EXTENSIONS OF TWO-LEVEL MINIMIZATION METHODS WITH APPLICATIONS TO MULTI-LEVEL LOGIC SYNTHESIS

by

Abdul A. Malik

Memorandum No. UCB/ERL M89/132

14 December 1989

EXTENSIONS OF TWO-LEVEL MINIMIZATION METHODS WITH APPLICATIONS TO MULTI-LEVEL LOGIC SYNTHESIS

by

Abdul A. Malik

Memorandum No. UCB/ERL M89/132

14 December 1989

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

EXTENSIONS OF TWO-LEVEL MINIMIZATION METHODS WITH APPLICATIONS TO MULTI-LEVEL LOGIC SYNTHESIS

by

Abdul A. Malik

Memorandum No. UCB/ERL M89/132

14 December 1989

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Extensions of Two-level Minimization Methods with Applications to Multi-level Logic Synthesis

Abdul A. Malik

University of California
Berkeley, California

Department of Electrical Engineering
and Computer Sciences

Abstract

Some recent developments in multi-level synthesis have renewed interest in two-level minimization. This thesis makes contributions in the area of two-level minimization so that it can be used efficiently as an aid to multi-level logic optimization. In particular the following two problems are addressed.

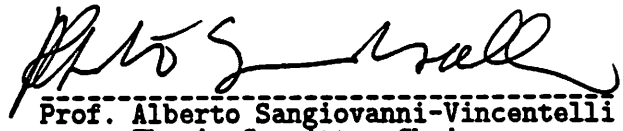
1. A commonly used heuristic algorithm for two-level logic minimization uses the offset of the function. There are functions with reasonable size onsets and don't care sets that have very large offsets. Minimization techniques that use the offsets are not very efficient with such functions. When two-level minimization was used to obtain local optimization in multi-level network, functions with large offsets were encountered frequently.

   An important contribution of this thesis is to introduce the notion of *reduced offsets*. It is shown that reduced offsets can be used in a similar way as the exact offset is used. The quality of minimization is maintained because the overall algorithm remains unchanged. Several reduced offsets may need to be computed instead of a single exact offset but each is usually much smaller than the exact offset and can be obtained much faster. Some experimental results are presented to show that for many functions with large offsets, the CPU time decreases when reduced offsets are used.

2. The generation of factored forms from two-level representaions is an important operation in multi-level minimization. The objective is to obtain factored forms with a small number of literals so that the number of literals in the multi-level network is minimized. Traditionally, the two level representation to be factored is treated as an algebraic polynomial and algebraic factoring techniques are used. Such approaches exclude Boolean operations and search only part of the solution space. In this thesis, a

new factoring algorithm is presented that uses Boolean properties and often produces better factored forms which are not possible to obtain with algebraic methods.

The algorithms presented in this thesis have been implemented in Two-level minimization program ESPRESSO and multi-level minimization environment MIS.



Prof. Alberto Sangiovanni-Vincentelli
Thesis Committee Chairman

Acknowledgements

I feel indebted to my research advisor professor Alberto Sangiovanni-Vincentelli for persuading me to do my Ph.D dissertation in the area of Logic Synthesis and for his continued guidance during the course of this dissertation. Professor Alberto Sangiovanni-Vincentelli and professor Richard Newton have been my research advisors for the last two and a half years. During that time, I have benefited from their encouragement, support and valuable advice.

I consider it a great privilege to have the opportunity to work very closely with professor Robert Brayton. One of my memorable experiences at Berkeley is the many exciting and thought provoking technical discussions that I had with professor Brayton. I am very grateful to professor Brayton for taking interest in my research.

My association with professors Sangiovanni, Brayton and Newton made the last two years, the most rewarding years of my entire college education. I am grateful to all three of them. I wish I had come into contact with them a little earlier during the course of my graduate work at Berkeley.

I am also thankful to professor Charles Stone of the Department of Statistics for his willingness to be on my Qualifying Exam and Thesis committees.

Thanks are also due to many present and past students in the department of Electrical Engineering and Computer Sciences for their help and for making my stay at Berkely a pleasant experience. In particular, I would like to mention Andrea Casotto, David Harrison, Steve Lewis, Guy Marong, Cho Moon, Rajeev Murgai, Tom Quarles, Omid Razavi, Richard Rudell, Res Saleh, Hamid Savoj, Ellen Sentovich, Narendra Shenoy, Jyuo-Min Shyu, Kanwar Jit Singh, Rick Spickelmier, Herve Touati and Albert Wang. I am also thankful to computer support staff members Brad Krebs and Kurt Pires.

Last but not the least, I acknowledge the partial funding from DARPA under contract number N00039-C-0182 and from NSF under contract number UCB-BSI6421.

Contents

Acknowledgements i

Table of Contents ii

List of Figures v

List of Tables vi

1 Introduction 1

2 Basic Definitions 3

2.1 Binary-valued Functions . . . 3

2.1.1 Completely-specified Function . . . 3

2.1.2 Incompletely-specified Function . . . 4

2.1.3 Tautology . . . 4

2.1.4 Minterm . . . 4

2.1.5 Onset, Don't Care Set and Offset . . . 4

2.1.6 Literal . . . 5

2.1.7 Product Term or Cube . . . 5

2.1.8 Cube-representation . . . 5

2.1.9 Universal Cube . . . 5

2.1.10 Containment of One Cube in Another . . . 6

2.1.11 Prime Cube . . . 6

2.1.12 Implicant and Prime Implicant . . . 6

2.1.13 Super Cube . . . 6

2.1.14 Overexpanded Cube . . . 6

2.1.15 Distance between two Cubes . . . 7

2.1.16 Orthogonality between two Cubes . . . 7

2.1.17 Intersection of two Cubes . . . 7

2.1.18 Sum-of-products . . . 7

2.1.19 Unate Function . . . 8

2.1.20 Unate Sum-of-products Representation . . . 8

2.1.21 Cofactor . . . 9

2.1.22 Shannon's Expansion . . . 9

2.2 Multi-valued Functions 9
2.2.1 Mv-function 9
2.2.2 Minterm 10
2.2.3 Onset, Don't Care Set, and Offset 10
2.2.4 Literal 10
2.2.5 Product Term or Cube 10
2.2.6 Containment of one Cube in another 11
2.2.7 Distance between two Cubes 12
2.2.8 Intersection of two Cubes 12
2.2.9 Positional Notation 12
2.2.10 Cover 13
2.2.11 Weakly Unate Function 13
2.2.12 Weakly Unate Cover 13
2.2.13 Strongly Unate Function 13
2.2.14 Cofactor 14
2.2.15 Generalized Shannon's Expansion 14
2.3 Factored-forms 14
2.3.1 Algebraic Division 14
2.3.2 Co-kernel and Kernel 15
2.3.3 Factored-form 15
2.3.4 Tree Representation of Factored From 15
2.3.5 Factored-Form as Representation of a Completely-Specified Function 16
2.3.6 Boolean Factored-form Minimization 16
2.4 Multi-level Network 16

3 Background and Contributions 18

4 Reduced Offsets for Minimization of Binary-valued Functions 22
4.1 Introduction 22
4.2 Reduced Offset for a Cube 24
4.3 Reduced Offset for a Set of Cubes 25
4.4 Generalized Cube 27
4.5 Reduction Operator R_p and Reduced Offset 29
4.6 Properties of R_p 31
4.7 Properties of the Reduced Offset 33
4.8 Unate Recursive Algorithm for the Reduced Offset for a Cube 35
4.8.1 Applying Reduction Operator to Unate Sums-of-Products 36
4.8.2 Merging 37
4.9 Overexpanded Cube 41
4.10 Improving Overexpanded Cube Computation 45
4.11 Using the Overexpanded Cube for Computing the Reduced Offset 48
4.12 Further Improvements in Computing the Reduced Offset 54
4.13 Limitations 59
4.14 Experimental Results 60

5 Reduced Offsets for Minimization of Multi-valued Functions 63
5.1 Introduction 63
5.2 Reduction operator R_p and Reduced Offset 64
5.3 Properties of R_p 67
5.4 Unate Recursive Algorithm for the Reduced Offset for a Cube 68
5.4.1 Recursive Shannon's Cofactoring 71
5.4.2 Applying Reduction Operator to Strongly Unate Sums-of-Products . 72
5.4.3 Merging 76
5.5 Overexpanded Cube 79
5.6 Using the Overexpanded Cube to find the Reduced Offset 82
5.7 Storing Cofactoring Tree 85
5.8 Experimental Results 89
5.9 Limitations 92
5.10 Conclusion 92

6 Boolean Minimization for Factored Forms 94
6.1 Introduction 94
6.2 Initial Factorization 96
6.3 Reducing Literals in Initial Factored Form 97
6.3.1 Expansion of g_1 and g_2 97
6.3.2 Removal of Cubes from g_1 and g_2 98
6.4 Choosing optimal $h = h_1 h_2 + \rho$ 99
6.5 Recursion 100
6.6 Example 103
6.7 Experimental Results 103
6.8 Conclusion 104

Bibliography 106

List of Figures

2.1 Tree representation of a factored-form. 17

4.1 Illustrating reduced offset 26
4.2 Table for intersecting two generalized cubes 28

List of Tables

4.1 Offset vs. reduced offsets for binaray valued functions. 62

5.1 Offset vs. reduced offsets for PLAs 91
5.2 Multi-valued reduced offsets vs. offset for Achilles' heel functions 92

6.1 Boolean factorization vs. Good factor and Quick factor 105

Chapter 1

Introduction

Ever since the advent of integrated circuits, they have been growing in complexity. Chips with more than a million transistors are now available. The increase in complexity of digital ICs was enabled by extensive use of computers in the design process. The field of study that addresses the use of computers for the design of ICs is known as *Computer Aided Design of Integrated Circuits* (CAD of ICs).

Integrated circuits can be divided into two groups: analog ICs and digital ICs. Analog integrated circuits work with data which is continuous in nature. Such circuits have rather limited applications. They are used in audio amplifiers, televisions, etc. Digital ICs deal with discretized data which is expressed as strings of 0s and 1s. Such circuits have wide range of applications. Continuous data is often discretized; processed using digital ICs; and finally converted back into continuous form. This techniques is an alternative to using analog circuits. In addition, digital ICs have a wide range of applications from children's toys to space shuttles.

An important part of the design of digital ICs is to obtain a description of the layout from the functional specification of the circuit. The functional specification is often in the form of logic equations. The circuit to be laid out must produce for any input, an output which conforms with the functional specification. The output must be produced within an allowable time once the input has been applied. In addition, the circuit must satisfy some testability requirements. The process of obtaining a layout description from functional specification is known as *logic synthesis*. Since the cost of the circuit increases with the area, the objective of logic synthesis is to obtain a circuit with the smallest area possible that meets the timing and testing requirements.

The description of the layout produced by logic synthesis may be in the form of interconnected modules (subcircuits). The implementation of the circuit as interconnected modules is known as multi-level implementation. The process of obtaining such an implementation from functional specification is known as *multi-level logic synthesis.* The layout description may be in the form of a Programmable Logic Array (PLA). PLA is a regular structure in which each path from the input to the output has at most two logic gates. Such an implementation is called two-level implementation. The process of obtaining a PLA based implementation is known as *two-level logic synthesis.* In addition to being a design style in itself, two-level logic synthesis has been successfully used as an aid in multi-level logic synthesis.

An aspect of logic synthesis is *logic minimization.* Given a representation for a functional specification in the form of a PLA or interconnected modules, logic minimization attempts to obtain another representation for the same functional specification whose layout would take less area than the original representation. Logic minimization can be two-level minimization or multi-level minimization depending on whether the representation used is in the form of PLA or interconnected modules.

The contribution of this thesis is to extend two-level logic minimization techniques so that they are more suitable for use in multi-level logic synthesis. Chapter 2 presents some basic definitions. Chapter 3 presents some historic background of two-level synthesis and explain the contributions of this thesis in more detail. Chapter 3 also presents the outline of the rest of the thesis.

Chapter 2

Basic Definitions

The purpose of this chapter is to present some basic definitions from the literature that will be used in the later chapters. Most of the definitions related to binary-valued functions are taken from [2]. The definitions needed to deal with multiple-valued functions are those used by Sasao [18,19,20,22,21] and later in ESPRESSO-MV [16]. Finally, definitions used in regard to factored-forms are those developed by Brayton et. el [4,5,6]. Any new definitions will be presented in each chapter as they are needed.

It is assumed that the reader is familiar with Boolean Algebra and the basic OR (+) and AND (*) operators defined for it. Such preliminaries are covered in most standard text books on Logic Design e.g [15]. First definitions related to binary-valued Boolean functions will be presented. They will be followed by definitions regarding multi-valued functions. Finally, some definitions about factored-forms will be presented. Many terms are common to binary-valued and multi-valued functions. Such terms are usually preceded with the phrase *binary-valued* or *multi-valued* if the ambiguity can not be resolved from the context.

2.1 Binary-valued Functions

2.1.1 Completely-specified Function

A completely specified function g with n inputs and m outputs is a mapping

$$g : B^n \rightarrow B^m$$

where $B = \{0,1\}$, B^n is called the domain and B^m is called the range of the function.

If $m = 1$, g is called a *completely-specified single-output function.* If $m > 1$, the functions is called a *completely specified multiple-output function.*

2.1.2 Incompletely-specified Function

An incompletely-specified function $\mathcal{F}$ with n inputs and m outputs is a mapping

$$h : B^n \rightarrow Y^m$$

where $B = \{0, 1\}$, and $Y = \{0, 1, *\}$. B^n is called the domain and Y^m is called the range of the function.

$\mathcal{F}$ is called an *incompletely-specified single-output function* if $m = 1$. It is called an *incompletely specified multiple-output function* if $m > 1$.

A completely-specified function is a special case of an incompletely-specified function which occurs when no minterm in the domain of the incompletely-specified function is mapped to $*$. The phrase *completely-specified* or *incompletely-specified* is usually omitted if the meaning is clear from the context.

2.1.3 Tautology

An incompletely-specified function $\mathcal{F} : B^n \rightarrow Y^m$ is called a tautology if $\forall s \in B^n$, $\mathcal{F}(s) = \{1\}^m$.

2.1.4 Minterm

A point in the domain of a completely or incompletely specified function is called a minterm of the function.

2.1.5 Onset, Don't Care Set and Offset

An incompletely-specified function $\mathcal{F} : B^n \rightarrow Y$ maps each minterm to 0, 1, or $*$. The onset f, don't care set d and offset r of $\mathcal{F}$ are sets of minterms as defined below.

$$\begin{aligned} f &= \{s \in B^n | \mathcal{F}(s) = 1\} \\ d &= \{s \in B^n | \mathcal{F}(s) = *\} \\ r &= \{s \in B^n | \mathcal{F}(s) = 0\} \end{aligned}$$

Customarily, each one of these sets is represented by a completely-specified function that maps a minterm to 1 if and only if the minterm belongs to the set.

f, d, and r partition B^n. A minterm in the don't care set is allowed to have either value 0 or 1. An incompletely-specified single-output function is often denoted by triplet (f, d, r). A completely-specified single-output function can be thought of as a special case of an incompletely-specified single-output function with $d = \emptyset$.

2.1.6 Literal

A literal is a variable or its complement. A literal $D(\overline{D})$ represents the set of all minterms for which variable D takes on value 1(0). D and $\overline{D}$ are complements of each other.

2.1.7 Product Term or Cube

A product term is a product or conjunction of one or more literals. A product term represents the intersection of sets of minterms represented by all of the literals in it. A product term in B^n with l literals is a hypercube of $n - l$ dimensions. Hence a product terms is also called a cube.

2.1.8 Cube-representation

An incompletely-specified single-output function $\mathcal{F} = B^n \rightarrow Y$ can be represented by a cube with n dimensions. Each vertex of the cube corresponds to a minterm of $\mathcal{F}$. Since each minterm is exactly in one of onset, don't care set, or offset, the vertex corresponding to the minterm is labeled as onset-vertex, don't care-vertex or offset-vertex.

2.1.9 Universal Cube

If all vertices of the cube representing an incompletely-specified single-output function are labeled as onset-vertices then the cube is called a universal cube. The incompletely-specified function in that case is a tautology.

EXAMPLE 2.1 :

Consider a function with 3 variables: x_1, x_2 and x_3. The product term $x_1\overline{x}_2x_3$ represents

$$\begin{aligned} & \{(x_1,x_2,x_3) \in B^3 | x_1 = 1\} \cap \{(x_1,x_2,x_3) \in B^3 | x_2 = 0\} \cap \{(x_1,x_2,x_3) \in B^3 | x_3 = 1\} \\ = \; & \{(x_1,x_2,x_3) \in B^3 | x_1 = 1 \; and \; x_2 = 0 \; and \; x_3 = 1\} \end{aligned}$$

The product term is a cube of dimension 0.

2.1.10 Containment of One Cube in Another

A cube c_2 contains c_1 ($c_1 \subseteq c_2$) if the set of minterms represented by c_1 is a subset of that represented by c_2. A minterm $s \in c_1$ if s is in the set of minterms represented by c_1. A cube c is contained in a completely-specified single-output function h ($c \subseteq h$) if $\forall s \in c$, $h(s) = 1$.

2.1.11 Prime Cube

A cube (product term) p is called a prime cube of $\mathcal{F} = (f, d, r)$ if $p \subseteq f \cup d$ and no other cube p' exists such that $p \subset p' \subseteq f \cup d$.

2.1.12 Implicant and Prime Implicant

Given a cube c and an incompletely-specified function $\mathcal{F} = (f, d, r)$, c is called an implicant of $\mathcal{F}$ if $c \subseteq f \cup d$. If c is a prime cube of $\mathcal{F}$ then it is called a prime implicant.

2.1.13 Super Cube

q is the super cube of a set of cubes, S, if $\forall p \in S$, $p \subseteq q$ and $q \subset q'$ for any other cube q' such that $\forall p \in S$, $p \subseteq q'$.

2.1.14 Overexpanded Cube

If S is the set of all prime cubes of $\mathcal{F} = (f, d, r)$ that contain cube $p \subseteq f \cup d$ then the super cube of S is called the overexpanded cube of p. In other words, the overexpanded is the smallest cube that contains all primes of $\mathcal{F}$ that contain p.

2.1.15 Distance between two Cubes

The distance between two cubes c_1 and c_2 denoted as $dist(c_1, c_2)$ is the number of literals in c_1 whose complements are present in c_2. Also, $dist(c_1, c_2) = dist(c_2, c_1)$.

EXAMPLE 2.2 :

$$c_1 = x_1\overline{x}_2x_3$$
$$c_2 = x_2\overline{x}_3x_5x_6$$

The distance between c_1 and c_2 is 2 because there are two literals $\overline{x}_2$ and x_3 in c_1 whose complements x_2 and $\overline{x}_3$ are present in c_2.

2.1.16 Orthogonality between two Cubes

Two cubes c_1 and c_2 are orthogonal to each other if $dist(c_1, c_2) = dist(c_2, c_1) > 0$.

2.1.17 Intersection of two Cubes

If two cubes are orthogonal to each other, their intersection or product is the null cube. Otherwise, the intersection consists of the product of all literals that appear in either cube.

2.1.18 Sum-of-products

A sum-of-products is a sum or disjunction of product terms. A sum-of-products represents the union of sets of minterms represented by all the product-terms in it. It also represents the completely-specified single-output function whose onset is that set of minterms.

EXAMPLE 2.3 :

Consider a function with three variables x_1, x_2 and x_3. The sum-of-products $\overline{x}_1x_3 + x_2\overline{x}_3$ represents

$$S = \{(x_1, x_2, x_3) \in B^3 | x_1 = 0 \; and \; x_3 = 1\} \cup \{(x_1, x_2, x_3) \in B^3 | x_2 = 1 \; and \; x_3 = 0\}$$

It also represents function $f : B^3 \rightarrow B$ such that

$$\begin{aligned} f(s) &= 1 \; if \; s \in S \\ &= 0 \; otherwise. \end{aligned}$$

Each minterm can be represented by a product term with n literals. Hence any $S \subseteq B^n$ can be represented by a sum-of-products.

An incompletely-specified single-output function $\mathcal{F} = (f, d, r)$ is represented by specifying completely-specified functions f, d and r which can be represented using sums-of-products. Usually, only f and d are specified. If r is needed, it can be computed by complementing $f \cup d$. Each output of a multiple-output function is a single-output function. A multiple-output function can be represented by its single-output functions using sums-of-products.

2.1.19 Unate Function

Consider a pair of minterms (s', s) of a completely-specified function g such that s' has literal $\overline{x}_i$ and s is obtained from s' by replacing literal $\overline{x}_i$ by x_i.

g is *monotone increasing in variable* x_i if $\forall(s', s)$, $g(s') \geq g(s)$ for each output.

g is *monotone decreasing in variable* x_i if $\forall(s', s)$, $g(s) \geq g(s')$ for each output.

g is *unate in variable* x_i if it is either monotone increasing or monotone decreasing in variable x_i.

g is a *unate function* if it is unate in all of its input variables.

2.1.20 Unate Sum-of-products Representation

A sum-of-products f is unate if for every variable x in it, f contains either x or $\overline{x}$ but not both.

EXAMPLE 2.4 :

$$\begin{aligned} f_1 &= A\overline{B} + \overline{B}C + AC \\ f_2 &= AB + \overline{A}C \end{aligned}$$

f_1 is unate but f_2 is not unate because f_2 has both A and $\overline{A}$.

A function represented by a unate sum-of-products is unate. Every unate function has at least one unate sum-of-products representation. However, the function represented by a non-unate sum-of-products is not necessarily a non-unate function.

2.1.21 Cofactor

The cofactor of a sum-of-products f with respect to literal $x_i(\overline{x}_i)$, denoted by $f_{x_i}(f_{\overline{x}_i})$, is obtained by substituting 1 (0) for x_i in f.

EXAMPLE 2.5 :

$$\begin{aligned} f &= A\overline{B} + \overline{B}C + \overline{A}C \\ f_A &= \overline{B} + \overline{B}C \\ f_{\overline{A}} &= \overline{B}C + C \end{aligned}$$

2.1.22 Shannon's Expansion

For a sum-of-products f with variable x

$$f = xf_x + \overline{x}f_{\overline{x}}$$

2.2 Multi-valued Functions

This section presents definitions regarding multi-valued functions and their properties which will be used later. For simplicity, *multi-valued* will be abbreviated as mv- if needed.

2.2.1 Mv-function

An mv-function with n inputs is defined as a mapping

$$\mathcal{F}: P_1 \times P_2 \times \cdots \times P_n \rightarrow Y$$

where $P_i = \{0, 1, \ldots, p_i - 1\}$ and p_i is the number of values that i^{th} variable may take on, $Y = \{0, 1, *\}$; $P_1 \times P_2 \times \cdots \times P_n$ is called the domain of the function.

2.2.2 Minterm

A point in the domain of an mv-function f is called a minterm of f.

2.2.3 Onset, Don't Care Set, and Offset

An incompletely-specified mv-function $\mathcal{F}$ maps each minterm to 0, 1, or $*$. The onset f, don't care set d and offset r of $\mathcal{F}$ are sets of minterms as defined below.

$$\begin{aligned} f &= \{s \in B^n | \mathcal{F}(s) = 1\} \\ d &= \{s \in B^n | \mathcal{F}(s) = *\} \\ r &= \{s \in B^n | \mathcal{F}(s) = 0\} \end{aligned}$$

A minterm in d is allowed to have either value 0 or 1. If $d = \emptyset$ then $\mathcal{F}$ is a completely-specified mv-function. An incompletely-specified mv-function is denoted by the triplet (f, d, r).

2.2.4 Literal

If X_i is the i^{th} variable of $\mathcal{F}$ and $S_i \subseteq P_i$ then $X_i^{S_i}$ is a literal. If $j \in S_i$, the literal is said to have value j in it. $X_i^{S_i}$ represents the set of minterms for which $X_i = j$ for any value $j \in S_i$. If $S_i = P_i$, the literal is called a *full literal* because the set of minterms represented by it is the entire domain of the function. If $S_i = \emptyset$, the literal is called an *empty literal.*

2.2.5 Product Term or Cube

A product term is a product or conjunction of one or more literals. It represents the intersection of the sets of minterms represented by literals in it. Without loss of generality, a product term related to a function of n variables can be represented as

$$X_1^{S_1} X_2^{S_2} \cdots X_n^{S_n}$$

where some of the literals in it may be full literals. It is customary to leave full literals out of the product terms. The product term p is said to *depend* on variable X_i if $X_i^{S_i}$ is not a full literal. If the cube has a null literal then it is called a null cube.

EXAMPLE 2.6 :

Consider the product term

$$X_1^{\{0,1\}} X_2^{\{1\}} X_3^{\{0,1,2\}}$$

in the domain $P_1 \times P_2 \times P_3$ where $p_1 = 3$; $p_2 = 2$; $p_3 = 3$. The product term has a full literal in variable X_3. It can also be represented without the full literal as

$$X_1^{\{0,1\}} X_2^{\{1\}}$$

This product term depends on variables X_1 and X_2 but not on variable X_3.

A product term is also called a cube. However, it does not represent a hypercube in the domain of the mv-function unlike a product term for a binary-valued function.

Let s be a minterm, c be a cube, and h be a completely specified mv-function. The definitions for $s \in c$ and $c \subseteq h$ are identical to those in the binary-valued case.

The definitions of *prime cube, super cube, overexpanded cube* and *sum-of-products* are also the same as for the binary-valued case. The properties discussed for binary sums-of-products in the end of section 2.1.18 also hold for the multi-valued sums-of-products.

2.2.6 Containment of one Cube in another

Consider two cubes

$$c_1 = X_1^{S_1} X_2^{S_2} \cdots X_n^{S_n}$$
$$c_2 = X_1^{T_1} X_2^{T_2} \cdots X_n^{T_n}$$

c_2 contains c_1 (written as $c_1 \subseteq c_2$) if $S_i \subseteq T_i$ for $1 \leq i \leq n$.

If neither c_1 nor c_2 is a null cube and if $c_1 \subseteq c_2$ then the set of minterms represented by c_1 is a subset of that represented by c_2. According to this definition, no null cube is automatically contained in every cube as would be the case for binary-valued cubes.

2.2.7 Distance between two Cubes

Consider two cubes

$$c_1 = X_1^{S_1} X_2^{S_2} \cdots X_n^{S_n}$$
$$c_2 = X_1^{T_1} X_2^{T_2} \cdots X_n^{T_n}$$

c_1 and c_2 are said to conflict in variable X_i for $1 \leq i \leq n$ if $S_i \cap T_i = \emptyset$. The distance between c_1 and c_2 denoted as $dist(c_1, c_2)$ is the number of variables in which c_1 and c_2 conflict. If $dist(c_1, c_2) > 0$ then c_1 and c_2 are orthogonal to each other.

EXAMPLE 2.7 :

For c_1 and c_2 mentioned above, let $n = 3$, $S_1 = \{0,1\}$, $S_2 = \{1,4\}$ and $S_3 = \emptyset$, $T_1 = \{2,3\}$, $T_2 = \{1,3\}$ and $T_3 = \{0,1\}$.

Since $S_1 \cap T_1 = \emptyset$, $S_2 \cap T_2 = \{1\} \neq \emptyset$ and $S_3 \cap T_3 = \emptyset$, $dist(c_1, c_2) = 2$.

2.2.8 Intersection of two Cubes

Consider two cubes

$$c_1 = X_1^{S_1} X_2^{S_2} \cdots X_n^{S_n}$$
$$c_2 = X_1^{T_1} X_2^{T_2} \cdots X_n^{T_n}$$

The intersection of c_1 and c_2 is

$$c_1 \cap c_2 = X_1^{S_1 \cap T_1} X_2^{S_2 \cap T_2} \cdots X_n^{S_n \cap T_n}$$

If c_1 and c_2 are orthogonal to each other then $c_1 \cap c_2$ has at least one null literal.

2.2.9 Positional Notation

Let $c = X_1^{S_1} X_2^{S_2} \cdots X_n^{S_n}$ be a product term. This product term can be represented just by the sets $S_1, S_2 \cdots S_n$. Since for $1 \leq i \leq n$, $S_i \subseteq P_i$ and P_i are finite sets with usually a small number of values p_i, it is convenient to represent S_i as bit vectors. Hence c can be represented by the following bit vector.

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

where $c_i^j = 1$ if $j \in S_i$, otherwise $c_i^j = 0$. This is called the positional cube notation. $c_i = c_i^0 c_i^1 \cdots c_i^{P_i-1}$ represents the binary vector for S_i. $|c_i|$ represents the number of 1's in c_i.

A sum-of-products is represented in positional notation as a matrix where each row corresponds to a product term and each column to some c_i^j.

2.2.10 Cover

A cover is a sum-of-products represented in positional notation [16]. A cover for an incompletely specified function $\mathcal{F} = (f, d, r)$ is a sum-of-products in positional notation that contains the entire f and possibly some part of d [2].

2.2.11 Weakly Unate Function

A completely-specified function f is weakly unate in variable X_i if $\exists j \in P_i$ such that changing the value of variable X_i form j to any other value $k \in P_i$ causes the function value to change from 0 to 1, if it changes at all.

If it can be deduced that a minterm s with $X_i = j$ is in the onset of f, then so is s' obtained by replacing j by some other value $k \in P_i$.

If f is weakly unate in all of its variables then it is a weakly-unate function.

2.2.12 Weakly Unate Cover

A cover is weakly-unate in variable X_i if there exists a $j \in P_i$ such that all cubes that depend on variable X_i contain a 0 in the column corresponding to j. If the cover is weakly unate in all of its variables then it is a weakly-unate cover.

A weakly-unate cover represents a weakly-unate function. However, a non-weakly-unate cover may still be that of a weakly-unate function.

2.2.13 Strongly Unate Function

A completely-specified mv-function f is strongly unate in variable X_i if the elements of P_i can be totally ordered via $\preceq$ such that changing the value of variable X_i from value j to some value other value k, such that $j \preceq k$, causes the function value to change from 0 to 1 if it changes at all. If f is strongly unate in X_i then it is also weakly unate in X_i. If f is strongly unate in all of its variables then f is a strongly-unate function.

2.2.14 Cofactor

The cofactor of a product term S with respect to another product term T (S_T) is empty if S and T are orthogonal. Otherwise, it is the product term $X_1^{S_1 \cup \overline{T}_1} X_2^{S_2 \cup \overline{T}_2} \cdots X_n^{S_n \cup \overline{T}_n}$ where $\overline{T}_i = \{j \in P_i | j \notin T_i\}$.

The cofactor of a sum-of-products g with respect to cube T (g_T) is the sum of cofactors of all the cubes in g with respect to T.

2.2.15 Generalized Shannon's Expansion

Let $c_0, c_1, \cdots c_{l-1}$ be product terms in the domain of a completely-specified mv-function f such that $\sum_{j=0}^{j=l-1} c_j = 1$ (i.e tautology) and $c_j c_k = \emptyset$ for $j \neq k$ then

$$f = \sum_{j=0}^{j=l-1} c_j f_{c_j}$$

A special case of generalized Shannon's expansion takes place when $c_j = X_i^{\{j\}}$ and $l = p_i$.

2.3 Factored-forms

2.3.1 Algebraic Division

The algebraic division of a binary-valued sum-of-products f by a cube c (f/c) is the quotient of dividing f by c while treating f as an algebraic polynomial. Each literal in f is treated as an algebraic variable and c as an algebraic term.

EXAMPLE 2.8 :

$$\begin{aligned} f &= A\overline{B}C + \overline{A}C + A\overline{B}D \\ c &= A\overline{B} \\ f/c &= C + D \end{aligned}$$

2.3.2 Co-kernel and Kernel

A co-kernel c of a binary-valued sum-of-products f is a cube such that f/c has at least two cubes and there are no literals common to each cube in it. f/c is called the kernel of f.

EXAMPLE 2.9 :

$$f = A\overline{B}CE + \overline{A}C + A\overline{B}DE$$

$c_1 = A\overline{B}$ is not a co-kernel because $f/c_1 = CE + DE$ has literal E in each cube. However, $c_2 = A\overline{B}E$ is a co-kernel and $f/c_2 = C + D$ is the corresponding kernel.

2.3.3 Factored-form

A factored-form is a sum-form or a product-form. A sum-form is a literal or a sum of two or more product-forms. A product-form is a literal or a product of two or more sum-forms.

EXAMPLE 2.10 :

$$A(B + (\overline{B} + C)E) + D$$

is a factored-form because it is a sum of two product-forms: $A(B + (\overline{B} + C)E)$ and D. The first one is a product-form because it is a product of two sum-forms: A and $B + (\overline{B} + C)E$. $B + (\overline{B} + C)E$ is sum of two product-forms: B and $(\overline{B} + C)E$. The later is a product of sum-forms: $\overline{B} + C$ and E.

2.3.4 Tree Representation of Factored From

A factored-form can be represented by a tree whose leafs represent literals that appear in the factored-form and whose remaining nodes represent intermediate sum-forms and product-forms. A node labeled (*) represents a product-form consisting of products of sum-forms represented by its children nodes. Similarly, a node labeled (+) represents a sum-form consisting of sum of product-forms represented by its children nodes.

EXAMPLE 2.11 :

The tree representation of the factored-form in in example 2.10 is shown in 2.3.4.

2.3.5 Factored-Form as Representation of a Completely-Specified Function

A factored from F contains a completely specified function f ($f \subseteq F$) if F evaluates to 1 for any minterm in the onset of f. The factored-form F represents f if in addition, F evaluates to 0 for any minterm in the offset of f.

2.3.6 Boolean Factored-form Minimization

Given a cover l and the don't care set d for an incompletely specified binary-valued function $\mathcal{F} = (f, d, r)$ with single output, Boolean factored-form minimization attempts to obtain a factored-form F with the minimum number of literals such that $f \subseteq F \subseteq f \cup d$.

2.4 Multi-level Network

A multi-level network is a Directed Acyclic Graph (DAG) that represents a completely-specified multiple-output binary-valued function. The source nodes of the DAG correspond to the inputs and the sink nodes correspond to the outputs of the function. Each intermediate-node represents a completely-specified single-output binary-valued function and corresponds to an intermediate variable that represents the function. If the function at node A depends on the variable corresponding to the function at node B then there is an edge from B to A in the DAG. For any input to the DAG, the outputs correspond to the mapping of the input by the function.

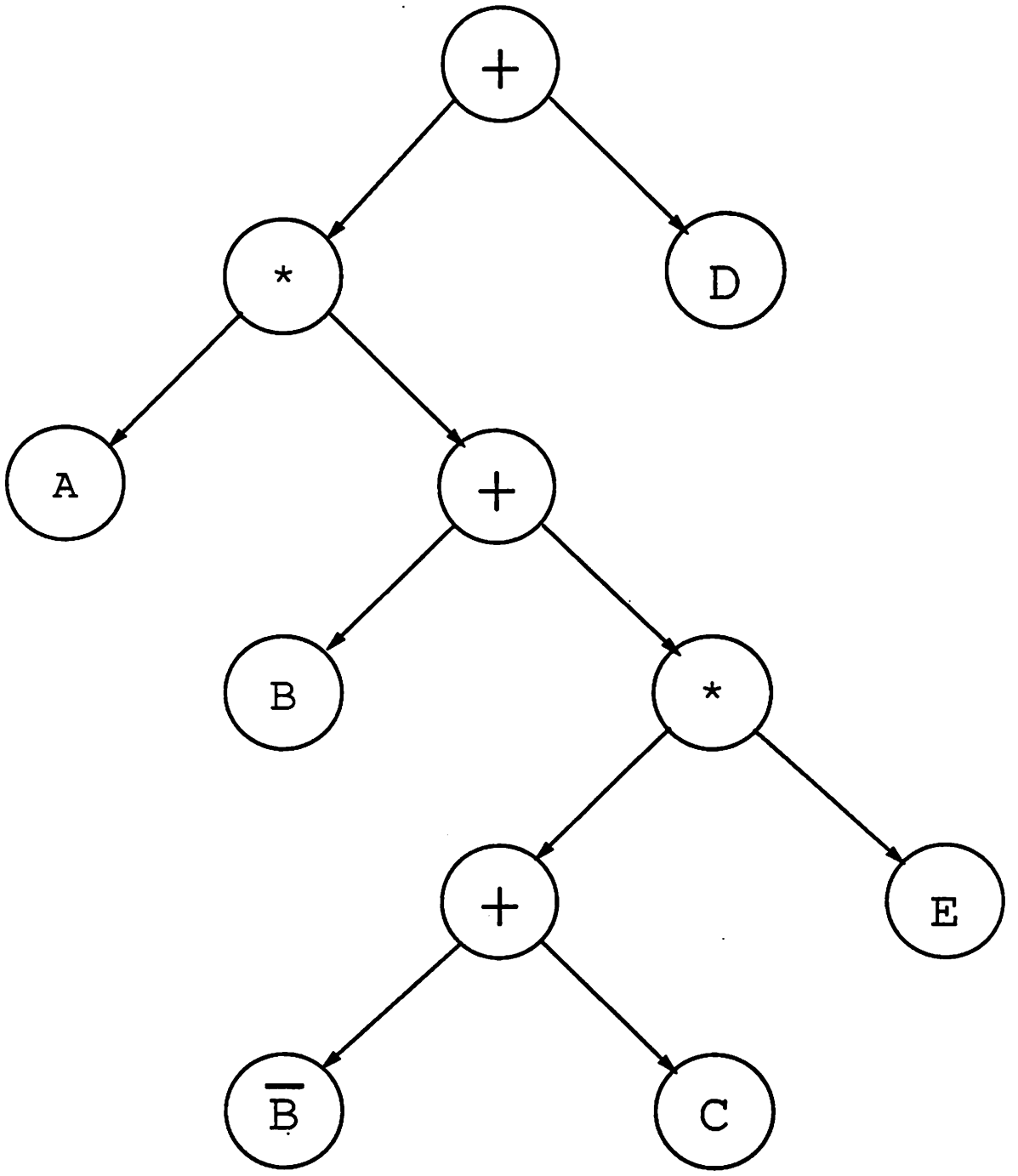


Figure 2.1: Tree representation of a factored-form.

Chapter 3

Background and Contributions

Two-level logic minimization algorithms have been around for several decades. Karnaugh and Veitch maps were among the earliest methods used. They are still discussed in most elementary text books on Digital Circuit Design, e.g [15]. Such methods were suitable for simplification by hand but were useful only for functions up to 5 variables. Quine and McCluskey [13] developed an algorithm for exact minimization that can be used for functions with larger number of variables. Their algorithm requires listing of all the minterms in the onset. Since a function with n variables can have 2^n minterms, the algorithm is not very useful for functions with even a moderate number of variables (around 20).

Quine and McCluskey's algorithm generates all the prime implicants of the function then transforms the problem of obtaining the minimum subset of primes into that of finding a minimum column cover of a matrix. The number of prime implicants of a function with n variables can be as high as $3^n/n$. This is an obstacle in the way of applying the algorithm for large problems. The problem of finding minimum column cover of a matrix is NP-complete. This is another source of possibly large computation. Later, several methods for exact minimization were developed but they all required computations of all the primes and therefore had limited use.

With the introduction of Programmable Logic Arrays (PLA) and the advances in Integrated Circuit Design, a need for minimization techniques that could handle functions with a large number of inputs and outputs (typically around 50 inputs and 50 outputs) became apparent. Many researchers resorted to heuristic minimization to handle large functions. A typical operation done in heuristic minimization is called expansion of cubes

(product terms). This involves removing literals from cubes to make them prime. The cube is said to be expanded as the literals are removed from it.

A literal can be removed from a cube only if the expanded cube is still an implicant of the function. Two methods have emerged to test whether a given cube is an implicant of a function. One uses tautology as done in PRESTO [7]. The other uses the offset which is computed by complementing the union of the onset and the don't care set. This is done in MINI [10] and ESPRESSO [2]. An experiment done in ESPRESSO-I [2,3] concluded that the technique using offset produced superior results in general and was usually faster than using tautology.

A common criticism against using the offset is that there are many functions which have reasonable size onsets and don't care sets but the offsets are too big. A common example of such a function is Achilles' heel function [2] where the size of the offset is 3^n cubes if the onset has n cubes. ESPRESSO attempted to solve this problem by a technique called output splitting [2]. The technique may give up some optimization in the interest of keeping the offset small. Another approach to address this problem has been to develop more efficient tautology-based algorithms [9,23].

Functions with large offsets occasionally occur in the traditional application of Two-Level Minimization for reducing the size of PLAs. However, it has been found from experience with MIS [5] that such functions arise commonly when two-level minimization is used for simplification of functions at individual nodes within a multi-level network. Since each node in the network has a variable and a function associated with it, the number of variables in the network is large. The covers of most functions in the network are small but the don't care sets tend to be very large due to the network topology. If the full don't care set is passed to ESPRESSO it often runs out of memory or takes a long time to compute the offset. Therefore, it is often necessary to compromise quality by deleting part of the don't care set before calling ESPRESSO.

A contribution of this thesis is formulation of a new concept called the *reduced offset for a cube.* The reduced offset for a cube is never larger than the offset and is found to be much smaller than the offset in practice. Yet, it can be used in the same way as the offset during expansion of a cube. Consequently, quality of minimization is maintained. An efficient algorithm for computing the reduced offset for a cube has been developed. The key ingredient in the algorithm is that the exact offset of the function is never computed.

A new algorithm for obtaining the *overexpanded cube* was also discovered. The

overexpanded cube of a cube p is defined as the smallest cube containing all the implicants of the function that contain p. The overexpanded cube consists of all the literals that cannot be removed from the cube. An efficient algorithm for computing the overexpanded cube is important in its own right. It is known that the overexpanded cube can be obtained by tautology or by using the offset. The new algorithm is faster than using tautology and does not use the offset. Hence, it is useful in tautology-based algorithms where knowledge of the overexpanded cube beforehand can save time in expansion of the cube. In this research, the overexpanded cube is used to speed up the computation of the reduced offset.

The algorithms for computing reduced offsets and overexpanded cube have been implemented in ESPRESSO with an interface to the Multi-Level logic optimization interactive system MIS. The execution time for minimizing individual nodes has been greatly reduced and it is now possible to minimize some nodes impossible to minimize with ESPRESSO.

Since, the logic functions at internal nodes of a multi-level network are single-output Boolean functions, the theory related to the reduced offset was initially developed for Boolean functions with single outputs. A Boolean function is a special case of multi-valued function. A multi-valued function is similar to a Boolean function except that each variable x_i in the function can take on any value from a finite set of values P_i. For a Boolean function, $P_i = \{0, 1\}$ for every variable x_i in the function. Therefore, the study of multi-Valued minimization provides a larger view of the problem of minimization of Boolean functions. The problem of minimizing a multiple-output Boolean function can be treated as that of Multi-Valued minimization where all the outputs together are considered as a single Multi-Valued variable [18]. This approach was taken in ESPRESSO-MV [16]. Finally, Two-level Multi-Valued logic minimization has been found useful for optimal encoding of states for Finite-State Machines [8]. Offset is used in multi-valued minimization in the same way as it is used in binary-valued minimization. A multi-valued function may have reasonable size onset and don't care set but unreasonably large offset, making it impossible to use the offset for minimization. Hence, it was desirable to extend the theory of reduced offset to multi-valued functions. The extension is also presented in this thesis.

Another contribution of this thesis is related to *factored-form minimization*. The objective in factored-form minimization is to produce factored-forms with the minimum number of literals for the given Boolean function. Traditionally this problem has been tackled by using algebraic techniques [5], [6], [4]. First the sum-of-products representation of the Boolean function is obtained usually by using a two-level logic minimizer. Then

this representation is treated as an algebraic polynomial. Each Boolean variable and its complement are treated as two different algebraic variables. The factored-form obtained is *algebraically equivalent* to the original sum-of-products representation. In other words, if the algebraic factors are multiplied out, the resulting polynomial will be the same as the original sum-of-products representation. However, any factored-form which is equivalent in the *Boolean* sense is a valid factored-form. If a factored-form is algebraically equivalent to a given sum-of-products expression then it is also equivalent to the same expression in the *Boolean* sense but the converse is not true. As a result, when the algebraic factors are used, only a subset of the valid solutions are available.

A heuristic algorithm is presented in this thesis for Boolean factored-form minimization that makes use of the operations which are the basis of ESPRESSO. The algorithm generates Boolean factored-forms and explores a much larger region of the solution space than algebraic factoring operations. The quality of the factored-forms obtained is good. However, the CPU time spent is higher than that for algebraic factored-form minimization.

The reduced offset for binary-valued function is presented in Chapter 3. The extension of reduced offset to multi-valued minimization is presented in Chapter 4. The algorithm for Boolean factored form minimization is presented in Chapter 5.

Chapter 4

Reduced Offsets for Minimization of Binary-valued Functions

4.1 Introduction

A two level logic minimization problem is generally posed as minimization of a boolean function given a cover of the function and a representation of its don't care set. The objective of minimization is generally to decrease the total number of cubes (product terms) and the total number of literals in the cover. Generally, as an important sub-task, the number of literals in each cube needs to be reduced, either to obtain a cube with the minimum possible literals or a cube that contains as many other cubes in the cover as possible. The expanded cube is then added to the cover and those cubes contained in the expanded cube are removed.

Let g be the union of the onset f and don't care set d of an incompletely specified function $\mathcal{F} = (f, d, r)$. The expansion for a cube in the cover of $\mathcal{F}$ is valid only if the expanded cube is contained in g. Two well-known methods exist to make this test. The first uses tautology. The test if an expanded cube is covered by g can be posed as a tautology test. This method is used by PRESTO [7]. Testing if a function is a tautology takes exponential time in the worst case. Also, several tautology tests may be necessary for each cube.

The other method makes use of the simple fact that if the expanded cube is contained in g, it must not intersect any cube in the offset $r = \overline{g}$. If it is not given, the

computation of the offset is also expensive in the worst case but this needs to be done only once for the entire run. Further, if the offset is used, usually the expansion can be done quickly and in a more global way because it is easier to see effective directions of expansion. For example, using the offset, it is easy to find the literals that can be removed without affecting any other literal, or the literals that must be retained in the expanded cube. The offset method is used by ESPRESSO [2] and MINI [10].

The problem with using the offset is that there are functions which have reasonable size onset and don't care set but the offset is unreasonably large. An example is the Achilles' heel function with n cubes:

$$f = x_1x_2x_3 + x_4x_5x_6 + \cdots + x_{3n-2}x_{3n-1}x_{3n}$$

The don't care set is empty. The number of cubes in the minimum representation of the offset of this function is 3^n.

It was found from experience with MIS [5] that such functions arise commonly when two level minimization is used as part of a multi-level minimization process. There, two level minimization is used for simplification of functions at individual nodes within a multilevel network. Each node in the network has a variable and a function associated with it [1]. As a result the number of variables in the network is large. The cover of most functions in the network is small but the don't care sets tend to be very large due to the network topology. If the full don't care set is passed to ESPRESSO it often runs out of memory or takes a long time to compute the offset. Therefore, it is often necessary to compromise quality by deleting part of the don't care set before calling ESPRESSO [17].

To solve this problem, the notion of the reduced offset of a cube is introduced. The reduced offset for a cube is never larger than the entire offset of the function and in practice has been found to be much smaller. Yet, it can be used in the same way as the complete offset during expansion of the cube. Consequently, the quality of results is maintained.

If the reduced offset is computed for each cube that needs to be expanded, it may be time consuming. Fortunately, it is possible to obtain a reduced offset for a set of cubes which can be used to expand each cube in the set in the same way as the offset would be used. Computation of the reduced offset for a set of cubes can be done without using the offset. The size of the reduced offset for a set of cubes increases as the number of cubes in the set increases but it is never larger than the entire offset.

The work on reduced offsets initially focussed on minimization of binary valued

functions with single output because these are the type of functions that correspond to individual nodes in multi-level networks. This chapter presents the application of reduced offsets for minimization of binary valued single output functions. The theory was later extended to the domain of multi-valued logic minimization. It will be presented in a later chapter.

In this chapter, an efficient algorithm will be presented for computing the reduced offset for a cube without using the complete offset of the function. During the research on reduced offsets, a new algorithm for obtaining the overexpanded cube for a cube was discovered. For binary valued functions, the overexpanded cube consists of all the literals that must remain in any valid expansion of the cube. An efficient algorithm for generation of the overexpanded cube is important in its own right. The overexpanded cube can be obtained by tautology or by using the offset. The algorithm presented here is more efficient than using tautology and does not use the offset. Hence, it is useful in tautology based algorithms where knowledge of the overexpanded cube beforehand can save time in expansion of the cube. The overexpanded cube is of interest here because it is used to speed up the computation of the reduced offset.

4.2 Reduced Offset for a Cube

To illustrate that the entire offset is not needed for expanding a cube, consider the incompletely specified function $\mathcal{F} = (f, d, r)$ shown in figure 4.1. The clear vertices represent minterms in the onset. The filled vertices represent the minterms in the offset. The vertex with concentric circles represents the minterm in the don't care set. Suppose cube $p = \overline{A}\,\overline{B}C$ is to be expanded. Because of the presence of $\overline{A}B$, and $A\overline{B}$ in the offset, p can not expand to cover the onset minterm ABC. As far as the expansion of p is concerned, ABC could just as well be in the offset. If it is added to the offset, the offset is simplified. It becomes $A + B$, which is all that is needed for correct expansion of p.

I now introduce the notion of *the reduced offset for a cube* and explain how the reduced offset can be computed easily if the offset is available. Later an algorithm will be presented that does not require the offset.

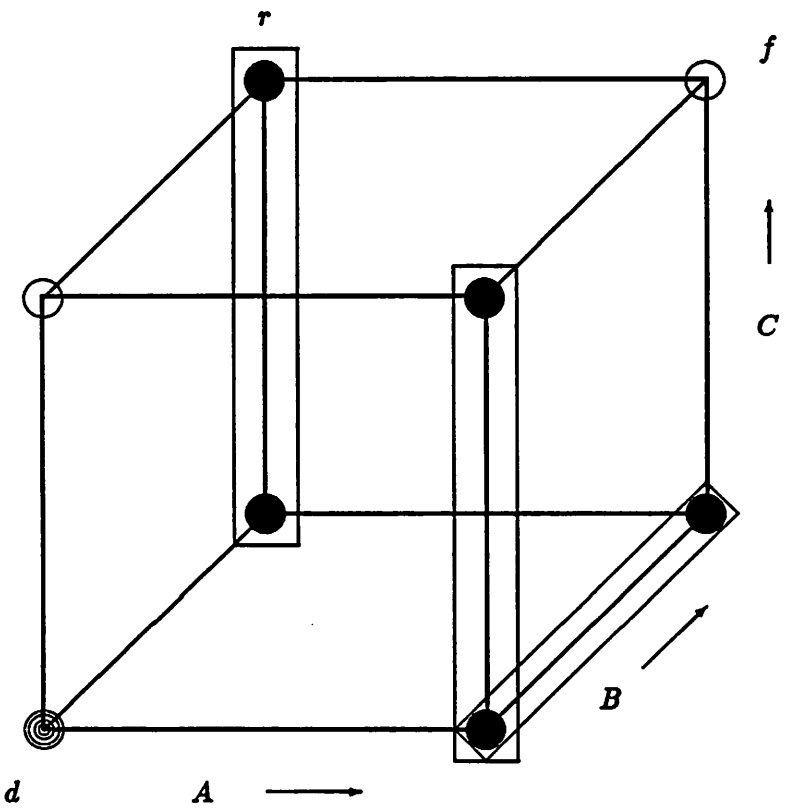
Note that a cube p in the cover can be expanded by removing one or more literals provided the new cube is orthogonal to each of the cubes in the offset. For the example of figure 4.1, consider the expansion of the cube $p = \overline{A}\,\overline{B}$ in the cover and an offset cube

$q = AB\overline{C}$. p does not intersect q because p contains literals $\overline{A}$ and $\overline{B}$ whereas q contains complements of these literals. Literal $\overline{C}$ in q does not affect the orthogonality between cubes p and q because literal C does not appear in p. Since an expansion of p is obtained by removing some of the literals in p, the presence of $\overline{C}$ in q will not affect orthogonality between q and any expansion of p. Hence, $\overline{C}$ can be removed from q. In general, removing from an offset cube any literal that is not the complement of a literal in cube p, does not result in any loss of information as far as the expansion of p is concerned. As a result of removing literals, some cubes become larger and may subsume others, thus reducing the size of the offset. For this example, removal of literals $\overline{A}$, $\overline{B}$, C and $\overline{C}$ from cubes in the offset will result in $A + B + AB$. Since AB is contained in A as well as B, it can be dropped to obtain $A + B$.

$A + B$ is in fact the reduced offset for p. It has only two cubes and two literal whereas the exact offset has three cubes and six literals. The reduced offset is obtained by removing all literals from the cubes of the offset except those that are the complements of the literals in p. The reduced offset for a cube is a unate function and has a unique minimum representation in sum-of-products form. This minimum representation is obtained by removing any cube contained in some other cube in the reduced offset. Whatever representation is used for r, the reduced offset for a cube has a unique minimum representation.

4.3 Reduced Offset for a Set of Cubes

To see how the concept of reduced offset *for a cube* can be extended to that of the reduced offset for *a set of cubes*, consider an incompletely specified function with $r = A\overline{B} + \overline{A}B + A\overline{C} + C\overline{D}$ and two cubes $p_1 = \overline{A}\,\overline{B}CD$ and $p_2 = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$ to be expanded. Expansion of p_1 will not be affected by the removal of literals $\overline{A}$, $\overline{B}$, C and D. Similarly removal of literals $\overline{A}$, $\overline{B}$, $\overline{C}$ and $\overline{D}$ will not affect expansion of p_2. Hence removal of literals $\overline{A}$ and $\overline{B}$ will not affect either expansion of p_1 or p_2. Removal of these literals from r gives $A + B + C\overline{D}$ which is in fact the reduced offset for $\{p_1, p_2\}$. Reduced offset of a set of cubes is obtained by removing those literals from the offset that will not affect the expansion of any cube in the set. The reduced offset of a set of cubes is usually smaller than the offset but larger than the reduced offset for any cube in the set. Unlike the reduced offset for a cube, reduced offset for a set of cubes is not necessarily unate. Therefore, it does not necessarily have a unique minimum.



$$f = \overline{A}\,\overline{B}C + ABC$$

$$d = \overline{A}\,\overline{B}\,\overline{C}$$

$$r = \overline{A}B + A\overline{B} + A\overline{C}$$

Figure 4.1: Illustrating reduced offset

I will next define the notion of a generalized cube which will be helpful in handling the reduced offset for a set of cubes.

4.4 Generalized Cube

Definition 4.4.1 *A* **generalized cube or a generalized product term** *is a product of one or more literals (at most one literal of each variable). A literal can be an asserted literal x, a complemented literal $\overline{x}$, or a null literal $\hat{x}$.*

EXAMPLE 4.1 :

All of the following are generalized cubes: $p_1 = A\overline{C}E$, $p_2 = \overline{A}\hat{C}D$, $p_3 = ABC$, $p_4 = C\hat{E}$, and $p_5 = \hat{A}\hat{C}\hat{E}$

If the generalized cube does not have a null literal then it is just an ordinary cube. On the other hand, if it has a null literal then it is a null cube. The set of minterms represented by a null literal or a null cube is empty.

In the rest of the chapter, a *cube* will mean an ordinary (non-null) cube. A generalized cube will mean that the cube may or may not be null.

A null cube can not occur in a sum-of-products but it can result from the intersection (product) of two or more generalized cubes.

Definition 4.4.2 *$\tilde{p}$ is the product (intersection) of two generalized cubes; p_1 and p_2 (written as $\tilde{p} = p_1 p_2$) if for each variable x present in either cube, a literal for $\tilde{p}$ in variable x is obtained according to the table in figure 4.2.*

If neither p_1 nor p_2 is a null cube then the above definition is equivalent to that of the intersection of two cubes, presented in Chapter 2. If the product of n generalized cubes is desired for $n > 2$, it is obtained by intersecting q_{n-1} with p_n where q_{n-1} is the intersection of the first $n-1$ generalized cubes.

EXAMPLE 4.2 : Consider the five cubes in Example 4.1. Let q_i be the product of first i cubes and q be the product of all five cubes then $q_2 = \hat{A}\hat{C}DE$, $q_3 = \hat{A}B\hat{C}DE$, $q_4 = \hat{A}B\hat{C}D\hat{E}$, and $q = q_5 = \hat{A}B\hat{C}D\hat{E}$.

p_2 \ p_1	$\hat{x}$	$\overline{x}$	x	NP
$\hat{x}$	$\hat{x}$	$\hat{x}$	$\hat{x}$	$\hat{x}$
$\overline{x}$	$\hat{x}$	$\overline{x}$	$\hat{x}$	$\overline{x}$
x	$\hat{x}$	$\hat{x}$	x	x
NP	$\hat{x}$	$\overline{x}$	x	NP

NP means variable x is *not present.*

Figure 4.2: Table for intersecting two generalized cubes

A null cube is not automatically contained in every other generalized cube. The definition of one generalized cube containing another is given below:

Definition 4.4.3 *A* **generalized cube p_1 contains another generalized cube p_2** *(written as $p_2 \subseteq p_1$) if each variable in p_1 is present in p_2 and each literal in p_1 contains the literal of the same variable in p_2.*

Definition 4.4.4 *A* **literal contains another literal of the same variable** *if the set of minterms represented by the former literal contains that represented by the later literal.*

According to this definition $\hat{x}$ is contained in x as well as in $\overline{x}$. It can be shown that if $\tilde{p}$ is the intersection of generalized cubes $p_1, p_2 \cdots p_n$ then $\tilde{p} \subseteq p_i$ for $1 \leq i \leq n$. Note that q in Example 4.2 is contained in every cube p_i in Example 4.1.

EXAMPLE 4.3 : Let $p_1 = A\overline{B}C$, $p_2 = \hat{A}\overline{B}DE$, $p_3 = A\overline{B}D$, and $p_4 = \overline{A}\,\overline{B}D$. Then $p_2 \subseteq p_3$ and $p_2 \subseteq p_4$ but $p_2 \not\subseteq p_1$.

4.5 Reduction Operator R_p and Reduced Offset

To facilitate the computation of the reduced offset, the notion of the reduction operator R_p is now introduced.

Definition 4.5.1 *Let g be a sum-of-products and p be a generalized cube. The* **reduction operator** $R_p(g)$ *removes all literals from the cubes of g except those which are complements of non-null literals in p.*

Definition 4.5.2 *Let p be a generalized cube and c be a non-null cube. Then $|R_p(c)|$ is the number of literals in $R_p(c)$.*

If p is not a null cube either, then $|R_p(c)| = |R_c(p)|$ is the *distance* between cubes p and c.

The reduced offset for a generalized cube can now be formally defined using the reduction operator R_p.

Definition 4.5.3 *If r is the offset of a function and p is a generalized cube then $R_p(r)$ is* **the reduced offset of r for p.**

The reduced offset for a set of cubes is defined in terms of the reduced offset for a cube as follows:

Definition 4.5.4 *Let S be a set of cubes and $\tilde{p}$ be the intersection of all the cubes in S then* **the reduced offset for S** *is the reduced offset for $\tilde{p}$.*

The usefulness of the offset r in expanding a cube p stems from the fact that an expansion p' of p is valid if and only if p' is orthogonal to r. Whether a reduced offset is for a cube or for a set of cubes, it can be used for expanding cubes in the same way as the offset because both are sums-of-products and the following Theorem and Corollaries hold which relate the orthogonality of p' to r with that to the reduced offset.

Theorem 4.5.1 *Let p be a generalized cube and p' be a (non-null) cube such that $p \subseteq p'$. Let g be a sum-of-products. p' is orthogonal to g if and only if it is orthogonal to $g_p = R_p(g)$.*

Proof: Since g_p is obtained by removing literals from g, $g \subseteq g_p$. Therefore, if p' is orthogonal to g_p (i.e no minterm is common to both p' and g_p) it is also orthogonal to g.

What remains to be shown is that if p' is orthogonal to g, it is also orthogonal to g_p. Let

$$\begin{aligned} g &= c_1 + c_2 + \cdots + c_n \\ R_p(c_i) &= c_i' \end{aligned}$$

Then

$$g_p = c_1' + c_2' + \cdots + c_n'$$

Consider a minterm $s \in g_p$. s must be in one or more cubes in g_p. Let c_i' be one such cube. Without loss of generality suppose that

$$\begin{aligned} c_i &= x_1 x_2 \cdots x_m x_{m+1} \cdots x_n \\ c_i' &= x_1 x_2 \cdots x_m \end{aligned}$$

Thus p does not have literals $\overline{x}_i$ or $\hat{x}_i$ for $m+1 \leq i \leq n$. Since $p \subseteq p'$, p' does not have these literals either. If p' is orthogonal to g, it is also orthogonal to c_i. Hence it must have a literal $\overline{x}_j$ for some j between 1 and m. In that case p' is also orthogonal to c_i' and $s \notin p'$. Since this is true for any minterm $s \in g_p$, p' is orthogonal to g_p. ■

Corollary 4.5.2 *Let $\mathcal{F} = (f, d, r)$ be an incompletely specified function. Let p be a generalized cube and p' be a (non-null) cube such that $p \subseteq p'$. p' is a valid expansion of p if and only if p' is orthogonal to the reduced offset for p.*

Proof: p' is a valid expansion of p if and only if it is orthogonal to r. The corollary follows from Theorem 4.5.1 with $g = r$. ■

Corollary 4.5.3 *Let $\mathcal{F} = (f, d, r)$ be an incompletely specified function and S be a set of cubes, each of which is contained in $f \cup d$. Let $q \in S$ and q' be an expansion of q. q' is a valid expansion of q if and only if it is orthogonal to the reduced offset for S.*

Proof: Let $\tilde{q}$ be the intersection of cubes in S. Then the reduced offset for S is $R_{\tilde{q}}(r)$: the reduced offset for $\tilde{q}$.

Clearly $\tilde{q} \subseteq q \subseteq q'$. With $p = \tilde{q}$, $p' = q'$, and $g = r$ in Theorem 4.5.1, it follows that p' is orthogonal to r if and only if it is orthogonal to $R_{\tilde{q}}(r)$, i.e the reduced offset for S. ■

4.6 Properties of R_p

The reduction operator has following useful properties:

Property 4.6.1 *Let f_1 and f_2 be two sums-of-products and p be a generalized cube then:*

$$R_p(f_1 + f_2) = R_p(f_1) + R_p(f_2)$$

Proof: Let

$$\begin{aligned} f_1 &= c_1 + c_2 + \cdots + c_n \\ f_2 &= b_1 + b_2 + \cdots + b_m \end{aligned}$$

Removing some literals from f_i for $i = 1, 2$ is the same as removing each of those literals from each cube in f_i.

$$\begin{aligned} R_p(f_1) &= R_p(c_1) + R_p(c_2) + \cdots + R_p(c_n) \\ R_p(f_2) &= R_p(b_1) + R_p(b_2) + \cdots + R_p(b_n) \end{aligned}$$

Hence

$$R_p(f_1) + R_p(f_2) = R_p(c_1) + R_p(c_2) + \cdots + R_p(c_n) + R_p(b_1) + R_p(b_2) + \cdots + R_p(b_m)$$

Now consider

$$f_1 + f_2 = c_1 + c_2 + \cdots c_n + b_1 + b_2 + \cdots b_m$$

Therefore

$$\begin{aligned} R_p(f_1 + f_2) &= R_p(c_1) + R_p(c_2) + \cdots + R_p(c_n) + R_p(b_1) + R_p(b_2) + \cdots + R_p(b_m) \\ &= R_p(f_1) + R_p(f_2) \end{aligned}$$

■

Property 4.6.2 *Let f_1 and f_2 be two sums-of-products and p be a generalized cube. Let S_1 be the set of literals of f_1 and S_2 be the set of literals of f_2. If for every literal $x \in S_1$, $\overline{x} \notin S_2$ then:*

$$R_p(f_1 f_2) = R_p(f_1) R_p(f_2)$$

Proof: Let

$$f_1 = c_1 + c_2 + \cdots + c_n$$
$$f_2 = b_1 + b_2 + \cdots + b_m$$

Then

$$R_p(f_1)\, R_p(f_2) = \prod_{i=1}^{n} \prod_{j=1}^{m} R_p(c_i) R_p(b_j)$$

Also

$$f_1\, f_2 = \prod_{i=1}^{n} \prod_{j=1}^{m} c_i b_j$$
$$R_p(f_1\, f_2) = \prod_{i=1}^{n} \prod_{j=1}^{m} R_p(c_i\, b_j)$$

Let L_c^i and L_b^j be the sets of literals in c_i and b_j for some $1 \leq i \leq n$ and $1 \leq j \leq m$ respectively. Let L_p be the set of literals which are not removed by R_p. From the definition of R_p, $L_c^i \cap L_p$ is the set of literals in $R_p(c_i)$ and $L_b^j \cap L_p$ is the set of literals in $R_p(b_j)$. Hence $R_p(c_i)R_p(b_j)$ consists of literals in $(L_c^i \cap L_p) \cup (L_b^j \cap L_p)$. The condition in the statement ensures that $c_i b_j \neq \emptyset$. In that case $b_i\, c_j$ consists of literals in $L_c^i \cup L_b^j$ and $R_p(b_i\, c_j)$ consists of literals in $L_p \cap (L_c^i \cup L_b^j)$. But $L_p \cap (L_c^i \cup L_b^j) = (L_p \cap L_c^i) \cup (L_p \cap L_b^j)$. Hence $R_p(c_i\, b_j) = R_p(c_i)\, R_p(b_j)$. Now from the above equation:

$$\begin{aligned} R_p(f_1\, f_2) &= \prod_{i=1}^{n} \prod_{j=1}^{m} R_p(c_i)\, R_p(b_j) \\ &= R_p(f_1) R_p(f_2) \end{aligned}$$

■

Property 4.6.3 *Let p_1 and p_2 be two generalized cubes such that $p_1 \subseteq p_2$ and g be a sum-of-products. Then*

$$R_{p_2}(g) = R_{p_2}(R_{p_1}(g))$$

Proof: Let S_1 and S_2 be sets of literals which are removed by the application of R_{p_1} and R_{p_2} respectively. Since $p_1 \subseteq p_2$, S_2 contains all literals in S_1. Now consider $R_{p_2}(R_{p_1}(g))$. Since any literal which is removed by R_{p_1} would be removed by the subsequent application of R_{p_2}, $R_{p_2}(R_{p_1})(g) = R_{p_2}(g)$. ■

The next section presents some properties of the reduced offset.

4.7 Properties of the Reduced Offset

In addition to the fact that the reduced offset can be used in lieu of the offset for expansion of cubes, it has the following properties that may be useful in some applications.

Property 4.7.1 *The reduced offset for a (non-null) cube is a unate function.*

Proof: Let p be a (non-null) cube and r_p be the reduced offset for p. By definition $r_p = R_p(r)$. If a variable is not present in p then r_p does not have that variable. Let x be a variable in p. p has either literal x or $\overline{x}$ but not $\hat{x}$. Therefore r_p will have literal $\overline{x}$ or x but not both. Hence $r_p = R_p(r)$ is a unate sum-of-products, and hence a unate function. ■

Property 4.7.2 *Let r_p and r_q be reduced offsets for generalized cubes p and q such that $p \subseteq q$. Then $r_q = R_q(r_p)$.*

Proof: It follows from Property 4.6.3 of the reduction operator with $p_1 = p$, $p_2 = q$ and $g = r$ that

$$R_q(r) = R_q(R_p(r))$$

By the definition of the reduced offset:

$$r_q = R_p(r_p)$$

■

This property is useful for obtaining the reduced offset for q from the reduced offset for p if $p \subseteq q$.

Property 4.7.3 *Let $\mathcal{F} = (f, d, r)$ be an incompletely specified function. Let p be a generalized cube contained in $g = f \cup d$ and r_p be the reduced offset for p. Then a cube q is a prime of $\overline{r_p}$ if and only if q is a prime of g and $p \subseteq q$.*

Proof: If p can be expanded to q then q must be orthogonal to r_p according to Corollary 4.5.2. Therefore $q \subseteq \overline{r}_p$. Now consider any other cube $q' \supset q$. If q is a prime of $g = f \cup d$ then $q' \not\subseteq g$; equivalently q' is not orthogonal to r. Therefore q' is not orthogonal to r_p according to Corollary 4.5.2. In that case $q' \not\subseteq \overline{r}_p$. Hence q is a prime of $\overline{r}_p$.

If q is a prime of $\overline{r_p}$ then q is orthogonal to r_p. According to Corollary 4.5.2, it is also orthogonal to r. Hence $q \subseteq g$. Now consider a cube $q' \supset q$. $q' \not\subseteq \overline{r}_p$ because q is a

prime of r_p. Therefore, q' is not orthogonal to r_p. Hence, according to Corollary 4.5.2, q' is not orthogonal to r; equivalently $q' \not\subseteq g$. This proves that q is a prime of g. What remains to be shown is that $p \subseteq q$. Assume for a contradiction that $p \not\subseteq q$. Then the following two possibilities exist:

1. Some variable x is present in q but not in p. It follows from the definition of r_p that variable x is not present in r_p. If $\tilde{q}$ is obtained by removing variable x from q then $\tilde{q}$ is also orthogonal to r_p. Consequently, $q \subset \tilde{q} \subseteq \overline{r}_p$. But this can not be possible because q is a prime of r_p.

2. Some literal x is present in q and its complement $\overline{x}$ is present in p. By the definition of r_p, literal $\overline{x}$ is not present in r_p. If $\tilde{q}$ is obtained by removing variable x from q then $\tilde{q}$ is also orthogonal to r_p. Consequently, $q \subseteq \tilde{q} \subseteq \overline{r}_p$. But this can not be possible because q is a prime of r_p.

Since none of the above two conditions can occur, it is concluded that $p \subseteq q$. ■

For a given cube $p \subseteq f \cup d$, it is sometimes necessary to find all the primes of $\mathcal{F} = (f, d, r)$ that contain p. If p is a (non-null) cube then r_p is unate. Therefore, $\overline{r}_p$ is unate also. Based on a property of unate functions, if every cube in $\overline{r}_p$ that is contained in some other cube in $\overline{r}_p$ is removed, then the remaining cubes are all the primes of $\overline{r}_p$ [2]. If p is a null cube then r_p is generally not unate. It is still possible to obtain all the primes of $\overline{r}_p$ by first getting a product-of-sums representation for $\overline{r}_p$ by applying DeMorgan's complementation on r_p and then using Nelson's Theorem [14]. The theorem states that if a product-of-sums representation of $\overline{r}_p$ is available, then all primes of $\overline{r}_p$ are obtained by first multiplying out the product-of-sums representation to get a sum-of-products representation and then removing each cube from the sum-of-products representation that is contained in any other cube.

Property 4.7.4 *Let S be the set of all the cubes in the onset f of $\mathcal{F} = (f, d, r)$ and r_S be the reduced offset for S. Then r_S is a cover for $\mathcal{G} = (r, d, f)$ i.e*

$$r \subseteq r_S \subseteq r \cup d$$

Proof: Since r_S is obtained by removing zero or more literals from r, $r \subseteq r_S$.

Let p be the generalized cube obtained by the intersection of all the cubes in the onset. Then by definition $r_S = r_p$. Any cube q in f is a valid expansion of p. Therefore q is orthogonal to r_S. Consequently f is orthogonal to r_S. Hence, $r_S \subseteq \overline{f} = r \cup d$. ■

Having defined the reduction operator, the reduced offset and their properties, I present an algorithm for generating the reduced offset for a cube p using the unate recursive paradigm on $f \cup d$. Thus the offset r is not required.

4.8 Unate Recursive Algorithm for the Reduced Offset for a Cube

The unate recursive paradigm has been used successfully for several logic operations including complementation of a function. Unate functions have many nice properties which make it easier to operate on them. In a unate recursive paradigm, a non-unate function is broken down into its unate cofactors using Shannon's expansion recursively. Then the operation is applied to the unate cofactors rather than the original function and the results are merged together. The unate recursive paradigm applied to the computation of the reduced offset is given below. The reduced offset was defined earlier as produced by the reduction operator acting on the offset but this requires knowing the offset. To be useful, the algorithm for computing the reduced offset has to avoid computation of the offset.

Let p be a cube and $\mathcal{F} = (f, d, r)$ be a function whose reduced offset for p is desired. Let the reduced offset be denoted by r_p. By definition $r_p = R_p(\overline{g})$ where $g = f \cup d$. From Shannon's expansion,

$$\overline{g} = x_i \overline{g}_{x_i} + \overline{x}_i \overline{g}_{\overline{x}_i}$$

where x_i is a binate (non-unate) variable in g. The complementation and cofactoring operations commute. Hence,

$$\overline{g} = x_i \overline{g_{x_i}} + \overline{x}_i \overline{g_{\overline{x}_i}}$$

Applying the reduction operator on both sides of the equation and using property 4.6.1 of the operator, we get

$$r_p = R_p(\overline{g}) = R_p(x_i \overline{g_{x_i}}) + R_p(\overline{x}_i \overline{g_{\overline{x}_i}})$$

Since $\overline{g_{x_i}}$ and $\overline{g_{\overline{x}_i}}$ are independent of x_i, property 4.6.2 of R_p gives:

$$r_p = R_p(\overline{g}) = R_p(x_i)R_p(\overline{g_{x_i}}) + R_p(\overline{x}_i)R_p(\overline{g_{\overline{x}_i}})$$

If either $\overline{g_{x_i}}$ or $\overline{g_{\overline{x}_i}}$ is not unate, it can be decomposed further with respect to a binate variable in it. The process continues until all cofactors become unate.

4.8.1 Applying Reduction Operator to Unate Sums-of-Products

The following Theorem can be used for obtaining the reduced offset for a cube when a cover U is unate.

Theorem 4.8.1 *Let U be a unate sum-of-products and p be a generalized cube. Then*

$$R_p(\overline{U}) = \overline{V}$$

where V is obtained from U by removing all cubes that don't contain p.

Proof: Let $q_1, q_2, q_3 \cdots q_m$ be the cubes in U that don't contain p. Since

$$\overline{U} = \overline{V}\overline{q}_1\overline{q}_2 \cdots \overline{q}_m$$

Applying operator R_p to both sides of the above equation.

$$R_p(\overline{U}) = R_p(\overline{V}\overline{q}_1\overline{q}_2 \cdots \overline{q}_m)$$

Since U is unate, for any variable x, either U is independent of x or it has only x or $\overline{x}$ but not both. The literals in $\overline{V}, \overline{q}_1, \overline{q}_2 \cdots \overline{q}_m$ are only complements of the literals in U. As a result, property 4.6.2 of R_p can be used and thus

$$R_p(\overline{U}) = R_p(\overline{V})R_p(\overline{q}_1)R_p(\overline{q}_2) \cdots R_p(\overline{q}_m)$$

Without loss of generality, suppose that $q_i = x_{i_1}x_{i_2} \cdots x_{i_l}$ for $1 \leq i \leq m$. Then $\overline{q}_i = \overline{x}_{i_1} + \overline{x}_{i_2} \cdots \overline{x}_{i_l}$. Since q_i does not contain p, at least one literal in q_i is either the complement of a non-null literal in p or it is of a variable not present in p. Let x_j be that literal. In either case $\overline{x}_j$ will be removed when R_p acts on $\overline{q}_i$. Since $\overline{x}_j$ is a single literal cube, the removal of $\overline{x}_j$ will result in making it a tautology. Therefore $R_p(\overline{q}_i)$ is a tautology and the above equation becomes

$$R_p(\overline{U}) = R_p(\overline{V})$$

Now consider a literal x_i present in V. Since V is unate, $\overline{V}$ is also unate and has $\overline{x}_i$ but not x_i. Since each cube in V contains p, p either has x_i or $\hat{x}_i$. In either case R_p will not remove $\overline{x}_i$ from $\overline{V}$. Hence

$$R_p(\overline{U}) = R_p(\overline{V}) = \overline{V}$$

■

The above Theorem is quite effective. On one hand it reduces the number of cubes in the unate cofactor that must be complemented and on the other hand it reduces the number of variables in the cofactor to a subset of those in p. As a result, complementing V is much faster than complementing U. Also $\overline{V}$ is much smaller than $\overline{U}$.

To illustrate the effectiveness of this Theorem, consider the Achilles' heel function

$$f = x_1x_2x_3 + x_4x_5x_6 + \cdots + x_{3n-2}x_{3n-1}x_{3n}$$

The don't care set is empty. Suppose that p is the i^{th} cube in the function; $p = x_{3i-2}x_{3i-1}x_{3i}$. The reduced offset for p is $R_p(\overline{f})$. f is unate therefore Theorem 4.8.1 can be applied to f. Since no other cube in f contains p, $V = p$ and $\overline{V} = \overline{p} = \overline{x}_{3i-2} + \overline{x}_{3i-1} + \overline{x}_{3i}$. Thus the reduced offset contains only three cubes whereas the offset $\overline{f}$ has 3^n cubes.

4.8.2 Merging

If h is a unate cofactor of g, $R_p(\overline{h})$ can be obtained from Theorem 4.8.1. Otherwise, h has cofactors h_x and $h_{\overline{x}}$ with respect to variable x. First $R_p(\overline{h_x})$ and $R_p(\overline{h_{\overline{x}}})$ are found. Then $R_p(\overline{h})$ is obtained by the following equation:

$$R_p(\overline{h}) = R_p(x)R_p(\overline{h_x}) + R_p(\overline{x})R_p(\overline{h_{\overline{x}}})$$

To keep the number of cubes in $R_p(\overline{h})$ small, the cubes contained in other cubes can be dropped. The resulting sum-of-products is called *minimal with respect to single cube containment.* This will involve testing every cube in $R_p(\overline{h})$ for containment in every other cube in $R_p(\overline{h})$. However, if each of $R_p(\overline{h_x})$ and $R_p(\overline{h_{\overline{x}}})$ is minimal with respect to single cube containment, the minimal $R_p(\overline{h})$ can be obtained by dropping cubes from $R_p(\overline{h_x})$ and $R_p(\overline{h_{\overline{x}}})$ directly to obtain ρ_x and $\rho_{\overline{x}}$ such that $R_p(x)\rho_x + R_p(\overline{x})\rho_{\overline{x}}$ gives $R_p(\overline{h})$ which is minimal with respect to single cube containment. Depending on what literal of variable x is present in p, this can be done by the following rules:

1. If p has $\hat{x}$ then $R_p(x) = x$ and $R_p(\overline{x}) = \overline{x}$. No cube in $xR_p(\overline{h_x})$ may be contained in any cube in $\overline{x}R_p(\overline{h_{\overline{x}}})$ and vice versa. Therefore,

$$x\rho_x + \overline{x}\rho_{\overline{x}}$$

where $\rho_x = R_p(\overline{h_x})$ and $\rho_{\overline{x}} = R_p(\overline{h_{\overline{x}}})$.

2. If p has $\overline{x}$ then $R_p(x) = x$ and $R_p(\overline{x}) = 1$, Since $R_p(\overline{h_x})$ and $R_p(\overline{h_{\overline{x}}})$ are independent of variable x, no cube in $R_p(\overline{h_{\overline{x}}})$ may be contained in a cube of $xR_p(\overline{h_x})$. Therefore,

$$x\rho_x + \rho_{\overline{x}}$$

where $\rho_{\overline{x}} = R_p(\overline{h_{\overline{x}}})$ and ρ_x is obtained by removing each cube from $R_p(\overline{h_x})$ which is contained in a cube in $R_p(\overline{h_{\overline{x}}})$.

3. If p has literal x then the situation is the same as above with the roles of x and $\overline{x}$ switched.

4. If p is independent of variable x then $R_p(x) = R_p(\overline{x}) = 1$. Therefore,

$$\rho_x + \rho_{\overline{x}}$$

where ρ_x is obtained by removing each cube from $R_p(\overline{h_x})$ which is contained in a cube in $R_p(\overline{h_{\overline{x}}})$ and $\rho_{\overline{x}}$ is obtained by removing each cube from ρ_x which is contained in a cube in $R_p(\overline{h_x})$

In this scheme, only those cubes are eliminated which are contained in some other cubes. A cube in $R_p(\overline{h})$ may be redundant even if it is not contained in some other cube. A better scheme can be devised that would exclude any redundant cubes from $R_p(\overline{h})$ but at the cost of additional computation. If p does not have $\hat{x}$ and both $R_p(\overline{h_x})$ and $R_p(\overline{h_{\overline{x}}})$ are unate then the scheme shown above will provide a unate representation of $R_p(\overline{h})$ which is minimal with respect to single cube containment. Therefore $R_p(\overline{h})$ has the minimum number of product terms [2]. If p is not a null cube then it does not have any null literals. In that case, this scheme gives the minimum representation for every $R_p(\overline{h})$ and therefore for the reduced offset.

Subroutine *compute_ros*() shown in Algorithm 4.1 contains pseudo-code for computing the reduced offset. It makes use of subroutine *merge*() which is shown in Algorithm 4.2. A problem with Algorithm 4.1 is that every unate cofactor of $g = f \cup d$ is considered

Algorithm 4.1

****

Input: A cofactor h of $g = f \cup d$ and a generalized cube p.

Output: $R_p(\overline{h})$.

```
compute_ros(h, p) {
    If (h is unate) {
        Obtain V by removing those cubes from h that don't contain p.
        Return(V̄)
    } Else {
        Choose a binate variable x in h.
        Obtain cofactors h_x and h_x̄.
        t1 = compute_ros(h_x, p)
        t2 = compute_ros(h_x̄, p)
        t = merge(p, x, t1, t2)
        Return(t)
    }
}
```

Algorithm 4.2

****

Input: Cube p, splitting variable x, $t_1 = R_p(\overline{g}_x)$ *and* $t_2 = R_p(\overline{g}_{\overline{x}})$

Output: $r_p = R_p(\overline{g})$

merge(p, x, t_1, t_2 *)*

{

If (p has $\hat{x}$*)* {

Return($xt_1 + \overline{x}t_2$*)*

} *Else if (p has* $\overline{x}$*)* {

Remove each cube from t_1 *which is contained in any cube of* t_2 *to obtain* ρ_1.

Return($x\rho_1 + t_2$*)*

} *Else if (p has x)* {

Remove each cube from t_2 *which is contained in any cube of* t_1 *to obtain* ρ_2.

Return($t_1 + \overline{x}\rho_2$*)*

} *Else* {

Remove each cube from t_1 *which is contained in any cube of* t_2 *to obtain* ρ_1.

Remove each cube from t_2 *which is contained in any cube of* ρ_1 *to obtain* ρ_2.

Return($\rho_1 + \rho_2$*)*

}

}

in *compute_ros*() which may be very time consuming if g has a very large number of unate cofactors. The algorithm can be improved with the use of the overexpanded cube so that many cofactors need not even be computed. The next section considers the relationship between the overexpanded cube and the reduced offset and provides an efficient algorithm for computing the overexpanded cube.

4.9 Overexpanded Cube

The definition of the overexpanded cube for a (non-null) cube was presented earlier. The overexpanded cube is of interest because it will be used to improve Algorithm 4.1. However, Algorithm 4.1 is for computing the reduced offset for a *generalized cube.* It is therefore necessary to extend the notion of overexpanded cube to that of a generalized cube. The generalized cube is a vehicle to handle reduced offset for a set of cubes.

Definition 4.9.1 *Let $\mathcal{F} = (f, d, r)$ be an incompletely specified function and p be a generalized cube such that $p \subseteq p' \subseteq f \cup d$ for some non-null cube p'. The overexpanded cube q_p of p is the super cube of all primes of $\mathcal{F}$ that contain p.*

The requirement that $p \subseteq p' \subseteq f \cup d$ for some non-null cube p' is to ensure that the overexpanded cube is not a null cube. If that condition is not met then the overexpanded cube is not defined. The generalized cube is a vehicle to handle reduced offset for a set of cubes according to Definition 4.5.4. Therefore only those generalized cubes are of interest which are formed by the intersection of two or more (non-null) cubes of $f \cup d$. Any such generalized cube satisfies this requirement because it is contained in every cube in the intersection.

For a given incompletely specified function $\mathcal{F} = (f, d, r)$ and a cube $p \subseteq g = f \cup d$, a literal x_j present in p is in the overexpanded cube q_p if and only if $p' \not\subseteq g$ where p' is obtained from p by removing literal x_j [2]. If p is a generalized cube then the problem of finding q_p becomes more difficult. One way to obtain the overexpanded cube for a null cube is as follows:

1. Let P be a set of cubes obtained by replacing all null literals in p by asserted or complemented literals. If p has m null literals then P has 2^m cubes.

2. Remove every cube from P that is not contained in $g = f \cup d$.

3. Let S be the set of overexpanded cubes of cubes in P.

4. The overexpanded cube of p is the super cube of S. If $S = \emptyset$ then its super cube is not defined.

Any prime cube of $\mathcal{F}$ that contains p also contains some cube in P. The overexpanded cube is the super cube (smallest cube containing) all such primes of $\mathcal{F}$. Equivalently, the overexpanded cube is the super cube of S.

The test of whether a non-null cube p' is contained in g can be made using the tautology or the offset. $p' \subseteq g$ if and only if the cofactor of g with respect to p is a tautology. Equivalently, $p' \subseteq g$ if and only if p' is orthogonal to the offset.

An algorithm for overexpanded cube that uses the offset is of no use here because the overexpanded cube is needed to expedite finding the reduced offset and the reduced offset is used to avoid computation of the offset. If tautology is to be used for finding the overexpanded cube, several tautology tests are needed. Also, each tautology test may take a long time. The algorithm presented here is more efficient compared with the one that uses tautology. It finds all literals in the overexpanded cube simultaneously. The key to the algorithm is the relationship between the overexpanded cube and the reduced offset which is given by the following Theorem:

Theorem 4.9.1 *Let $\mathcal{F} = (f, d, r)$ be an incompletely specified function. Let $p \subseteq f \cup d$ be a generalized cube, q_p be its overexpanded cube and r_p be the reduced offset for it. If a single literal cube x_i is present in r_p then q_p has literal $\overline{x}_i$.*

Proof: Let p' be a prime cube of $\mathcal{F}$ such that $p \subseteq p'$. p' must be orthogonal to r_p. If x_i is a cube in r_p then p' must have literal $\overline{x}_i$ so that it is orthogonal to cube x_i. Since any prime of $\mathcal{F}$ containing p has literal $\overline{x}_i$, so does q_p: the smallest cube containing all such primes. ■

If q is the cube obtained by multiplying out the single literal cubes in r_p then $q_p \subseteq q$. However, $q_p \not\supseteq q$ because the converse of this theorem is not true as shown in the example below:

EXAMPLE 4.4 :

$$p = A\hat{B}C$$

$$r_p = \overline{A} + B\overline{C} + \overline{B}\,\overline{C}$$

It can be shown that the overexpanded cube of p is $q_p = AC$. However, cube $\overline{C}$ is not present in r_p.

Cube $\overline{C}$ does not appear in r_p in the above example. However, r_p can be re-written as $\overline{A} + \overline{C}$. This shows that cube C is *contained* in r_p. An obvious question to ask at this point is whether a single literal cube x_i is contained in r_p if literal $\overline{x}_i$ is present in q_p. The following theorem provides the answer:

Theorem 4.9.2 *Let p be a generalized cube, q_p be its overexpanded cube, and r_p be the reduced offset for p. If a literal $\overline{x}_i$ is present in q_p then r_p contains cube x_i.*

Proof: Let $p_1, p_2, \cdots p_m$ be all primes of the underlying incompletely specified function that contain p. Since q_p has literal $\overline{x}_i$, it is present in every p_j for $1 \leq j \leq m$. Hence $p_j = \overline{x}_i q_j$ where q_j is independent of variable x_i.

Let

$$\begin{aligned} l &= p_1 + p_2 \cdots + p_m \\ &= \overline{x}_i(q_1 + q_2 \cdots + q_m) \\ &= \overline{x}_i\, t \end{aligned}$$

where

$$t = q_1 + q_2 \cdots + q_m$$

According to Property 4.7.3 of the reduced offset:

$$r_p = \overline{l} = x_i + \overline{t}$$

Cube x_i is contained in r_p because it appears in the above representation of r_p. ■

In the special case when p is not a null cube, the following stronger theorem holds:

Theorem 4.9.3 *Let $\mathcal{F} = (f, d, r)$ be an incompletely specified function. Let $p \subseteq f \cup d$ be a (non-null) cube and q_p be its overexpanded cube. Let $\tilde{r}_p$ be a representation of the reduced offset for p consisting only of literals which are complements of literals in p. A literal x_i is in q_p if and only if cube $\overline{x}_i$ is present in $\tilde{r}_p$.*

Proof: Since a non-null cube is a generalized cube, it follows from Theorem 4.9.3 that if cube $\overline{x}_i$ is in $\tilde{r}_p$ then q_p has literal x_i.

Now suppose that cube $\overline{x}_i$ is not present in $\tilde{r}_p$. Let p' be a cube obtained from p by removing x_i. Since p' differs p only in variable x_i, p' is automatically orthogonal to any cube in $\tilde{r}_p$ that is independent of variable x_i. Any cube in r_p with literal $\overline{x}_i$ has at least one more literal x_j. Since $\tilde{r}_p$ consists only of literals which are complements of literals in p, p has literal $\overline{x}_j$. Therefore, p' also has literal $\overline{x}_j$. As a result, p' is orthogonal to $\tilde{r}_p$. p' is a prime of $\mathcal{F}$ or is contained in a prime of $\mathcal{F}$. In either case, there exists a prime of $\mathcal{F}$ containing p that does not have literal x_i. Hence, q_p does not have literal x_i. ■

Theorem 4.9.3 reduces the problem of finding the overexpanded cube of a cube to that of identifying the single literal cubes in the reduced offset. Once all the single literal cubes in the reduced offset are known, the overexpanded cube can be obtained by the product of their complements. However, the single literal cubes must be deduced without computing the reduced offset. The rest of this section and the next section will be devoted to describing algorithms for the overexpanded cube q_p of a given cube p based on deducing single literal cubes in r_p. If p is a null cube, then these algorithms will provide some cube $q \subseteq q_p$ according to Theorem 4.9.1. It will be explained in a latter section, how a cube $q \subseteq q_p$ for a generalized cube p can be used to improve the algorithm for computing the reduced offset for p.

To see how the single literal cubes come about in the reduced offset, consider the algorithm described in section 4.8 for the reduced offset. The recursive application of Shannon's expansion in the algorithm amounts to decomposing $g = f \cup d$ into unate cofactors such that

$$g = \sum c_i U_i$$

where U_i is a cofactor of g with respect to cube c_i, the c_i are orthogonal cubes, and $\sum c_i$ is a tautology. It is known from the unate recursive algorithm for complementation that:

$$\overline{g} = \sum c_i \overline{U}_i$$

For a given generalized cube p, applying the reduction operator R_p on both sides, we get:

$$R_p(\overline{g}) = \sum R_p(c_i) R_p(\overline{U}_i)$$

Using Theorem 4.8.1 gives $R_p(\overline{U}_i) = \overline{V}$. Therefore:

$$R_p(\overline{g}) = \sum R_p(c_i) \overline{V}_i$$

There are two possible ways of getting a single literal cube in $R_p(\overline{g})$:

1. A single literal cube x appears in $\overline{V}_j$ for some j and $R_p(c_j)$ is a tautology. x can be present in $\overline{V}_j$ if and only if literal $\overline{x}$ is present in every cube of V_j.

2. $R_p(c_j)$ is a single literal cube for some j and $\overline{V}_j$ is a tautology. This happens when $|R_p(c_j)| = 1$ and $V_j = \emptyset$ i.e U_j does not have any cube that contains p.

In the first case, the single literal cube in $R_p(\overline{g})$ is x. In the second case, the single literal cube is $\overline{R_p(c_j)}$. In either case, the overexpanded cube has the complement of the single literal cube present in $R_p(\overline{g})$.

Subroutine *find_oc*() shown in Algorithm 4.3 computes the overexpanded cube based on these observations. It is a recursive algorithm. For the first call to *find_oc*(), $q_p = 1$, $c = 1$ and $h = f \cup d$. Only those cofactors with cofactoring cubes c are looked at, for which $|R_p(c)| \leq 1$.

It is interesting to note that many literals in the overexpanded cube can be obtained simultaneously in *find_oc_unate*() if $|R_p(c)| = 0$. This is unlike a tautology based algorithm where each literal in p is tested for inclusion in q_p individually. It is also interesting to note that no cube-specific information is used until *find_oc_unate*() is called. If the overexpanded cubes for several p's are desired, then it is desirable to store at least part of the cofactoring tree of $f \cup d$ so that the cofactors h_x and $h_{\overline{x}}$ in *find_oc*() need not be computed for each p. h is a cofactor of $f \cup d$. This is another reason for the efficiency of this algorithm over the tautology based one.

4.10 Improving Overexpanded Cube Computation

Algorithm 4.3 can be improved significantly if some way can be found to drop cubes from h during recursive Shannon cofactoring. This will help in getting to the unate cofactors fast. Theorem 4.10.1 provides one way to drop cubes.

Theorem 4.10.1 *Let p be a cube whose overexpanded cube q_p is needed and q a cube in $g = f \cup d$. If $|R_p(q)| \geq 2$ then q can be removed for computing q_p.*

Proof: $|R_p(q)|$ is the number of literals in $R_p(q)$. Only those unate cofactors of g with cofactoring cubes c_i need be considered for which $|R_p(c_i)| \leq 1$. Each cube in any unate

Algorithm 4.3

****

Input: p, overexpanded cube q_p so far, cofactor h of $g = f \cup d$, and cofactoring cube c such that $h = g_c$.

Output: Updated overexpanded cube q_p for p.

```
find_oc(p, q_p, c, h)
{
    If (h is unate) {
        find_oc_unate(p, c, q_p, h)
    } Else { /* h is not unate */
            Choose a binate variable x in h.
            If (|R_p(cx)| ≤ 1)
                find_oc(p, q, cx, h_x)
            If (|R_p(cx̄)| ≤ 1)
                find_oc(p, q, cx̄, h_x̄)
    }
}

find_oc_unate(p, c, q_p, h)
{
    If (|R_p(c)| = 0) {
        Form V from h by removing those cubes that don't contain p.
        For each literal x_i present in every cube of V
            q_p = q_p x̄_i
    } Else { /* |R_p(c)| = 1 */
        If (no cube in h contains p) /* V = ∅ */
            q_p = q R_p(c)‾
    }
}
```

cofactor is the result of cofactoring a cube in g. Consider a unate cofactor U of g with cofactoring cube c. Let s be a cube in U which is the result of cofactoring q with respect to c. If $|R_p(c)| = 0$ then $|R_p(s)| \geq 2$ and if $|R_p(c)| = 1$ then $|R_p(s)| \geq 1$. In either case, s does not contain p because it is orthogonal to p. Therefore it can be dropped from U. Hence q can be dropped from g. ■

In addition to dropping some cubes in the beginning, it is possible to drop additional cubes during recursion according to the following theorem.

Theorem 4.10.2 *Let p be a generalized cube to be expanded, h a non-unate cofactor of $f \cup d$ and q a cube of h. If for some unate variable, x, of h, the literal of x in q does not contain that in p, then q can be removed for computing either the reduced offset or the overexpanded cube for p.*

Proof: Since h is unate in x, during a recursive Shannon cofactoring of h, x will not be used as a cofactoring variable. Each cube in the unate cofactors of h is the result of cofactoring some cube in h. Let r be a cube that resulted from q. The literal of x in r does not contain that in p. Hence, r does not contain p and will be removed according to Theorem 4.8.1. Since any result of q will ultimately be removed, then q can be removed. ■

In order to get further improvement of algorithm 4.3, some cofactors h in $find\_oc()$ can be dropped without any loss of information. Note that if $|R_p(c)| = 0$ in the subroutine then only those variables can be added to q_p by any unate cofactor of h which are present in both h and p. Hence, if all variables in h and p are already in q_p then there is no need to pursue h. Any variable present in the cofactoring cube c is not present in the cofactor h. As a result, the number of variables left in h decreases as Shannon's cofactoring goes on.

If $|R_p(c)| = 1$ for h in $find\_oc()$ then only the literal $x = \overline{R_p(c)}$ can be added to q_p by calling $find\_oc()$ with h. If x is already in q_p, there is no need to pursue h.

On the other hand, if $x = \overline{R_p(c)}$ is not in q_p then h can not be dropped. However, some cubes in h may be dropped according to the following Theorem:

Theorem 4.10.3 *Let h be a cofactor of $f \cup d$ with cofactoring cube c, p a cube whose overexpanded cube is desired. Let $|R_p(c)| = 1$ and $\tilde{p}$ obtained by removing $x = \overline{R_p(c)}$ from p, and $h_{\tilde{p}}$ the cofactor of h with respect to $\tilde{p}$. Then x is added to q_p by $find\_oc(p, c, q_p, h)$ if and only if it is added by $find\_oc(p, c, q_p, h_{\tilde{p}})$.*

Proof: Without loss of generality, suppose that $\tilde{p} = x_{i_1} x_{i_2} \cdots x_{i_m}$ then $\tilde{p} + \overline{x}_{i_1} + \overline{x}_{i_2} + \cdots + \overline{x}_{i_m} = 1$. Hence

$$h = \sum_{j=0}^{m} c_j h_{c_j}$$

where h_{c_j} is a cofactor of h with cofactoring cube c_j, $c_0 = \tilde{p}$, and $c_j = \overline{x}_{i_j}$ for $1 \leq j \leq m$. Each h_{c_j} is a cofactor of $g = f \cup d$ also with cofactoring cube $c\,c_j$. x is added to q_p by $find\_oc(p, c, q_p, h)$ if and only if a unate cofactor of some h_{c_j} with $|R_p(c\,c_j)| = 1$ exists with no cube that contains p. Since x_{i_j} is a literal in p, $|R_p(c\,c_j)| = 2$ for $1 \leq j \leq m$ therefore the only h_{c_j} that can have such a unate cofactor is $h_{c_0} = h_{\tilde{p}}$. But in that case $find\_oc(p, c, q_p, h_{\tilde{p}})$ will also add x to q_p. ■

Algorithms 4.4, 4.5 and 4.6 present improved subroutines to compute the overexpanded cube. Initially, $find\_oc0()$ is called with $q_p = 1$, $c = 1$, and $h = f \cup d$. As soon as $|R_p(c)|$ becomes 1, $find\_oc0()$ calls $find\_oc1()$. When h becomes unate in $find\_oc0()$, it calls $find\_oc\_unate0()$. Similarly, when h becomes unate in $find\_oc1()$, it calls $find\_oc\_uante1()$. Subroutine $find\_oc0()$ is shown in algorithm 4.4. Subroutine $find\_oc1()$ is shown in algorithm 4.5. Subroutines $find\_oc\_unate0()$ and $find\_oc\_unate1()$ are shown in 4.6.

4.11 Using the Overexpanded Cube for Computing the Reduced Offset

The algorithms presented in the last two sections compute the overexpanded cube for a non-null cube exactly. However, for a null cube p, they only provide a cube q that contains the overexpanded cube q_p. This section describes how the knowledge of any cube $q \supseteq q_p$ can be used to expedite computation of the reduced offset. This allows the possibility of $q = q_p$. The key to the usefulness of q and q_p is the following Lemma:

Lemma 4.11.1 *Let p be a generalized cube, q_p be its overexpanded cube and r_p be the reduced offset for it. If q is a cube such that $q_p \subseteq q$ then*

$$r_p = \overline{q} + t_p^q$$

where t_p^q is the sum of cubes in r_p, none of which is orthogonal to q.

Algorithm 4.4

****

Input: p, overexpanded cube q_p so far, cofactor h of $g = f \cup d$, and cofactoring cube c such that $h = g_c$ and $|R_p(c)| = 0$.

Output: Updated overexpanded cube q_p for p.

```
find_oc0(p, q_p, c, h) {
    If (c = 1) {
        Remove each cube from h for which |R_p(c)| > 1.
    } Else {
        Let S be the set of variables present in both p and h.
        If (each variable in S is in q_p)
            return;
    }
    If (h is unate) {
        find_oc_unate0(p, c, q_p, h)
    } Else { /* h is not unate */
        Remove each cube from h that does not contain p in the unate variables.
        Choose a binate variable x in p.
        For (y = x and x̄) {
            If (R_p(cy)| = 0) {
                find_oc0(p, q_p, cy, h)
            } Else if (ȳ not in q_p) {
                Obtain p̃ by removing variable x from p.
                h_p̃ = cofactor of h with respect to p̃.
                find_oc1(p, q_p, c, h)
            }
        }
    }
}
```

Algorithm 4.5

****

Input: p, overexpanded cube q_p so far, cofactor h of $g = f \cup d$, and cofactoring cube c such that $h = g_c$ and $|R_p(c)| = 1$.

Output: Updated overexpanded cube q_p for p.

```
find_oc1(p, q_p, c, h) {
    If (h is unate) {
        find_oc_unate1(p, c, q_p, h)
    } Else { /* h is not unate */
        Remove each cube from h that does not contain p in the unate variables.
    }
    Choose a binate variable x in p.
        If (|R_p(cx)| = 1) {
            find_oc1(p, q_p, cx, h)
        }
        If (|R_p(c x̄)| = 1) AND (R_p(c) not in q_p) {
            find_oc1(p, q_p, cx̄, h)
        }
}
```