2 \rangle\rangle, R - 1)$$

where Q_1 and Q_2 are product terms of the function and MIN is the arithmetic minimum. The *sum* of product terms is defined to be the truncated sum of the terms. Any function can be written as a *sum-of-products* expression.

For simplicity all examples will be given in three-valued or four-valued logic, but the algorithm presented is applicable to any radix.

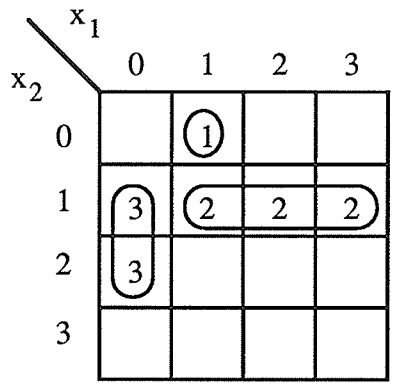


Figure 3.1 Map of a four-valued function.

Example 3.1. Consider the function shown in Figure 3.1 which can be expressed as a TSUM of 3 product terms.

$$F(x_1,x_2) = 1x_1^1 x_2^0 \Diamond 2x_1^{1,2,3} x_2^1 \Diamond 3x_1^0 x_2^{1,2}$$

Note that the constant 3 can be omitted from the final term.

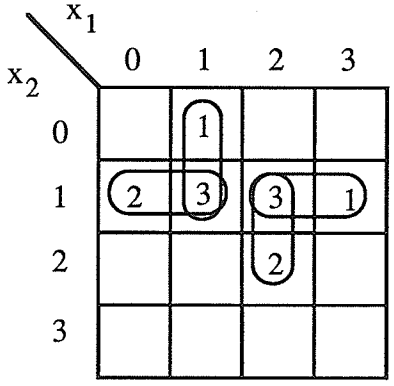


Figure 3.2 Map of the function used in Example 3.2.

For certain functions the minimal solution contains terms which combine to realize a larger value. Example 3.2 illustrates such a function.

Example 3.2. The function shown in Figure 3.2 can be expressed as the following sum-of-products expression:

$$F(x_1,x_2) = 1x_1^{1}\, x_2^{0,1} \lozenge 1x_1^{2,3}\, x_2^{1} \lozenge 2x_1^{0,1}\, x_2^{1} \lozenge 2x_1^{2}\, x_2^{1,2}$$

Here each '3' value is realized by summing a '2' and a '1'.

The minimal solution may contain terms which are not prime implicants as demonstrated in Example 3.1 where the '1' and the '2' terms are not prime, since each is part of a larger term. In fact for this example, no minimum TSUM cover exists where all the product terms are prime implicants. Hence, a minimization procedure cannot be restricted to considering only prime implicants. The set of implicants which must be considered can become very large.

Lemma 3.1. A function of radix R with n input variables may have up to $(R - 1)\omega^n$ implicants, where $\omega = 2^R - 1$.

Proof. Any implicant can be written as

$$c x_1^{S_1}\, x_2^{S_2}\, x_3^{S_3} \ldots . x_n^{S_n}$$

There are (R - 1) choices for c. Each S_i is a subset of P. P has $2^R - 1$ non-empty subsets. Hence, the number of implicants follows. All possible product terms are implicants of the constant function f(X) = (R - 1). Q.E.D.

Clearly it is not feasible to consider all possible implicants of a function. For example, a 4-valued function with three input variables may have up to 10,125 implicants. Some heuristic must be applied in order to reduce the set of implicants which will be considered.

Direct cover minimization was described in Section 2.3.1. A direct cover method for multiple-valued minimization using the TSUM operator was first suggested by Pomper and Armstrong [POM81]. Their algorithm selects a minterm at random and finds all prime

implicants which include the selected minterm. The "best" prime implicant is then added to the solution, and the function is modified accordingly. The "best" prime implicant is the one that covers the most minterms which are not don't-cares. The advantage of this algorithm is its speed. No claim of a minimal or near minimal solution is made. The randomness makes the analysis of this algorithm extremely difficult.

Besslich [BES86] has presented a very general direct cover minimization which can be readily adapted to any algebra. His algorithm can be summarized as follows:

- select the most isolated uncovered minterm α;
- calculate the efficiency coefficient for all implicants that contain α (the efficiency coefficient is obtained by dividing the number of minterms which the implicant covers by the cost associated with the implicant);
- include the most efficient implicant in the solution;
- repeat the process until all minterms are covered.

This algorithm seems to be well suited for minimizations using the maximum operator. When dealing with the TSUM, making an implicant more efficient, *i.e.* bigger, may make the remaining function more complex. If we apply Besslich's algorithm to the function given in Example 3.1, we obtain

$$F(x_1,x_2) = 1x_1^{1}x_2^{0,1} \lozenge 1x_2^{1} \lozenge 1x_1^{0,2,3}x_2^{1} \lozenge 3x_1^{0}x_2^{1,2}$$

which is not minimal. Any minterm which evaluates to R - 1 can be realized by a sum which may exceed this value, since the sum is always truncated at R - 1.

3.3 THE ALGORITHM

It is important to select the first minterm intelligently. It has been suggested that the most "isolated" minterm is the best choice [RHY77]. Isolation is a measure of how many possible combinations a minterm or a product term has with neighbouring terms. The

higher the "isolation", the fewer combinations exist. This concept is formalized in Definition 3.4 below.

The importance of selecting an isolated minterm is illustrated by the binary example shown in Figure 3.3 [BES86]. If a direct cover approach begins with one of the minterms in the central four squares of the map, the prime implicant x_2x_4 will be included in the solution. Clearly it is redundant. If a direct cover approach begins with each 'isolated' minterm in turn, the minimal solution of four prime implicants will be found.

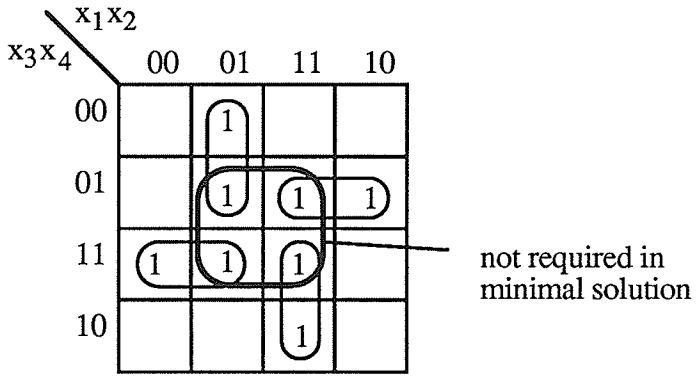


Figure 3.3 A binary function with a redundant term.

Definition 3.1. (identical to Definition 2.3) Two minterms are said to be *adjacent* if they differ in one input variable x_i. The minterms are termed *adjacent relative to* x_i. Each minterm has n(R - 1) adjacencies.

Definition 3.2. Let α and β be two adjacent minterms. β is said to be an *expandable adjacency* of α if $f(\alpha) \leq f(\beta)$. For minterm α the *number of expandable adjacencies* is denoted by EA_α.

Definition 3.3. For minterm α the *number of directions of expandable adjacency* of α (DEA_α) is

$$DEA_\alpha = \sum_{i=1}^{n} \gamma_i$$

where γ_i is 1 if α has an expandable adjacency with respect to x_i, and is 0 otherwise.

Definition 3.4. The isolation factor of a minterm α , denoted IF_α, is

$$IF_\alpha = \frac{1}{DEA_\alpha (R\text{-}1) + EA_\alpha + 1}$$

The algorithm must be able to handle don't-care conditions. Don't-care conditions may be specified in the original function and new don't-care conditions may be introduced during the minimization process. Don't-care conditions will be given the value of R (the radix). With this value the above definition of expandable adjacency still holds. A minterm adjacent to a don't-care can always be expanded in that direction.

The basic idea behind the algorithm is quite simple, consisting of an iteration of the following two steps:

- select the most isolated minterm α, *i.e.* the one with the highest isolation factor;
- consider all implicants that contain α and have the same value as α, and select the one which will make the remaining function as simple as possible. A metric for the simplicity of a function is defined below.

Definition 3.4. A *function break* occurs when two adjacent minterms have different values.

The method used here for measuring the complexity of a function is to count the number of function breaks. When an implicant Q is considered for inclusion in the solution, the total number of breaks in the function which remains to be realized need not be computed. Rather, only the change in the number of breaks need be considered. This is called the *break count reduction* (BCR).

Algorithm: *Determining the break count reduction for the product term Q of f(X)*

i) BCR $\leftarrow$ 0.

ii) Let M be the set of minterms which are contained in Q.

iii) For each $\alpha \in M$ such that $f(\alpha) \neq R$:

iii.a) For k=1, 2,.., n

iii.a.1) if (there exists a minterm $ß \notin M$ adjacent to α relative to x_k such that $f(ß) = f(\alpha)$ - «M») or ($f(\alpha)$ = «Q»), then

BCR $\leftarrow$ BCR + 1;

iii.a.2) if there exists a minterm $ß \notin M$ adjacent to α relative to x_k such that $f(ß) = f(\alpha)$ and $f(\alpha) \neq R - 1$, then

BCR $\leftarrow$ BCR - 1.

Step iii.a.2 requires some explanation. A break introduced for a minterm α, with $f(\alpha) = R - 1$, is not counted since the sum at α may exceed R - 1 since it is truncated to R - 1.

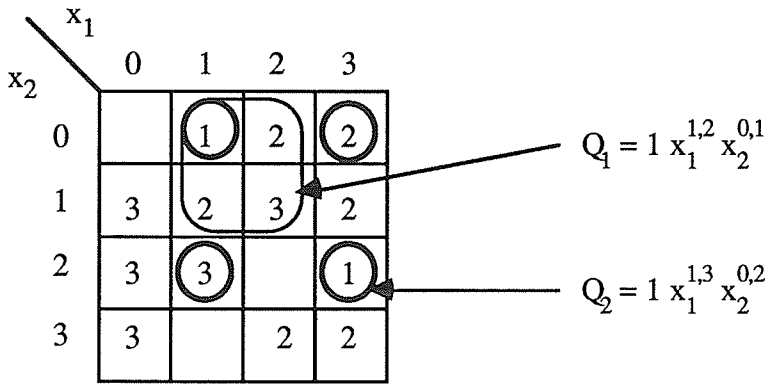


Figure 3.4 Two implicants considered in Example 3.3.

Example 3.3. Consider the implicant Q_1 of the function shown in Figure 3.4. Q_1 has an BCR of 1, which is obtained as follows:

minterm	k = 1	k = 2	total
$x_1^1 x_2^0$	1	1	2
$x_1^1 x_2^1$	-1	0	-1
$x_1^2 x_2^0$	-1	-1	-2
$x_1^2 x_2^1$	1	1	2
		BCR =	1

On the other hand, the implicant Q_2, of the same function, has a BCR of 3, which is obtained as follows:

minterm	k = 1	k = 2	total
$x_1^1 x_2^0$	1	1	2
$x_1^1 x_2^2$	0	1	1
$x_1^3 x_2^0$	-1	-1	-2
$x_1^3 x_2^2$	1	1	2
		BCR =	3

This leads to the conclusion that Q_2 is likely a better choice than Q_1 to cover the minterm $1\, x_1^1 x_2^0$

Algorithm: ***DCM*** *Direct cover minimization for multiple-valued functions using the truncated sum*

Let f(X) be the function to be minimized and let g(X) be a copy of f(X). The radix of f(X) is denoted by R.

STEP 1 Let M be the set of all minterms ß such that $1 \leq g(ß) \leq R - 1$ and $f(ß) \neq R - 1$.

$$\text{MinValue} \leftarrow \begin{cases} \text{minimum value of } g(ß) \text{ for } ß \in M, & \text{if } M \neq \phi \\ R - 1, & \text{otherwise} \end{cases}$$

All non-zero values of g(X) less than MinValue are replaced by MinValue.

STEP 2 If MinValue < R - 1 then

Find the IFs for each minterm of g(X) whose value is equal to MinValue for which the value of f(X) was not R - 1.

else

Find the IFs for each minterm whose value is less than R. (All remaining minterms can be considered to have value R - 1).

STEP 3 Let α be the minterm with the maximum IF. If more than one such minterm exists, one is selected arbitrarily.

STEP 4 Find the BCR for all implicants that include the minterm α. The value of the implicants considered must be $g(\alpha)$.

STEP 5 Let Q_{max} be the implicant with the maximum BCR which contains the minterm α.

STEP 6 Add Q_{max} to the solution.

STEP 7 Set $g(\alpha) \leftarrow g(\alpha) - «Q_{max}»$ for each minterm α included in Q_{max} for which $g(\alpha) \neq R$. If $g(\alpha) = 0$ and $f(\alpha) = R - 1$ then set $g(\alpha) \leftarrow R$ (this ensures that α is treated a don't-care from this point on).

STEP 8 If there is any minterm α such that $1 \leq g(\alpha) \leq R - 1$ then go to STEP 1.

Minterms with the smallest nonzero value are covered first (STEP 1) since they have fewer possibilities of being realized by "summing" other implicants. The function keeps changing during the minimization procedure (STEP 7). Once an implicant is added to the solution, the function must be modified accordingly. If the value of R - 1 has been reached

for a minterm α for which $f(\alpha) = R - 1$, $g(\alpha)$ can now be treated as a don't-care minterm.

The algorithm to find the BCR in STEP 4 requires a slight modification. In step iii.a.2 of the BCR calculation the value of $g(\alpha)$ is compared to R - 1. $g(\alpha)$ must be replaced by $f(\alpha)$.

3.4 EXAMPLES

In this section, examples are given to illustrate the steps of the DCM algorithm. For simplicity the product term

$$c x_1^{S_1} x_2^{S_2} x_3^{S_3} \ldots . x_n^{S_n}$$

will be written as $c(S_1)(S_2)(S_3) \ldots (S_n)$.

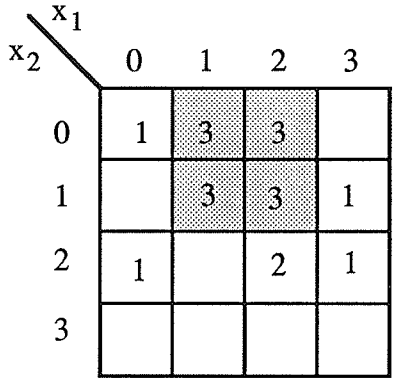


Figure 3.5 Map[1] of the function used in Example 3.4.

Example 3.4. Consider the two-variable 4-valued function depicted in Figure 3.5. The minimization proceeds as follows:

STEP 1 MinValue $\leftarrow$ 1.

STEP 2 The isolation factors for the minterms with value 1 are as follows[2]:

minterm	(0,0)	(0,2)	(3,1)	(3,2)

[1] The minterms for which the function evaluates to R - 1 are shaded because they are treated differently during the minimization process.

[2] In practice the inverse of the isolation factor is calculated to avoid the computationally expensive divide operation.

IF	1/10	1/10	1/10	1/10	

STEP 3 $\alpha \leftarrow (0,0)$.

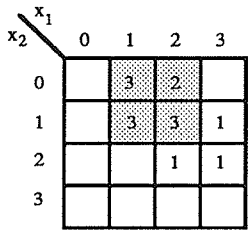
STEP 4 The six implicants which contain α have the following break count reductions:

implicant	1(0)(0)	1(0,1)(0)	1(0,2)(0)	1(0,1,2)(0)	1(0)(0,2)	1(0,2)(0,2)
BCR	1	1	2	2	3	4

STEP 5 $Q_{max} \leftarrow 1\ x_1^{0,2}\ x_2^{0,2}$.

STEP 6 Add $1\ x_1^{0,2}\ x_2^{0,2}$ to the solution.

STEP 7 g(X) now becomes:



STEP 8 Go to STEP 1.

STEP 1 MinValue $\leftarrow$ 1.

STEP 2 The isolation factors for the minterms with value 1 are as follows:

minterm	(2,2)	(3,1)	(3,2)
IF	1/10	1/10	1/9

STEP 3 $\alpha \leftarrow (3,2)$.

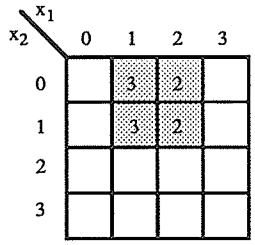
STEP 4 The four implicants which contain α have the following break count reductions:

implicant	1(3)(2)	1(2,3)(2)	1(3)(1,2)	1(2,3)(1,2)
BCR	0	3	3	7

STEP 5 $Q_{max} \leftarrow 1\ x_1^{2,3}\ x_2^{1,2}$.

STEP 6 Add $1\ x_1^{2,3}\ x_2^{1,2}$ to the solution.

STEP 7 g(X) now becomes:



STEP 8 Go to STEP 1.

STEP 1 MinValue ← 3.

(MinValue is assigned the value R - 1 since the two minterms which evaluate to 2 had the value 3 in f(X), therefore the sum is allowed to exceed R - 1).

STEP 2 The isolation factors for the minterms with value 3 are as follows.

minterm	(1,0)	(1,1)	(2,0)	(2,1)
IF	1/9	1/9	1/9	1/9

STEP 3 $\alpha \leftarrow (1,0)$.

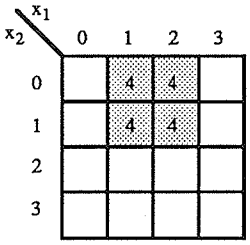
STEP 4 The four implicants which contain α have the following break count reductions:

implicant	3(1)(0)	3(1,2)(0)	3(1)(0,1)	3(1,2)(0,1)
BCR	2	4	4	8

STEP 5 $Q_{max} \leftarrow x_1^{1,2} x_2^{0,1}$.

STEP 6 Add $x_1^{1,2} x_2^{0,1}$ to the solution.

STEP 7 g(X) now becomes:



STEP 8 The termination condition has been reached. The result is

$$f(x_1,x_2) = 1\, x_1^{0,2} x_2^{0,2} \lozenge 1\, x_1^{2,3} x_2^{1,2} \lozenge x_1^{1,2} x_2^{0,1} \quad .$$

A class of functions which requires relatively many product terms is the set of Latin Square functions [BEN85].

Definition 3.5. $f(x_1,x_2)$ is a *Latin Square Function* if $f(x_1,c)$ and $f(c,x_2)$ assume all R possible logic values for every $c \in \{0,1,\ldots,R-1\}$. On the map of a latin square function each row and each column is a permutation of $\{0,1,\ldots,R-1\}$.

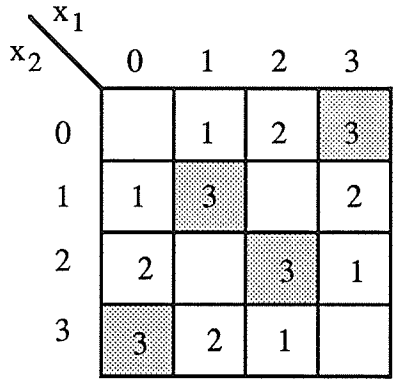


Figure 3.6 A latin square function.

The DCM was applied to the four latin square functions given in [DUE86]. For one of the function (shown in Figure 3.6) a better solution was found. The solution given in [DUE86] used 8 product terms. As shown below, the function only requires 7 product terms. The minimization proceeds as follows:

Example 3.5.

STEP 1 MinValue $\leftarrow$ 1.

STEP 2 The isolation factors for the minterms with value 1 are as follows:

minterm	(0,1)	(1,0)	(2,3)	(3,2)
IF	1/11	1/11	1/11	1/11

STEP 3 $\alpha \leftarrow (0,1)$.

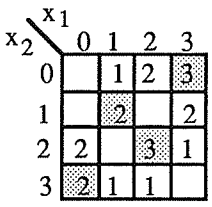
STEP 4 The nine implicants which contain α have the following break count reductions:

implicant	1(0)(1)	1(0)(1,3)	1(0,3)(1)	1(0)(1,2)	1(0)(1,2,3)
BCR	2	4	3	3	4
implicant	1(0,3)(1,2)	1(0,1)(1)	1(0,1)(1,3)	1(0,1,3)(1)	
BCR	4	4	6	4	

STEP 5 $Q_{max} \leftarrow 1(0,1)(1,3)$.

STEP 6 Add 1(0,1)(1,3) to the solution.

STEP 7 g(X) now becomes:



STEP 8 Go to STEP 1.

STEP 1 MinValue ← 1.

STEP 2 The isolation factors for the minterms with value 1 are as follows.

minterm	(1,0)	(1,3)	(2,3)	(3,2)
IF	1/11	1/11	1/11	1/11

STEP 3 $\alpha \leftarrow (1,0)$.

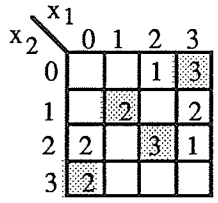
STEP 4 The nine implicants which contain α have the following break count reductions:

implicant	1(1)(0)	1(1)(0,3)	1(1,3)(0)	1(1,2)(0)	1(1,2)(0,3)
BCR	1	3	3	2	6
implicant	1(1,2,3)(0)	1(1)(0,1)	1(1)(0,1,3)	1(0,1)(0,1)	
BCR	3	2	3	4	

STEP 5 $Q_{max} \leftarrow 1(1,2)(0,3)$.

STEP 6 Add 1(1,2)(0,3) to the solution.

STEP 7 g(X) now becomes:



STEP 8 Go to STEP 1.

STEP 1 MinValue ← 1.

STEP 2 The isolation factors for the minterms with value 1 are as follows.

minterm	(2,0)	(3,2)
IF	1/10	1/11

STEP 3 $\alpha \leftarrow (2,0)$.

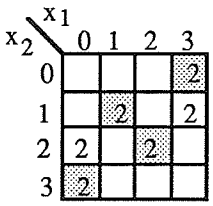
STEP 4 The four implicants which contain α have the following break count reductions:

implicant	1(2)(0)	1(2,3)(0)	1(2)(0,2)	1(2,3)(0,2)
BCR	2	3	3	6

STEP 5 $Q_{max} \leftarrow 1(2,3)(0,2)$.

STEP 6 Add 1(2,3)(0,2) to the solution.

STEP 7 g(X) now becomes:



STEP 8 Go to STEP 1.

STEP 1 MinValue ← 2.

STEP 2 The isolation factors for the minterms with value 2 are as follows.

minterm	(0,2)	(3,1)
IF	1/9	1/9

STEP 3 $\alpha \leftarrow (0,2)$.

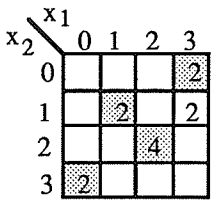
STEP 4 The three implicants which contain α have the following break count reductions:

implicant	2(0)(2)	2(0,2)(2)	2(0)(2,3)
BCR	0	3	3

STEP 5 $Q_{max} \leftarrow 2(0,2)(2)$[3].

STEP 6 Add 2(0,2)(2) to the solution.

STEP 7 g(X) now becomes:



STEP 8 Go to STEP 1.

STEP 1 MinValue ← 2.

STEP 2 (3,1) is the only minterm with value 2, which did not have the value 3 in f(X), has an isolation factor of 1/9.

STEP 3 α ← (3,1).

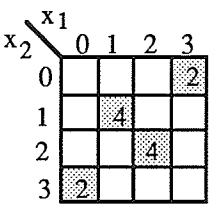
STEP 4 The three implicants which contain α have the following break count reductions:

implicant	2(3)(1)	2(1,3)(1)	2(3)(1,2)
BCR	0	3	3

STEP 5 $Q_{max} \leftarrow 2(1,3)(1)$.

STEP 6 Add 2(1,3)(1) to the solution

STEP 7 g(X) now becomes:



STEP 8 Go to STEP 1.

[3] The tie between 2(0,2)(2) and 2(0)(2,3) is arbitrarily broken.

The last two iterations of the algorithm are trivial. The two minterms which remain uncovered, (0,3) and (3,0), can each be covered by a single implicant, 3(0)(3) and 3(3)(0) respectively. These two implicants are added to the solution — the termination condition has been reached. The result is

$$f(x_1,x_2) = 1\,x_1^{0,1}x_2^{1,3} \lozenge 1\,x_1^{1,2}x_2^{0,3} \lozenge 1\,x_1^{2,3}x_2^{0,2} \lozenge 2\,x_1^{0,2}x_2^{2} \lozenge 2\,x_1^{1,3}x_2^{1} \lozenge 3\,x_1^{0}x_2^{3} \lozenge 3\,x_1^{3}x_2^{0}$$

The algorithm does not always produce a minimal result. Example 3.6 shows a three-variable three-valued function for which the minimal sum-of-product expression was not obtained.

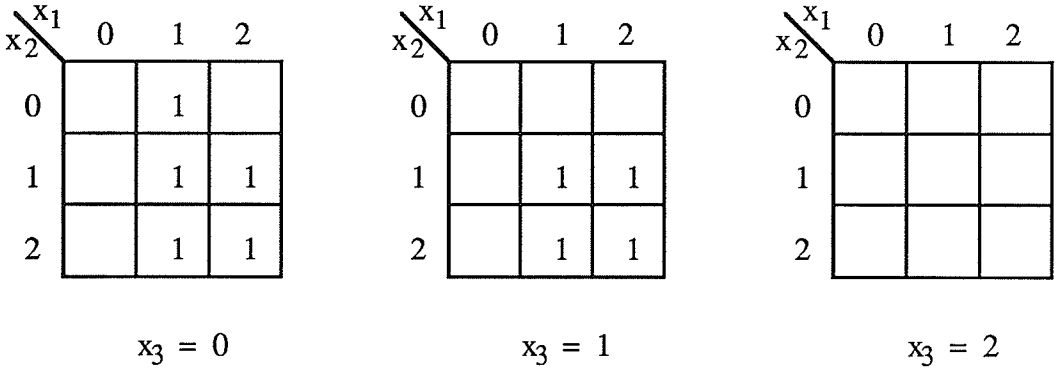


Figure 3.7 Three-valued three-variable function used in Example 3.6.

Example 3.6. Consider the function shown in Figure 3.7. The unique minimal sum-of-products expression is

$$f(x_1,x_2,x_3) = 1\,x_1^{1}x_2^{0}x_3^{0} \lozenge 1\,x_1^{1,2}x_2^{1,2}x_3^{0,1}$$

The minimization of the above function proceeds as follows:

STEP 1 MinValue ← 1.

STEP 2 The isolation factors for the minterms with value 1 are as follows:

minterm	(1,0,0)	(1,1,0)	(1,2,0)	(2,1,0)	(2,2,0)
IF	1/6	1/14	1/14	1/13	1/13
minterm	(1,1,1)	(1,2,1)	(1,2,1)	(2,2,1)	
IF	1/13	1/13	1/13	1/13	

STEP 3 $\alpha \leftarrow (1,0,0)$.

STEP 4 The four implicants which contain α have the following break count reductions:

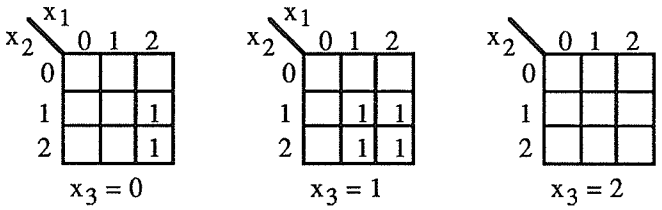
implicant	1(1)(0)(0)	1(1)(0,1)(0)	1(1)(0,2)(0)	1(1)(0,1,2)(0)
BCR	2	2	2	5

STEP 5 $Q_{max} \leftarrow 1(1)(0,1,2)(0)$.

STEP 6 Add 1(1)(0,1,2)(0) to the solution.

This will not lead to a minimal solution. The "best" minterm to cover α is 1(1)(0)(0), but it does not have the maximum BCR.

STEP 7 g(X) now becomes:



STEP 8 Go to STEP 1.

STEP 1 MinValue $\leftarrow$ 1.

STEP 2 The isolation factors for the minterms with value 1 are as follows:

minterm	(2,1,0)	(2,2,0)	(1,1,1)	(1,2,1)	(1,2,1)	(2,2,1)
IF	1/9	1/9	1/9	1/9	1/13	1/13

STEP 3 $\alpha \leftarrow (2,1,0)$.

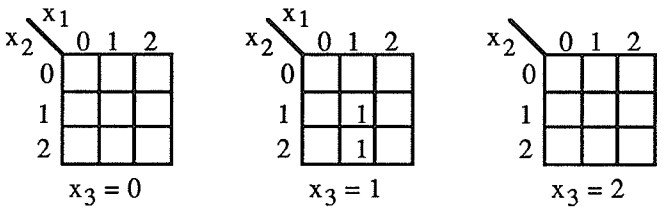
STEP 4 The four implicants which contain α have the following break count reductions:

implicant	1(2)(1)(0)	1(2)(1,2)(0)	1(2)(1)(0,1)	1(2)(1,2)(0,1)
BCR	1	4	3	10

STEP 5 $Q_{max} \leftarrow 1(2)(1,2)(0,1)$.

STEP 6 Add 1(2)(1,2)(0,1) to the solution.

STEP 7 g(X) now becomes:



STEP 8 Go to STEP 1.

STEP 1 MinValue $\leftarrow$ 1.

STEP 2 The isolation factors for the minterms with value 1 are as follows:

minterm	(1,1,1)	(1,2,1)
IF	1/5	1/5

STEP 3 $\alpha \leftarrow$ (1,1,1).

STEP 4 The two implicants which contain α have the following break count reductions:

implicant	1(1)(1)(1)	1(1)(1,2)(1)
BCR	2	6

STEP 5 $Q_{max} \leftarrow 1(1)(1,2)(1)$.

STEP 6 Add 1(1)(1,2)(1) to the solution.

STEP 7 g(X) = 0 for all X.

STEP 8 The termination condition has been reached. The final result is

$$f(x_1,x_2,x_3) = 1\, x_1^{1} x_2^{0,1,2} x_3^{0} \lozenge 1\, x_1^{2} x_2^{1,2} x_3^{0,1} \lozenge 1\, x_1^{1} x_2^{1,2} x_3^{1}$$

which is obviously not minimal.

Example 3.7. Consider the function

$$f(x_1,x_2,x_3) = 1\, x_1^1 x_2^0 x_3^0 \lozenge 2\, x_1^{12} x_2^{01} x_3^{01}$$

This function is similar to the one given in Example 3.6. For this function the algorithm finds the minimal solution. Again the most isolated minterm (1,0,0) is covered first. This time the implicant 1(1)(0)(0) has the highest BCR and the minimal sum-of-products expression is obtained.

3.5 RESULTS

The algorithm has been implemented in APL. APL provides a suitable environment for experimentation with an algorithm. Since APL programs are interpreted and hence execute rather slowly, the algorithm is currently being implemented in a compiled language.

The APL program was tested using several four-valued functions. Some of the results are summarized in Table 3.1. The function MINi realizes the minimum of its input variables. Similarly, MAXi and SUMi realize the maximum and the arithmetic sum modulo 4 of the input variables. MAX3X is identical to MAX3 except that the 0 function value in MAX3 is 1 in MAX3X. This doubles the number of implicants which must be considered for the first minterm covered.

The minimization of the maximum function is extremely time consuming. This is not surprising, since the most isolated minterm will be contained in $2^{(R-1)n-1}$ implicants. For all of these implicants the BCR must be calculated. For MAX4 an implicant must be chosen from a set of 2^{11} implicants. It took over 232 seconds of cpu time on an Amdahl 580 to select the first implicant for MAX4.

It is interesting to compare the computation time required in solving MINi, MAXi, and SUMi. For MINi the number of terms in the solution is always three and independent of the value of i. MINi and MAXi do not contain any minterms with a high isolation factor. In MINi 4^i implicants are considered to cover the first minterm, whereas for the

second minterm only 2^i terms are considered, and only one term is considered for the last minterm. The DCM relies heavily on the isolation factor to limit the number of implicants to be considered. For SUMi some of the minterms are more "isolated", and hence the time required to find the first implicant is only a small fraction of the total time.

function	n	cost	max. time to choose 1 implicant	total time
MIN2	2	3	0.182	0.237
SUM2	2	6	0.111	0.388
MAX2	2	5	0.354	0.900
MIN3	3	3	1.738	1.958
SUM3	3	20	0.401	3.714
MAX3	3	11	8.644	23.024
MAX3X	3	6	19.994	29.713
MIN4	4	3	17.313	17.991
SUM4	4	73	1.385	46.110
MAX4	4	19	232.461	626.340
MIN5	5	3	178.070	180.433
SUM5	5	276	5.009	745.126

Notes:

a) all functions use four logic levels;

b) n is the number of inputs;

c) the cost is measured by the number of product terms in the solution;

d) c.p.u. time is for the APL execution on an Amdahl 580.

Table 3.1 Execution times for a DCM implementation.

3.6 REMARKS

A new algorithm to minimize multiple-valued logic functions using the truncated-SUM (TSUM) operator has been presented. A direct cover method, together with a heuristic selection of the implicants is used. Implicants are chosen so that the function remaining to be realized is as simple as possible. The "simplicity" of a function is measured in terms of the number of breaks between adjacent minterms.

The algorithm appears to produce a near-minimal cover. However, it is impossible to quantify this since there is no known algorithm which will produce minimal results in reasonable time.

The main drawback of the algorithm is its exponential complexity. Selecting the most isolated minterm first is one attempt to deal with this problem. A way of reducing the set of implicants considered to cover a minterm is also necessary. This is a subject of ongoing research.

Chapter 4

DIRECTED SEARCH MINIMIZATION OF MULTIPLE-VALUED FUNCTIONS

4.1 OVERVIEW

Classical binary minimization algorithms begin with the generation of all prime implicants (see Section 2.2.1). The second step is the selection of a minimum, or near minimum, number of prime implicants which cover the function. This approach is readily extended to the multiple-valued case [SMI84] but it is very inefficient.

In 1977, Rhyne, Noe, McKinney, and Pooch [RHY77] proposed the directed search algorithm (DSA) for the minimization of single-output binary functions. A review of the DSA algorithm, together with examples, was given in Section 2.2.2. During the prime implicant generating process, the DSA recognizes essential and pseudo-essential prime implicants. Not all prime implicants are necessarily generated and, typically, the actual number is a small fraction of the total number of prime implicants. The DSA is able to detect cycles but, unfortunately, it does not resolve them. Standard cycle resolution methods or heuristics must be used.

Serra [SER84] has extended the directed search algorithm to multiple-output binary minimization. In this chapter, an extension of the DSA, termed DSA-MV, to handle functions with multiple-valued inputs and a binary output is introduced. Any multiple-output binary or multiple-valued problem can be mapped to this type of function so the algorithm presented is applicable to a large class of problems. The method presented here has been found to produce equivalent results for the examples given in [SER84]. Results are considered equivalent if they use the same number of product terms.

The handling of the multiple-output problem as a multiple-valued input variable

eliminates the need for flags an complex heuristics used by Serra [SER84]. Empirical results have shown that the ranking of DSA-MV is more efficient than the one introduced by Serra, *i.e.* the expansion of the chosen minterm results in an essential prime implicant more frequently.

For multiple-valued functions the solutions found take the form

$$f(x_1,\ldots,x_n) = \sum_{m=1}^{r-1} m \bullet f_m(x_1,\ldots,x_n) \qquad (4.1)$$

where the f_m are decisive functions (assuming only the values 0 and R-1) expressed in sum of products form.

A number of difficulties associated with multiple-valued minimization have been identified by Muzio and Miller [MUZ79]. The problems arise from the costing of the literals required to form product terms. These problems are avoided in this chapter since the target circuit implementation is a PLA. This allows the use of heuristics which consider the cost of all product terms to be equal. Heuristics are required since exact minimization, even for PLAs, is computationally very expensive. The algorithm has been found to produce minimal or nearly minimal results for a variety of problems.

Any binary problem with n inputs and m outputs can be represented by a multiple-valued input function *g* with a single binary output. *g* has n + 1 variables, where $p_i = 2$ for i = 1, 2 . . . n , and $p_{n+1} = m$. The example below shows how the truth table of *g* is derived from the truth table of the f_i.

Example 4.1.

x_1	x_2	f_1	f_2	f_3
0	0	1	0	1
0	1	1	1	0
1	0	0	1	1
1	1	1	1	1

x_1	x_2	x_3	g	
0	0	0	1	
0	1	0	1	(from f_1)
1	0	0	0	
1	1	0	1	
0	0	1	0	
0	1	1	1	(from f_2)
1	0	1	1	
1	1	1	1	
0	0	2	1	
0	1	2	0	(from f_3)
1	0	2	1	
1	1	2	1	

Sasao [SAS78] has shown that minimizing g is equivalent to minimizing the f_i. It is clear that this is also true for multiple-valued input functions.

If a realization of the form of (4.1) is sought, the minimization of a multiple-valued output function can be transformed to the minimization of a set of binary-valued output functions as shown in Appendix A.

The product term

$$x_1^{S_1} x_2^{S_2} \ldots x_n^{S_n}$$

is denoted by the binary vector:

$$c_1^0 c_1^1 \ldots c_1^{p_1 - 1} - c_2^0 c_2^1 \ldots c_2^{p_2 - 1} - \ldots - c_n^0 c_n^1 \ldots c_n^{p_n - 1}$$

where

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

This is known as the *cube* notation [SU84]. A cube is said to have n coordinates where the bit string $c_i^0 c_i^1 \ldots c_i^{p_i - 1}$ represents the i^{th} coordinate of the cube. A minterm contains a single 1 in each coordinate.

Example 4.2. Let P= {3, 4, 4}. The product term

$$x_1^{0,1} x_2^{1} x_3^{0,2,3}$$

is represented by the cube 110-0100-1011.

Definition 4.1. (identical to Definition 2.1) Two minterms are said to be *adjacent* if they differ in only one of their input variables.

Definition 4.2. Let M_1 be a minterm in the ON-set of the function *f*. A second minterm M_2 is said to be an *expandable adjacency* of M_1 if and only if M_1 and M_2 are adjacent and M_2 is in the ON-set or the DC-set of *f*.

4.2 THE ALGORITHM

Directed search minimization starts with the list of minterms which are in the ON-set. A minterm is selected from the ON-set and expanded into all possible prime implicants containing that minterm. During the expansion process a tree is created. Clearly, it is advantageous to keep the size of the tree as small as possible. By selecting a minterm which offers very few possibilities for expansion the growth of the tree can be limited (at least the initial branching possibilities are keep as low as possible). Therefore, the first step in the minimization procedure is to find all expandable adjacencies for each minterm in the ON-set.

The expandable adjacencies of a minterm can be represented by a bit string similar to the cube notation of the minterm. Let M be a minterm and let M_c be the cube representing M. Let $Q_1, Q_2, \ldots Q_r$ be the minterms which are expandable adjacencies of M and let Q_{c1}, $Q_{c2}, \ldots Q_{cr}$ be their respective cube representations. The bit string representing the expandable adjacencies of M, EAV_M (Expandable Adjacency Vector), is obtained as follows

$$EAV_M = (Q_{c1} + Q_{c2} + \ldots + Q_{cr}) \bullet (M_c)'$$

where +, •, and ' represent the bitwise OR, AND, and COMPLEMENT operations respectively.

Example 4.3. Let

$M =$ 010 - 1000 - 0010,
$Q_1 =$ 100 - 1000 - 0010,
$Q_2 =$ 001 - 1000 - 0010,
and $Q_3 =$ 010 - 1000 - 0100.
Then $EAV_M =$ 101 - 0000 - 0100.

Each expandable adjacency vector has an associated weight EAVW which consists of a pair of integers. The integers in EAVW are obtained as follows:

- The first integer indicates the number of coordinates of EAV which are not all zeros.
- The second number is the total number of ones in the EAV.

Example 4.4. Consider the function shown in Figure 4.1, with P = {4, 4, 3}. The expandable adjacencies with the corresponding weight pairs are shown in Table 4.1.

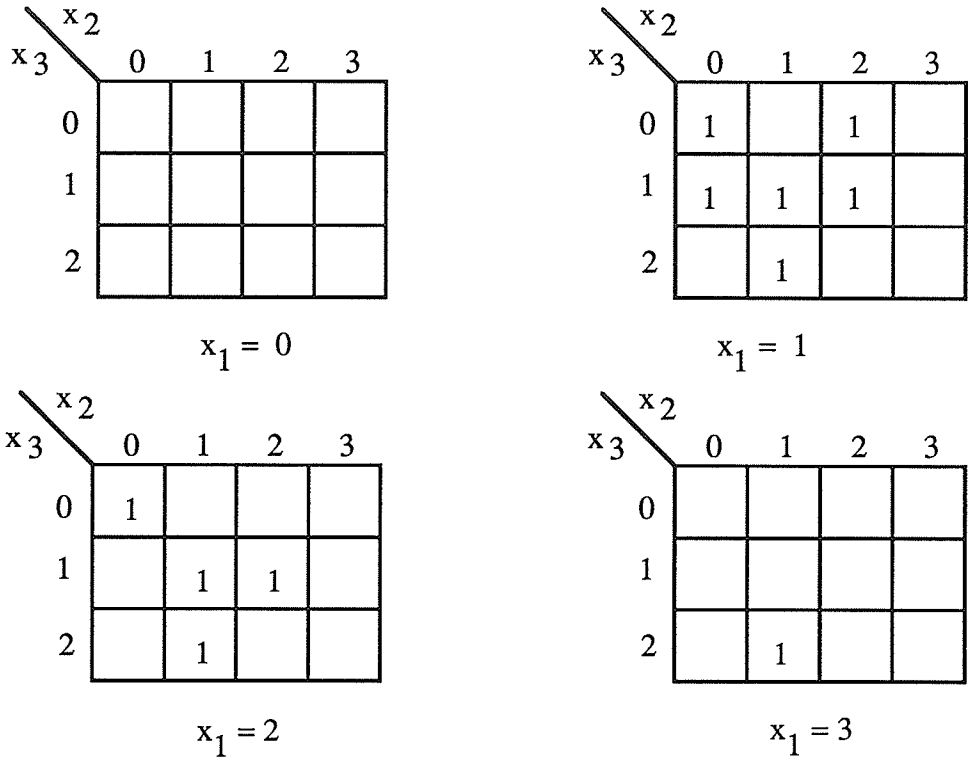


Figure 4.1 Sample function used in Example 4.4.

ON-set	EAV	EAVW
x_1 x_2 x_3	x_1 x_2 x_3	
0100 - 1000 - 100	0010 - 0010 - 010	(3,3)
0010 - 1000 - 100	0100 - 0000 - 000	(1,1)
0100 - 0010 - 100	0000 - 1000 - 010	(2,2)
0100 - 1000 - 010	0000 - 0110 - 100	(2,3)
0100 - 0100 - 010	0010 - 1010 - 001	(3,4)
0100 - 0010 - 010	0010 - 1100 - 100	(3,4)
0010 - 0100 - 010	0100 - 0010 - 001	(3,3)
0010 - 0010 - 010	0100 - 0100 - 000	(2,2)
0100 - 0100 - 001	0011 - 0000 - 010	(2,3)
0010 - 0100 - 001	0101 - 0000 - 010	(2,3)
0001 - 0100 - 001	0110 - 0000 - 000	(1,2)

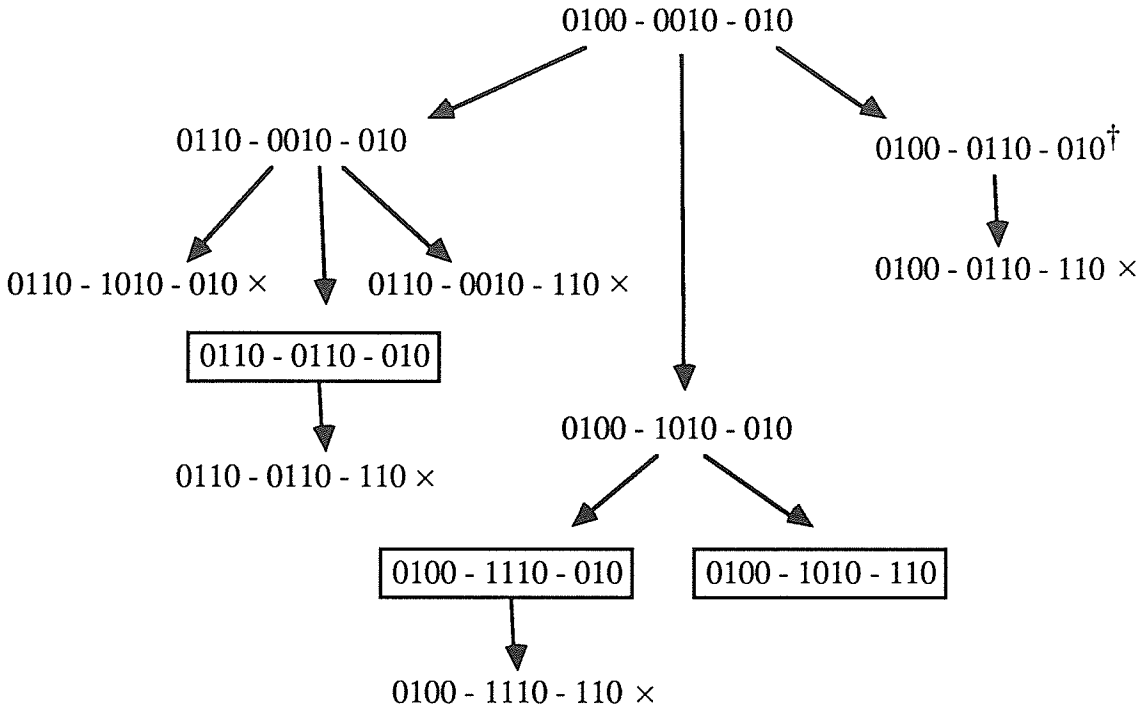
Table 4.1 Expandable adjacencies with the corresponding weight pairs (Example 4.4).

Let A and B be minterms of *f*. Let $EAVW_A = (a_1,a_2)$ and $EAVW_B = (b_1,b_2)$. (a_1,a_2) is said to be less than (b_1,b_2) if $a_1 < b_1$ or if $a_1 = b_1$ and $a_2 < b_2$. For convenience, this is denoted by $(a_1,a_2) < (b_1,b_2)$.

The general strategy of the directed search is to start at one minterm and expand it into all possible prime implicants which include it. The complete expansion of the minterm 0100 - 0010 - 010 from the function given in Figure 4.1 would yield the tree shown in Figure 4.2. Each prime implicant is enclosed in a rectangle. Branches which yield terms which are not implicants of the function are marked by (×). Clearly, these need be expanded no further.

The expansion tree for a particular minterm M is generated from left to right, depth first. Each path in the tree corresponds to a subset of the EAV for M. Generating the tree in a canonic order allows for pruning, and all possible expansions need not be tried. At each node in the tree the possible expansions are determined by examining the EAV 1 bits to the right of the 1 bit corresponding to the adjacency leading to this node. The valid possibilities are tried in order from left to right. Under this ordering, no path need be followed if its expansions are entirely contained in a path which has led to a prime implicant. The effect is clear in Figure 4.2 where there are eleven branches whereas the EAV has four 1 bits and hence fifteen non-empty subsets so that the complete expansion tree has fifteen branches. There may be as few as *s* branches in the tree, where *s* is the number of ones in the EAV. This happens when the minterm which is expanded is contained in a single prime implicant of the function, *i.e.* the prime implicant is essential.

After pruning all branches which are not implicants of the function, *i.e.* those marked by (×) in Figure 4.2, most leaf nodes are prime implicants. Unfortunately, there are exceptions to this rule. The rightmost branch in Figure 4.2 shows a leaf node which is not a prime implicant.



† 0100 - 0110 - 010 is not a prime implicant since it is contained in 0100 - 1110 - 010.

Figure 4.2 Expansion tree for minterm 0100 - 0010 - 010.

The choice of the minterm in Figure 4.2 is not the best one possible, since we don't know if any of the prime implicants generated are essential. It is usually more profitable to start at a minterm which has a low EAVW. For example, the two minterms 0010 - 1000 - 100 and 0001 - 0001 - 001, with EAVW (1,1) and (1,2) respectively, yield the trees shown in Figure 4.3. Both minterms expand into a unique prime implicant. Therefore both prime implicants are essential — they will be part of the final cover. Once an essential prime implicant is determined, all minterms it covers can be changed to don't-cares since including them in the rest of the solution is optional.

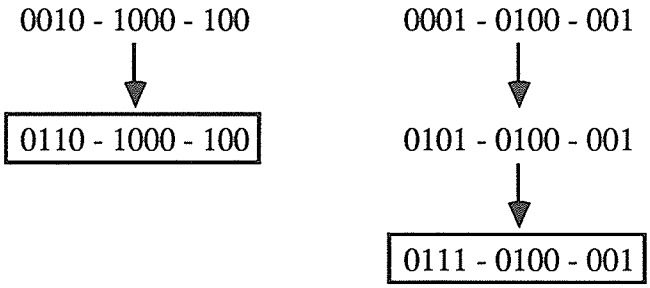


Figure 4.3 Expansion trees for two minterms.

Next the two minterms with EAVW equal to (2,2) are expanded (Figure 4.4). A minterm is only considered for expansion if it is still in the ON-set *i.e.* it has not been covered so far.

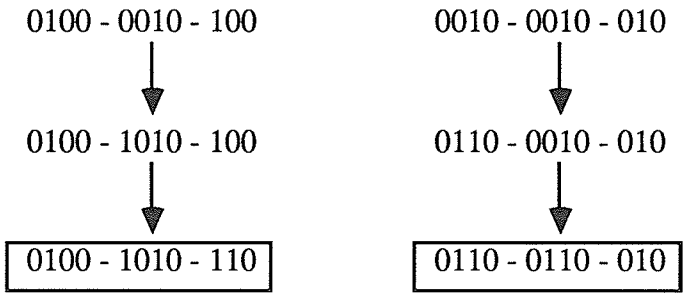


Figure 4.4 Expansion trees for minterms 0100 - 0010 - 100 and 0010 - 0010 - 010.

Again both prime implicants are essential. The four prime implicants identified in Figure 4.3 and 4.4 cover the entire function. Unfortunately, not all functions can be covered by essential prime implicants.

A prime implicant will, in general, cover a set of minterms from the ON-set and a set of minterms from the DC-set. A prime implicant including only minterms from the DC-set is always dominated (and always discarded). An advantage of the directed search approach is that dominated prime implicants are identified during prime implicant generation and no prime implicant table is required for this purpose.

A detailed description of the new directed search algorithm follows:

Algorithm: *DSA-MV Directed search for multiple-valued input functions with a single binary output*

1) Find the EAVs and EAVWs for all minterms in the ON-set.
2) For each minterm with an EAVW < $(1,\infty)$ find the prime implicant which covers it. These prime implicants must be essential and can therefore be added to the solution. Remove all covered minterms from the ON-set and include them in the DC-set.
3) If the ON-set is empty, then stop; else select the minterm with the lowest EAVW from the ON-set. Form the pruned expansion tree from this minterm.
4) Remove any dominated prime implicants.
5) If there is only one prime implicant in the expansion tree, then add it to the solution, adjust the ON-set and DC-set accordingly, and go to step 3.
6) Select a minterm which meets the following criteria:
   - it is in the ON-set;
   - it has not been expanded;
   - it is covered by some prime implicant in a previous expansion tree.

   If several minterms meet the above criteria then select the one with the lowest EAVW. If no such minterm exists go to step 9, else build a new pruned expansion tree from the selected minterm.
7) Remove all dominated prime implicants from all current expansion trees. If no prime implicants remain to be considered, go to step 3.
8) If any expansion tree contains a single prime implicant and the minterm which originated the tree is still in the ON-set, add the prime implicant to the solution, adjust the ON-set and the DC-set accordingly, and go to step 7; else go to step 6.
9) A cycle exists. Select the prime implicant which covers the most minterms in the ON-set, add it to the solution and go to step 7.

Step 6 requires some explanation. At this point, the algorithm has expanded one (or more) minterms and found that none are covered by a single essential or pseudo-essential prime implicant (one that becomes the only prime implicant covering an expanded minterm after dominated prime implicants are removed). The idea of step 6 is to further expand a minterm encountered in a previous expansion in the hope of finding that the minterm is covered by either an essential or a pseudo-essential prime implicant. Care must be taken not to generate any part of the parent expansion tree from which the minterm is selected, but this is straightforward due to the canonic order in which the trees are generated.

The way in which a cycle is resolved in step 9 is simplistic. Selecting the prime implicant which covers the largest number of minterms does not always result in a minimal cover. If a minimal solution is required, this step must be replaced by a more sophisticated algorithm. For example, McCluskey describes an algorithm to resolve cycles in [MCC56].

In order to detect a cycle, all minterms which are covered by the cycle must be expanded. Expansion of the final minterms in this process is unlikely to find any new prime implicants. Hence, it seems a reasonable heuristic to halt the expansion process when a "substantial" number of minterms have been expanded and to assume the presence of a cycle. The key is to determine when to stop the expansion process. In the implementation of the algorithm, a limit was set to the number of prime implicants in the current expansion trees. Once the limit is surpassed a cycle is assumed to be present. Substantial time saving has been obtained using this heuristic. A minimal cover is not guaranteed.

4.3 EXAMPLES

Example 4.5. Consider the 4-valued function with three input variables shown in Figure 4.5. The ON-set with the expandable adjacency vectors and the corresponding EVAW is shown in Table 4.2. Two minterms (0100 - 0100 - 0001 and 0001 - 0010 - 0100) have a

EVAW of (2,4). Minterm 0100 - 0100 - 0001 is arbitrarily selected to be expanded first. The expansion tree results in two prime implicants as shown in Figure 4.6 (expanded minterms are underlined). PI2 is deleted, since it is dominated by PI1. Therefore, PI1 is pseudo-essential and is added to the solution.

Minterm 0001 - 0010 - 0100 is expanded next. As shown in Figure 4.7 it yields two prime implicants. Minterm 0001 - 0001 - 0010 is the candidate for the next expansion. The expanded minterm is contained in the prime implicants PI3 and PI5. PI3 is chosen to be part of the solution, since it dominates PI5. The pruned expansion tree now contains only one prime implicant (PI4). PI4 does not contain any minterm which has been expanded and is still in the ON-set. Therefore, step 6 must be executed again. The expansion of 0001 - 0010 - 0001 yields 3 new prime implicants (PI6, PI7, and PI8). PI7 is chosen to be part of the solution, since it dominates all other prime implicants in the expansion tree. The expansion tree is now empty.

Finally, minterm 1000 - 1000 - 1000 is expanded (Figure 4.9). PI9 dominates PI10 and is therefore added to the solution. All minterms in the ON-set are covered. The solution includes the prime implicants PI1, PI3, PI7, and PI9. Formally, the solution is given by the expression:

$$F(x_1,x_2,x_3) = x_1^{1,2}\, x_2^{0,1,2}\, x_3^{3} + x_1^{3}\, x_2^{2,3}\, x_3^{1,2} + x_1^{1,2,3}\, x_2^{2}\, x_3^{0,3} + x_1^{0,1,2}\, x_2^{0,2}\, x_3^{0}$$

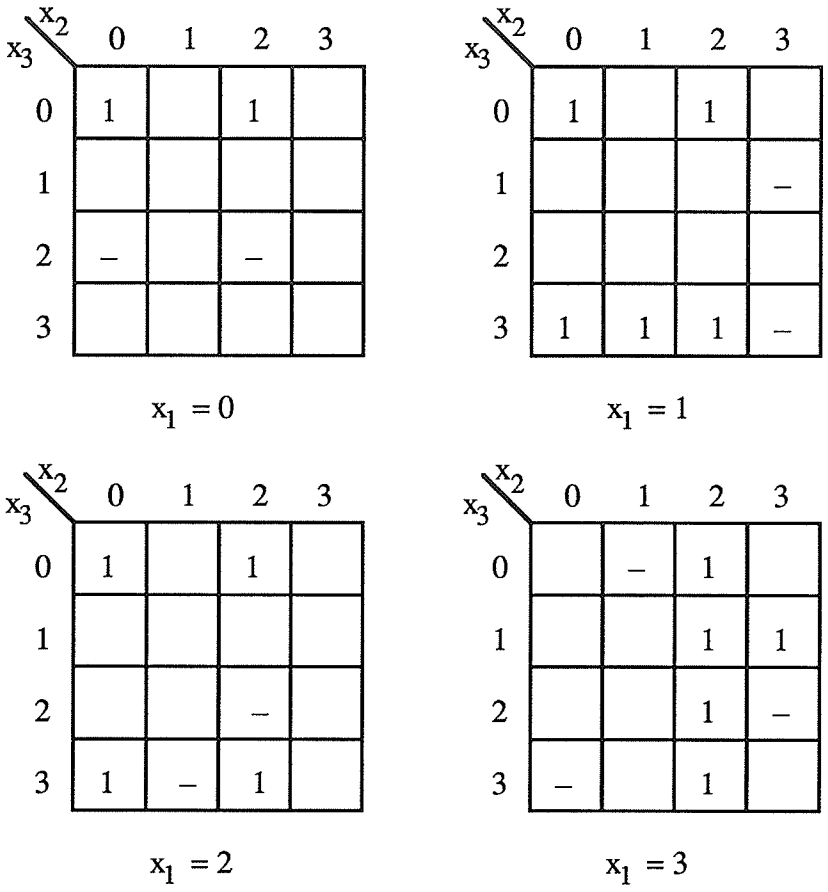


Figure 4.5 Map of the function used in Example 4.5.

ON-set			EAV			EAVW
x_1	x_2	x_3	x_1	x_2	x_3	
1000	1000	1000	0110	0010	0010	(3,4)
1000	0010	1000	0111	1000	0010	(3,5)
0100	1000	1000	1010	0010	0001	(3,4)
0100	1000	0001	0011	0111	1000	(3,6)
0100	0100	0001	0010	1011	0000	(2,4)
0100	0010	1000	1011	1000	0001	(3,5)
0100	0010	0001	0011	1101	1000	(3,6)
0010	1000	1000	1100	0010	0001	(3,4)
0010	1000	0001	0101	0110	1000	(3,5)
0010	0010	1000	1101	1000	0011	(3,6)
0010	0010	0001	0101	1100	1010	(3,6)
0001	0010	1000	1110	0100	0111	(3,7)
0001	0010	0100	0000	0001	1011	(2,4)
0001	0010	0010	1010	0001	1101	(3,6)
0001	0010	0001	0110	1000	1110	(3,6)
0001	0001	0100	0100	0010	0010	(3,3)

Table 4.2 Expandable adjacencies with the corresponding weight pairs (Example 4.5).

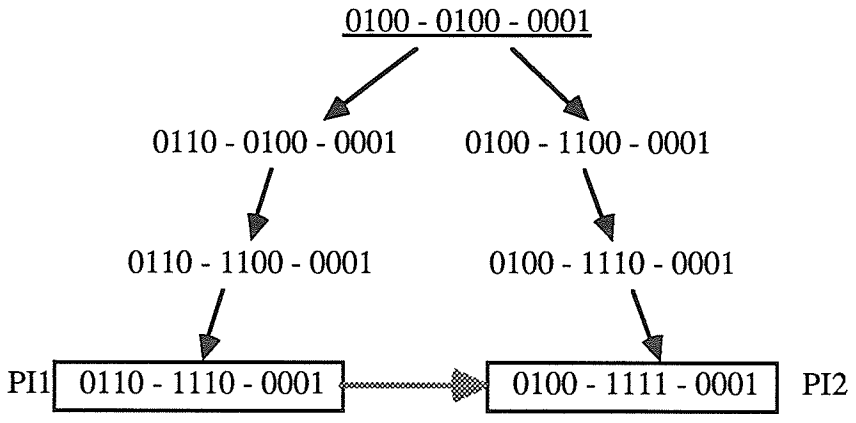


Figure 4.6 The expansion tree for minterm 0100 - 0100 - 0001 (Example 4.5).

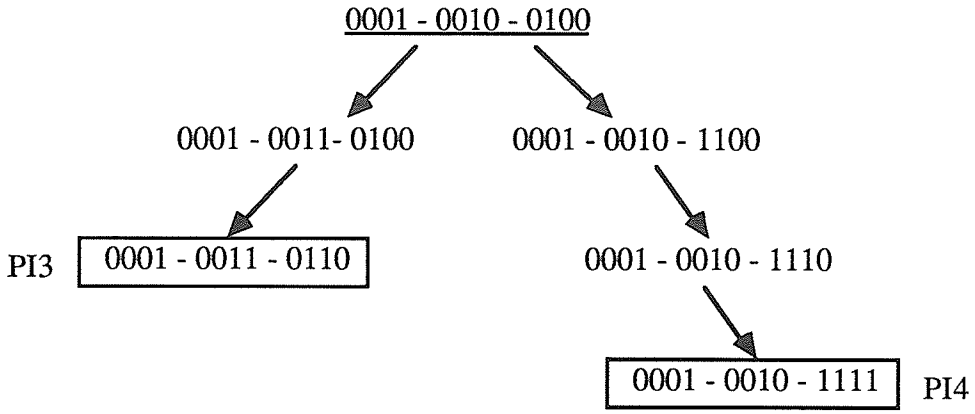


Figure 4.7 A pruned expansion tree.

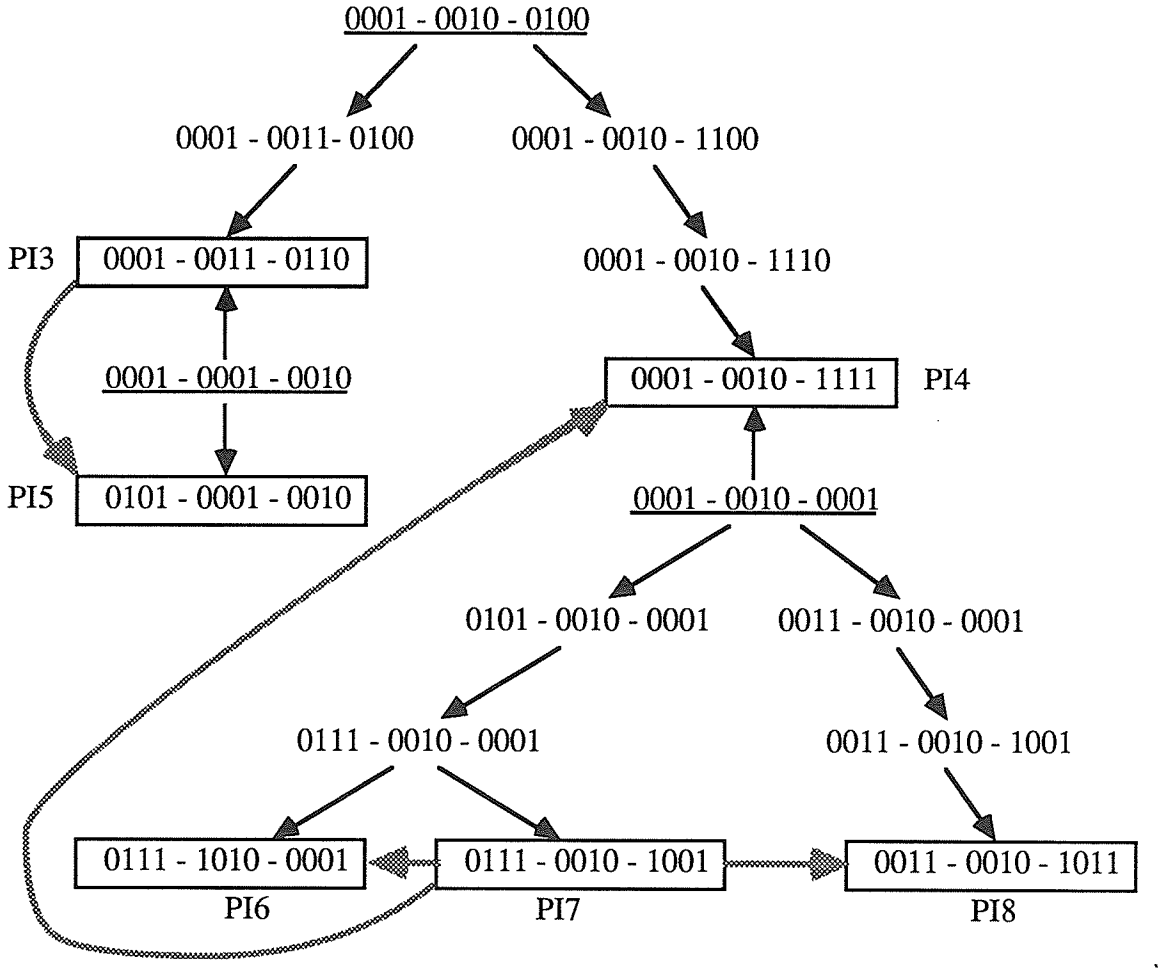


Figure 4.8 A pruned expansion tree.

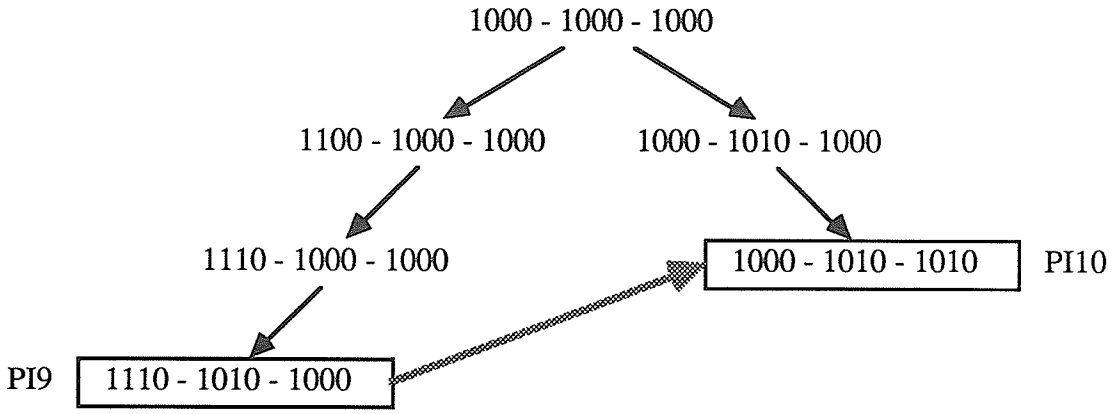


Figure 4.9 The expansion tree for minterm 1000 - 1000 - 1000 (Example 4.5).

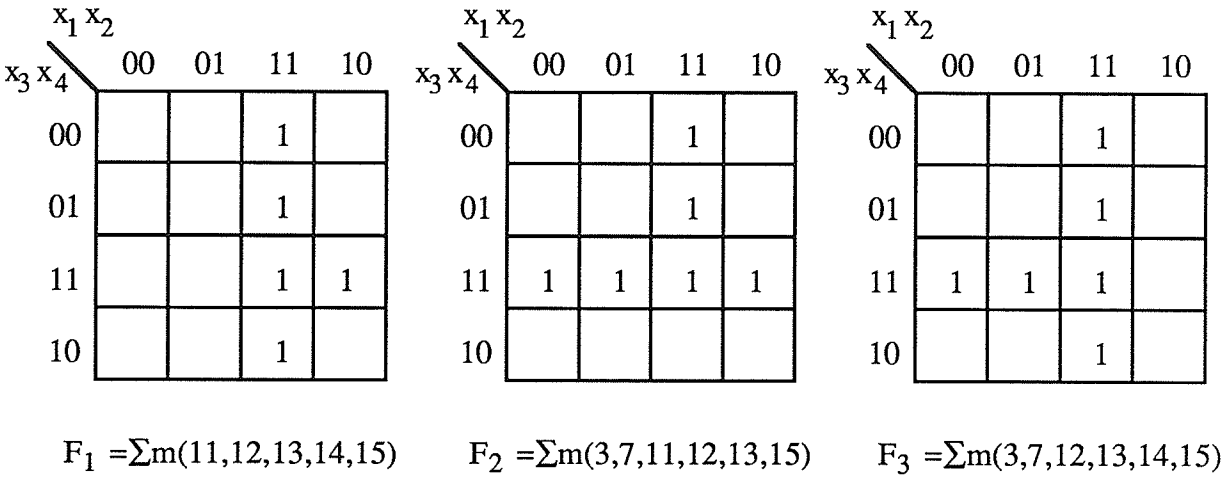


Figure 4.10 A binary multiple output problem.

Example 4.6. Consider the binary multiple output problem shown in Figure 4.10 [SER84]. First, the three functions are mapped into one multiple-valued function (Table 4.3). The ON-set with the expandable adjacency vectors and the corresponding EVAWs is also shown in Table 4.3. Minterms are expanded in the order shown in Table 4.4. Each of the four minterms results in an essential prime implicant. The minimal sum of products for the three functions are:

$$F_1(x_1,x_2,x_3,x_4) = x_1 x_3 x_4 + x_1 x_2 \bar{x}_3 + x_1 x_2$$

$$F_2\ (x_1,x_2,x_3,x_4) = x_1 x_3 x_4 + \bar{x}_1 x_3 x_4 + x_1 x_2 \bar{x}_3$$

$$F_3\ (x_1,x_2,x_3,x_4) = \bar{x}_1 x_3 x_4 + x_1 x_2 \bar{x}_3 + x_1 x_2$$

The sum of product expression for F_1 contains a redundant term ($x_1 x_2 \bar{x}_3$) . The cost of a PLA implementation will not decrease if the redundant term is removed since this term is required in F_1 and F_2. The solution in [SER84] is similar to the one given above. However, Serra uses a different ranking for the minterm expansion. The first two minterms are expanded as shown in this example. Minterm 15, which is included in all three functions, is expanded next. This expansion results in four prime implicants. Further expansions are needed to find the essential ones. The weight ranking presented in this chapter is more efficient, since only essential prime implicants are generated for this example.

	minterm	ON-set					EAV					EAVW
		x_1	x_2	x_3	x_4	x_5	x_1	x_2	x_3	x_4	x_5	
	11	01	10	01	01	100	00	01	00	00	010	(2,2)
	12	01	01	10	10	100	00	00	01	01	011	(3,4)
F_1	13	01	01	10	01	100	00	00	01	10	011	(3,4)
	14	01	01	01	10	100	00	00	10	01	001	(3,3)
	15	01	01	01	01	100	00	10	10	10	011	(4,5)
	3	10	10	01	01	010	01	01	00	00	001	(3,3)
	7	10	01	01	01	010	01	10	00	00	001	(3,3)
F_2	11	01	10	01	01	010	10	01	00	00	100	(3,3)
	12	01	01	10	10	010	00	00	00	01	101	(2,3)
	13	01	01	10	01	010	00	00	01	10	101	(3,4)
	15	01	01	01	01	010	10	10	10	00	101	(4,5)
	3	10	10	01	01	001	00	01	00	00	010	(2,2)
	7	10	01	01	01	001	01	10	00	00	010	(3,3)
F_3	12	01	01	10	10	001	00	00	01	01	110	(3,4)
	13	01	01	10	01	001	00	00	01	10	110	(3,4)
	14	01	01	01	10	001	00	00	10	01	100	(3,3)
	15	01	01	01	01	001	10	00	10	10	110	(4,5)

Table 4.3 Expandable adjacencies with the corresponding weight pairs (Example 4.6).

	minterms	prime implicants
1)	01 - 10 - 01 - 01 - 100	01 - 11 - 01 - 01 - 110
2)	10 - 10 - 01 - 01 - 001	10 - 11 - 01 - 01 - 011
3)	01 - 01 - 10 - 10 - 010	01 - 01 - 10 - 11 - 111
4)	01 - 01 - 01 - 10 - 100	01 - 01 - 11 - 11 - 101

Table 4.4 Expansion of minterm with the corresponding prime implicants (Example 4.6).

The algorithm was applied to each example presented by Serra [SER84]. These are all binary examples. Most are multiple-output problems. In each case, the identical solution, or one with the same number of product terms, was found by the DSA-MV

algorithm. The algorithm presented here is simpler than Serra's algorithm which employs complex heuristics to accommodate multiple outputs.

4.4 RESULTS

A prototype program has been implemented in Pascal and executed on an Amdahl 580. As a benchmark, the algorithm has been applied to certain two-valued multiple-output functions. Execution times are quoted to show the increase in complexity with respect to the increase in the number of inputs/outputs and the number of prime implicants of the function.

Let ADDn be the function which adds 2 n-bit numbers and produces a result of n + 1 bits. This function is described in [BRA84] as being particularly time consuming to minimize. Multiplication has also proven to be an interesting function. SQR6 is a 6-input function where the 12 outputs are the square of the inputs. SYM9 is a 9-input, 1-output, function which is equal to 1 if and only if the number of 1's in the input is 3, 4, 5, or 6. SYM9 has 1680 prime implicants, none of which are essential. Table 4.5 summarizes the results obtained in the minimization of the ADDn, MULT4, SQR6, and SYM9 function. By limiting the maximum number of prime implicants generated before assuming the presence of a cycle, it is possible to cut the minimization time of some functions, in half without increasing the size of the final result. Unfortunately, it is generally very difficult to determine the optimal value for the limiting number of prime implicants.

The performance of the DSA-MV algorithm could not be compared directly with other algorithms on the same computer. At the time this research was conducted, the source code for the Espresso algorithms [RUD87] was not available. While source code for the McBoole algorithm [DAG86] was available, time did not permit the required conversions to run it on the Amdahl. An empirical comparison of the DSA-MV algorithm with McBoole[DAG86] and Espresso-MV[RUD87] is underway.

According to the timing given in [DAG86] Espresso IIC required 83.0 sec to minimize the function ADD4, whereas McBoole required 27.0. One must keep in mind that the time measurements were taken on a VAX 750, which is about a factor of twenty slower than an Amdahl 580. The speed of our algorithm is therefore of the same order as McBoole and Espresso IIC. All three algorithms obtained the same number of terms in the final solution.

Execution time for the minimization of the MULT4 function was 305.7 and 859.1 for Expresso IIC and McBoole respectively. The solution found by McBoole contained 124 terms, and the solution found by Espresso IIC contained 133 terms. According to one referee of [DUE88] the minimum number of terms is 121. DSA-MV produced solutions with 123 to 133 terms. It is somewhat disturbing that the best solution was not found by giving our algorithm the highest bound on the number of prime implicants that could be generated before a cycle was assumed.

A comparison of the results obtained from Espresso, McBoole and DSA-MV is shown in Table 4.6. It shows that the results obtained by the DSA-MV algorithm are indeed very good. Unfortunately, it is only possible to compare the sizes of the solutions, because details of the solutions produced McBoole an Espresso were not published.

function	n	m	cpu time sec.	terms	limit on number of prime imp.
ADD3	6	4	0.31	31	nil
ADD3	6	4	0.27	31	12
ADD4	8	5	6.12	75	nil
ADD4	8	5	3.45	75	100
ADD4	8	5	2.63	75	50
ADD4	8	5	2.20	78	25
ADD5	10	6	148.32	171	nil
ADD5	10	6	116.09	171	200
ADD5	10	6	74.95	174	100
MULT4	8	8	70.71	127	nil
MULT4	8	8	63.11	128	500
MULT4	8	8	51.30	127	400
MULT4	8	8	36.84	123	300
MULT4	8	8	27.07	127	200
MULT4	8	8	15.01	133	100
SQR6	6	12	11.26	50	nil
SQR6	6	12	3.82	48	100
SYM9	9	1	180.93	84	1000
SYM9	9	1	62.02	89	500
SYM9	9	1	20.93	100	100

Notes:

a) n is the number of inputs — m is the number of outputs;

b) c.p.u. time is for optimized PVS Pascal on an Amdahl 580 (NOCHECK option specified);

c) the number of terms is the number of prime implicants in the solution;

d) the limit on the number of prime implicants as applied to the currently active expansion trees.

Table 4.5 Results produced by an implementation of DSA-MV.

Function	Espresso-MV -fast	-exact	McBoole	DSA	minimum†
SQR6	51	50	49*	48	47
MULT4	131	128	124**	123	121
ADD5	167	167		171	167
SYM9	88	84		84	84
SYM10	231	210		210	?

† Minima for these functions were supplied by one of the referees of [DUE88].

* Branching abandoned after 6 nested cycles.

** Branching abandoned after 8 nested cycles.

Table 4.6 Comparisons of Espresso, McBoole, and DSA-MV.

4.5 REMARKS

It has been shown that the directed search algorithm can be extended to accommodate multiple-valued input variables. Essential and pseudo-essential prime implicants are detected during the generation of prime implicants. The benefit of this early detection is that typically not all prime implicants are generated. A heuristic was presented which speeds up the algorithm by limiting the number of prime implicants generated before one is chosen to appear in the solution.

The algorithm is suitable for the minimization of medium size functions. Functions with up to 10 binary input variables and 6 outputs can be handled in reasonable time. The current implementation is not appropriate for minimization problems with a much larger number of variables. The DSA-MV algorithm has some clear advantages:

1) It is simple. The principles which underlie each step are easy to understand.
2) The ranking of minterms together with the use of heuristics eliminate the need for generating all prime implicants.
3) It can minimize multiple-valued input functions.

4) By applying suitable translations, it can be used to minimize multiple-valued functions as well as multiple-output functions.

The examples shown in the previous section suggest that simple heuristics can produce reasonable results. More investigation is required to determine if the presented heuristics are optimum. There may be a better way to limit the growth of the expansion tree. A number of alternatives are under investigation.

Chapter 5

RCM: A RECURSIVE CONSENSUS MINIMIZATION ALGORITHM

5.1 OVERVIEW

A major drawback of the directed search minimization algorithm is that it must start with a list of minterms — a disadvantage that it shares with the Quine-McCluskey method. If the function is specified as a list of product terms, where each product term covers more than a single minterm, all minterms must be generated. This is particularly regrettable if the function is given as a minimal or near-minimal sum-of-products expression.

The iterated consensus algorithm (described in Section 2.2.1) improved on the Quine-McCluskey method by starting with a set of product terms. The new recursive consensus minimization algorithm presented in this chapter combines the advantages of the directed search and the iterated consensus.

The traditional iterated consensus algorithm [QUI55] is only concerned with the generation of prime implicants — the selection of a minimal cover is independent of the prime implicant generation. However, during this process valuable information about the relationship between cubes can be obtained. In the algorithm presented here, intersecting terms are detected during the generation of consensus terms. A list containing all intersecting terms is associated with each product term.

With this additional information, essential and pseudo-essential prime implicants can be recognized early. This in turn simplifies the choice of a minimal cover. Unfortunately, the cyclic problem must still be solved by algebraic or heuristic means. Nevertheless, the minimization of a cyclic function is simplified with the additional information kept along with each prime implicant.

5.2 PRELIMINARIES

Definition 5.1. The *distance* between two product terms is the number of variables which appear complemented in one term and uncomplemented in the other.

Example 5.1. Consider the function $F(x_1,x_2,x_3,x_4) = \sum m(0,1,3,4,5,6,11,14,15)$. A partial list of pairs of product terms and their distances are shown in Table 5.1.

product terms		distance
$\bar{x}_1\bar{x}_3$	$\bar{x}_1\bar{x}_2x_4$	0
$x_1x_3x_4$	$x_2x_3\bar{x}_4$	1
$\bar{x}_1x_2\bar{x}_3$	$x_1\bar{x}_2x_3x_4$	3

Table 5.1 Product terms and their distance.

Lemma 5.1. Let P and Q be two product terms of the function F. There exists a minterm which is in P, and also in Q, if and only if the distance between P and Q is zero.

Proof. Let P and Q be at distance zero. PQ contains at least one minterm since no variable that appears complemented in P appears uncomplemented in Q and no variable that appears uncomplemented in P appears complemented in Q.

Let α be a minterm such that P contains α and Q contains α. Let M be the product term that covers α only. The literals involved in P must be a subset of the literals contained in M. The same is true for the literals of Q. Therefore, there is no variable that appears complemented in one term and uncomplemented in the other — the distance between P and Q is zero. Q.E.D.

Example 5.2. Consider the function given in Example 5.1. The distance between the product terms $\bar{x}_1\bar{x}_3$ and $\bar{x}_1\bar{x}_2x_4$ is zero and their intersection is $\bar{x}_1\bar{x}_2\bar{x}_3x_4$. The

corresponding Karnaugh map is shown in Figure 5.1.

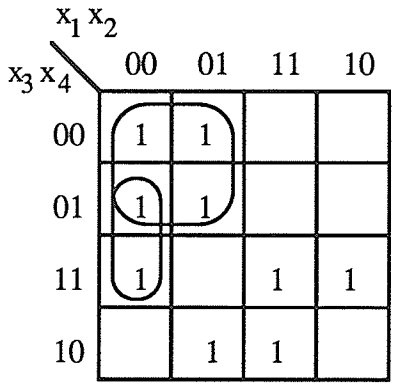


Figure 5.1 Two product terms at distance zero.

Definition 5.2. The *consensus* between two product terms Px_i and $Q\bar{x}_i$ with distance equal to one is defined to be PQ.

Example 5.3. Consider the function given in Example 5.1. The consensus term of $x_1x_3x_4$ and $x_2x_3\bar{x}_4$ is $x_1x_2x_3$. The two product terms and their consensus term are shown in Figure 5.2.

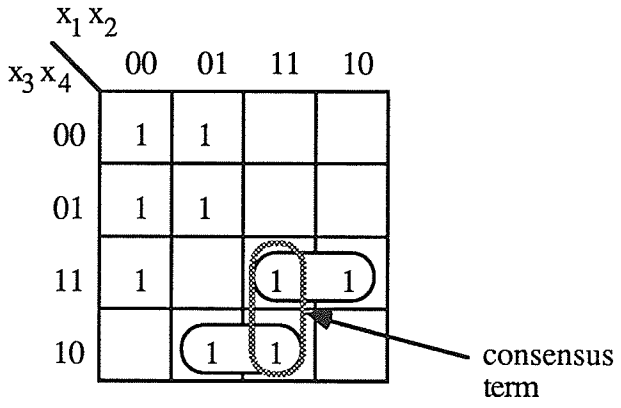


Figure 5.2 Two product terms and their consensus.

Definition 5.3. The *sharp* operation of two product terms P and Q of F, denoted

$P \# Q$, is defined to be $P\overline{Q}$.

Example 5.4.

$$\bar{x}_1\bar{x}_3 \# \bar{x}_1\bar{x}_2x_4 = \bar{x}_1\bar{x}_3\overline{(\bar{x}_1\bar{x}_2x_4)} = \bar{x}_1x_2\bar{x}_3 + \bar{x}_1\bar{x}_3\bar{x}_4$$

Lemma 5.2. Let P and Q be product terms of the function F such that the distance between them is greater than zero. $P \# Q = P$.

Proof. The intersection of P and Q is empty (Lemma 5.1). Therefore, $\overline{Q}$ contains all minterms which are contained in P. Q.E.D.

5.3 THE BINARY ALGORITHM

The recursive consensus minimization (RCM) starts with two lists of cubes (product terms) of the function to be minimized. The first list covers all minterms in the ON-set plus all minterms in the DC-set and a second list covers all minterms in the DC-set. The two lists are called OnList and DcList, respectively. RCM differs from the traditional iterated consensus [QUI55] in two ways:

1) the order in which cubes are selected for expansion;
2) the information retained on the intersection between cubes.

The new ordering helps to detect prime implicants early in the generation process. The information on the interaction between cubes allows for an immediate detection of essential and pseudo-essential prime implicants.

The algorithm will, for simplicity and brevity, be described in terms of PASCAL-like pseudo-code. Each cube is stored in the following data structure:

```
Cube = record
        Zero : set of [1..N];          { complemented variables }
        One  : set of [1..N];          { uncomplemented variables }
        Kind : (Deleted,PICandidate,Implicant);
        PIKind : (Essential, NonEssential);
```

```
DCFlag : Boolean;          { true if it is a don't-care cube }
Intersections : LinkPtr;   { pointer to a list of intersecting cubes}
Next : CubePtr;            { pointer to next cube in the list }
end;
```

The subscript of a complemented variable appears only in set Zero. The subscript of an uncomplemented variable appears only in set One. The subscript of a missing variable will appear in both sets. Initially, the Kind of each cube will be Implicant. A cube becomes a PICandidate (prime implicant candidate) if it is thought to be a prime implicant. A particular ordering and a special structure of the cubes, in the OnList, results in the marking of a cube as a PICandidate when it is not a prime implicant. This phenomenon will be explained later. The PIKind field in the Cube record is only meaningful if the Kind of the cube is PICandidate. A cube marked Essential is always a prime implicant (see Lemma 5.3 below). The information about intersecting cubes is kept in the Intersections list. Each Link in the Intersections list is stored in the following data structure:

```
Link = record
    InterCube : CubePtr;   { Intersecting cube }
    Next  : LinkPtr;       { next Link in the list }
    end;
```

Example 5.5. Let F be a function with 6 input variables. The cube $\bar{x}_2 x_4 \bar{x}_5$ will be stored in the cube C as follows:

```
C.Zero ← [1,2,3,5,6]
C.One ← [1,3,4,6]
C.Kind ← Implicant
C.PIKind ← NonEssential
C.DCFlag ← false
C.Intersections ← Nil
```

Each cube is linked with its intersecting cubes. Cubes are linked on two different occasions. If two cubes are found to be at distance zero, and one cube is not a subset of

the other, then they are linked. If two cubes have a consensus, the consensus cube is linked to each of its parents (unless the parent is deleted). This information is useful when checking for essential prime implicants. Only intersecting cubes need to be sharped with the prime implicant in question. Furthermore, if dominated prime implicants of Q are to be removed, they must intersect with Q. Since all these cubes are on the intersection list, there is no need to check all prime implicants in the OnList.

Cubes which cover only don't-care minterms must be placed at the end of the OnList. They are not expanded, but they must be used in the expansion of all other cubes in the OnList. Once all implicants in OnList have been expanded, OnList contains only prime implicants (except the cubes which cover only don't-care minterms, since they are not expanded). All essential and pseudo-essential prime implicants are marked Essential. Not all prime implicants of the function will be in the OnList, since dominated cubes will have been deleted during the process.

The procedure GeneratePIs and Expand identify candidate prime implicants. GeneratePIs invokes Expand repeatedly.

```
Procedure GeneratePIs(OnList);
for each Cube in the OnList
    if Cube is not marked Deleted and Cube is not in the DcList then
        Expand(OnList,Cube)    { Cube can also be seen as the first element in a list }
end for
end GeneratePIs

Procedure Expand (OnList,CubeList)
FirstCube ← first element in the CubeList
CubeElement ← second element in the CubeList
while (CubeElement ≠ nil) and (FirstCube is not deleted) do
    Distance ← distance between CubeElement and FirstCube
    if Distance = 0 then
        IntersectionCube ← intersection of FirstCube and CubeElement
        if IntersectionCube = CubeElement then
```

```
                delete CubeElement
            else if IntersectionCube = FirstCube then
                delete FirstCube
            if CubeElement and FirstCube are not deleted then
                LinkCubes(CubeElement, FirstCube)
        else if Distance = 1 then
            ConsensusCube ← consensus of FirstCube and CubeElement
            if ConsensusCube is not a subset of any cube in the OnList then
                remove all subsets of ConsensusCube from the OnList
                mark ConsensusCube as Implicant
                insert ConsensusCube to the front of OnList
                if FirstCube is not deleted then
                    LinkCubes(ConsensusCube, FirstCube)
                if CubeElement is not deleted then
                    LinkCubes(ConsensusCube, CubeElement)
                Expand(OnList)
end while
if FirstCube has not been deleted then
    mark FirstCube as PICandidate
    CheckEssential(FirstCube)
end Expand
```

The procedure Expand finds the consensus cubes between the first cube and the remaining (undeleted) cubes in CubeList. CubeList is a sub-list of OnList. Intersecting cubes (cubes at distance zero) must be checked to determine if one cube completely covers the other (*i.e.* the intersection is equal to one of the cubes); if this is the case, the covered cube must be deleted, otherwise the cubes must be linked to each other. If a consensus cube is found which is not already in the list, then this new cube is added to the front of the list and the Expand procedure is called recursively. The recursive expansion of the newly created cube enables the early recognition of prime implicants. Furthermore, it can be determined if a prime implicant is essential.

The procedure LinkCubes allocates a Link for each cube. The Link contains a pointer

to the other cube and is inserted in the corresponding intersection list.

If FirstCube has not been deleted during the expansion process it is most likely a prime implicant. The consensus of FirstCube and any cube at distance one from it has also been expanded. This will ensure, in most cases, that the prime implicant which includes FirstCube is found during the expansion. The only case in which FirstCube is not a prime implicant occurs when all cubes that are part of the prime implicant which includes FirstCube intersect with it. This is illustrated in Example 5.6. This causes no concern since the cube will be deleted later on when the prime implicant is generated.

Example 5.6. Consider the four-variable function shown in Figure 5.3. The OnList is given as follows (using the positional notation introduced in Chapter 2):

$-1-1$ FirstCube
$0\,1\,0-$
$1\,1\,0-$
$-1\,1-$

FirstCube is at distance zero from all other cubes, and therefore no consensus term is generated. Clearly FirstCube is not a prime implicant since it is part of the cube $-1--$.

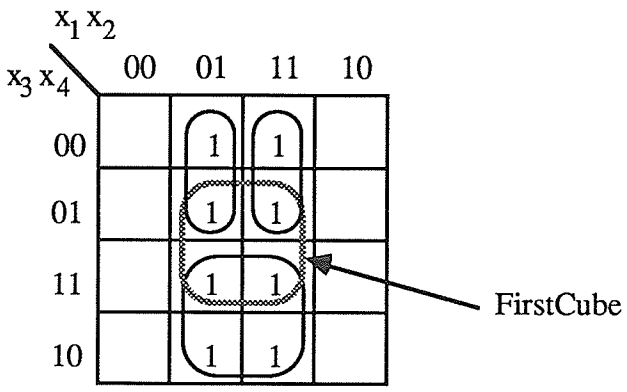


Figure 5.3 Function used in Example 5.6.

The pseudo-code for the procedure to determine whether or not a cube is essential is given below.

Procedure CheckEssential (PI: Cube)
delete all cubes in the OnList which are dominated by PI
NotDCList ← Sharp(PI, DCList)
UniqueList ← Sharp(NotDCList, PI.Intersections)
if UniqueList is not empty then
    PI is essential
    insert PI in the DCList
    if a cube linked to PI is dominated by a nonessential prime implicant PI* then
        CheckEssential(PI*)
end CheckEssential

The CheckEssential procedure is fairly straightforward. Cubes dominated by PI are deleted. All cubes that intersect with PI are known, since they are in the intersection list. During the expansion process, the distance between PI and every cube in the OnList was determined, and cubes at distance zero were linked. UniqueList will cover all minterms which are only covered by PI. Clearly, PI is essential if UniqueList is not empty (some minterms are only covered by PI). As pointed out earlier, a cube can be marked as a PICandidate even when it is not a prime implicant. The following lemma shows that such a cube is never marked Essential.

Lemma 5.3. A cube (PI) marked as Essential, in the CheckEssential procedure, is a prime implicant of the given function.

Proof. Assume that there exists a prime implicant Q which covers PI, and PI ≠ Q, *i.e.* assume PI is not a prime implicant. Clearly, Q is not in the OnList — otherwise PI would have been deleted. UniqueList contains a cube P which is covered by the cube Q. P has no cubes at distance 0 in the OnList (all cubes at distance 0 have been sharped with PI). There must be a cube C in the OnList at distance 1 from P. This implies that the distance between PI and C is 1. This is a contradiction, since CheckEssential is only called after all consensus cubes that can be formed with PI have been expanded. Q.E.D.

Lemma 5.4. At the termination of the procedure GeneratePIs, OnList contains only prime implicants of the function (with the exception of the don't-care cubes, which are not expanded).

Proof. Assume that OnList contains a cube Q which is not prime and which contains at least one true minterm of the function. Q is not essential, by Lemma 5.3. Therefore, Q is completely covered by other cubes in the OnList. The consensus of these cubes yields the prime implicant which includes Q. This is a contradiction, since the generation of such a cube would result in the deletion of Q. Q.E.D.

If, after applying GeneratePIs, the OnList contains any cube which is marked as NonEssential and is not in the DcList, the function is cyclic. In order to break the cycle, a cube must be chosen to be part of the solution. For convenience, such a cube will be marked Essential, even though it is not essential. This, in turn, may make other NonEssential cubes pseudo-essential. The pseudo-code for the BreakCycle procedure follows.

Procedure BreakCycle (OnList)
let CycleList be the list of all cubes C, where C ∈ OnList,
C.PIkind = NonEssential,
and C.DCFlag is false
while CycleList is not empty do
    let PI ∈ CycleList be the cube which covers the largest number of uncovered
        minterms of F
    mark PI Essential
    insert PI in the DcList
    if a cube linked to PI is dominated by a nonessential prime implicant PI* then
        CheckEssential(PI*)
end BreakCycle

5.4 BINARY EXAMPLES

The examples given in this section will clarify the steps in the RCM algorithm.

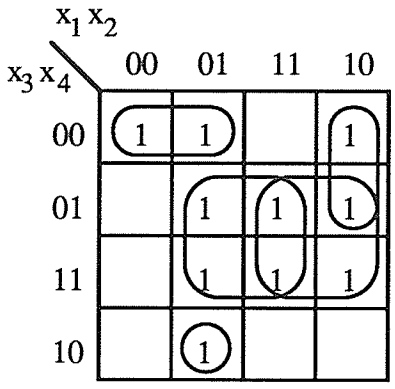


Figure 5.3 Function used in Example 5.6.

Example 5.6. Consider the function shown in Figure 5.3. The function is specified with the following five cubes:

$x_1x_2x_3x_4$

0 – 0 0

– 1 – 1

0 1 1 0

1 – – 1

1 0 0 –

The minimization would proceed as shown below. For identification purposes each cube is assigned a unique identifier (Id #). The intersections will be shown as a list of Id numbers. The initial OnList contains the following information.

Id #	cube	Kind	PIKind	Intersections
1	0 – 0 0	Implicant	NonEssential	[ ]
2	– 1 – 1	Implicant	NonEssential	[ ]
3	0 1 1 0	Implicant	NonEssential	[ ]
4	1 – – 1	Implicant	NonEssential	[ ]
5	1 0 0 –	Implicant	NonEssential	[ ]

Cube 1 is expanded first. The first cube at distance 1 is cube 2. The consensus of these cubes is added to the front of the OnList.

Id #	cube	Kind	PIKind	Intersections
6	0 1 0 –	Implicant	NonEssential	[1,2]
1	0 – 0 0	Implicant	NonEssential	[6]
2	– 1 – 1	Implicant	NonEssential	[6]
3	0 1 1 0	Implicant	NonEssential	[ ]
4	1 – – 1	Implicant	NonEssential	[ ]
5	1 0 0 –	Implicant	NonEssential	[ ]

Expand is called recursively with cube 6 as its argument. Cube 3 is at distance 1 from cube 6. Their consensus includes cube 3. Therefore, cube 3 is deleted.

Id #	cube	Kind	PIKind	Intersections
7	0 1 – 0	Implicant	NonEssential	[6]
6	0 1 0 –	Implicant	NonEssential	[1,2,7]
1	0 – 0 0	Implicant	NonEssential	[6]
2	– 1 – 1	Implicant	NonEssential	[6]
3	0 1 1 0	Deleted	NonEssential	[ ]
4	1 – – 1	Implicant	NonEssential	[ ]
5	1 0 0 –	Implicant	NonEssential	[ ]

Expand is called recursively with cube 7 as its argument. Cube 2 is at distance 1 from cube 7. Their consensus includes cubes 6 and 7. Therefore, both cubes are deleted. During this expansion it was also found that cubes 1 and 7 intersect. It turns out that this information is not very useful since cubes 6 and 7 are deleted.

Id #	cube	Kind	PIKind	Intersections
8	0 1 – –	Implicant	NonEssential	[2]
7	0 1 – 0	Deleted	NonEssential	[6,1]
6	0 1 0 –	Deleted	NonEssential	[1,2,7]
1	0 – 0 0	Implicant	NonEssential	[6,7]
2	– 1 – 1	Implicant	NonEssential	[6,8]
4	1 – – 1	Implicant	NonEssential	[ ]
5	1 0 0 –	Implicant	NonEssential	[ ]

Expand is called recursively with cube 8 as argument. Cube 8 has no useful consensus with any of the cubes in the OnList (*i.e.* all consensus cubes that can be obtained with cube 8 are already covered by a cube in the list). Cube 8 also intersects with cube 1.

Id #	cube	Kind	PIKind	Intersections
8	0 1 – –	PICandidate	NonEssential	[2,1]
1	0 – 0 0	Implicant	NonEssential	[8]
2	– 1 – 1	Implicant	NonEssential	[8]
4	1 – – 1	Implicant	NonEssential	[ ]
5	1 0 0 –	Implicant	NonEssential	[ ]

Cube 8 is now checked to determine if it is essential. The sharp of cube 8 with cubes 1 and 2 (the intersecting cubes) results in the implicant 0 1 1 0. Cube 8 is marked as Essential and is now part of the minimal cover. All minterms covered by cube 8 need not be covered again. Therefore, cube 8 can be considered a don't-care cube for the remaining steps in the minimization process. A copy of cube 8 is created and inserted in the DcList. The algorithm now returns (by unwrapping the recursion) to the expansion of cube 1. A consensus of cube 1 and 5 yields – 0 0 0.

	Id #	cube	Kind	PIKind	Intersections
	9	– 0 0 0	Implicant	NonEssential	[1,5]
dc[1]	8	0 1 – –	PICandidate	Essential	[2,1]
	1	0 – 0 0	Implicant	NonEssential	[8,9]
	2	– 1 – 1	Implicant	NonEssential	[8]
	4	1 – – 1	Implicant	NonEssential	[ ]
	5	1 0 0 –	Implicant	NonEssential	[9]

The consensus of cube 9 and 4 is 1 0 0 –, which is already in the OnList (cube 5). Cube 9 does not yield any new consensus cubes. Cube 9 dominates cube 1. Cube 1 is deleted and cube 9 is found to be essential.

[1] This cube is also in the DcList.

	Id #	cube	Kind	PIKind	Intersections
dc	9	– 0 0 0	PICandidate	Essential	[1,5]
dc	8	0 1 – –	PICandidate	Essential	[2,1]
	1	0 – 0 0	Deleted	NonEssential	[8,9]
	2	– 1 – 1	Implicant	NonEssential	[8]
	4	1 – – 1	Implicant	NonEssential	[ ]
	5	1 0 0 –	Implicant	NonEssential	[9]

The expansion of cube 2 yields no new implicants. Cubes 2 and 4 intersect. Cube 2 is not essential since the sharp with its intersecting cubes (4 and 8) is empty.

	Id #	cube	Kind	PIKind	Intersections
dc	9	– 0 0 0	PICandidate	Essential	[5]
dc	8	0 1 – –	PICandidate	Essential	[2]
	2	– 1 – 1	PICandidate	NonEssential	[8,4]
	4	1 – – 1	Implicant	NonEssential	[2]
	5	1 0 0 –	Implicant	NonEssential	[9]

No implicants are generated during the expansion of cube 4. Cubes 4 and 5 intersect. Cube 4 dominates both of its intersecting cubes (2 and 5).

	Id #	cube	Kind	PIKind	Intersections
dc	9	– 0 0 0	PICandidate	Essential	[5]
dc	8	0 1 – –	PICandidate	Essential	[2]
	2	– 1 – 1	deleted	NonEssential	[8,4]
dc	4	1 – – 1	PICandidate	Essential	[2,5]
	5	1 0 0 –	deleted	NonEssential	[9,4]

The final OnList contains only essential (cubes 4 and 8) and pseudo-essential (cube 9) prime implicants — it contains the solution to given problem.

	Id #	cube	Kind	PIKind	Intersections
dc	9	– 0 0 0	PICandidate	Essential	[ ]
dc	8	0 1 – –	PICandidate	Essential	[ ]
dc	4	1 – – 1	PICandidate	Essential	[ ]

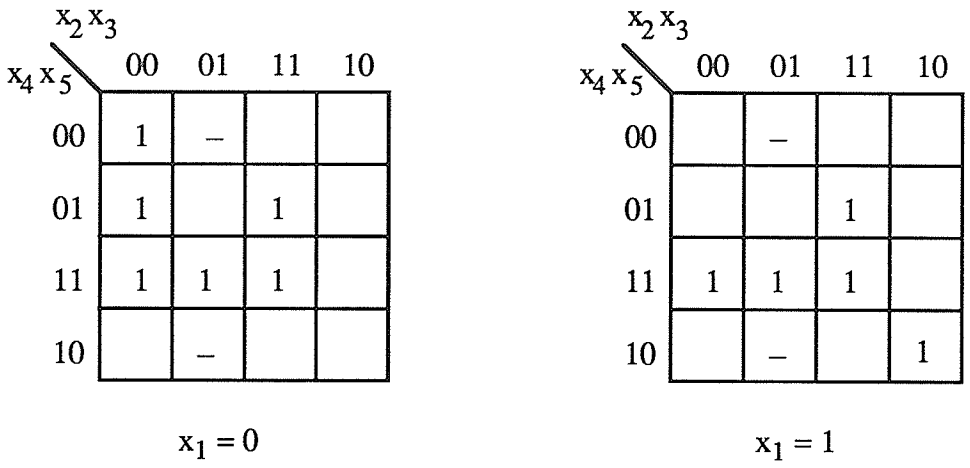


Figure 5.4 Function used in Example 5.7.

Example 5.7. Consider the 5-variable function shown in Figure 5.4. The function is specified with the following lists of cubes:

OnList	DcList
1 1 0 1 0	– 0 1 – 0
– 0 0 1 1	
– – 1 1 1	
– 1 1 – 1	
0 0 0 0 –	
– 0 1 – 0	

The DcList contains the don't-care cube. Cubes are never deleted from the DcList. A don't-care cube Q may be deleted from the OnList if a cube is generated of which Q is a subset. Don't-care cubes are not linked to intersecting cubes because they are dominated by any cube, but they should not be deleted. The initial OnList is shown below.

	Id #	cube	Kind	PIKind	Intersections
	1	1 1 0 1 0	Implicant	NonEssential	[ ]
	2	– 0 0 1 1	Implicant	NonEssential	[ ]
	3	– – 1 1 1	Implicant	NonEssential	[ ]
	4	– 1 1 – 1	Implicant	NonEssential	[ ]
	5	0 0 0 0 –	Implicant	NonEssential	[ ]
dc	6	– 0 1 – 0	Implicant	NonEssential	[ ]

Cube 1 is expanded first. No consensus can be formed. Cube 1 is found to be essential. Cube 2 is expanded next. The consensus of cube 2 and 3 (cube 7) is added to the OnList. Cube 2 is a subset of cube 7 and is therefore deleted.

	Id #	cube	Kind	PIKind	Intersections
	7	– 0 – 1 1	Implicant	NonEssential	[3]
	1	1 1 0 1 0	PICandidate	Essential	[ ]
	2	– 0 0 1 1	Deleted	NonEssential	[ ]
	3	– – 1 1 1	Implicant	NonEssential	[7]
	4	– 1 1 – 1	Implicant	NonEssential	[ ]
	5	0 0 0 0 –	Implicant	NonEssential	[ ]
dc	6	– 0 1 – 0	Implicant	NonEssential	[ ]

Cube 7 has a consensus with cube 5. Cube 8 does not have a useful consensus with any cube in the list. Cube 8 is not essential.

	Id #	cube	Kind	PIKind	Intersections
	8	0 0 0 – 1	PICandidate	NonEssential	[5,7]
	7	– 0 – 1 1	Implicant	NonEssential	[3,8]
dc	1	1 1 0 1 0	PICandidate	Essential	[ ]
	3	– – 1 1 1	Implicant	NonEssential	[7]
	4	– 1 1 – 1	Implicant	NonEssential	[ ]
	5	0 0 0 0 –	Implicant	NonEssential	[8]
dc	6	– 0 1 – 0	Implicant	NonEssential	[ ]

The expansion of cube 7 continues. Cubes 7 and 6 yield – 0 1 1 –. No useful consensus cubes can be obtained with cube 9. The expansion of cube 7 terminates. Cube 7 is

essential. It dominates cube 9.

	Id #	cube	Kind	PIKind	Intersections
	9	– 0 1 1 –	Deleted	NonEssential	[3,7]
	8	0 0 0 – 1	PICandidate	NonEssential	[5,7]
dc	7	– 0 – 1 1	PICandidate	Essential	[3,8]
dc	1	1 1 0 1 0	PICandidate	Essential	[ ]
	3	– – 1 1 1	Implicant	NonEssential	[7]
	4	– 1 1 – 1	Implicant	NonEssential	[ ]
	5	0 0 0 0 –	Implicant	NonEssential	[8]
dc	6	– 0 1 – 0	Implicant	NonEssential	[ ]

The expansion of cubes 3 and 6 yields cube 10 which is not essential. Cube 3 is not essential. Cube 3 dominates cube 10, which is deleted. The expansion of cube 4 does not yield a new cube. Cube 4 is essential, and it dominates cube 3.

	Id #	cube	Kind	PIKind	Intersections
	10	– 0 1 1 –	Deleted	NonEssential	[3,7]
	8	0 0 0 – 1	PICandidate	NonEssential	[5,7]
dc	7	– 0 – 1 1	PICandidate	Essential	[3,8]
dc	1	1 1 0 1 0	PICandidate	Essential	[ ]
	3	– – 1 1 1	Deleted	NonEssential	[4,7]
dc	4	– 1 1 – 1	PICandidate	Essential	[3]
	5	0 0 0 0 –	Implicant	NonEssential	[8]
dc	6	– 0 1 – 0	Implicant	NonEssential	[ ]

The consensus of cubes 5 and 6 yields cube 11. Cube 11 is found to be a non-essential prime implicant.

	Id #	cube	Kind	PIKind	Intersections
	11	0 0 – 0 0	PICandidate	NonEssential	[5]
dc	8	0 0 0 – 1	PICandidate	NonEssential	[5,7]
dc	7	– 0 – 1 1	PICandidate	Essential	[8]
dc	1	1 1 0 1 0	PICandidate	Essential	[ ]
dc	4	– 1 1 – 1	PICandidate	Essential	[ ]
	5	0 0 0 0 –	Implicant	NonEssential	[8,11]
dc	6	– 0 1 – 0	Implicant	NonEssential	[ ]

Cube 5 dominate both its intersecting cubes (8 and 11). Hence, cube 5 is essential and cube 8 and 11 are deleted. The minimal solution is shown below. The don't-care cube (6) has been removed.

	Id #	cube	Kind	PIKind	Intersections
dc	7	– 0 – 1 1	PICandidate	Essential	[ ]
dc	1	1 1 0 1 0	PICandidate	Essential	[ ]
dc	4	– 1 1 – 1	PICandidate	Essential	[ ]
dc	5	0 0 0 0 –	PICandidate	Essential	[ ]

5.5 RCM-MV: THE MULTIPLE-VALUED EXTENSION OF RCM

The cube notation introduced in Section 4.2 will be used to represent multiple-valued logic product terms. The definitions for *distance*, *consensus*, and the *sharp* operator, together with multiple-valued examples, are given below.

Definition 5.4. Let P and Q be two cubes of a multiple-valued function. Let S be a cube obtained as follows:

$$S = AND(P,Q)$$

where AND is the bitwise AND of the cube notation of P and Q. The *distance* of P and Q is the number of coordinates in S which contains only zeros.

Example 5.8. Table 5.2 shows some cubes of a 3-valued logic function and their corresponding distance.

P	Q	AND(P,Q)	distance
011–111–100	110–110–110	010–110–100	0
011–111–100	100–101–101	000–101–100	1
011–111–100	100–111–011	000–111–000	2

Table 5.2 Cubes and their distance.

Definition 5.5. Let $P = p_1 - p_2 - \ldots - p_n$ be a cube of a multiple-valued logic function F where p_i represents the i^{th} coordinate of P. Similarly, let $Q = q_1 - q_2 - \ldots - q_n$ be a cube of F such that the distance between P and Q is 1. Let $j \in \{1,2, \ldots ,n\}$ such that $AND(p_j,q_j)$ is equal to a bitstring consisting only of zeros. The i^{th} coordinate of the *consensus* cube $W = w_1 - w_2 - \ldots - w_n$ of P and Q can be obtained as follows:

$$w_i = \begin{cases} AND(p_i, q_i), & \text{if } i \neq j \\ OR(p_i, q_i), & \text{if } i = j \end{cases}$$

where AND and OR are the bitwise AND and OR operations respectively.

Example 5.9. Some cubes with their corresponding consensus cube are shown in Table 5.3.

P	Q	consensus of P and Q
011–111–100	100–101–101	111–101–100
001–110–101	100–101–011	101–100–001
111–010–011	110–001–111	110–011–011

Table 5.3 Product terms and their consensus.

Definition 5.6. The *sharp* operation P # Q, is defined as a sum of cubes which covers all minterms which are covered by P and not covered by Q. The procedural definition of # is given below. Consider two cubes $P = p_1 - p_2 - \ldots - p_n$ and $Q = q_1 - q_2 - \ldots - q_n$.

$P \# Q = C_1 + C_2 + \ldots + C_n$, where the cube C_i is given by

$C_1 = AND(p_1,NOT(q_1)) - p_2 - \ldots - p_n$

$C_2 = p_1 - AND(p_2,NOT(q_2)) - \ldots - p_n$

. . . . . .

$C_n = p_1 - p_2 - \ldots - AND(p_n,NOT(q_n))$

AND is the bitwise AND operation and NOT is the binary complement operation. If C_i becomes a null cube, *i.e.* one coordinate consists only of zeros, then C_i is removed from the list.

Example 5.10. Consider the cubes P = 011–111–100–101 and Q = 110–101–101–110. P # Q is given by

C_1 = 001–111–100–101

C_2 = 011–010–100–101

C_3 = 011–111–000–101 null cube (delete)

C_4 = 011–111–100–001

The data structure used to store each multiple-valued cube is slightly different from the one used for a binary cube. R is the radix of the function and N denotes the number of input-variables.

```
Cube = record
    Ones : array[0..R-1] of set of [1..N];  { ones in each coordinate }
    Kind : (Deleted,PICandidate,Implicant);
    PIKind : (Essential, NonEssential);
    DCFlag : Boolean;                       { true if it is a don't care cube }
    Intersections : LinkPtr;                { pointer to a list of intersecting cubes}
    Next : CubePtr;                         { pointer to next cube in the list }
    end;
```

Example 5.11. Let F be a 4-valued function with 3 input variables. The cube P = 0110–1111–1000 will be stored as follows:

P.Ones[0] ← [2,3]
P.Ones[1] ← [1,2]
P.Ones[2] ← [1,2]
P.Ones[3] ← [2]
P.Kind ← Implicant
P.PIKind ← NonEssential

P.DCFlag ← false
P.Intersections ← Nil

P.Ones is an array indexed by the logical value where each element is a set of the function variables.

The pseudo-code for the basic operations is given below.

```
Function Distance(CubeA,CubeB);
TempSet ← [ ]     { empty set }
Distance ← 0
for I ← 0 to (R - 1) do
      TempSet ← TempSet + CubeA.Ones[I]*CubeB.Ones[I]
end for
for I ← 1 to N do
      if I in TempSet then
            Distance ← Distance + 1
end for
end Distance

Procedure Consensus(CubeA,CubeB,CubeR);
{CubeA and CubeB are at distance 1}
let Position be the coordinate in which CubeA and CubeB don't intersect
for I ← 0 to (R - 1) do
      CubeR[I] ← CubeR[I] + CubeA.Ones[I] *CubeB.Ones[I]
      if (Position in CubeA.Ones[I]) or (Position in CubeB.Ones[I]) then
            CubeR[I] ← CubeR[I] + [Position]
end for
end Consensus

Procedure Sharp(CubeA,CubeB,Result);    {Result is a list of cubes }
Result ← empty list
if Distance(CubeA,CubeB) > 0 then
      add CubeA to Result
```

```
else
    TempSet ← [ ]    { empty set }
    for I ← 0 to (R - 1) do
        TempSet ← TempSet + (CubeA.Ones[I] - CubeA.Ones[I]*CubeB.Ones[I])
    end for
    for I ← 1 to N do
        if I in TempSet then
            let NewCube be a copy of CubeA
            for J ← 0 to (R - 1) do
                if I in CubeA.Ones[J]*CubeB.Ones[J] then
                    NewCube.Ones[J] ← NewCube.Ones[J] - [I]
            end for
            insert NewCube in the list Result
    end for
end Sharp
```

With the new definitions given above, the procedures GeneratePIs, Expand, and CheckEssential can be immediately applied as presented in Section 5.3.

5.6 MULTIPLE-VALUED EXAMPLES

Example 5.12. Consider the 4-valued function with three input variables represented by the following five cubes:

0001-0100-0010
0100-1010-1100
0110-0100-0110
0010-1000-1000
0010-0010-0100

The initial OnList contains the following information.

	Id #	cube	Kind	PIKind	Intersections
	1	0001-0100-0010	Implicant	NonEssential	[ ]
	2	0100-1010-1100	Implicant	NonEssential	[ ]
	3	0110-0100-0110	Implicant	NonEssential	[ ]
	4	0010-1000-1000	Implicant	NonEssential	[ ]
	5	0010-0010-0100	Implicant	NonEssential	[ ]

The consensus of cubes 1 and 3 yields cube 6. Cube 1 is a subset of cube 6 and is therefore deleted. Cube 6 yields no further consensus cubes. Cube 6 is essential.

	Id #	cube	Kind	PIKind	Intersections
dc	6	0111-0100-0010	PICandidate	Essential	[3]
	1	0001-0100-0010	Deleted	NonEssential	[ ]
	2	0100-1010-1100	Implicant	NonEssential	[ ]
	3	0110-0100-0110	Implicant	NonEssential	[6]
	4	0010-1000-1000	Implicant	NonEssential	[ ]
	5	0010-0010-0100	Implicant	NonEssential	[ ]

Cube 2 is expanded next. The consensus of cubes 2 and 3 yields cube 7. Cube 7 is at distance 1 from cube 5. Cube 5 is a subset of cube 8.

	Id #	cube	Kind	PIKind	Intersections
	8	0110-0010-0100	Implicant	NonEssential	[7]
	7	0100-1110-0100	Implicant	NonEssential	[2,3,8]
dc	6	0111-0100-0010	PICandidate	Essential	[3]
	2	0100-1010-1100	Implicant	NonEssential	[7]
	3	0110-0100-0110	Implicant	NonEssential	[6,7]
	4	0010-1000-1000	Implicant	NonEssential	[ ]
	5	0010-0010-0100	Deleted	NonEssential	[ ]

The consensus of cubes 8 and 3 yields cube 9. Cube 8 is a subset of cube 9. Cube 9 is essential. Cube 3 is dominated by cube 9.

	Id #	cube	Kind	PIKind	Intersections
dc	9	0110-0110-0100	PICandidate	Essential	[2,3,7]
	8	0110-0010-0100	Deleted	NonEssential	[7]
	7	0100-1110-0100	Implicant	NonEssential	[2,3,9]
dc	6	0111-0100-0010	PICandidate	Essential	[3]
	2	0100-1010-1100	Implicant	NonEssential	[7,9]
	3	0110-0100-0110	Deleted	NonEssential	[6,7,9]
	4	0010-1000-1000	Implicant	NonEssential	[ ]

Cube 7 can not be expanded any further. Cube 7 is not essential. The expansion of cube 2 yields cube 10 (the consensus of cubes 2 and 4). Cube 4 is a subset of cube 10. Cube 10 is essential.

	Id #	cube	Kind	PIKind	Intersections
dc	10	0110-1000-1000	PICandidate	Essential	[2]
dc	9	0110-0110-0100	PICandidate	Essential	[2,7]
	7	0100-1110-0100	PICandidate	NonEssential	[2,9]
dc	6	0111-0100-0010	PICandidate	Essential	[ ]
	2	0100-1010-1100	Implicant	NonEssential	[7,9,10]
	4	0010-1000-1000	Deleted	NonEssential	[ ]

Cube 2 is essential. Cube 2 dominates cube 7. The minimal solution is shown below.

	Id #	cube	Kind	PIKind	Intersections
dc	10	0110-1000-1000	PICandidate	Essential	[2]
dc	9	0110-0110-0100	PICandidate	Essential	[2]
dc	6	0111-0100-0010	PICandidate	Essential	[ ]
dc	2	0100-1010-1100	PICandidate	Essential	[9,10]

Example 5.13. Consider the binary multiple-output problem described in Example 4.6. The Karnaugh maps for the three functions are shown in Figure 4.10. The three functions can be mapped into a multiple-valued function as shown in Table 4.3. The list of cubes used in initial OnList represent the minimal sum-of-products for each binary function, without considering the multiple output. The given cubes are:

F_1	01–01–11–11–100
	01–11–01–01–100

F_2	01–01–10–11–010
	11–11–01–01–010

F_3	01–01–11–11–001
	10–11–01–01–001

The initial OnList contains the following information.

Id #	cube	Kind	PIKind	Intersections
1	01–01–11–11–100	Implicant	NonEssential	[ ]
2	01–11–01–01–100	Implicant	NonEssential	[ ]
3	01–01–10–11–010	Implicant	NonEssential	[ ]
4	11–11–01–01–010	Implicant	NonEssential	[ ]
5	01–01–11–11–001	Implicant	NonEssential	[ ]
6	10–11–01–01–001	Implicant	NonEssential	[ ]

Cubes 1 and 3 yield cube 7. Cube 3 is a subset of cube 7. The consensus of cubes 7 and 4 yields cube 8. Cubes 8 and 1 yield cube 9. Cube 8 is a subset of cube 9. The consensus of cubes 9 and 5 yields cube 10. Cube 9 is a subset of cube 10.

Id #	cube	Kind	PIKind	Intersections
10	01–01–11–01–111	Implicant	NonEssential	[5]
9	01–01–11–01–110	Deleted	NonEssential	[ ]
8	01–01–11–01–010	Deleted	NonEssential	[ ]
7	01–01–10–11–110	Implicant	NonEssential	[1]
1	01–01–11–11–100	Implicant	NonEssential	[2,7]
2	01–11–01–01–100	Implicant	NonEssential	[1]
3	01–01–10–11–010	Deleted	NonEssential	[ ]
4	11–11–01–01–010	Implicant	NonEssential	[ ]
5	01–01–11–11–001	Implicant	NonEssential	[10]
6	10–11–01–01–001	Implicant	NonEssential	[ ]

The expansion of cube 10 yields cube 11 (the consensus of cubes 10 and 6). The

consensus of cubes 11 and 4 yields cube 12. Cube 12 does not generate any useful consensus cubes. Cube 12 is not essential. The expansion of cube 10 also terminates. Cube 10 is not essential.

Id #	cube	Kind	PIKind	Intersections
12	11–01–01–01–011	PICandidate	NonEssential	[4,5,6,10]
11	11–01–01–01–001	Deleted	NonEssential	[ ]
10	01–01–11–01–111	PICandidate	NonEssential	[1,2,4,5,7,12]
7	01–01–10–11–110	Implicant	NonEssential	[1,10]
1	01–01–11–11–100	Implicant	NonEssential	[2,7,10]
2	01–11–01–01–100	Implicant	NonEssential	[1,10]
4	11–11–01–01–010	Implicant	NonEssential	[10,12]
5	01–01–11–11–001	Implicant	NonEssential	[10,12]
6	10–11–01–01–001	Implicant	NonEssential	[12]

The expansion of cube 7 continues. The consensus of cubes 7 and 5 yields cube 13. Cube 7 is a subset of cube 13. Cube 13 is essential.

	Id #	cube	Kind	PIKind	Intersections
dc	13	01–01–10–11–111	PICandidate	Essential	[1,10]
	12	11–01–01–01–011	PICandidate	NonEssential	[4,5,6,10]
	10	01–01–11–01–111	PICandidate	NonEssential	[1,2,4,5,12]
	7	01–01–10–11–110	Deleted	NonEssential	[ ]
	1	01–01–11–11–100	Implicant	NonEssential	[2,10]
	2	01–11–01–01–100	Implicant	NonEssential	[1,10]
	4	11–11–01–01–010	Implicant	NonEssential	[10,12]
	5	01–01–11–11–001	Implicant	NonEssential	[10,12]
	6	10–11–01–01–001	Implicant	NonEssential	[12]

Cubes 1 and 5 yield cube 14. Both cubes are subsets of cube 14. Cube 14 is essential. Cube 12 is checked to determine if it is essential since it dominates cube 10 which intersects with cube 14. Cube 12 is not essential.

	Id #	cube	Kind	PIKind	Intersections
dc	14	01–01–11–11–101	PICandidate	Essential	[2,12,13]
dc	13	01–01–10–11–111	PICandidate	Essential	[14]
	12	11–01–01–01–011	PICandidate	NonEssential	[4,6,14]
	10	01–01–11–01–111	Deleted	NonEssential	[ ]
	1	01–01–11–11–100	Deleted	NonEssential	[ ]
	2	01–11–01–01–100	Implicant	NonEssential	[14]
	4	11–11–01–01–010	Implicant	NonEssential	[12]
	5	01–01–11–11–001	Deleted	NonEssential	[ ]
	6	10–11–01–01–001	Implicant	NonEssential	[12]

Cube 2 is expanded next. Cubes 2 and 4 yield cube 15. Cube 2 is a subset of cube 15. The consensus of cubes 15 and 13 yields cube 16. Cube 16 is immediately deleted since it is dominated by cube 15. Cube 15 is essential.

	Id #	cube	Kind	PIKind	Intersections
	16	01–01–11–01–110	Deleted	NonEssential	[ ]
dc	15	01–11–01–01–110	PICandidate	Essential	[4,12,14]
dc	14	01–01–11–11–101	PICandidate	Essential	[12,13,15]
dc	13	01–01–10–11–111	PICandidate	Essential	[14]
	12	11–01–01–01–011	PICandidate	NonEssential	[4,6,14,15]
	2	01–11–01–01–100	Deleted	NonEssential	[ ]
	4	11–11–01–01–010	Implicant	NonEssential	[12,15]
	6	10–11–01–01–001	Implicant	NonEssential	[12]

The consensus of cubes 4 and 6 yields cube 17. Cube 6 is a subset of cube 17. Cube 17 dominates cubes 12 and 4. Finally, 4 essential cubes remain in OnList. The identical result was obtained by the DSA-MV (see Example 4.6)

	Id #	cube	Kind	PIKind	Intersections
dc	17	10–11–01–01–011	PICandidate	Essential	[ ]
dc	15	01–11–01–01–110	PICandidate	Essential	[14]
dc	14	01–01–11–11–101	PICandidate	Essential	[13,15]
dc	13	01–01–10–11–111	PICandidate	Essential	[14]

5.7 REMARKS

Prototypes of both algorithms have been implemented in Turbo Pascal on a Macintosh computer. The results of all examples given in this chapter (Sections 5.4 and 5.6) were obtained using these computer programs.

The implementation of the algorithms presented in this chapter is straightforward. Once the binary implementation was complete, the extension from RCM to RCM-MV was accomplished by simply replacing the data structure to store a cube and the procedures Distance, Consensus, and Sharp, together with new input and output routines.

Due to the new ordering of the cubes in OnList, which enables the detection of essential and pseudo-essential prime implicants, as well as the deletion of all dominated cubes, the number of elements in OnList is kept small.

Chapter 6

MINIMIZATION WITH WINDOW LITERALS

6.1 OVERVIEW

A wide variety of literal operations for multiple-valued logic have been proposed (see Section 2.3). The three minimization procedures presented in the preceding chapters use the generalized literal operation (see Definition 1.7). The window literal is defined as follows:

$$ {}^{a}x_i^{b} = \begin{cases} 1 & \text{if } a \le x_i \le b \\ 0 & \text{otherwise} \end{cases} \qquad (6.1) $$

Window literal are a subset of generalized literal. For example, a four-valued variable has 10 distinct non-trivial window literals, whereas the number of generalized literals is 15. In general, an R-valued variable has

$$ \frac{R\,(R+1)}{2} $$

window literals and

$$ R^2 - 1 $$

generalized literals. The number of literals is an important factor if the function is to be implemented using a PLA, since all literals must be generated. The number of literals determines the number of "rows" of the PLA needed for each input variable. The number of product terms of the function determines the number of PLA "columns". A sum-of-products expression is likely to have more terms using the window literal than the generalized literal.

The cube notation for window literals is similar to the cube notation for generalized

literals (Section 4.1). The product term

$$x_1^{a_1 b_1} x_2^{a_2 b_2} \ldots x_n^{a_n b_n}$$

is denoted by the binary vector:

$$c_1^0 c_1^1 \ldots c_1^{p_1 - 1} - c_2^0 c_2^1 \ldots c_2^{p_2 - 1} - \ldots - c_n^0 c_n^1 \ldots c_n^{p_n - 1}$$

where

$$c_i^j = \begin{cases} 1 & \text{if } a_i \le j \le b_i \\ 0 & \text{otherwise} \end{cases}$$

Example 6.1. If P= {4, 3, 4, 4}, then the product term

$$x_1^{0\,2} x_2^{1\,1} x_3^{2\,3} x_4^{1\,2}$$

is represented by the cube 1110–010–0011–0110.

The definition of adjacency must be changed slightly and the definition of expandable adjacencies in the direct cover algorithm is different from the one in the directed search algorithm.

Definition 6.1. Two minterms are said to be *adjacent* if they differ in exactly one of their input variables, and the magnitude of the difference in that input variable is exactly one. Let α and ß be adjacent minterms, let x_i be the input variable in which they differ

$$\alpha = (a_1, a_2, \ldots, a_i, \ldots, a_n)$$

$$ß = (a_1, a_2, \ldots, b_i, \ldots, a_n)$$

then $|a_i - b_i| = 1$

Any minterm has between *n* and *2n* adjacent minterms since a minterm has at least one adjacent minterm with respect to each coordinate and at most two. In contrast, the number of adjacencies for the generalized literal operator is:

$$\sum_{i=1}^{n}(p_i - 1)$$

6.2 DIRECT COVER ALGORITHM FOR TRUNCATED SUM MINIMIZATION

6.2.1 The Extension

Since we are dealing with functions with multiple-valued outputs, the literal operation will yield the values R-1 and 0 (not 1 and 0 as shown in (6.1)). The definition of expandable adjacency from Chapter 3 is used in this section (Definition 3.2). A minterm may be expanded beyond its adjacent minterms (Figure 6.1). This must be considered when ranking minterms.

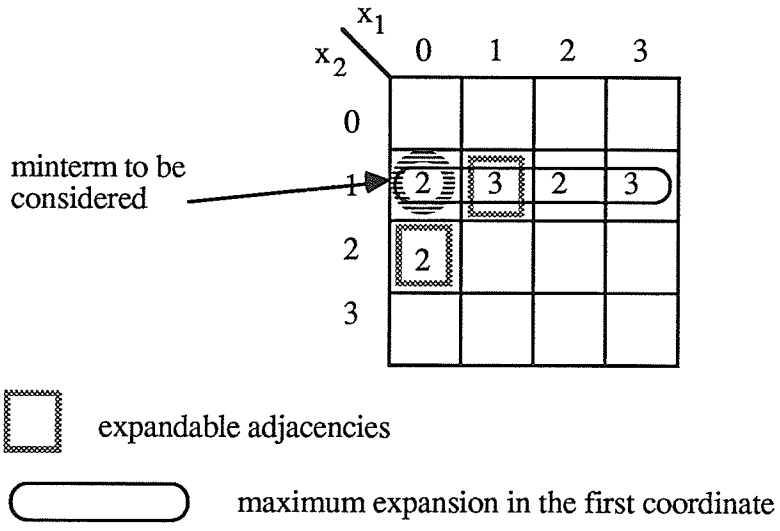


Figure 6.1 A minterm may be expanded beyond its adjacent minterms.

Definition 6.2. Given the minterm

$$\alpha = (a_1, a_2, \ldots, a_i, \ldots, a_n)$$

let Q be an implicant of the function of the form

$$Q = c\, x_1^{a_1} x_1^{a_1}\, x_2^{a_2} x_2^{a_2} \cdots x_i^{k_i} x_i^{l_i} \cdots x_n^{a_n} x_n^{a_n}$$

where $k_i \le a_i \le l_i$, $c = \ll f(\alpha) \gg$, and Q is not contained in any other implicant of the same form, *i.e.* Q is the largest implicant of this type. All minterms contained in Q , excluding α, are said to be the *extendible minterms* of α in the i^{th} coordinate. The total number of extendible minterms of α is denoted by EM_α.

Definition 6.3. The *isolation factor* of a minterm α is

$$IF_\alpha = \frac{1}{EA_\alpha (R - 1) + EM_\alpha + 1}$$

With these three new definitions the algorithm to determine the break count reduction (BCR) can be used without modification. Furthermore, the minimization algorithm presented in Chapter 3 can be applied as given. This algorithm will be referred to as DCM^W.

6.2.2 Examples

Tirumalai and Butler [TIR88] analyzed several minimization algorithms for multiple-valued PLAs using the truncated sum operator and window literals. Tirumalai and Butler's adaptation of the DCM algorithm (DCM^*) gave results which are not as good as the extension of the DCM algorithm (DCM^W) presented in Section 6.2.1. For the following three examples [BUT88] DCM^W gave better results than DCM^*.

For simplicity, the product term

$$c\, {}^{a_1}x_1^{b_1}\, {}^{a_2}x_2^{b_2} \ldots {}^{a_n}x_n^{b_n}$$

will be written as $c(a_1,b_1)(a_2,b_2) \ldots (a_n,b_n)$.

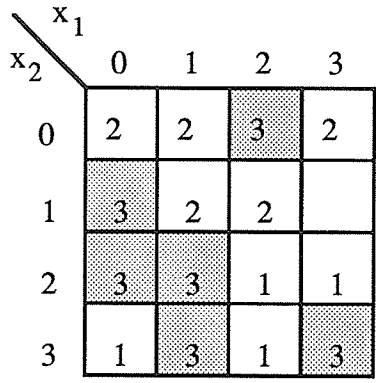


Figure 6.2 Map[1] of the function used in Example 6.2.

Example 6.2. Consider the function shown in Figure 6.2. The function was realized with 8 implicants using the DCM* algorithm [BUT88]. With DCM^{W} a minimal cover consisting of 6 implicants was obtained. The minimization proceeds as follows:

STEP 1 MinValue ← 1.

STEP 2 The isolation factors for the minterms with value 1 are as follows:

minterm	(2,2)	(3,2)	(0,3)	(2,3)
IF	1/19	1/11	1/13	1/16

STEP 3 $\alpha \leftarrow$ (3,2).

STEP 4 The eight implicants which contain α have the following break count reductions:

implicant	1(3,3)(2,2)	1(3,3)(2,3)	1(2,3)(2,2)	1(2,3)(2,3)
BCR	0	0	0	1
implicant	1(1,3)(2,2)	1(1,3)(2,3)	1(0,3)(2,2)	1(0,3)(2,3)
BCR	1	2	1	2

STEP 5 Qmax ←1(0,3)(2,3)[2].

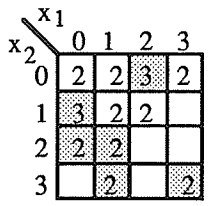
STEP 6 Add 1(0,3)(2,3) to the solution.

---

[1] The minterms for which the function evaluates to R - 1 are shaded because they are treated differently during the minimization process.

[2] The last implicant with the highest BRC is selected. This choice is arbitrary, but consistent throughout the examples in this chapter.

STEP 7 g(X) now becomes:



STEP 1 MinValue ← 2.

STEP 2 The isolation factors for the minterms with value 2 are as follows:

minterm	(0,0)	(1,0)	(3,0)	(1,1)	(2,1)
IF	1/12	1/16	1/7	1/18	1/10

Note: minterms (0,2), (1,2), (1,3), and (3,3) are not considered since they evaluated to 3 in f(X).

STEP 3 $\alpha \leftarrow$ (3,0).

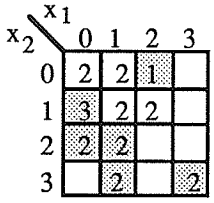
STEP 4 The four implicants which contain α have the following break count reductions:

implicant	2(3,3)(0,0)	2(2,3)(0,0)	2(1,3)(0,0)	2(0,3)(0,0)
BCR	1	1	-1	0

STEP 5 Qmax ←2(2,3)(0,0).

STEP 6 Add 2(2,3)(0,0) to the solution.

STEP 7 g(X) now becomes:



STEP 1 MinValue ← 2.

(MinValue is not assigned the value 1 because the minterm which evaluates to 1 is shaded — it originally evaluated to 3)

STEP 2 The isolation factors for the minterms with value 2 are as follows:

minterm	(0,0)	(1,0)	(1,1)	(2,1)
IF	1/11	1/15	1/18	1/10

STEP 3 $\alpha \leftarrow (1,2)$.

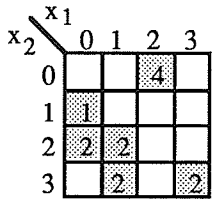
STEP 4 The six implicants which contain α have the following break count reductions:

2(2,2)(1,1)	2(2,2)(0,1)	2(1,2)(1,1)	2(1,2)(0,1)	2(0,2)(1,1)	2(0,2)(0,1)
1	2	0	1	0	2

STEP 5 Qmax $\leftarrow$ 2(0,2)(0,1).

STEP 6 Add 2(0,2)(0,1) to the solution.

STEP 7 g(X) now becomes[3]:



STEP 1 MinValue $\leftarrow$ 3.

(all remaining minterms for which «g(X)» < 4, had the value 3 in f(X).)

STEP 2 The isolation factors for the minterms with value 3 are as follows:

minterm	(0,1)	(0,2)	(1,2)	(1,3)	(3,3)
IF	1/5	1/9	1/9	1/5	1/1

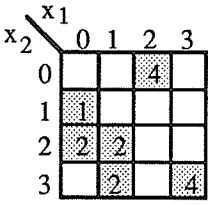
STEP 3 $\alpha \leftarrow (3,3)$.

STEP 4 There is only one implicant 3(3,3)(3,3) with BRC 2.

STEP 5 Qmax $\leftarrow$ 3(3,3)(3,3).

STEP 6 Add 3(3,3)(3,3) to the solution.

STEP 7 g(X) now becomes:



[3] Minterms with value 4 are don't-care minterms.

STEP 1 MinValue ← 3.

STEP 2 The isolation factors for the minterms with value 3 are as follows:

minterm	(0,1)	(0,2)	(1,2)	(1,3)
IF	1/5	1/9	1/9	1/5

STEP 3 $\alpha \leftarrow (0,1)$.

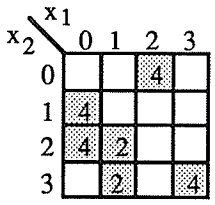
STEP 4 The two implicants which contain α have the following break count reductions:

implicant	3(0,0)(1,1)	3(0,0)(1,2)
BCR	2	3

STEP 5 Qmax ← 3(0,0)(1,2).

STEP 6 Add 3(0,0)(1,2) to the solution.

STEP 7 g(X) now becomes:



STEP 1 MinValue ← 3.

STEP 2 The isolation factors for the minterms with value 3 are as follows:

minterm	(1,2)	(1,3)
IF	1/9	1/5

STEP 3 $\alpha \leftarrow (1,3)$.

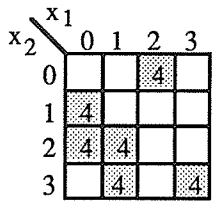
STEP 4 The two implicants which contain α have the following break count reductions:

implicant	3(1,1)(3,3)	3(1,1)(2,3)
BCR	2	4

STEP 5 Qmax ← 3(1,1)(2,3).

STEP 6 Add 3(1,1)(2,3) to the solution.

STEP 7 g(X) now becomes:



STEP 8 The termination condition has been reached. The result is

$$f(x_1,x_2) = 1\,x_1^{0\,3\,2\,3}x_2^{} \lozenge\; 2\,x_1^{2\,3\,0\,0}x_2^{} \lozenge\; 2\,x_1^{0\,2\,0\,1}x_2^{} \lozenge\; 3\,x_1^{3\,3\,3\,3}x_2^{} \lozenge\; 3\,x_1^{0\,0\,1\,2}x_2^{} \lozenge\; 3\,x_1^{1\,1\,2\,3}x_2^{}$$

The result obtained by DCM* [BUT88] is

$$F(x_1, x_2) = 1\,x_1^{0\,3\,2\,3}x_2^{} \lozenge\; 2\,x_1^{3\,3\,3\,3}x_2^{} \lozenge\; 2\,x_1^{1\,1\,0\,3}x_2^{} \lozenge\; 2\,x_1^{2\,2\,1\,1}x_2^{} \lozenge$$

$$2\,x_1^{3\,3\,0\,0}x_2^{} \lozenge\; 2\,x_1^{0\,0\,0\,2}x_2^{} \lozenge\; 1\,x_1^{0\,0\,1\,1}x_2^{} \lozenge\; 3\,x_1^{2\,2\,0\,0}x_2^{}$$

Example 6.3. Consider the function shown in Figure 6.3. The function was realized with 9 implicants using the DCM* algorithm [BUT88]. A minimal cover consisting of 7 implicants was obtained using the DCM^W algorithm. The terms were obtained in the following order

most isolated minterm	best implicant
$x_1^{1\,1}x_2^{2\,2}$	$1\,x_1^{1\,1}x_2^{2\,2}$
$x_1^{0\,0}x_2^{1\,1}$	$1\,x_1^{0\,0}x_2^{1\,3}$
$x_1^{2\,2}x_2^{3\,3}$	$2\,x_1^{2\,3}x_2^{3\,3}$
$x_1^{1\,1}x_2^{0\,0}$	$2\,x_1^{1\,3}x_2^{0\,0}$
$x_1^{3\,3}x_2^{1\,1}$	$2\,x_1^{3\,3}x_2^{0\,3}$

$^{2}_{}x_1^{2}\,^{1}_{}x_2^{1}$ $\qquad$ $3\,^{2}_{}x_1^{2}\,^{1}_{}x_2^{1}$

$^{3}_{}x_1^{3}\,^{2}_{}x_2^{2}$ $\qquad$ $3\,^{3}_{}x_1^{3}\,^{2}_{}x_2^{3}$

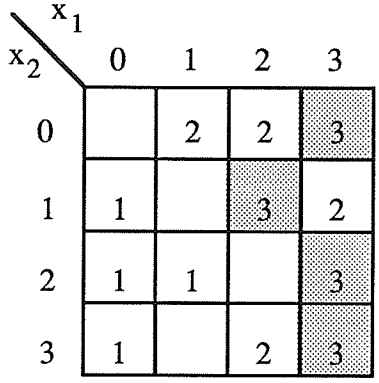
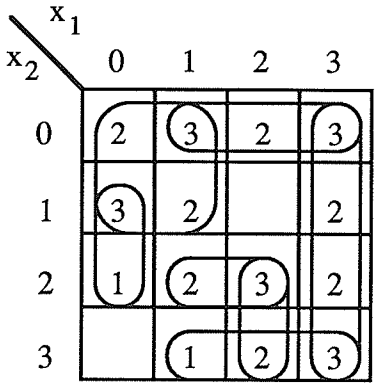


Figure 6.3 Map of the function used in Example 6.3.



$$f(x_1,x_2) = 1\,^{0}_{}x_1^{0}\,^{1}_{}x_2^{2} \lozenge 1\,^{1}_{}x_1^{3}\,^{3}_{}x_2^{3} \lozenge 1\,^{2}_{}x_1^{2}\,^{2}_{}x_2^{3} \lozenge$$
$$2\,^{3}_{}x_1^{3}\,^{0}_{}x_2^{3} \lozenge 2\,^{1}_{}x_1^{3}\,^{0}_{}x_2^{0} \lozenge 2\,^{0}_{}x_1^{1}\,^{0}_{}x_2^{1} \lozenge$$
$$2\,^{1}_{}x_1^{2}\,^{2}_{}x_2^{2}$$

Figure 6.4 Map of the function used in Example 6.4.

Example 6.4. Consider the function shown in Figure 6.4. DCM* produced a realization with 10 implicants [BUT88]. The minimal realization of this function consists of 7 implicants (see Figure 6.4). The DCM^{W} algorithm produced a solution with 8 terms which is not minimal, but better than the solution proposed by DCM*. The execution of DCM^{W} proceeds as follows:

STEP 1 MinValue ← 1.

STEP 2 The isolation factors for the minterms with value 1 are as follows:

minterm	(0,2)	(1,3)
IF	1/12	1/12

STEP 3 $\alpha \leftarrow (0,2)$.

STEP 4 The eight implicants which contain α have the following break count reductions:

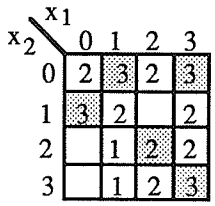
implicant	1(0,0)(2,2)	1(0,0)(1,2)	1(0,0)(0,2)	1(0,1)(2,2)
BCR	1	3	2	1
implicant	1(0,1)(1,2)	1(0,1)(0,2)	1(0,2)(2,2)	1(0,3)(2,2)
BCR	3	3	3	1

STEP 5 Qmax ← 1(0,2)(2,2).

Note: this choice will not lead to the minimal solution. The implicant 1(0,0)(1,2) should have been chosen at this point. The BRC for the "best" implicant 1(0,0)(1,2) is the same as the BRC for the chosen implicant 1(0,2)(2,2).

STEP 6 Add 1(0,2)(2,2) to the solution.

STEP 7 g(X) now becomes:



STEP 1 MinValue ← 1.

STEP 2 The isolation factors for the minterms with value 1 are as follows:

minterm	(1,2)	(1,3)
IF	1/15	1/12

STEP 3 $\alpha \leftarrow (1,3)$.

STEP 4 The eight implicants which contain α have the following break count reductions:

implicant	1(1,1)(3,3)	1(1,1)(2,3)	1(1,1)(1,3)	1(1,1)(0,3)
BCR	0	2	2	4
implicant	1(1,2)(3,3)	1(1,2)(2,3)	1(1,3)(3,3)	1(1,3)(2,3)
BCR	-1	2	0	1

STEP 5 Qmax ← 1(1,1)(0,3) .

STEP 6 Add 1(1,1)(0,3) to the solution.

STEP 7 g(X) now becomes:

x_2 \ x_1	0	1	2	3
0	2	2	2	3
1	3	1		2
2			2	2
3			2	3

STEP 1 MinValue ← 1.

STEP 2 The only minterm which evaluates to 1 (1,1) has an isolation factor of 1/9.

STEP 3 α ← (1,1).

STEP 4 The four implicants which contain α have the following break count reductions:

implicant	1(1,1)(1,1)	1(1,1)(0,1)	1(0,1)(1,1)	1(0,1)(0,1)
BCR	2	2	3	2

STEP 5 Qmax ← 1(0,1)(1,1).

STEP 6 Add 1(0,1)(1,1) to the solution.

STEP 7 g(X) now becomes:

x_2 \ x_1	0	1	2	3
0	2	2	2	3
1	2			2
2			2	2
3			2	3

STEP 1 MinValue ← 2.

STEP 2 The isolation factors for the minterms with value 2 are as follows:

minterm	(0,0)	(2,0)	(3,1)	(3,2)	(2,3)
IF	1/11	1/10	1/10	1/14	1/9

STEP 3 $\alpha \leftarrow (2,3)$

STEP 4 The four implicants which contain α have the following break count reductions:

implicant	2(2,2)(3,3)	2(2,2)(2,3)	2(2,3)(3,3)	2(2,3)(2,3)
BCR	0	3	0	2

STEP 5 Qmax $\leftarrow$ 2(2,2)(2,3).

STEP 6 Add 2(2,2)(2,3) to the solution.

STEP 7 g(X) now becomes:

x_2 \ x_1	0	1	2	3
0	2	2	2	3
1	2			2
2			4	2
3				3

STEP 1 MinValue $\leftarrow$ 2.

STEP 2 The isolation factors for the minterms with value 2 are as follows:

minterm	(0,0)	(2,0)	(3,1)	(3,2)
IF	1/11	1/10	1/10	1/14

STEP 3 $\alpha \leftarrow (2,0)$.

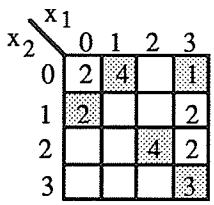
STEP 4 The six implicants which contain α have the following break count reductions:

2(2,2)(0,0)	2(2,3)(0,0)	2(1,2)(0,0)	2(1,3)(0,0)	2(0,2)(0,0)	2(0,3)(0,0)
0	0	2	2	1	1

STEP 5 Qmax $\leftarrow$ 2(1,3)(0,0).

STEP 6 Add 2(1,3)(0,0) to the solution.

STEP 7 g(X) now becomes:



STEP 1 MinValue ← 2.

STEP 2 The isolation factors for the minterms with value 2 are as follows:

minterm	(0,0)	(3,1)	(3,2)
IF	1/9	1/10	1/14

STEP 3 $\alpha \leftarrow (0,0)$.

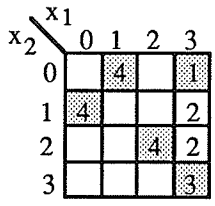
STEP 4 The three implicants which contain α have the following break count reductions:

implicant	2(0,0)(0,0)	2(0,0)(0,1)	2(0,1)(0,0)
BCR	-1	2	-1

STEP 5 Qmax ← 2(0,0)(0,1).

STEP 6 Add 2(0,0)(0,1) to the solution.

STEP 7 g(X) now becomes:



STEP 1 MinValue ← 2.

STEP 2 The isolation factors for the minterms with value 2 are as follows:

minterm	(3,1)	(3,2)
IF	1/10	1/14

STEP 3 $\alpha \leftarrow (3,1)$.

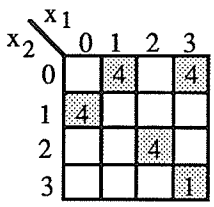
STEP 4 The six implicants which contain α have the following break count reductions:

2(3,3)(1,1)	2(3,3)(1,2)	2(3,3)(1,3)	2(3,3)(0,1)	2(3,3)(0,2)	2(3,3)(0,3)
-1	0	0	1	2	2

STEP 5 Qmax ← 2(3,3)(0,3).

STEP 6 Add 2(3,3)(0,3) to the solution.

STEP 7 g(X) now becomes:



STEP 1 MinValue ← 3.

STEP 2 The only minterm which evaluates to 3 (3,3) has an isolation factor of 1/1.

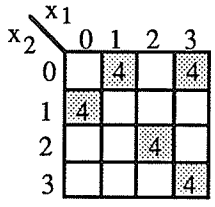
STEP 3 α ← (3,3).

STEP 4 The only implicant 3(3,3)(3,3) has a BRC of 2.

STEP 5 Qmax ← 3(3,3)(3,3).

STEP 6 Add 3(3,3)(3,3) to the solution.

STEP 7 g(X) now becomes:



STEP 8 The termination condition has been reached. The result is

$$f(x_1,x_2) = 1\,{}^{0}_{1}x^{2}_{2}x^{2}_{2} \lozenge\ 1\,{}^{1}_{1}x^{1}_{1}x^{0}_{2}x^{3}_{2} \lozenge\ 1\,{}^{0}_{1}x^{1}_{1}x^{1}_{2} \lozenge\ 2\,{}^{2}_{2}x^{2}_{1}x^{3}_{2} \lozenge\ 2\,{}^{1}_{2}x^{3}_{1}x^{0}_{2} \lozenge 2\,{}^{0}_{2}x^{0}_{1}x^{1}_{2} \lozenge\ 2\,{}^{3}_{2}x^{3}_{1}x^{0}_{2} \lozenge\ 3\,{}^{3}_{3}x^{3}_{1}x^{3}_{2}$$

The result obtained by DCM* [BUT88] is

$$F(x_1, x_2) = 1\,{}^{0}x_1^{0}\,{}^{1}x_2^{2} \lozenge 1\,{}^{1}x_1^{3}\,{}^{3}x_2^{3} \lozenge 1\,{}^{2}x_1^{2}\,{}^{2}x_2^{3} \lozenge 2\,{}^{1}x_1^{3}\,{}^{2}x_2^{2} \lozenge 2\,{}^{3}x_1^{3}\,{}^{3}x_2^{3} \lozenge$$

$$2\,{}^{3}x_1^{3}\,{}^{1}x_2^{1} \lozenge 2\,{}^{0}x_1^{2}\,{}^{0}x_2^{0} \lozenge 1\,{}^{1}x_1^{1}\,{}^{0}x_2^{0} \lozenge 2\,{}^{0}x_1^{1}\,{}^{1}x_2^{1} \lozenge 3\,{}^{3}x_1^{3}\,{}^{0}x_2^{0}$$

6.2.3 Remarks

The DCM^W minimization algorithm produced better results than Tirumalai and Butler's [TIR88] extension of the DCM algorithm. However, a minimal result is not always produced by DCM^W (see Example 6.4). The break reduction count often produces ties for different implicants. In Example 6.4, the "best" implicant (the implicant which is part of a minimal solution) always had the highest BRC. Frequently, several terms had the same BRC and the "best" implicant was not always chosen to be part of the solution. Additional investigation is needed to find an effective way to break BRC ties. One possibility is to build a partial tree containing all of the "best" choices.

6.3 DIRECTED SEARCH MINIMIZATION

6.3.1 The Extension

The definition of expandable adjacency from Chapter 4 (Definition 4.2) is used here. In this section we are dealing with multiple-valued input, binary-valued output functions. As pointed out in Section 6.2.1, a minterm may be expanded beyond its adjacent minterms (Figure 6.2).

Definition 6.4. Given the minterm

$$\alpha = (a_1, a_2, \ldots, a_i, \ldots, a_n)$$

let Q be an implicant of the function of the form

$$Q = {}^{a_1}x_1^{a_1}\,{}^{a_2}x_2^{a_2} \ldots {}^{k_i}x_i^{l_i} \ldots {}^{a_n}x_n^{a_n}$$

where $k_i \leq a_i \leq l_i$ and Q is not contained in any other implicant of the same form. All minterms contained in Q, excluding α, are said to be the *extendible minterms* of α in the i^{th} coordinate.

The extendible minterms of a minterm can be represented by a bit string similar to the cube notation of the minterm. Let M be a minterm and let M_c be the cube representing M. Let $Q_1, Q_2, \ldots Q_r$ be the extendible minterms of M and let $Q_{c1}, Q_{c2}, \ldots Q_{cr}$ be their respective cube representations. The bit string representing the extendible minterms of M, EMV_M (Extendible Minterms Vector), is obtained as follows:

$$EMV_M = (Q_{c1} + Q_{c2} + \ldots + Q_{cr}) \bullet (M_c)'$$

where +, •, and ' represent the bitwise OR, AND, and COMPLEMENT operations respectively.

Each extendible minterms vector has associated with it a weight $EMVW_M$ which consists of a pair of integers. The integers in $EMVW_M$ are obtained as follows:

- The first integer indicates the number of adjacent minterms of M which are also extendible minterms of M.
- The second integer is the number of ones in the EMV_M.

The following modifications are needed to adapt the DSA-MV algorithm (see Section 4.2), to handle window literals — replace EAV by EMV and replace EAVW by EMVW. The modified version of the directed search algorithm is called DSA-MV^W.

6.3.2 Examples

Example 6.5. Consider the three-variable function with radix three shown in Figure 6.5. The minterms in the ON-set with their corresponding extendible minterm vectors and weights are shown in Table 6.1. Minterms are expanded in the order shown in Table 6.2. Each of the expansions results in an essential prime implicant. The final result is

$$F(x_1,x_2,x_3) = \overset{0\,0\,0\,1}{x_1\ x_2} + \overset{0\,0\,0\,1}{x_2\ x_3} + \overset{0\,1\,0\,1\,1\,2}{x_1\ x_2\ x_3}$$

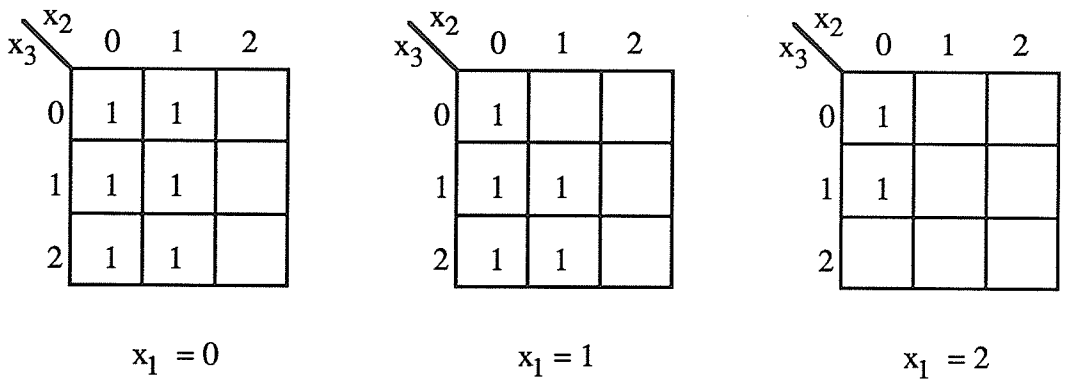


Figure 6.5 Function used in Example 6.5.

ON-set	EMV	EMVW
100–100–100	011–010–011	(3,5)
100–100–010	011–010–101	(4,5)
100–100–001	010–010–110	(3,4)
100–010–100	000–100–011	(2,3)
100–010–010	010–100–101	(4,4)
100–010–001	010–100–110	(3,4)
010–100–100	101–000–011	(3,4)
010–100–010	101–010–101	(5,5)
010–100–001	100–010–110	(3,4)
010–010–010	100–100–001	(3,3)
010–010–001	100–100–010	(3,3)
001–100–100	110–000–010	(2,3)
001–100–010	110–000–100	(2,3)

Table 6.1 Extendible minterms with the corresponding weight pairs (Example 6.5).

	minterm	prime implicant
1)	100–010–100	100–110–111
2)	001–100–100	111–100–110
3)	010–010–010	110–110–011

Table 6.2 Expanded minterms with the corresponding prime implicants (Example 6.5).

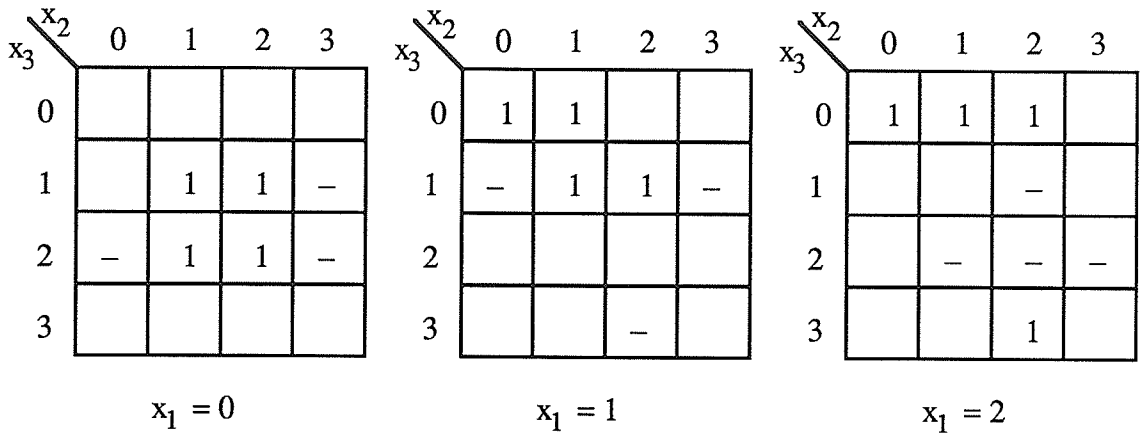


Figure 6.6 Map of the function used in Example 6.6.

Example 6.6. Consider the three-variable function with P = {3,4,4}, shown in Figure 6.6. The minterms in the ON-set with their corresponding extendible minterm vectors and weights are shown in Table 6.3. The minterm with the lowest weight (001–1000–1000) is expanded first. The pruned expansion tree is shown in Figure 6.7 (expanded minterms are underlined). Two prime implicants (PI1 and PI2) cover the expanded minterm. Minterm (001–0010–1000) meets the criteria of step 6 in DSA-MV^W, and so it is expanded next (Figure 6.8). Two prime implicants, PI2 and PI3, cover the expanded minterm.

Next, minterm (001–0010–0001) is expanded. PI4 is added to the tree (Figure 6.9). PI4 is dominated by PI3. PI3 becomes pseudo-essential. Now PI1 dominates PI2 — PI1 is pseudo-essential. The expansion tree is now empty.

Minterm 100–0100–0100 is expanded next (Figure 6.10). Two prime implicants are left in the pruned expansion tree (PI5 and PI6). The expansion of minterm 100–0100–0010 yields one new prime implicant (PI7). PI7 is dominated by PI6. PI6 is pseudo-essential — it is added to the solution. Prime implicant PI5 covers the only 2 remaining minterms in the ON-set. Therefore, even if the expansion of one of the two remaining minterms in the ON-set may yield new prime implicants, they will all be

dominated by PI5 — PI5 is added to the solution[4]. The minimal sum-of-products expression is

$$F(x_1,x_2,x_3) = {}^{1\,2}x_1\,{}^{0\,1}x_2\,{}^{0\,0}x_3 + {}^{2\,2}x_1\,{}^{2\,2}x_2\,{}^{0\,3}x_3 + {}^{0\,0}x_1\,{}^{1\,3}x_2\,{}^{1\,2}x_3 + {}^{0\,1}x_1\,{}^{1\,3}x_2\,{}^{1\,1}x_3$$

ON-set	EMV	EMVW
100–0100–0100	010–0011–0010	(3,4)
100–0100–0010	000–1011–0100	(3,4)
100–0010–0100	011–0101–0010	(4,5)
100–0010–0010	000–1011–0100	(3,4)
010–1000–1000	001–0100–0100	(3,3)
010–0100–1000	001–1000–0100	(3,3)
010–0100–0100	100–1011–1000	(4,5)
010–0010–0100	101–1101–0000	(4,5)
001–1000–1000	010–0110–0000	(2,3)
001–0100–1000	010–1010–0000	(3,3)
001–0010–1000	000–1100–0111	(2,5)
001–0010–0001	010–0000–1110	(2,4)

Table 6.3 Extendible minterms with the corresponding weight pairs (Example 6.6).

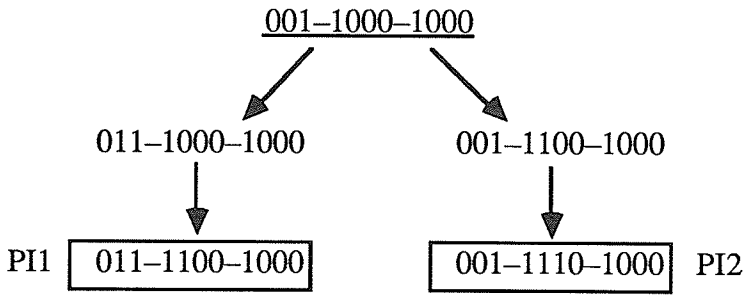


Figure 6.7 The pruned expansion tree for minterm (001–1000–1000).

[4] This shortcut may not be recognized by a programmed version of the algorithm. An additional expansion would be required.

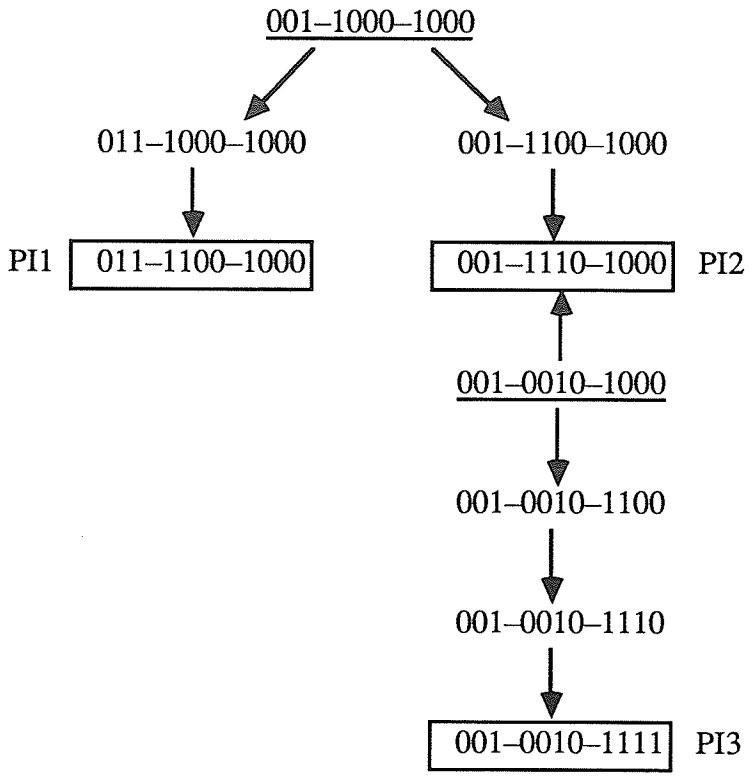


Figure 6.8 Pruned expansion tree.

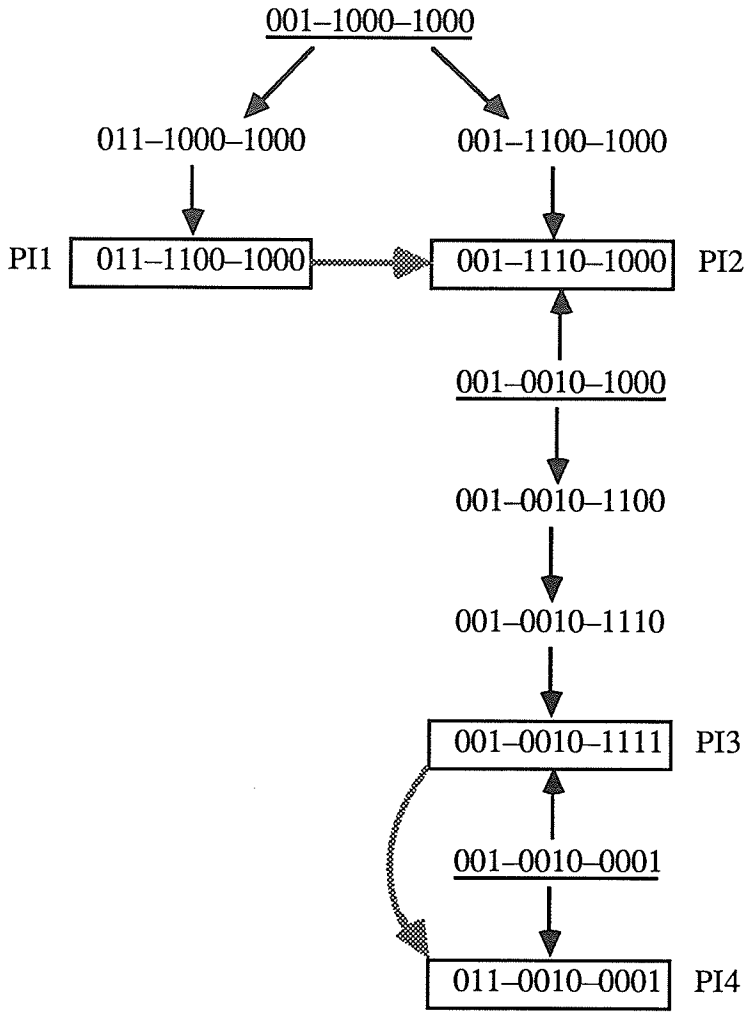


Figure 6.9 A pruned expansion tree.

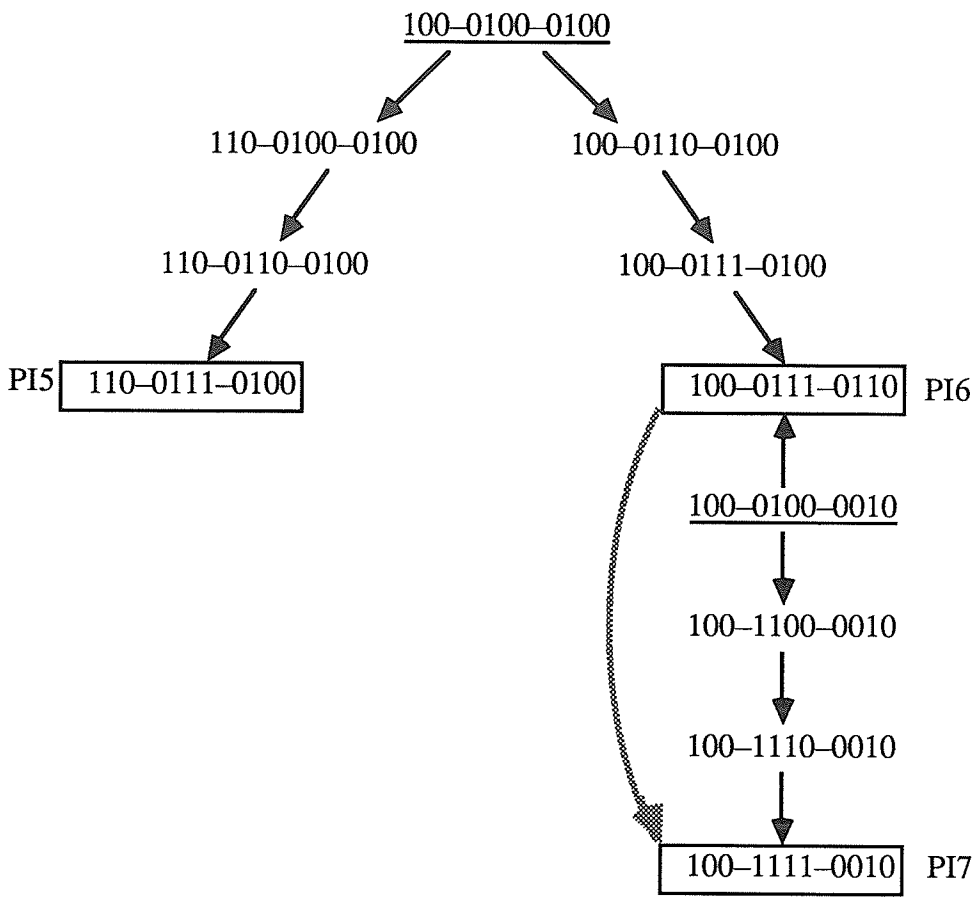


Figure 6.10 A pruned expansion tree (Example 6.6).

6.4 RECURSIVE CONSENSUS MINIMIZATION

6.4.1 The Extension

The adaptation of the new recursive consensus minimization algorithm (RCM-MV) to handle window literals requires minor changes in the basic definitions. In particular, the consensus of two product terms at distance 1 does not always exist — an additional relation must be satisfied. Furthermore, a new procedural definition for the sharp operation is required.

The definition of the distance between two cubes given in Chapter 5 (Definition 5.4) will be used in this section.

Example 6.7. Table 6.4 shows some cubes of a 4-valued function and their corresponding distance.

P	Q	AND(P,Q)	distance
0110–1110–0100	1110–0110–1110	0110–0110–0100	0
0110–1110–0100	1100–1100–0011	0100–0110–0000	1
0110–1110–0100	1100–1100–0001	0100–0110–0000	1
0110–1110–0100	0001–1110–0011	0000–1110–0000	2

Table 6.4 Cubes and their distance.

Unfortunately, two cubes at distance one do not necessarily have a consensus cube. For example, the cubes 0110–1110–0100 and 1100–1100–0001 are at distance 1, and their consensus, according to Definition 5.5, is 0100–0110–0101, but x_3 can not be expressed using a single window literal. Therefore, two cubes must satisfy an additional condition to have a consensus cube.

Let $P = p_1 - p_2 - \ldots - p_n$ be a cube of a multiple-valued logic function F where p_i represents the i^{th} coordinate of P. Similarly, let $Q = q_1 - q_2 - \ldots - q_n$ be a cube of F such that the distance between P and Q is 1. Let $j \in \{1,2,\ldots,n\}$ such that $AND(p_j,q_j)$ is equal to a bitstring consisting of zeros only. P and Q have a consensus (as defined in Definition 5.5) if and only if all 1's in OR (p_j,q_j) are consecutive, where OR is the bitwise OR operation.

Example 6.8. Table 6.5 shows some cubes with their corresponding consensus cube.

P	Q	consensus of P and Q
0110–1110–0100	1110–0110–0011	0110–0110–0111
1100–0110–0111	0011–1100–0001	1111–0100–0001
1100–0110–0111	0110–1000–1110	0100–1110–0110

Table 6.5 Product terms and their consensus.

Each product term using the generalized literals can be expressed as a sum of product terms using window literals. Some examples are given below.

1101–0110–1100 = 1100–0110–1100 + 0001–0110–1100

1101–0101–0110 = 1100–0100–0110 + 1100–0001–0110 + 0001–0100–0110 + 0001–0001–0110

The sharp operator has been defined in Section 5.5 (Definition 5.6). The procedural definition of # has to be modified is such a way that all product terms are using the window literal. This is illustrated by Example 6.9.

Example 6.9. Consider the cubes P = 1110–0110–0011 and Q = 0100–1100–0111. According to the procedural definition (Definition 5.6) P # Q is given by

C_1 = 1010–0110–0011,

C_2 = 1110–0010–0011,

C_3 = 1110–0110–0000 null cube (delete).

Cube C_1 can not be expressed as one product term using window literals. 1010–0110–0011 = 0010–0110–0011 + 1000–0110–0011. Hence,

P # Q = 0010–0110–0011 + 1000–0110–0011 + 1110–0010–0011

The changes described in this section affect the procedure Expand and the procedure Sharp. In the procedure Expand, an additional check must be made before Consensus is called. In the Sharp procedure each cube must only use window literals. No other procedure is affected. The algorithm which implements the changes described above will be called RCM-MV^W.

6.4.2 Examples

Example 6.10. Consider the three-variable function shown in Figure 6.5. The initial OnList contains the following information.

Id #	cube	Kind	PIKind	Intersections
1	100–110–100	Implicant	NonEssential	[]
2	110–110–011	Implicant	NonEssential	[]
3	011–100–110	Implicant	NonEssential	[]

The consensus of cubes 1 and 2 yields cube 4. Cube 1 is deleted since it is contained in cube 4. The consensus of cubes 4 and 3 yields cube 5. Cube 3 is deleted since it is contained in cube 5.

Id #	cube	Kind	PIKind	Intersections
5	111–100–110	Implicant	NonEssential	[2,4]
4	100–110–111	Implicant	NonEssential	[2,5]
1	100–110–100	Deleted	NonEssential	[]
2	110–110–011	Implicant	NonEssential	[4,5]
3	011–100–110	Deleted	NonEssential	[]

No further consensus term can be formed with cube 5. Cube 5 is essential. Cube 4 is also found to be essential. The final OnList contains the minimal solution.

	Id #	cube	Kind	PIKind	Intersections
dc	5	111–100–110	Implicant	NonEssential	[2,4]
dc	4	100–110–111	Implicant	NonEssential	[2,5]
dc	2	110–110–011	Implicant	NonEssential	[4,5]

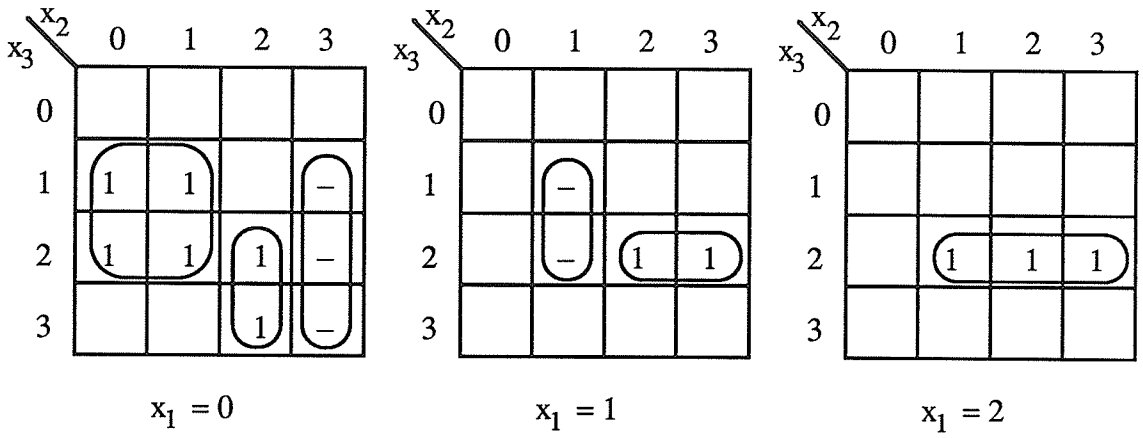


Figure 6.11 Function used in Example 6.11.

Example 6.11. Consider the function shown in Figure 6.11. The function can be represented by the following cubes.

OnList	DcList
001–0111–0010	100–0001–0111
010–0011–0010	010–0100–0110
100–1100–0110	
100–0100–0011	

The initial OnList contains the following information.

	Id #	cube	Kind	PIKind	Intersections
	1	001–0111–0010	Implicant	NonEssential	[]
	2	010–0011–0010	Implicant	NonEssential	[]
	3	100–1100–0110	Implicant	NonEssential	[]
	4	100–0100–0011	Implicant	NonEssential	[]
dc	5	100–0001–0111	Implicant	NonEssential	[]
dc	6	010–0100–0110	Implicant	NonEssential	[]

The consensus cubes are obtained in the following order. Cubes 1 and 2 yield cube 7 (cube 2 is deleted). Cubes 7 and 4 yield cube 8. Cubes 8 and 3 yield cube 9. Cubes 9 and 5 yield cube 10 (cube 9 is deleted). Cubes 10 and 7 yield cube 11 (cubes 7 and 8 are

deleted).

	Id #	cube	Kind	PIKind	Intersections
	11	111–0011–0010	Implicant	NonEssential	[10]
	10	100–1111–0010	Implicant	NonEssential	[11]
	9	100–1110–0010	Deleted	NonEssential	[]
	8	111–0010–0010	Deleted	NonEssential	[]
	7	011–0011–0010	Deleted	NonEssential	[]
	1	001–0111–0010	Implicant	NonEssential	[]
	2	010–0011–0010	Deleted	NonEssential	[]
	3	100–1100–0110	Implicant	NonEssential	[]
	4	100–0100–0011	Implicant	NonEssential	[]
dc	5	100–0001–0111	Implicant	NonEssential	[]
dc	6	010–0100–0110	Implicant	NonEssential	[]

The consensus of cubes 11 and 6 yields cube 12. Cubes 12 and 10 yield cube 13 (cube 12 is deleted). Cubes 13 and 1 yield cube 14 (cubes 13, 11, and 1 are deleted). The expansion of cube 14 terminates — it is found to be essential. The expansion of cube 10 terminates. Cube 10 is not essential.

	Id #	cube	Kind	PIKind	Intersections
dc	14	111–0111–0010	PICandidate	Essential	[3,4,10]
	13	110–0111–0010	Deleted	NonEssential	[]
	12	010–0111–0010	Deleted	NonEssential	[]
	11	111–0011–0010	Deleted	NonEssential	[]
	10	100–1111–0010	PICandidate	NonEssential	[3,4,14]
	1	001–0111–0010	Deleted	NonEssential	[]
	3	100–1100–0110	Implicant	NonEssential	[10,14]
	4	100–0100–0011	Implicant	NonEssential	[10,14]
dc	5	100–0001–0111	Implicant	NonEssential	[]
dc	6	010–0100–0110	Implicant	NonEssential	[]

Cube 3 is expanded next. Cubes 3 and 6 yield cube 15 (cube 6 is deleted). Cube 15 is an non-essential prime implicant. The expansion of cube 3 terminates. Cube 3 is essential (it

dominates cube 15).

	Id #	cube	Kind	PIKind	Intersections
	15	110–0100–0110	Deleted	NonEssential	[]
dc	14	111–0111–0010	PICandidate	Essential	[3,4]
	10	100–1111–0010	Deleted	NonEssential	[]
dc	3	100–1100–0110	PICandidate	Essential	[14]
	4	100–0100–0011	Implicant	NonEssential	[14]
dc	5	100–0001–0111	Implicant	NonEssential	[]
dc	6	010–0100–0110	Deleted	NonEssential	[]

The expansion of cube 4 (consensus with cube 5) yields cube 16 (cube 4 is deleted). Cubes 16 and 3 yield cube 17. Cube 17 is not essential. Cube 16 is essential (it dominates cube 17).

	Id #	cube	Kind	PIKind	Intersections
	17	100–1111–0010	Deleted	NonEssential	[]
dc	16	100–0011–0011	PICandidate	Essential	[14]
dc	14	111–0111–0010	PICandidate	Essential	[3,16]
dc	3	100–1100–0110	PICandidate	Essential	[14]
	4	100–0100–0011	Deleted	NonEssential	[]
dc	5	100–0001–0111	Implicant	NonEssential	[]

Finally, the minimal solution is given below.

	Id #	cube	Kind	PIKind	Intersections
dc	16	100–0011–0011	PICandidate	Essential	[14]
dc	14	111–0111–0010	PICandidate	Essential	[3,16]
dc	3	100–1100–0110	PICandidate	Essential	[14]

6.5 REMARKS

The algorithms presented in Chapters 3 to 5 are easily extended to accommodate window literals. In each case, only the basic definitions, such as adjacency, expandable minterms, consensus, etc., required some changes. The main thrust of the algorithms is left unchanged.

Tirumalai and Butler [TIR88] compared four different minimization algorithms using the truncated SUM operator and window literals. Minimization results from a set of 7,000 randomly generated four-valued, two-variable functions were compared. The extension of the direct cover algorithm presented here produced better results than the examples in [BUT88] which used the previously reported extension [TIR88]. It would be interesting to compute the results for all 7,000 functions using DCM^W and compare them with the results reported in [TIR88].

Tirumalai and Butler [TIR88] conjecture that no more than 10 implicants are needed in a minimal sum-of-products expression of a four-valued, two-variable function, using the truncated SUM and window literals. A function which needs 12 implicants is given in Example 6.12.

Example 6.12. Consider the function shown in Figure 6.12. A minimum sum-of-products expression, shown below, consists of 12 terms

$$F(x_1, x_2) = 1\,x_1^{0}x_2^{0}\,\lozenge\, 1\,x_1^{0}x_2^{3}\,\lozenge\, 1\,x_1^{0}x_2^{0}\,\lozenge\, 1\,x_1^{2}x_2^{0}\,\lozenge\, 1\,x_1^{3}x_2^{0}\,\lozenge\, 3\,x_1^{2}x_2^{1}\,\lozenge$$

$$3\,x_1^{1}x_2^{2}\,\lozenge\, 1\,x_1^{3}x_2^{3}\,\lozenge\, 1\,x_1^{0}x_2^{3}\,\lozenge\, 1\,x_1^{0}x_2^{3}\,\lozenge\, 1\,x_1^{0}x_2^{3}\,\lozenge\, 1\,x_1^{3}x_2^{3}$$

Essentially, one implicant is needed for each non-zero minterm. It is easy to see that no minterm can be covered by extending adjacent minterms.

x_2 \ x_1	0	1	2	3
0	3	1	2	3
1	1		3	
2		3		1
3	3	2	1	3

Figure 6.12 A function which needs 12 implicants.

Chapter 7

CONCLUSION

This thesis provides a comprehensive treatment of multiple-valued logic minimization. The truncated SUM operator as well as the maximum operator are employed by the proposed algorithms. The algorithms presented in Chapters 3 to 5 use the generalized literal operation. However, the algorithms are easily adapted to handle window literals as shown in Chapter 6. This demonstrates the flexibility of the algorithms. Flexibility is an important feature since new literal operators may come into use as alternative technologies evolve.

Algorithms which minimize multiple-valued logic functions can be used to minimize the multiple-output problem. The multiple-output problem is transformed into a single multiple-valued function with a binary output. Several examples have been given to illustrate this approach.

The direct cover algorithm presented in Chapter 3 is not suitable to solve the binary multiple-output problem. In binary logic, the truncated SUM is equivalent to the OR operation. For two reasons it is not efficient to use the DCM algorithm to solve this problem. First, it is known that a minimal solution consists of prime implicants only. The DCM algorithm considers all implicants. Second, a break introduced with a minterm with value r - 1 (the only non-zero value in a binary function) is not counted. Therefore, new breaks will never be introduced.

Directed search minimization (Chapter 4) offers two significant advantages over traditional minimization algorithms. First, it has the ability to detect essential and pseudo-essential prime implicants early during the generation process. Second, not all prime implicants are necessarily generated. Unfortunately, cycles must be resolved using

traditional techniques and it must start with a list of minterms. The first disadvantage was diminished by adding a heuristic which limits the growth of the tree and solves the cycle problem before all prime implicants are generated. The results are not always optimal but are, in general, very good.

The recursive consensus minimization algorithm presented in Chapter 5 combines the advantages of the directed search minimization with the advantage of the iterated consensus. The starting point is a list of terms, which are not necessarily minterms. The recursive expansion of terms leads to an early generation of prime implicants. With the additional information kept along with each term, it is possible to detect essential and pseudo-essential prime implicants during the generation process.

The strength of the proposed algorithms lies in their simplicity. The heuristics employed are easily understood. All algorithms have been implemented as computer programs. It is worth noting that the coding of the algorithms is straightforward. These programs can be integrated into CAD software systems.

PLAs have found wide application in VLSI implementations. The advantages of structured implementations leads to the conclusion that multiple-valued logic implementation will take a PLA-like structure. In fact, several multiple-valued logic PLAs have been proposed [SAS86a, KER86, TIR84]. All results produced by the algorithms are in sum-of-products form and are thus geared towards a PLA implementation.

At this time, the practical applications of multiple-valued logic minimization are limited to research implementations of multiple-valued logic circuits and to solve the binary multiple-output problem. Nevertheless, the benefits of efficient multiple-valued logic minimization algorithms will become evident with the advances in multiple-valued logic circuit realizations. Etiemble and Israël [ETI88] presented a critical comparison of binary and multi-valued integrated circuits. According to their research, multiple-valued logic will not replace binary implementations, but the implementation of special purpose multiple-valued circuits offers a significant advantage over their binary counterparts. Furthermore,

the advances in optoelectronics research [HUR86], will make multiple-valued logic more attractive.

Several questions remain unanswered. For example, how good are the results produced by the DCM algorithm? Until an algorithm which will find minimal solutions in reasonable time exists, this question will remain unanswered. For some problems, where the answers were known, DCM did produce minimal results. The complexity is too high for any four-valued function with more than 5 inputs. The search for better, or additional, heuristics to be used in the DCM algorithm continues.

It may be possible to improve the heuristics used by the directed search algorithm. Other ways to limit the growth of the expansion trees have been investigated. Limiting the size of the expansion tree was the best alternative.

The initial list of terms presented to the RCM procedure is unordered. Through observation it was found that the order in the list has an effect on the efficiency of the algorithm. Clearly it is advantageous to generate the essential and pseudo-essential prime implicants first. Moreover, if the function contains disjoint cycles, it is beneficial to detect and solve them independently.

Testing is an important facet of digital logic design. The generation of test vectors has traditionally been divorced from the minimization process. Each product term in a sum-of-products expression which has been minimized using the RCM algorithm has an associated list of intersecting terms. This information is valuable for the generation of test vectors. Further research is needed to fully exploit this information.

Appendix A

In this appendix, it will be shown that any function with multiple-valued inputs and outputs can be mapped into a set of binary-valued functions with multiple-valued inputs. Note that the definitions of literals and product terms are, of necessity, different from those used in the body of the thesis.

Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of n input variables. Define the set $P = \{1, 2, \ldots R - 1\}$ which represents the values that the variable x_i can assume. An R-valued function *f* is a mapping

$$f: P \times P \times \ldots \times P \rightarrow P$$

Let x_i be an input variable and let S_i be a subset of P. The literal function is defined as follows:

$$x_i^{S_i} = \begin{cases} R - 1 & \text{if } x_i \in S_i \\ 0 & \text{if } x_i \notin S_i \end{cases}$$

A *product term*

$$c\, x_1^{S_1} x_2^{S_2} \ldots x_n^{S_n}$$

is defined to be the minimum of the literals and the constant $c \in \{1, 2 \ldots R - 1\}$. If a term contains $S_i = P$ the term is said to be independent of x_i. An independent variable may be omitted from the term. The *sum* of product terms is defined to be the maximum of the terms. Any function can be written as a *sum-of-products*.

Let $c_{ij}Q_{ij}$ be a product term where $Q_{ij} = x_1^{S_{ij1}} x_2^{S_{ij2}} \cdots x_n^{S_{ijn}}$ and $c_{ij} = j$. Any

sum-of-products can be written in the following form: $1(Q_{11} + Q_{21} + \ldots + Q_{q_1 1}) + 2(Q_{12} + Q_{22} + \ldots + Q_{q_2 2}) + \ldots + R - 1(Q_{1r-1} + Q_{2r-1} + \ldots + Q_{q_{r-1} r-1})$. Note that $Q_{1i} + Q_{2i} + \ldots + Q_{q_i i}$ is a binary function, since it can take on either the value 0 or R - 1.

To find the minimal expression for $i(Q_{1i} + Q_{2i} + \ldots + Q_{q_i i})$ the ON-set will consist of all minterms which are equal to i; the OFF-set will contain all minterms which are less than i; the DC-set is the union of the DC-set of the function and all minterms which are greater than i. Therefore, the minimization of an R-valued function is equivalent to the minimization of R - 1 multiple-valued input, binary output functions.

REFERENCES

[ABD86] M. H. Abd-El Barr, Z. G. Vranesic, and S. G. Zaky, "Synthesis of MVL functions for CCD implementation," *Proceedings of the 16th International Symposium on Multiple-Valued Logic*, May 1986, pp. 116 - 127.

[ALL84] C. M. Allen and D. D. Givone "The Allen-Givone implementation oriented algebra," in *Computer Science and Multiple-Valued Logic*, D. C. Rine Ed., 2nd Ed., North Holland, New York 1984, pp. 268 - 288.

[BAR61] T. C. Bartee, "Computer design of multiple-output logic networks," *IRE Transactions on Electronic Computers*, Vol. EC-10, 1961, pp. 21 - 30.

[BEN85] E. A. Bender, J. T. Butler, and H. G. Kerkhoff, "Comparing the SUM with the MAX operator for use in four-valued PLA's," *Proceedings of the 15th International Symposium on Multiple-Valued Logic*, May 1985, pp. 30 - 35.

[BES79] P. W. Besslich, "Anatomy of Boolean-function simplification," *Computers and Digital Techniques*, Vol. 2, No. 1, February 1979, pp. 7 - 12.

[BES83] P. W. Besslich, "Efficient computer method for EXOR logic design," *IEE Proceedings*, Vol. 130, Part E, November 1982, pp. 203 - 206.

[BES86] P. W. Besslich, "Heuristic Minimization of MVL functions: a direct cover approach," *IEEE Transactions on Computers*, Vol. C-35, February 1986, pp. 134 - 144.

[BES87] P. W. Besslich and H. Bässmann, "Synthesis of exclusive-OR logic functions using spectral techniques," preprint.

[BRA82] R. K. Brayton, J. D. Cohen, G. D. Hachtel, B. M. Trager, and D. Y. Y. Yun, "Fast recursive Boolean function manipulation," *Proceedings of the 1982 International Symposium on Circuits and Systems*, 1982, pp. 58 - 62.

[BRA84] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, Boston, 1884.

[BUT88] J. T. Butler, private communication.

[CHA78] A. K. Chandra and G. Markowsky, "On the number of prime implicants," *Discrete Mathematics 24*, 1978, pp. 7 - 11

[COM84] D. J. Comer, *Digital Logic and State Machine Design,* Holt, Rinehart and Winston, New York, 1984.

[DAG86] M. R. Dagenais, V. K. Agarwal, and N. C. Rumin, "McBOOLE: A new procedure for exact logic minimization," *IEEE Transactions on Computer Aided Design*, Vol. CAD-5, January 1986, pp. 229 - 238.

[DUE86] G. W. Dueck and D. M. Miller, "A 4-valued PLA using the MODSUM," *Proceedings of the 16th International Symposium on Multiple-Valued Logic*, May 1986, pp. 232 - 240.

[DUE87] G. W. Dueck and D. M. Miller, "A direct cover MVL minimization using the truncated sum," *Proceedings of the 17th International Symposium on Multiple-Valued Logic*, May 1987, pp. 221 - 226.

[DUE88] G. W. Dueck and D. M. Miller, "Directed search minimization of multiple-valued functions," *Proceedings of the 18th International Symposium on Multiple-Valued Logic*, May 1988, pp. 218 - 225.

[DUN59] B. Dunham and R. Fridshal, "The problem of simplifying logical expressions," The Journal of Symbolic Logic, Vol. 24, March 1959, pp. 17 - 19.

[ETI88] D. Etiemble and M. Israël, "Comparison of binary and multivalued ICs according to VLSI criteria," *Computer,* April 1988, pp. 28 - 42.

[FLE75] H. Fleisher and L. I. Maisel, "An introduction to array logic," *IBM Journal of Research and Development*, March 1975, pp. 98 - 109.

[HON74] S. J. Hong, R. G. Cain, and D. L. Ostapko, "MINI: A heuristic approach for logic minimization," *IBM Journal of Research and Development*, September 1974, pp. 443 - 458.

[HUR84] S. L. Hurst, "Multiple-valued logic - Its status and its future," *IEEE Transactions on Computers*, Vol. C-33, December 1984, pp. 1160 - 1179.

[HUR86] S. L. Hurst, "A survey: developments in optoelctronics and its applicability to multiple-valued logic," *Proceedings of the 16th International Symposium on Multiple-Valued Logic*, May 1986, pp. 179 - 188.

[IGA79] Y. Igarashi, "An improved lower bound on the Maximum number of prime implicants," *The Transactions of the IECE of Japan*, Vol. E62, June 1979, pp. 389 - 394.

[JOH87] E. L. Johnson and M. A. Karim, *Digital Design A Pragmatic Approach,* PWS, Boston, 1987.

[KAM88] M. Kameyama, S. Kawahito, and T. Higuchi, "A multiplier chip with multiple-valued bidirectional current-mode logic circuits," *Computer,* April 1988, pp. 43 - 56.

[KAR53] M. Karnaugh, "The map method for synthesis of combinational logic circuits," *AIEE Transactions, part I Communication and Electronics*, Vol. 72, November 1953, pp. 593 - 599.

[KER82] H. G. Kerkhoff and H. A. J. Robroek, " The logic design of multiple-valued logic functions using charge coupled devices," *Proceedings of the 12th International Symposium on Multiple-Valued Logic*, May 1982, pp. 34 - 44.

[KER86] H. G. Kerkhoff and J. T. Butler, "Design of a high-radix programmable logic array using profiled peristaltic charge-coupled devices," *Proceedings of the 16th International Symposium on Multiple-Valued Logic*, May 1986, pp. 128 - 136.

[KER84] H. G. Kerkhoff, "Theory, design and applications of digital charge-coupled devices," Ph.D. Thesis, University of Twente, Enschede, Netherlands, 1984.

[LEE83] J. K. Lee and J. T. Butler, "Tabular methods for the design of CCD multiple-valued circuits," *Proceedings of the 13th International Symposium on Multiple-Valued Logic*, May 1983, pp. 162 - 170.

[MCC56] E. J. McCluskey, "Minimization of boolean functions," *The Bell System Technical Journal*, November. 1956, pp. 1417 - 1444.

[MCC79] E. J. McCluskey, "Logic design of multivalued I^2L logic circuits," *IEEE Transactions on Computers*, Vol. C-28, August 1979, pp. 546 - 559.

[MCM86] C. McMullen and J. Shearer, "Prime implicants, minimum covers, and the complexity of logic simplification," *IEEE Transactions on Computers*, Vol. C-35, August 1986, pp. 761 - 762.

[MIL79] D. M. Miller and J. C. Muzio, "On the minimization of many-valued functions," *Proceedings of the 9th International Symposium on Multiple-Valued Logic*, May 1979, pp. 294 - 299.

[MOR70] E. Morreale, "Recursive operators for prime implicant and irredundant normal form determination," *IEEE Transactions on Computers*, Vol. C-19, June 1970, pp. 504 - 509.

[MOT60] T. H. Mott, "Determination of the irredundant normal forms of a truth function by iterated consensus of the prime implicants," *IRE Transactions on Electronic Computers*, June 1960, pp. 245 - 252.

[MUL54] D. E. Muller, "Application of Boolean algebra to switching circuit design and error detection," *IRE Transactions on Electronic Computers*, September 1954, pp. 6 - 12.

[MUZ86] J. C. Muzio and T. C. Wesselkamper, *Multiple-Valued Switching Theory*, Adam Hilger, Boston, 1986.

[PAP79] G. Papakonstantiou, "Minimization of modulo-2 sums of products for switching functions," *IEEE Transactions on Computers*, Vol. C-28, February 1979, pp. 163 - 167.

[POM81] G. Pomper and J. R. Armstrong, "Representation of multivalued functions using the direct cover method," *IEEE Transactions on Computers*, Vol. C-30, September 1981, pp. 674 - 679.

[POS21] E. L. Post, "Introduction to a general theory of elementary propositions," *American Journal of Mathematics*, Vol. 43, 1921, pp. 163 - 185.

[QUI52] W. V. Quine, "The problem of simplifying truth functions," *American Mathematical Monthly*, Vol. 59, October. 1952, pp. 521 - 531.

[QUI55] W. V. Quine, "A way to simplify truth functions," *American Mathematical Monthly*, Vol. 62, November. 1955, pp. 627 - 631.

[REE54] I. S. Reed, "A class of multiple-error-correcting codes and the decoding scheme," *IRE Transactions on Information Theory*, IT-4, 1954, pp. 38 - 49.

[RED72] S. M. Reddy, "Easily testable realizations for logic functions," *IEEE Transactions on Computers*, C-21, November 1972, pp. 1183 - 1188.

[RHY77] V. T. Rhyne, P. S. Noe, M. H. McKinney, and U. W. Pooch, "A new technique for the fast minimization of switching functions," *IEEE Transactions on Computers*, C-26, August 1977, pp. 757 - 764.

[RUD87] R. Rudell and A. L. Sangiovanni-Vincentelli, "Multiple-valued minimization for PLA optimization," *Proceedings of the 17th International Symposium on Multiple-Valued Logic*, May 1987, pp. 198 - 208.

[SAL79] K. K. Saluja and E. H. Ong, "Minimization of Reed-Muller canonic expansion," *IEEE Transactions on Computers*, Vol. C-28, June 1979, pp. 535 - 537.

[SAS78] T. Sasao, "An application of multiple-valued logic to a design of programmable logic arrays," *Proceedings of the 8th International Symposium on Multiple-Valued Logic*, May 1978, pp. 65 - 72

[SAS86a] T. Sasao, "On the optimal design of multiple-valued PLA's," *Proceedings of the 16th International Symposium on Multiple-Valued Logic*, May 1986, pp. 214 - 223.

[SAS86b] T. Sasao and P. W. Besslich, "On the the complexity of MOD-2 sum PLA's," Technical Paper FTS86-17, Institute of Electronics, Communication and Information Engineering of Japan.

[SER84] M. Serra, "Directed search minimization of multiple-output networks," M.Sc. Thesis, University of Victoria, available as Technical Report DCS-42-IR, University of Victoria, Department of Computer Science, May 1984.

[SMI84] W. R. Smith III, "Minimization of multivalued functions," in *Computer Science and Multiple-Valued Logic*, D. C. Rine Ed., 2nd Ed., North Holland, New York 1984, pp. 227 - 267.

[SU84] S. Y. H. Su and P. T. Cheung, "Computer simplification of multi-valued switching functions," in *Computer Science and Multiple-Valued Logic*, D. C. Rine Ed., 2nd Ed., North Holland, New York 1984, pp. 195 - 226.

[TIR84] P. Tirumalai and J. T. Butler, "On the realization of multiple-valued functions using CCD PLA's," *Proceedings of the 14th International Symposium on Multiple-Valued Logic*, 1984, pp. 33 - 42.

[TIR88] P. Tirumalai and J. T. Butler, "Analysis of minimization algorithms for multiple-valued programmable logic arrays," *Proceedings of the 18th International Symposium on Multiple-Valued Logic*, 1988, pp. 226 - 236.

[TIS67] P. Tison, "Generalization of consensus theory and application to the minimization of boolean functions," *IEEE Transactions on Electronic Computers*, Vol. EC-16, August 1967, pp. 446 - 456.

[VRA70] Z. G. Vranesic, E. S. Lee, and K. C. Smith, "A many valued algebra for switching systems," *IEEE Transactions on Computers*, C-19, October 1970, pp. 964 - 971.

[WU82] X. Wu, X. Chen, and S. L. Hurst, "Mapping of Reed-Muller coefficients and the minimisation of exclusive OR-switching functions," *IEE Proceedings*, Vol. 129, Part E, January 1982, pp. 15 - 20.

[ZHA84] Y. Z. Zhang and P. J. W. Rayner, "Minimisation of Reed-Muller polynomials with fixed polarity," *IEE Proceedings*, Vol. 131, Part E,September 1984, pp. 177 - 186.