THE COMPLEXITY OF MONOTONE BOOLEAN FUNCTIONS AND AN ALGORITHM FOR FINDING SHORTEST PATHS IN A GRAPH

by

Peter Anthony Bloniarz

January 1979

© Massachusetts Institute of Technology

This report is a minor revision of a thesis submitted in August, 1977 in partial satisfaction of the requirements for the degree of Doctor of Philosophy in Computer Science

This research was partially supported by National Science Foundation Grants MCS76-14294, MCS74-12997-A04, and MCS77-19754. The writing was partially supported by NSF Grant MCS78-04346.

Massachusetts Institute of Technology
Laboratory for Computer Science

Cambridge Massachusetts 02139

This empty page was substituted for a blank page in the original document.

THE COMPLEXITY OF MONOTONE BOOLEAN FUNCTIONS AND AN ALGORITHM FOR FINDING SHORTEST PATHS IN A GRAPH

by

PETER ANTHONY BLONIARZ

Submitted to the

Department of Electrical Engineering and Computer Science
on August 12, 1977 in partial fulfillment of the requirements
for the Degree of Doctor of Philosophy

ABSTRACT

The first part of this thesis considers the complexity of Boolean functions. The main complexity measures used are the number of gates in combinational networks and the size of Boolean formulas. The case of monotone realizations, using only the operations of AND and OR, of monotone functions is emphasized.

For a particular class of monotone functions, the quadratic functions, the worst-case values for the monotone circuit complexity is shown to be proportional to $n^2/\log n$. The number of $\wedge$-gates necessary to compute any quadrastic function is also analyzed.

A technique for deriving bounds on monotone circuit size of threshold functions is applied to the "majority" function (threshold n/2) to establish a lower bound on its monotone circuit complexity of 3n-O(1). For the function "threshold 2", previously known lower bounds on the number of $\vee$-gates required are extended in the case of a circuit which has a minimal number of $\wedge$-gates.

As a result, it follows that no monotone circuit for "threshold 2" can simultaneously have both a minimal number of $\wedge$-gates and a minimal number of $\vee$-gates.

The complexity of combinations of functions on disjoint sets of variables is studied, and a gap between the formula and circuit size of a particular function is given.

Finally, we study the effect of allowing negation in a formula for monotonic functions. Examples are given both of functions in which using negations allows more succinct expressions, and functions in which it does not.

The second part of the thesis describes an algorithm for computing shortest paths in a graph. These results show that an algorithm originally proposed by Spira for this problem can have slow running time. The lacuna in his algorithm is repaired, and it is shown to have $O(n^2(\log n)^2)$ average running time over wide classes of graphs as Spira originally claimed. As a special case, a transitive closure algorithm with $O(n^2 \log n)$ average time is also described.

Thesis Supervisor: Albert R. Meyer, Professor of Computer Science.

Key Words: Boolean functions, circuit complexity, Boolean formula size, monotone networks, threshold functions, quadratic functions, algorithms, shortest paths, directed graphs, transitive closure, computational complexity.

ACKNOWLEDGEMENTS

This thesis is dedicated to my friend and companion Mary Ellen. Thanks for everything.

I'd like to thank all my friends who made my life more enjoyable because I knew them. Thanks to Albert and Ron for showing me what research was all about, and for having the faith and patience to see me through this. Thanks to Paul and Mary for their own unique contributions to the completion of this manuscript. Thanks to Wendy for typing this during her vacation, and Peter Elias for struggling through many versions of this thesis.

TABLE OF CONTENTS

Title Page.....1

Abstract.....3

Acknowledgements.....5

Table of Contents.....7

Chapter 1. Introduction.....9

Chapter 2. Definitions and Preliminary Results.....19

Section A. Boolean Functions.....19

Section B. Representations of Boolean Functions.....24

Section C. Complexity Measures.....42

Section D. Circuits and Turing Machines.....48

Section E. Miscellaneous Notations.....50

Chapter 3. Worst-Case Values for the Complexity of Quadratic Functions.....53

Section A. The Total Number of Gates.....55

Section B. Asymptotic Bounds on $\wedge$-Gates.....61

Section C. A Graph Problem.....69

Section D. Open Questions.....77

Chapter 4. Combinations of Functions.....79

Section A. The Formula Size of $\nu(f \times g)$.....82

Section B. Monotone Functions.....86

Section C. Open Questions.....92

Chapter 5. The Monotonic Circuit Complexity of Threshold Functions.....93

Section A. The Class of All Threshold Functions.....98

Section B. The Monotonic Circuit Complexity of Threshold 2....113

Section C. M-Circuits for Threshold 2 which are $\wedge$-minimal.....134

Section D. Efficient Circuits for Threshold Functions.....155

Section E. Open Questions.....158

Chapter 6. The Complexity of Monotone Functions in Other Bases...159

Section A. A Case where a Complete Basis Doesn't Help.....162

Section B. A Case where B_2 Can Help.....173

Section C. Open Questions.....175

Chapter 7. A Shortest-Path Algorithm.....177

Section A. Shortest Paths and Graph Distributions.....178

Section B. The Algorithm.....183

Section C. Analysis of the Algorithm.....192

Section D. Implementation.....198

Section E. A.I. Distributions.....202

Section F. Open Questions.....203

Appendix 1.....205

Appendix 2.....207

Appendix 3.....213

Bibliography.....219

Biography.....225

CHAPTER 1

Introduction

This thesis consists of two parts, each devoted to a different area in the theory of computation. It is primarily devoted to a study of the complexity of Boolean functions, with an emphasis on monotone Boolean functions. A secondary objective is the analysis of algorithms which compute the shortest distances between all points in a graph and which compute the transitive closure of a Boolean matrix.

In general, computational complexity theory asks questions about the computational resources required to solve a problem or compute a function. Traditional complexity theory has been primarily concerned with the difficulty of computing recursive functions on various models of machines such as Turing Machines or Random Access Machines; measures such as time or space on these machines are considered. Finite functions (i.e. those with finite domain), since they are computable by finite state machines, are inherently "easy" according to these definitions since they require only enough time and space to read the input and print the output. Several models of computing machines for finite functions have been proposed over the past years, and these have been studied as a means for defining the complexity of finite functions. Some of these, such as circuit complexity, receive their motivation from computer technol-

ogy, while others such as formula size are more mathematical in origin. In the work which has been done so far, these models are seen to have a rich mathematical structure which is only now beginning to be understood to any extent. The bulk of the research contained in this thesis is a study of several problems in this area of finite computation.

We restrict ourselves to the Boolean case in which the finite domain and range are vector spaces over the set {0,1}. The original work in this area was directed towards finding asymptotic results about the worst-case complexity of all Boolean functions on n variables. Work of Shannon [1949], Lupanov [1958,1962], and others [Krichevskii 1961] established that "most" Boolean functions on n variables have minimal formulas of size asymptotic with $2^n/\log n$,† and minimal circuits of size asymptotic with $2^n/n$. During the 1950's attention was focused more on individual "practical" problems, and on finding optimal or near-optimal canonical formulas and circuits for a function with a given input-output specification. Work of Quine [1952,1955] and others [McCluskey 1956, Karnaugh 1953] explored methods for finding efficient ways of computing or expressing a specific function.

Recent research has been directed primarily at finding exact values for the complexity of individual functions in different models of complexity. As an adjunct to this pursuit, other more

---

† In this report, all logarithms are to the base 2.

basic questions such as the relationship between complexity in different models have been studied. This search for optimal circuits and formulas is only indirectly motivated from an engineering point of view†, but is relevant to complexity theory.

Several authors [Pippenger and Fischer 1977, Schnorr 1976d] have demonstrated a connection between the Turing Machine complexity of a function and the circuit complexity of finite restrictions of that function. These results essentially show that a function is easy to compute with respect to oracle Turing Machine time if and only if it has a small circuit on each set of finite length inputs. This work relates to recent studies of non-deterministic polynomial time (*NP*) and to the *NP*-complete problems of Cook [1971] and Karp [1972] because it implies that if $P = NP$, then the finite restriction of any problem in *NP* has a polynomial-sized circuit. While this latter question is obviously still open, this observation has given impetus to many researchers to explore the possibility of proving large lower bounds on the complexity of specific Boolean functions.

---

† Considerations such as the number of gates in a circuit or the delay time are not as important as other factors such as the interconnections between gates and the fan-out of the gates.

The results obtained thus far have not been very encouraging in pursuit of this goal. The lower bounds obtained may be divided into two types. Diagonal-type arguments can obtain exponential lower bounds on the complexity of specific functions. For example, Stockmeyer [1974] and Meyer have shown that, since there are functions which are recognizable on a deterministic Turing Machine in space 2^n whose restrictions to inputs of fixed length have exponential circuit complexity, one can get a lower bound of c^n on any circuit which computes a restriction of certain logical languages. Thus far, results of this sort have been limited to Boolean functions which are binary codings of problems which are provably difficult to compute on a Turing Machine; the functions involved have not arisen as Boolean functions in their own right. Moreover, since the *NP*-complete problems are not yet provably difficult for a Turing Machine, these techniques do not apply to them.

Direct arguments for particular functions have not yielded much in the way of lower bounds. For single-output functions, no lower bounds on circuit complexity larger than linear in the number of inputs have yet been proven.

The order of these bounds has not been improved even when the allowable gates in the circuit are highly restricted. For the complete basis of all binary Boolean operations, the largest of these lower bounds is 2.5n recently discovered by Paul [1977]

and Stockmeyer [1977]. †For restricted bases, the largest known lower bound is 7n for particular functions [Redkin 1973]. Work contained in this thesis establishes lower bounds on the monotone circuit complexity of a particular threshold function which is equal to the largest lower bound achieved for this basis (namely 3n).

For multi-output functions, larger lower bounds have been achieved on the circuit complexity in certain bases by exploiting the interplay between the functions computed by different outputs. For example, Paterson [1975] and Mehlhorn and Galil [1976], indepently extending earlier work of Pratt [1974], have shown that the monotone circuit complexity of the multiplication of two nxn matrices is $2n^3-n^2$. ††Several other researchers have used different techniques to obtain lower bounds proportional to n log n on functions related to sorting problems [Lamagna and Savage 1974, Lamagna 1975].

We observe that in both cases the bounds obtained are far short of the exponential lower bounds theoretically possible for most functions.

---

† Recently, Lipton and Tarjon [1977] exhibited larger bounds for planar circuits, and Schnorr [1978b] has announced a 3n lower bound for arbitrary circuits.

†† Recently, these techniques have been extended to yield an $\Omega(n^2/\log^2 n)$ lower bound on the monotone circuit complexity of certain functions [Wegener 1978]

For formulas the results are also limited. If arbitrary connectives are allowed in the formula, the largest lower bounds are given by a general technique of Nečiporuk [1966] which he uses to establish lower bounds proportional to $n^2/\log n$ for a specific function. Harper and Savage use Nečiporuk's method to establish lower bounds on the complexity of the marriage problem [1972].

This technique is used later in this thesis to exhibit a monotone function for which negations don't allow a reduction in the formula size by more than a constant factor. Other lower bounds for formulas in which arbitrary binary connectives are allowed have been reported by several authors [Hansel 1964, Hodes and Specker 1968, Vilfan 1972, and Fischer, Meyer and Paterson 1975]. For more restricted sets of connectives, Khrapchenko [1971] established lower bounds on the $\{\wedge,\vee,\neg\}$ formula complexity for the parity function proportional to n^2, and this is the largest polynomial bound yet established for formula complexity.

The principal research contained in this thesis is the establishment of lower bounds on the complexity of specific Boolean functions, but includes more general work on asymptotic bounds on the complexity classes of Boolean functions and on the relationship between several complexity measures. In particular, we begin by studying the monotone circuit complexity of quadratic monotone Boolean functions. (Those for which each prime implicant is a

product of two distinct variables.) Using simple counting techniques we show that most quadratic functions require at least $\Omega(n^2/\log n)$[†] gates to compute in any circuit. On the other hand, we show that a monotone circuit of size proportional to $n^2/\log n$ exists for every quadratic function. When counting individual types of gates in a monotone circuit, we show that $n-1$ $\wedge$-gates suffice to compute any quadratic function, and exhibit a specific function for which $2n/3$ $\wedge$-gates are necessary in any monotone circuit.

In chapter 4 we examine the question of whether it may be easier to compute combinations of several functions than to compute the individual functions and then combine them. Both monotone and general circuits and formulas are considered. As a corollary to the work reported there, we establish the existence of a function of polynomial complexity which has smaller circuit complexity than formula complexity -- in fact their ratio is proportional to $n/\log\log n$.

Chapter 5 contains an examination of the monotone circuit complexity of the threshold functions. Using techniques similar to Paul [1977], Stockmeyer [1977], and Schnorr [1974], we establish larger lower bounds on their complexity than previously known. These bounds are in some cases quite small in comparison with the best known upper bounds. In particular, for the function "threshold $n/2$" of n variables, we establish a lower bound of

---

[†] For notation, see page 51.

3n-7 gates necessary in any monotone circuit; [†] on the other hand, the smallest known circuit has a number of gates proportional to $n(\log n)^2$.

For the function "threshold 2" of n variables, we present exact bounds on the number of $\wedge$-gates and $\vee$-gates for any monotone circuit. These bounds were observed by F.F. Yao[††]. We extend these bounds in the case of a circuit with the minimal number of $\wedge$-gates for values of n which are a power of 2, and show that for these values of n it is impossible to simultaneously achieve the minimum number of both types of gates. The latter result was again previously observed by F.F. Yao.

Finally, we study the effect of different bases on the formula complexity of monotone functions. In work done jointly with M. Paterson, we exhibit a function with monotone formula complexity proportional to $n^2/\log n$ for which any formula in an arbitrary basis can be no more than a constant factor smaller. On the other hand, in work done jointly with A. Meyer, we show that there are monotone Boolean functions for which the smallest monotone formula is larger by a factor of $\theta(n)$ than a formula for the function in the basis of all binary operators.

[†] Recently these techniques have been extended by the author to yield a 3n-O(1) lower bound on the $\{\wedge, \vee, \neg\}$ complexity of this function.

[††] Personal communication, 1975.

The final part of this thesis deals with the problems of determining the transitive closure of a Boolean matrix and determining the shortest distances between all points of a non-negatively weighted graph. P.M. Spira [1973] has published an algorithm to find the shortest distance matrix which he claims has an average running time (over a large class of weighted directed graphs) of $O(n^2 \log^2 n)$. Research by A. Meyer, M.J. Fischer, and this author has demonstrated classes for which his algorithm has slow average running time. A revision of his algorithm does indeed have $O(n^2 \log^2 n)$ average time, over even wider classes of graphs than Spira claimed. This result is presented in Chapter 7; it is also shown there that a further revision yields a simple $O(n^2 \log n)$ average time algorithm to compute the Boolean transitive closure over a wide class of probability distributions on matrices.

We begin by presenting some introductory remarks on Boolean functions.

CHAPTER 2

Definitions and Preliminary Results

Section A. Boolean Functions

An n-input, m-output Boolean function is a function with domain $\{0,1\}^n$ and range $\{0,1\}^m$. We will denote the set of all n-input, m-output Boolean functions by $B_{n,m}$. We will generally be interested in one-output Boolean functions and denote the set $B_{n,1}$ of all such functions with n inputs as B_n. If $f \in B_{n,m}$, we usually denote an arbitrary element of the domain of f by $\vec{x} = (x_1, x_2, \ldots, x_n)$, and refer to each x_i for $1 \leq i \leq n$ as a variable of f. Some specific Boolean functions we will refer to are the unary function negation or NOT (denoted $\neg$), and the binary functions disjunction or OR (v), conjunction or AND ($\wedge$ or $\cdot$) [†], EXCLUSIVE OR or mod 2 sum ($\oplus$), equivalence ($\equiv$); and the constant functions 0 and 1.

The sets of functions $B_{n,m}$ have been extensively studied from a mathematical point of view [Harrison 1965, MacLane and Birkhoff 1967]. We assume that the reader is familiar with elementary properties of these sets. We use notation which is standard and refer the reader to Harrison [1965] for elaboration and proofs of

---

† Frequently we will denote the conjunction of variables x and y by their concatenation xy.

results that we only state[†].

Boolean functions may be combined in standard fashion. In particular the composition of functions is defined as usual. If f and g are members of B_n for some n, we define $fvg \in B_n$ by $(fvg)(\vec{x}) = f(\vec{x})vg(\vec{x})$ for any $\vec{x} \in \{0,1\}^n$. If a collection of functions $(f_i \mid i \in I)$ is each a member of B_n, the disjunction $\bigvee_{i \in I} f_i \in B_n$ is defined in the obvious fashion. The conjunction of functions is defined similarly. In addition, we may define, from the functions $f \in B_{n_1,m_1}$ and $g \in B_{n_2,m_2}$, the function $fxg \in B_{n_1+n_2,m_1+m_2}$ by

$$fxg(x_1,x_2,\ldots,x_{n_1},y_1,y_2,\ldots,y_{n_2}) = (f(x_1,\ldots,x_{n_1}),\ g(y_1,\ldots,y_{n_2}))$$

(where we use the obvious isomorphism between $\{0,1\}^{m_1} x \{0,1\}^{m_2}$ and $\{0,1\}^{m_1+m_2}$). The combination $v(fxg)$ is defined, for $f \in B_{n_1}$

[†] Examination of the properties of Boolean functions indicates strong similarities among them. In fact, they may be grouped into pairs in which occurrences of $\wedge$ and $\vee$ are interchanged, as well as occurrences of 0 and 1. Such statements are called <u>duals</u> of each other. It is a well known fact that a Boolean statement is true if and only if its dual is true. This is known as the <u>pricipal of duality</u>, and permits some economy in proving properties of Boolean functions.

and $g \in B_{n_2}$, to be the obvious composition of $\vee$ and $\times$; that is

$\vee(f \times g)(x_1,\ldots,x_{n_1},y_1,\ldots,y_{n_2}) = f(x_1,\ldots,x_{n_1}) \vee g(y_1,\ldots,y_{n_2})$.

$\wedge(f \times g)$ is defined similarly.

Two properties of these combinations which we will use are stated in the following lemma:

2.1 Lemma: Suppose f and g are members of B_n for some $n \geq 0$. Then (a) if $f \vee g = \underline{0}$, the constant function 0, then $f = 0$ and $g = 0$

and (b) if $f \wedge g = \underline{1}$, then $f = \underline{1}$ and $g = \underline{1}$.

One specific set of Boolean functions is the set of projection functions. The i^{th} projection function of n variables, denoted Π_i^n, is defined by $\Pi_i^n(x_1,\ldots,x_n) = x_i$ for each i in $\{1,2,\ldots,n\}$. If f is an arbitrary member of $B_{n,m}$, we denote by f_i the function $\Pi_i^m \circ f$, the i^{th} component of f.

We say that a function $f \in B_{n,m}$ functionally depends on its i^{th} variable x_i if there are constants $\vec{c}$ and $\vec{d}$ in $\{0,1\}^n$ which differ only in their i^{th} positions for which $f(\vec{c}) \neq f(\vec{d})$. Finally, if A is some set of input variables for f, and $c_x \in \{0,1\}$ is a constant defined for each $x \in A$, then we denote the restricted function of the remaining variables which is obtained from f by setting each variable x in A to c_x by $f|_{x = c_x}$ for $x \in A$.

We are particularly interested in one subclass of Boolean functions, the monotone Boolean functions (m.b.f.'s). Observe that

the set $\{0,1\}^n$ is partially ordered by the rule

$$(x_1,\ldots,x_n) \leq (y_1,\ldots,y_n) \text{ iff } x_i \leq y_i \text{ for } i = 1,2,\ldots,n$$

(where $0 \leq 1$).

The set $B_{n.m}$ is partially ordered (in fact is a Boolean algebra) by defining, for $f,g \in B_{n,m}$,

$$f \leq g \text{ iff } f(\vec{x}) \leq g(\vec{x}) \text{ for all } \vec{x} \in \{0,1\}^n.$$

A Boolean function $f \in B^{n,m}$ is <u>monotonic increasing</u>, or <u>monotone</u> for short, if and only if it preserves the partial ordering $\leq$; that is, if $f(\vec{x}) \leq f(\vec{y})$ for every pair of inputs $\vec{x},\vec{y} \in \{0,1\}^n$ for which $\vec{x} \leq \vec{y}$.

Specific examples of m.b.f.'s include the function <u>threshold k</u> of n variables, denoted T_k^n, which is defined by $T_k^n(x_1,\ldots,x_n) = 1$ if and only if at least k of the variables $(x_1,\ldots,x_n)$ have the value 1. The threshold 1 function is the disjunction $x_1 \vee x_2 \vee \ldots \vee x_n$; the threshold n function is the conjunction $x_1 \wedge x_2 \wedge \ldots \wedge x_n$. Another m.b.f. is the Boolean multiplication of two $n \times n$ matrices A and B, a function in B_{2n^2,n^2} which is defined by $A \cdot B = C$ where $C_{ij} = \bigvee_{k=1}^{n} (A_{ik} \wedge B_{kj})$ for each i and j in $\{1,2,\ldots,n\}$. A third set of m.b.f.'s is Boolean convolution, in which we define $f \in B_{n,2n-1}$ by

$$f_k(x_0,\ldots,x_{n-1},y_0,\ldots,y_{n-1}) = \bigvee_{i+j=k}(x_i \wedge y_j)$$

for each $0 \leq k \leq 2n-2$.

The m.b.f.'s satisfy several properties which are not true of Boolean functions in general. The following facts are easily verified.

2.2 Lemma: Suppose f and $g \in B_n$ are monotone Boolean functions.

Then

(a) if $f \wedge g = \underline{0}$, then $f = \underline{0}$ or $g = \underline{0}$

and (b) if $f \vee g = \underline{1}$, then $f = \underline{1}$ or $g = \underline{1}$.

Further properties of the m.b.f.'s will be explored later in this chapter. It is useful at this point to point out one property of the ordering relationship on B_n.

2.3 Lemma: Suppose that f, g, and h are functions in B_n. If $f \leq h$ and $g \leq h$, then

(1) $f \wedge h = f$

(2) $f \vee h = h$

and (3) $f \wedge g \leq f \leq f \vee g \leq h$. †

---

† We remark that the dual of an arbitrary function $f \in B_n$ is the function $D(f)$ in B_n defined by $D(f)(\vec{x}) = \neg f(\neg x_1, \neg x_2, \ldots, \neg x_n)$. The truth of a statement

(cont. on next page)

Section B. Representations of Boolean Functions

There are many different ways of representing Boolean functions, some of which give rise to the complexity measures discussed in this thesis. One standard representation is the truth table of the value of the function on different inputs. The unifying representation used in this thesis is that of a combinational (or gate-type) switching circuit, and is identical with those studied by numerous authors [Harrison 1965, Savage 1976, Paul 1977, Schnorr 1974 and others]. The underlying structure is a graph, which we now define.

A directed graph D consists of a pair of finite sets (V,E), where V is a set whose members are called nodes or vertices, and $E \subseteq V \times V$ is a set whose members are called edges or arcs. Another notation for V and E is NODES(D) and EDGES(D) respectively. If $(v,w) \in$ EDGES(D) we say there is an edge from v to w in D. A path or chain of length k in D is a sequence of nodes $v_0, v_1, \ldots, v_k$ (for $k \geq 0$) of D such that there is an edge from v_i to v_{i+1} for every $i \in \{0,1,\ldots,k-1\}$. A graph D is acyclic if there is no path from any node A of D to itself other than

---

like Lemma 2.3 is unchanged if every function is replaced by its dual.

Observe that if $f \leq g$, then $D(f) \geq D(g)$, and that the dual of a monotone function is monotone.

the trivial path of length 0. For any node A of an acyclic graph D, we define the following subsets of nodes of D which pertain to A:

$Succ(A,D) = \{B \in NODES(D) \mid$ there is an edge from A to B$\}$ is the set of immediate successors of A,

$Succ^*(A,D) = \{B \in NODES(D) \mid$ there is a path from A to B in D$\}$,

$Succ^+(A,D) = \{B \in NODES(D) \mid$ there is an path of length $k \geq 1$ from A to B$\}$,

$Pred(A,D) = \{B \in NODES(D) \mid$ there is an edge from B to A$\}$ is the set of immediate predecessors of A,

$Pred^*(A,D) = \{B \in NODES(D) \mid$ there is a path from B to A in D$\}$,

and $Pred^+(A,D) = \{B \in NODES(D) \mid$ there is a path of length $k \geq 1$ from B to A$\}$.

If W is some subset of nodes of D, we may extend the above devinitions to W and speak, for example, of the set of immediate successors of W, defined by $Succ(W,D) = \bigcup_{A \in W} Succ(A,D)$. Finally, the indegree or fan-in (respectively outdegree or fan-out) of a node A is the number of edges directed into (out of) A in D and is denoted indeg(A,D) (outdeg(A,D)). In general, we will omit mention of the graph D in the above notation when the graph is clear from context.

Suppose Ω is a finite set of primitive functions $g_i \in B_{n_i}$ called the basis. A combinational circuit or network over the basis Ω, in short an Ω-circuit, is a directed acyclic graph N together with a labelling of the nodes and edges of N which is subject to the following constraints and definitions:

(1) For some positive integer n, at most n of the nodes with indegree zero are given distinct labels from the set $\{x_1, x_2, \ldots, x_n\}$; we refer to such nodes as input nodes and denote their set by INPUTS(N).

(2) There may be one node of indegree zero labelled with the symbol ZERO, and there may be one node of indegree zero labelled with the symbol ONE; such nodes are referred to as constant nodes.

(3) There are no other nodes of indegree zero other than those mentioned in (1) or (2).

and (4) Each node with indegree one or more is labelled with some member g_i of Ω. Each such node is a gate node, and their set is denoted GATES(N). Such a node G labelled with g_i must have indegree in N equal to the arity n_i of g_i; furthermore, each edge directed into G must be labelled with one of the integers $\{1, 2, \ldots, n_i\}$ in such a way that every edge into G gets a different label.

An example of such a network is given in Figure 2.1, where the basis is the set $\{\wedge,\vee,\neg\}$. (For figures, the arcs in the graph are always directed downward. Arbitrary nodes are designated by triangles, constants or inputs by rectangles, and gates by circles).

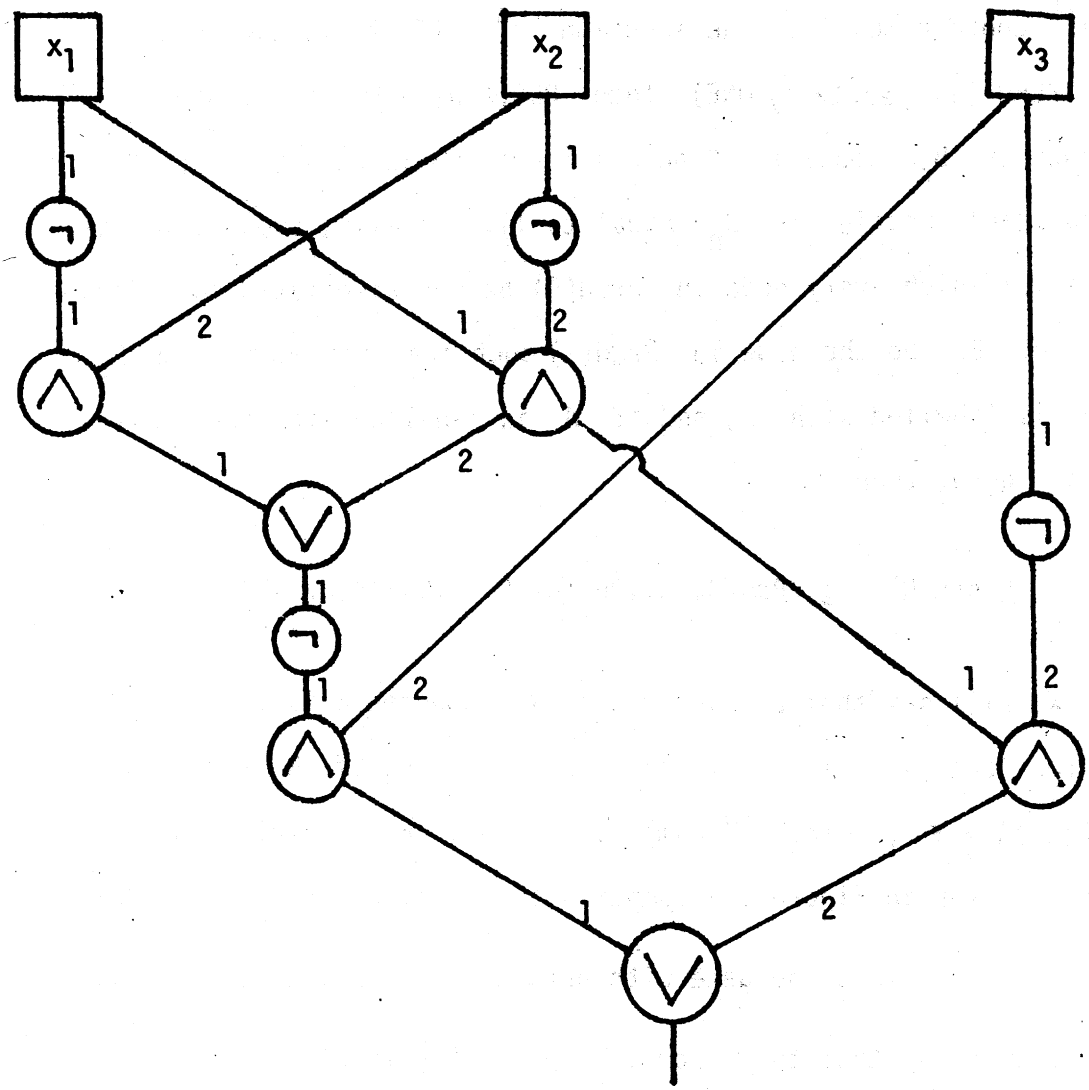

Fig. 2.1 A Combinational Circuit

If the input nodes to an Ω-circuit N are labelled with members of the set $\{x_1,\ldots,x_n\}$, we may associate a Boolean function in B_n with every node A in N by the following inductive rule. We denote the associated function by $Res(A,N)$. If A is an input node labelled with the variable x_i, then $Res(A,N)$ is the projection on the i^{th} coordinate, i.e. the function π_i^n. If A is the constant node ZERO (respectively ONE), then $RES(A,N) = \underline{0}$, the constant function in B_n equal to 0 on all inputs (respectively $RES(A,N) = \underline{1}$, the constant function in B_n equal to 1). Finally, suppose A is a gate for which every node in $Pred(A)$ has an associated function. If we let B_k be the node in $Pred(A)$ such that the edge from B_k to A is labelled with k, and if A is labelled with the basis element $g_i \in \Omega$, then

$$Res(A,N) = g_i(Res(B_1,N),Res(B_2,N),\ldots,Res(B_{n_i},N)).$$

We will say that an Ω-circuit N <u>computes</u> a function $f \in B_{n,m}$ iff for each i, $1 \leq i \leq m$, there is a node A_i of N such that $Res(A_i,N) = f_i$, the i^{th} component of f. For example, the circuit given in Figure 2.1 computes the function $f(x_1,x_2,x_3) = x_1 \oplus x_2 \oplus x_3$ since the gate with outdegree 0 computes this function. We may also say that the circuit computes the function $g(x_1,x_2,x_3) = (x_1, x_1 \wedge \neg x_2)$ since there are gates which compute each component of g. A node which computes a component of the designated function will be called an <u>output</u> node.

A basis Ω is called <u>semi-complete</u> iff given any function $f \epsilon B_n$ for arbitrary n, there is an Ω-circuit as described above which computes f. The basis is called <u>complete</u> if any f can be computed by an Ω - circuit which does not contain the nodes ZERO and ONE (i.e. constants are not available). Three complete bases in which we will be interested are B_2, the set of all binary Boolean functions, $B = B_1 \cup B_2$, and $U = \{\wedge, \vee, \neg\}$. An example of a semi-complete basis is the set $\{\oplus, \wedge\}$. An exhaustive characterization of all complete bases has been made by Post [1941], and the interested reader is referred to this work.

One incomplete basis which holds special interest is the set $M = \{\wedge, \vee\}$ which is important for the following reason.

<u>2.4 Theorem</u> : A Boolean function $f \epsilon B_{n,m}$ is monotone iff there is an M-circuit which computes f.

For a proof of Theorem 2.4 the reader is referred to [Harrison 1965 p. 189].

The bulk of this thesis is concerned with the results about M-circuits (also called <u>monotone circuits</u>) computing monotone Boolean functions.

One special class of circuits which holds special interest is the set of Boolean formulas. An <u>Ω-formula</u> is an Ω-circuit in which each node in the circuit has outdegree at most one, and which has a unique node of fan-out zero corresponding to the unique

output node. For a formula, we remove the restriction on the circuit that there be only a single node for each input variable and constant, and allow any number of nodes of indegree zero labelled with any input variable or constant, each node having outdegree one. For an example see Figure 2.2. The graph of any Ω-formula is a tree in which each leaf node (one with indegree zero) is labelled with a variable or constant, and each interior

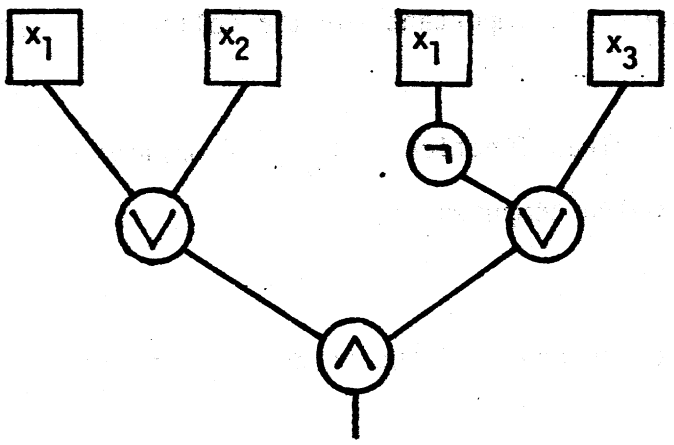

Fig. 2.2 The Formula $(x_1 \vee x_2) \wedge (\neg x_1 \vee x_3)$

node (every other node) is labelled with a gate. This formulation is easily seen to be equivalent to the usual inductive definition of an Ω-formula. Clearly a formula can compute only a single-output function.

Classically, Boolean formulas have been extensively studied as a means of representing Boolean functions. Many formula schemes have been proposed as canonical means for their representation, including the classical disjunctive normal form. In this method, a function $f \in B_n$ is represented as the formula

$$\bigvee_{(c_1,\ldots,c_n)\in A_f} x_1^{c_1} x_2^{c_2} \ldots x_n^{c_n}$$

where $$x^j = \begin{cases} x & \text{if } j = 1 \\ \neg x & \text{if } j = 0 \end{cases}$$

for any variable x, and A_f is those set of constants $\vec{c} \in \{0,1\}^n$ for which $f(\vec{c}) = 1$. Other canonical forms are discussed in [Savage 1976].

If F is a formula, and H is a node in F, then F_H will denote the subformula of F above H; that is, $Pred^*(H,F)$.

In this thesis, we will occasionally replace part of one circuit by another circuit and speak of combinations of circuits. For example, suppose that N and N′ are Ω-circuits which contain nodes A and A′ respectively. The circuit obtained by replacing A by A′ is constructed by first identifying the corresponding input and constant nodes of N and N′ and considering them as one network. Node A together with each arc directed into it is removed from the graph, and each edge (A,B) originally in circuit

N is replaced by the edge (A',B) (with the same label as previously). For example, see Figure 2.3. The replacement of a node in a circuit N

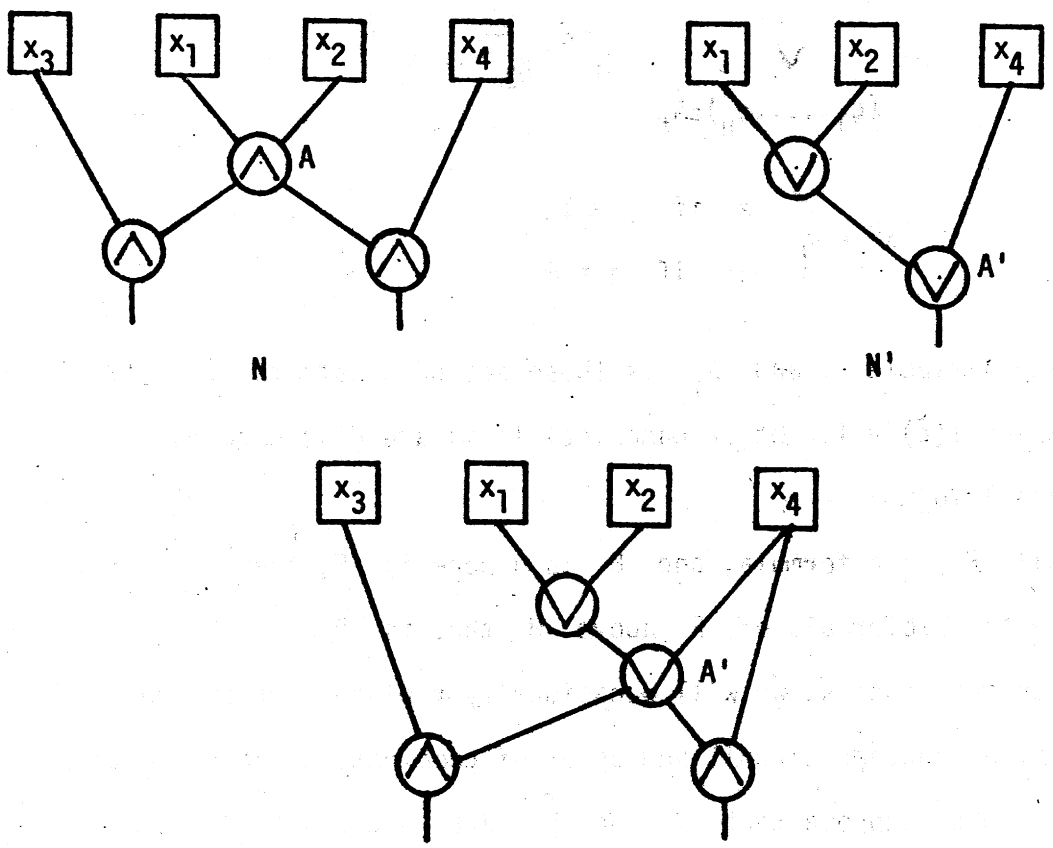

Fig. 2.3 The Replacement of A by A'

by a node in the same circuit is defined similarly, but substitution must not be made in a manner which introduces a cycle into the graph.

One may say little in general about the functions computed by a circuit in which one gate has been replaced by another. Specific situations may allow some conclusions to be drawn. For example, if node A in N is replaced by a node A' in N' for which $Res(A,N) = Res(A',N')$, then it is easy to prove by induction on the length of paths that every remaining node in N computes the same function as it did before the substitution.

One special replacement is the substitution of a constant for a variable. We say that N' is the circuit obtained from N <u>by setting variable x_i to 0</u> (respectively 1) if N' is obtained by replacing variable node x_i in N by the constant node ZERO (respectively ONE). One inductively defines setting a collection of variables to a constant. It is again easy to verify the following fact.

<u>2.5 Lemma</u>: Suppose N is an Ω-circuit, and $x_i \in INPUTS(N)$. If N' is the circuit obtained by setting variable x_i to 0, then for each remaining node A in N',

$$Res(A,N') = Res(A,N)|_{x_i = 0} \quad .$$

A similar statement may be made if x_1 is set to 1.

Observe that, depending on the basis, if a gate in a circuit has one input which is constant, then some simplification of the circuit may be made. For example, suppose the basis is the set B_2 of all binary Boolean functions. If a gate has a constant input then, except for trivial cases, it may be eliminated from the circuit since the output of that function is then a unary function of the other predecessor. Since this unary function may be absorbed into a preceding or succeding binary gate (if one such exists), then one may obtain a smaller circuit equivalent to the original. See Figure 2.4 for an example. Such simplifications may

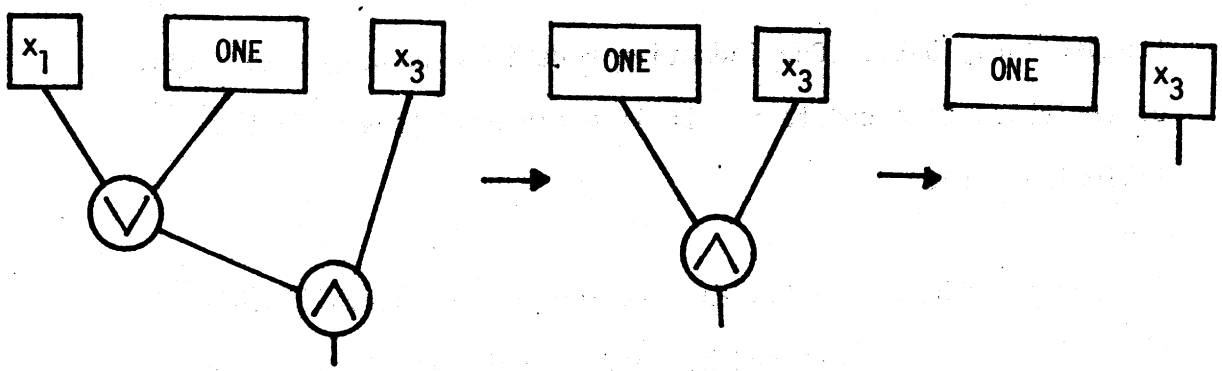

Fig. 2.4 Some Simplifications

always be made in the basis M. This elimination of gates will be a principal tool by which lower bounds on circuit complexity are obtained.

One additional representation for a Boolean function is as a set of prime implicants. In the interest of brevity we restrict ourselves to monotone Boolean functions, although a similar develop-men may be made for all Boolean functions.

A monome or product is a product of distinct variables, e.g. $x_1x_2x_5$ is a monome. In particular, we denote the empty monome by ϵ. A monome m includes a monome m' if every variable which appears in m' also appears in m. We denote the function defined by a monome m by $T(m)$, and if M is a set of monomes, then $T(M)$ will denote the function $\bigvee_{m\in M} T(m)$. $T(\epsilon)$ is the constant function $\underline{1}$.

Suppose that f is a monotone Boolean function. A monome m is an implicant of f iff $T(m) \leq f$; m is said to be a prime implicant of f if m includes no other implicant of f. We denote the set of all prime implicants of f by $PI(f)$[†].

For example, suppose $f(x_1,x_2,x_3)$ is the threshold 2 function of three variables, namely T_2^3. Then $x_1x_2x_3$ is an implicant of f, but is not a prime implicant since the monome

---

† One defines $\bigvee_{m\in\emptyset} T(m) = \underline{0}$. Hence $PI(\underline{0}) = \emptyset$, the empty set, and $PI(\underline{1}) = \{\epsilon\}$.

x_1x_2 is also an implicant of f. PI(f) is the set $\{x_1x_2, x_2x_3, x_1x_3\}$.†

The importance of the set of prime implicants is that they are a canonical representation for the m.b.f.'s, and are fairly easy to manipulate mathematically. A straightforward proof (see [Harrison 1965 p. 190] for example) demonstrates the following fact of their canonicity.

2.6 Lemma (Quine): Suppose $f \epsilon B_n$ is a monotone Boolean function. Then $f = \bigvee_{m \epsilon PI(f)} T(m)$. Moreover, if $\mathbb{P}$ is any set of monomes such that no monome in $\mathbb{P}$ includes another monome in $\mathbb{P}$, and $f = T(\mathbb{P})$, then $\mathbb{P} = PI(f)$.

The set of prime implicants of a combination of functions can be obtained from those of the constituent functions by means of the following simple set of rules. Suppose that f and g are m.b.f.'s in B_n. Then for any monome m, m is an implicant of the function fvg if and only if it is an implicant of f or an implicant of g.

---

† Note that we equate the monomes x_1x_2 and x_2x_1. Actually, the set of all monomes is equivalent to the free monoid on the elements $\{x_1, x_2, \ldots, x_n\}$ modulo the relations of the commutativity ($xy = yx$ for all variables x,y) and idempotency ($xx = x$ for all x). ϵ is the identity element for this algebraic system.

Moreover, $PI(f \vee g)$ is obtained from the union $PI(f) \cup PI(g)$ by eliminating all monomes which include another monome in the union. For example, if $PI(f_0) = \{x_1, x_2x_3, x_4\}$ and $PI(g_0) = \{x_1, x_2, x_3x_4\}$, then $PI(f_0 \vee g_0) = \{x_1, x_2, x_4\}$. If f and g are m.b.f.'s in B_n, and m is a monome such that $m \in PI(f \vee g)$, then $m \in PI(f)$ or $m \in PI(g)$, but the converse is not in general true. By Lemma 2.1(a), if $PI(f \vee g) = \emptyset$, then both $PI(f)$ and $PI(g)$ are empty sets.

The set of prime implicants of the function $f \wedge g$ may be obtained by forming the set of all products $m \cdot n$ of a monome $m \in PI(f)$ and a monome $n \in PI(g)$ (where duplicate variables in n and m are reduced by the rule $x \cdot x = x$), and then eliminating any products which include other products in the set. For f_0 and g_0 of the previous example, $PI(f_0 \wedge g_0) = \{x_1, x_2x_3, x_2x_4, x_3x_4\}$. Again, if $t \in PI(f \wedge g)$, then there are monomes $m \in PI(f)$ and $n \in PI(g)$ such that $m \cdot n = t$, but the converse is not in general true.

One may relate the ordering relation $\leq$ on the m.b.f.'s with their canonical representation using prime implicants by the following general result.

2.7 Lemma: Suppose f and $g \in B_n$ are m.b.f.'s. Then $f \leq g$ iff every monome $m \in PI(f)$ includes some monome $n \in PI(g)$.

Proof: Suppose $f \leq g$, and $m \in PI(f)$. By Lemma 2.6 and Lemma 2.3(c), $T(m) \leq \bigvee_{n \in PI(f)} T(n) = f \leq g$, so by the transitivity of $\leq$,

$T(m) \leq g$. Hence by definition m is an implicant of g. Define m_0 to be an implicant of g which is included in m and which includes no other implicant of g. Then m_0 is a prime implicant of g which is included in m.

Conversely, suppose every monome in $PI(f)$ includes some monome in $PI(g)$, and let $\vec{c} \in \{0,1\}^n$ be such that $f(\vec{c}) = 1$. By Lemma 2.6, there is some monome $m \in PI(f)$ such that $T(m)(\vec{c}) = 1$. By hypothesis, there is a monome n in $PI(g)$ included in m; By Lemma 2.3(c), we know that $T(m) \leq T(n)$. Hence $T(n)(\vec{c}) = 1$, and since n is an implicant of g, $g(\vec{c}) = 1$ as well.

□ Lemma 2.7.

One may define classes of Boolean functions in terms of structural properties of their set of prime implicants. For example, we will say that a m.b.f. $f \in B_n$ is <u>quadratic</u> if every monome in $PI(f)$ consists of the product of exactly two distinct variables; a m.b.f. $f \in B_{n,m}$ is quadratic iff every component f_i is quadratic (for $1 \leq i \leq m$).

The set of prime implicants of a function has been extensively used as a tool in explaining arguments in this field [Paterson 1975 Mehlhorn and Galil 1976, Lamagna 1975]. This representation is particularly helpful in describing several replacements which may be made in M-circuits which don't affect the output function computed. Several general theorems have been elucidated by Mehlhorn and

Galil [1976] of which special cases have been used by other authors; we will find use for one of them later in the thesis. If N is any M-circuit, and A is a node in N, then we will denote PI(Res(A,N)) by PI(A,N), and as usual omit mention of N whenever the circuit is obvious.

2.8 Lemma (Mehlhorn and Galil: Suppose N is an M-circuit, A is any node in N, and $PI(A) = \{m_0, m_1, \ldots, m_k\}$. If there is no output node B of N with $m_0 \cdot n \in PI(B)$ for some monome n, then one may replace A by a gate which computes the function $T(m_1, m_2, \ldots, m_k)$ without changing the function computed at any output gate.

For a proof of this result and several other general replacements for M-circuits the reader is referred to the paper [Mehlhorn and Galil 1976].

One final topic which pertains to circuits in general is the different notatic·s of dependence. We will say that a node A in a circuit N depends functionally on the variable x_i if Res(A,N) depends functionally on x_i. A different notion of structural (path or syntactic) dependence holds if there is a path from input node x_i to A in N, i.e. if $A \in Succ^*(x_i, N)$. Clearly if A depends functionally on x_i then it must also depend structurally on x_i, but in general the two notions do not

coincide. It is not known whether they coincide for minimal sized circuits over particular bases (M for example). However, it is possible to prove the following useful observation.

2.9 Lemma: Suppose a function $f \in B_n$ depends functionally on a variable x_i, and N is an Ω-circuit computing f with $Res(U,N) = f$ for a node U in N. Then there is a path from x_i to U in N such that every gate in the path depends functionally on x_i.

Proof: We construct the chain backwards from the output gate U. Let $U = U_0$. Inductively assume that we have constructed a path $U_k, U_{k-1}, \ldots, U_0$ in N such that every node in the path depends on x_i. If $U_k = x_i$, then the proof is complete. Otherwise, U_k must be a gate since no other input or constant node depends on x_i. Hence, since $Res(U_k)$ depends on x_i, there must be a node $B \in Pred(U_k)$ which depends functionally on x_i. We extend the path by defining $U_{k+1} = B$. (See Figure 2.5)

Since N contains no cycles, the nodes selected are all distinct. Since there are only a finite number of nodes in N, the above process must eventually halt.

□ Lemma 2.9

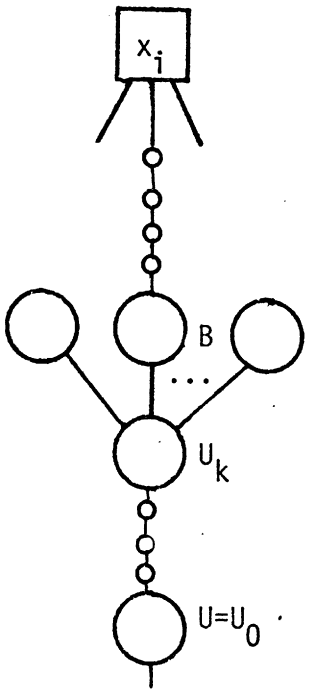

Fig. 2.5 The Extension

2.10 Corollary: If a function $f \in B_n$, which depends on x_i, is not the projection function Π_i^n, and if N is an Ω-circuit which computes f, then $\text{outdeg}(x_i, N) \geq 1$.

Section C. Complexity Measures

The models of computing Boolean functions introduced in the previous section naturally give rise to various complexity measures. If Ω is an arbitrary basis and N is an Ω-circuit, then we define $\hat{C}_\Omega(N)$ to be the number of gate-type nodes in N. If $f \in B_{n,m}$, then we define the combinational (or circuit) complexity of f with respect to Ω, denoted $C_\Omega(f)$, to be the minimum value of $\hat{C}_\Omega(N)$ for all Ω-circuits N which compute f. If f is not definable by any Ω-circuit, we define $C_\Omega(f) = \infty$.

For formulas, one possible complexity measure is to consider the number of gates as was done above for circuits. We instead use the number of occurrences of literals as our complexity measure. If F is an Ω-formula, we define $\hat{L}_\Omega(F)$ to be the number of occurrences of variable input nodes in the circuit associated with F (the number of leaves of the tree corresponding to F which are labelled with variables). Observe that if Ω is a binary basis, and F contains no constant nodes, then $\hat{L}_\Omega(F)$ is one more than the number of gates in F. The formula complexity of a function $f \in B_n$ with respect to Ω, denoted $L_\Omega(f)$, is the minimum of the values of $\hat{L}_\Omega(F)$ for all Ω-formulas F which define f. Again, if f is not definable by an Ω-formula, we define $L_\Omega(f) = \infty$. A minimal Ω-formula (respectively Ω-circuit) for f is an Ω-formula F (respectively Ω-circuit N) which computes f such that

$\hat{L}_\Omega(F) = L_\Omega(f)$ (respectively $\hat{C}_\Omega(N) = C_\Omega(f)$). Where there is no ambiguity we will use the same notations of complexity L_Ω for formulas and C_Ω for circuits as well as for functions.

For the particular monotone basis M, we will use the alternate notations MC and ML for C_M and L_M respectively. For this basis, we also define a complexity measure which depends on the types of gates used in the circuit or formula. If N is an M-circuit, define $MC_\vee(N)$ and $MC_\wedge(N)$ to be the number of v-gates and $\wedge$-gates respectively in N. If f is a m.b.f., we define $MC_\vee(f)$ (respectively $ML_\vee(f)$) to be the minimum value of $MC_\vee(N)$ as N ranges over all M-circuits (respectively formulas) which compute f (where here we consider a formula as a special type of circuit). We similarly define $MC_\wedge(f)$ and $ML_\wedge(f)$. An <u>v-minimal</u> (respectively <u>$\wedge$-minimal</u>) M-circuit for a m.b.f. f is one which has the minimal number $MC_\vee(f)$ of v-gates (respectively $MC_\wedge(f)$ of $\wedge$-gates).

The above are the complexity measures considered in the thesis, but other measures can be defined. For example, the depth of a circuit can be defined to be the length of the longest path in the circuit. This measure has been studied by Spira [1971] and more recently has been studied by several authors [McColl 1977, McColl and Paterson 1977, Paterson and Valiant 1976, and Borodin 1977]. One may similarly define the breadth of a circuit; this measure has recently been studied by Schnorr [1976b]. Finally, one may choose to consider only circuits in which the outdegree of

of each gate in the circuit is bounded above by some integer k. Johnson, Savage, and Welch established the result that if $k \geq 2$, such a complexity measure is proportional to C_Ω for any complete basis Ω, (for proof, see [Savage 1976 p.24]).

The relationship between these complexity measures of an individual function is well understood in some circumstances and unknown in others. Muller [1956] pointed out that in general, the circuit complexity of a function in any complete basis Ω is proportional to its C_{B_2} complexity. This is true since any gate in B_2 may be synthesized by an Ω-subcircuit and vice versa, so one can replace a gate by a subcircuit with only a constant factor increase in the total number of gates[†]. For formulas, the choice of complete basis can make a difference in formula size. For the parity function $f_0(x_1,\ldots,x_n) = x_1 \oplus x_2 \oplus \ldots \oplus x_n$, Khrapchenko [1971] demonstrated that $L_U(f_0)$ is proportional to n^2, while f_0 clearly has a formula of size n in the basis B_2. Pratt [1975] has recently shown that the maximal gap between the complexity in any pair of binary complete bases cannot be much larger by demonstrating that $L_U(f) \leq [L_{B_2}(f)]^{\log_3 10}$ for any $f \in B_n$.

The relationship between the B_2-complexity of a monotone function and its complexity in the incomplete basis M is not as

---

[†]In fact, $C_B(f) = C_{B_2}(f)$ for any function $f \in B_n$, and if f depends on more than one variable, then $C_B(f)$ is the minimal number of binary gates in any B-circuit which computes f. See [Savage 1976 p. 39] for a proof of this fact.

well understood. Specific results will be shown in Chapter 6.

Results about the maximum values of these complexity measures among all Boolean functions were mentioned in the introduction. In brief, the number of distinct circuits with at most a fixed number k of gates is bounded above by $ck^{c'k}$ for some constants c and c' depending on the basis. From this fact, one can easily show that if there are many different functions to compute, then there are not enough small circuits to compute them all, so "most" of them require a large number of gates. What is remarkable is that soem clever constructions exist which enable one to compute all Boolean functions in certain classes with circuits or formulas which are close to, or even asymptotic with, the lower bounds on size obtained by the above argument.

For the class of all Boolean functions, Shannon [1949] and Lupanov [1958] have shown that the maximal B_2 circuit complexity of any Boolean function $f \epsilon B_n$ is asymptotically $2^n/n$. Similarly, it is known [Krichevskii 1961, Lupanov 1962] that the maximal B_2 formula complexity is proportional to $2^n/\log n$.

Recent attention has been focused on the class of monotone Boolean functions. Kleitman and Markowsky [1975], extending work of numerous authors, have shown that there are asymptotically $2^{\binom{n}{\lfloor n/2 \rfloor}}$ m.b.f.'s of n variables, and one obtains a corresponding lower bound of order $2^n/n^{3/2}$ on the maximal B_2 (and M)

circuit complexity of m.b.f.'s. Pippenger [1976] has shown that this lower bound is attainable in showing that any m.b.f. $f \epsilon B_n$ can be computed in a B_2-circuit with asymptotically $O(2^n/n^{3/2})$ gates and an M-circuit with asymptotically $O(2^n \log n/n^{3/2})$ gates.

Lower bounds on the complexity of specific functions dominate the bulk of this thesis. It will be helpful to point out some facts about minimal circuits. If $f \epsilon B_{n,m}$ and Ω are arbitrary, then any minimal Ω-circuit for f has at most m gates of outdegree 0. This is so since any gate of outdegree 0 which is not an output gate may be eliminated from the circuit without changing the functions computed at the output nodes.

In a similar fashion, one can show that no two gates in a minimal Ω-circuit compute the same function. Suppose G and G' were distinct gates in a minimal Ω-circuit N for which $Res(G,N) = Res(G',N)$. Since N is acyclic, either $G \notin Succ^*(G')$ or $G' \notin Succ^*(G)$; w.l.o.g. we assume the former. Then we eliminate gate G, and replace G by G' in $Pred(H,N)$ for each gate $H \epsilon Succ(G,N)$. It is easily shown that the resulting circuit is acyclic, and that every remaining gate in the new circuit computes the same function as it did originally. This is a contradiction since N was a minimal Ω-circuit.

Suppose that a function $f \epsilon B_n$ depends on at least two variables Then any Ω-circuit or formula for f must contain at least one binary gate (where Ω is B_2, B, U, or M). By absorbing any

unary gates into the binary gates, one can show that in any minimal Ω-circuit for f there are no edges directed out from any constant nodes since each gate with one input constant can be simplified and eliminated. Similarly, no node G has two edges directed into the same node H, so outdeg(G) is equal to the number of nodes in Succ(G).

For minimal formulas, we point out that the notions of structural and functional dependence on a variable x_i coincide, To prove this, suppose that Ω is an arbitrary basis, and F is a minimal Ω-formula for $f \in B_n$. If H is a gate in F which does not depend functionally on the variable x_i, then we claim that there are no occurrences of the variable x_i in F_H. If there were, then one could replace all occurrences of x_i in F_H by the constant 0 (or 1) without changing the function computed at H and hence by the entire formula F. Since this reduces the number of occurrences of inputs in F, F_H must not contain any occurrences of x_i.

Finally, we point out one simple lower bound on the complexity of most functions.

2.11 Lemma (Savage): Suppose a function $f \in B_n$ depends on each of its variables. Then $C_{B_2}(f) \geq n-1$.

Proof: Suppose N is a minimal B_2-circuit for f which has k gates. Observe that since each input node has outdegree at least one and since there is at most one gate of outdegree 0, we have

$$\sum_{A \in Nodes(N)} \text{indegree } (A,N) = 2k$$

and

$$\sum_{A \in Nodes(N)} \text{outdegree}(A,N) \geq n+k-1$$

Hence, since

$$\sum_{A \in Nodes(N)} \text{indegree}(A,N) = \sum_{A \in Nodes(N)} \text{outdegree}(A,N) ,$$

we obtain $2k \geq n+k-1$ which establishes the lemma.

□ Lemma 2.11

Section D. Circuits and Turing Machines

One question which naturally arises concerns the relationship between the measures of complexity defined above and ordinary Turing Machine (T.M.) complexity. As mentioned above, every finite function is "easy" for a Turing Machine to compute since it may be computed by finite state machine. When one considers the finite subproblems of an infinite problem, however, the situation is

different. For concreteness, consider language recognition problems, i.e. functions $f:\{0,1\}^* \to \{0,1\}$ (where $\{0,1\}^*$ is the set of all finite-length words over the alphabet $\{0,1\}$), and let f_n be the restriction of f to words of length n. Consider the circuit complexity measure C_{B_2}.

One can show that there are functions f which are computable by a deterministic T.M. in space 2^n (i.e. a relatively easy function) for which $C_{B_2}(f_n)$ is near-maximal for all values of n (see [Stockmeyer 1974] for proof). On the other hand, one can exhibit arbitrarily complex, even arbitrarily non-recursive, functions f such that $C_{B_2}(f_n)$ is 1 for every $n \in \mathbb{N}$. There thus appears to be little correlation between the T.M. complexity of a function and the circuit complexity of its finite pieces.

A strong relationship between the two measures comes from a result due to Pippenger and Fischer [1977] which asserts the following: Suppose f is computable by a T.M. M operating in time $T(n)$ for all n. Then there is a constant c (depending on M's state diagram) such that $C(f_n) \leq cT(n)\log T(n)$. The proof is obtained by simulating machine M by an "oblivious" T.M. M' which operates in time $T \cdot \log T$ - the positions of the heads of M' depend only on the input length n. Then it is shown that any oblivious T.M. can have its operation on an input of length n simulated by a number of gates proportional to its running time.

This remark has many applications [Paul 1976]. As an example, Fischer and Meyer [1971] have shown that Strassen's fast integer matrix multiplication algorithm can be used to multiply two Boolean $n \times n$ matrices in time $O(n^{\log_2 7} \log n \log\log n \log\log\log n)$ on a Turing Machine. Using the oblivious construction of a circuit simulating the T.M., one thus obtains a B_2-circuit for matrix multiplication with $O(n^{2.82})$ gates. (Recall that Paterson [1975] and Mehlhorn and Galil [1976] have shown that any M-circuit for this problem must have $\Omega(n^3)$ gates.)

While Fischer's result is nice as a technical tool, it is obvious by the earlier remarks that the converse result does not hold. Schnorr [1976d] has strengthened the result by considering oracle T.M.'s in which one work tape initially holds some string A in $\{0,1\}^*$. Define, for $f:\{0,1\}^* \to \{0,1\}$,

$TC(f)(n) = \min\{|M| \cdot T^A(n) \cdot \log S^A(n) \mid M \text{ with oracle } A \text{ computes } f\}$

where M ranges over all oracle T.M.'s (see [Schnorr 1976d] for conventions), T^A and S^A are the time and space required by M, and $|M|$ is the number of instructions in program M. Then there exists some fixed polynomial p and a constant c such that

$$C_{B_2}(f) \leq c \cdot TC(f) \text{ and } TC(f) \leq p(C_{B_2}(f)).$$

Section E. Miscellaneous Notations

Frequently we will be concerned with the rate of growth of a numeric function rather than its exact value. Suppose f and g

are functions defined on the natural numbers. Then we write

$f = O(g)$ iff there is a constant $c > 0$ such that $\limsup_{n\to\infty} (f(n)/g(n)) \le c$
(i.e. if $f(n) \le c\, g(n)$ for all sufficiently large n),

$f = o(g)$ iff $\lim_{n\to\infty} f(n)/g(n) = 0$,

$f = \Omega(g)$ iff there is a constant $c > 0$ such that $\limsup_{n\to\infty} (f(n)/g(n)) \ge c$
(i.e. if $f(n) \ge c\, g(n)$ for all sufficiently large n),

$f = \Theta(g)$ iff $f = O(g)$ and $f = \Omega(g)$,

$f \sim g$ iff $\lim_{n\to\infty} f(n)/g(n) = 1$,

and

$f \lesssim g$ iff $\limsup_{n\to\infty} f(n)/g(n) \le 1$.

If $f = \Theta(g)$ we say that f is <u>proportional</u> to g, if $f \sim g$ we say that f is <u>asymptotically equal</u> to g, and if $f \lesssim g$ we say that f is <u>asymptotically less than</u> g.

If A is any finite set, then $|A|$, <u>the cardinality</u> of A, is the number of elements in A. A <u>partition</u> of A is a collection of pairwise disjoint sets $B_1, B_2, \ldots, B_k$ such that $\bigcup_{i=1}^{k} B_k = A$.
If A and B are sets, then $A-B = \{x \mid x\in A \text{ and } x\notin B\}$.

If X is a set of variables, an <u>X-variable</u> is a member of X.

$\mathbb{N}$ is the set of natural numbers, and R is the set of real numbers; $R^* = \{r\in R \mid r \ge 0\}\cup\{\infty\}$.

If $k \le \ell$ are natural numbers, the set of consecutive natural numbers $\{k, k+1, k+2, \ldots, \ell-1, \ell\}$ is denoted $[k:\ell]$.

If x is a real number, then $\lfloor x \rfloor$ is the greatest integer less than or equal to x; $\lceil x \rceil$ is the least integer greater than or equal to x.

"Iff" is an abbreviation for "if and only if", and "w.l.o.g." is an abbreviation for "without loss of generality".

ϕ is the empty set.

If k and ℓ are natural numbers, then $\binom{k}{\ell} = \frac{k!}{\ell!(n-\ell)!}$.

CHAPTER 3

Worst-Case Values for the Complexity of Quadratic Functions

We begin by considering several complexity measures for monotone Boolean functions, and examine the complexity of quadratic m.b.f.'s under these measures. Recall that a monotone function $f \in B_{n,m}$ is quadratic iff for each $i \in [1:m]$, component f_i of f has a set of prime implicants $PI(f_i)$ equal to $\{x_j \cdot x_k \mid \{j,k\} \in A_i\}$, where A_i is some collection of subsets of $[1:n]$ of cardinality two; equivalently, $f_i(x_1,\ldots,x_n) = \bigvee_{\{j,k\} \in A_i} (x_j \cdot x_k)$. The "threshold two" function of n variables T_2^n is quadratic since $PI(T_2^n) = \{x_j \cdot x_k \mid j,k \in [1:n], j \neq k\}$. Boolean matrix multiplication and Boolean convolution are additional examples.

In this chapter we primarily consider the measures MC, $MC_\wedge$, and $MC_\vee$ for circuits, and corresponding measures ML, $ML_\wedge$, and $ML_\vee$ for formulas. For any M-circuit N, we have the relationship $MC_\wedge(N) + MC_\vee(N) = MC(N)$. For an M-formula F which does not contain any constant inputs, we have the corresponding equation $ML_\wedge(F) + ML_\vee(F) = ML(F) - 1$*. Hence for an arbitrary monotone function $f \in B_n$ we have the relationships

---

*The number of gates in an M-formula is one less than the number of occurrences of variable and constant inputs.

$$MC_{\wedge}(f) + MC_{\vee}(f) \leq MC(f)$$

and

$$ML_{\wedge}(f) + ML_{\vee}(f) \leq ML(f) - 1,$$

but at least the first inequality may be strict for specific functions f. For example, in Chapter 5 we will show that $MC_{\wedge}(T_2^n) = \lceil \log n \rceil$ and $MC_{\vee}(T_2^n) = 2n-4$, but will also show that if n is a power of 2, any M-circuit for T_2^n which has exactly $\lceil \log n \rceil$ $\wedge$-gates must also have at least $2n + 3\lceil \log n \rceil - 9$ $\vee$-gates, and hence any M-circuit for T_2^n must have at least one more gate than the sum $MC_{\wedge}(T_2^n) + MC_{\vee}(T_2^n)$ for these values of n.

Two areas in the complexity of single-output quadratic m.b.f.'s are presented in this chapter. The asymptotic value of the largest complexity for all quadratic m.b.f.'s is explored first. It is shown that every quadratic m.b.f. in B_n has an M-circuit complexity of $O(n^2/\log n)$. That this upper bound is within a constant factor of the best possible result is shown by proving that "most" quadratic m.b.f.'s have circuit complexity over the larger basis B_2 of $\Omega(n^2/\log n)$.

The other area is the determination of the number of $\wedge$-gates necessary and sufficient to compute any quadratic m.b.f. in B_n. We show that $n-1$ $\wedge$-gates are sufficient in any M-formula, and exhibit a quadratic m.b.f. for which $\lfloor 2/3n \rfloor$ $\wedge$-gates are necessary in any M-circuit. By restricting the types of M-circuits allowed, we can demonstrate a quadratic function for which $n - o(n)$ $\wedge$-gates

are necessary in any restricted M-circuit, due to an observation by V.Chvatal.

Section A. The Total Number of Gates

We first demonstrate closely matching upper and lower bounds for the worst-case complexity of quadratic monotone Boolean functions.

Definition: Suppose $n \in \mathbb{N}$. Define

$$QC(n) = \max\{MC(f) \mid f \text{ is a quadratic m.b.f. in } B_n\},$$

$$QC_\wedge(n) = \max\{MC_\wedge(f) \mid f \text{ is a quadratic m.b.f. in } B_n\},$$

and $$QC_\vee(n) = \max\{MC_\vee(f) \mid f \text{ is a quadratic m.b.f. in } B_n\}.$$

We similarly define $QL(n)$, $QL_\wedge(n)$, and $QL_\vee(n)$.

We can use standard counting techniques as found in numerous proofs [Shannon 1949, Lupunov 1962, Fischer 1974] to establish lower bounds on QC.

3.1 Theorem: Suppose $\varepsilon > 0$. Then most quadratic m.b.f.'s f in B_n have

$$C_{B_2}(f) \geq (1 - \varepsilon)n^2/(4 \log n)$$

for n sufficiently large, where B_2 is the complete basis of all two-input Boolean functions.

Proof: Let $Q_n \subseteq B_n$ denote the set of all quadratic m.b.f.'s of n variables. Since there are $\binom{n}{2}$ subsets of the variables $\{x_1, x_2, \ldots x_n\}$ with two members, $|Q_n| = 2^{\binom{n}{2}} - 1$.

Suppose $f \in Q_n$. We know that there is some B_2-circuit N which

computes f since B_2 is a complete basis. If N contains two gates which compute the same function, then as in Chapter 2, one of these may be eliminated to yield a B_2-circuit with fewer gates which also computes f. Also, if f is not a constant function, it is possible to construct a B_2-circuit for f in which each constant node has outdegree zero. We thus consider only B_2-circuits in which each gate computes a distinct function, and in which constant nodes have outdegree zero; we call such circuits reduced.

Now suppose $q \in \mathbb{N}$ and N is a reduced B_2-circuit of size $\leq q$ which computes a function $f \in Q_n$. By possibly adding additional "dummy" gates, we may assume that there is a reduced B_2-circuit of size exactly q in which one of the gates computes f. Let R_q denote the number of quadratic m.b.f.'s computed by such reduced circuits of size q. Since each gate has two inputs, and can be labelled with any of the 16 binary Boolean functions, there are at most $16(q + n)^2$ possible assignments of function-input pair combinations for each gate in a reduced B_2-circuit, giving a total of at most $[16(q + n)^2]^q$ such assignments of reduced B_2-circuits. However, many of these labellings describe the same circuit -- in fact, there are q! different enumerations of the computational nodes of any fixed reduced circuit. Hence, there are at most $16^q(q + n)^{2q}/q!$ reduced circuits of size q. Since the (quadratic) output function may be computed at any gate of the circuit, we have

$$R_q \le q \cdot 16^q (q+n)^{2q}/q!$$

$$\le (16^q (q+n)^{2q+1} \cdot e^q)/(\sqrt{2\pi q} \cdot q^q) \quad \text{(using Stirling's approximation)}$$

$$\le \sqrt{(2/\pi)}(64e)^q \cdot q^{q+(1/2)} \quad \text{(when } q \ge n \text{)}$$

$$\le (cq)^q . \quad \text{(for some } c \ge 0 \text{ and } q \text{ sufficiently large)}$$

Now, suppose that $q \le (1-\varepsilon)\binom{n}{2} / \log \binom{n}{2}$. Then

$$R_q \le [c(1-\varepsilon)\binom{n}{2} / \log \binom{n}{2}]^{(1-\varepsilon)\binom{n}{2} / \log \binom{n}{2}}$$

$$\le \binom{n}{2}^{(1-\varepsilon)\binom{n}{2} / \log \binom{n}{2}} \quad \text{(when } \log \binom{n}{2} \ge c(1-\varepsilon) \text{)}$$

$$\le 2^{(1-\varepsilon)\binom{n}{2}} .$$

Hence the fraction of all quadratic m.b.f.'s counted in R_q is

$$R_q / 2^{\binom{n}{2}} \le 2^{(1-\varepsilon)\binom{n}{2}} / 2^{\binom{n}{2}} = 2^{-\varepsilon\binom{n}{2}}$$

which tends to 0 as $n \to \infty$. Thus most quadratic functions have B_2 complexity at least $(1-\varepsilon)\binom{n}{2} / \log \binom{n}{2} \sim (1-\varepsilon)(n^2)/(4 \log n)$.

□ Theorem 3.1

3.2 Corollary: $QC(n) = \Omega(n^2/\log n)$.

Proof: For any quadratic m.b.f. f, $MC(f) \ge C_{B_2}(f)$ since

$M \subseteq B_2$. □ Corollary 3.2

We can prove that all quadratic m.b.f.'s can be computed using a monotone circuit which is no more than a small constant factor larger than this bound. The idea of the proof is similar to that used in the Four Russian's algorithm for transitive closure [Arlazarov et al. 1970], and has been used previously for constructing efficient circuits for other classes of functions [Tarjan 1976].

3.3 Theorem: Suppose $f \in B_n$ is a monotone quadratic Boolean function. Then

$$MC(f) \leq 4\, n^2/\log n + O(n).$$

Proof: Let $m = \lfloor \log n \rfloor$. We partition the input variables of f into $\lceil n/m \rceil$ subsets, the i^{th} subset S_i consisting of the set $S_i = \{x_j \mid (i-1)m < j \leq i \cdot m,\ j \leq n\}$ for $i \in [1:n/m]$. Each subset, with the possible exception of the last, has m variables in it. If $f \in Q_n$ is given, for each input variable x_i and each subset S_j we define

$$A_{ij} = \{x_k \in S_j \mid k \geq i \text{ and } x_i x_k \in PI(f)\}.$$

Note that $A_{ij} = \emptyset$ for $j < \lceil i/m \rceil$. Let $\hat{A}_{ij}$ be the Boolean function of the disjunction of all variables in A_{ij} (recall that by convention $\hat{A}_{ij}$ is the constant function $\underline{0}$ when $A_{ij} = \emptyset$). We clearly

can express f as the factorization

$$f(x_1,\ldots,x_n) = \bigvee_{i=1}^{n} (x_i \wedge (\bigvee_{j=\lceil i/m \rceil}^{\lceil n/m \rceil} \hat{A}_{ij})) \qquad (1)$$

Our goal will be to compute the functions $\hat{A}_{ij}$ efficiently, and use the decomposition (1) to compute f. We make use of the following result [Tarjan 1976] which allows efficient computation of sets of disjunctions.

3.4 Lemma: Suppose X is a set of m variables, and $f_1,f_2,\ldots,f_k$ are monotone Boolean functions which are the disjunction of subsets of variables of X. Then there exists a circuit N consisting solely of v-gates which computes $F = \{f_1,f_2,\ldots,f_k\}$ such that $C_{\{v\}}(N) \leq 4km/\log k$.

Proof of lemma: Let $r = \lfloor \log k \rfloor$. We decompose the set of X variables into $\lceil m/r \rceil$ subsets $T_1,T_2,\ldots,T_{\lceil m/r \rceil}$ each containing no more than r variables. For each i, the circuit N initially computes the disjunction of all possible subsets of T_i. This can be done in a manner which has one disjunction for each subset of size ≥ 2 by computing the disjunction of smaller subsets first, and then constructing the disjunction of a larger set with one additional variable by using a single v-gate to add this variable to the previously constructed disjunction of the smaller set. Hence, this can be performed using at most $2^r - r - 1$ disjunctions for each $i \in [1:\lceil m/r \rceil]$, giving a total of

$(2^r - r - 1)(\lceil m/r \rceil) < k\lceil m/r \rceil$ disjunctions. Since each function f_j can be constructed by computing the disjunction of $\lceil m/r \rceil$ of these previously computed functions (one per subset T_i), we can compute F with at most an additional $k(\lceil m/r \rceil - 1)$ v-gates. Since $\lceil m/r \rceil \leq 2m/\log k$, the total number of v-gates used in this construction is at most $2k\lceil m/r \rceil - k \leq 4km/\log k$.

□ Lemma 3.4

Returning to the proof of Theorem 3.3, suppose that j is fixed. Note that $A_{ij} \neq \emptyset$ for at most $j \cdot m$ values of i, namely for each $i \in [1:j \cdot m]$. Hence the computation of the functions A_{ij} for all $j \cdot m$ values of i can be accomplished using the method of Lemma 3.4 with at most $4jm^2/\log(jm)$ v-gates. Thus, one can compute all functions A_{ij} for $i \in [1:n]$, $j \in [1:\lceil n/m \rceil]$ using at most

$$\sum_{j=1}^{\lceil n/m \rceil} (4jm^2)/(\log(jm)) = 4m^2 \left(\sum_{j=1}^{\lceil n/m \rceil} j/\log(jm) \right) \quad *$$

$$= 4m^2[n^2/(2m^2 \log n) + O(n/\log^2 n)]$$

$$= 2n^2/\log n + O(n) \qquad \text{v-gates.}$$

---

*To sum the series, note that

$$\sum_{j=1}^{\lceil n/m \rceil} j/\log(jm) \leq \int_2^{n/m} (x/\log(xm))\,dx + 2n/(m \log n)$$

$$\leq (1/m^2) \int_2^n x/\log x \; dx + 2n/(m \log n)$$

(cont. on next page)

Finally, combining these disjunctions to compute f using (1) can be done using an additional $n-1$ $\wedge$-gates (since $\hat{A}_{nj} = \underline{0}$ for all j); the number of additional $\vee$-gates is at most

$$\sum_{i=1}^{n} (\lceil n/m \rceil - \lceil i/m \rceil) + n - 1 \leq \sum_{i=1}^{n} ((n-i)/m + 1) + n - 1$$

$$= (\sum_{j=0}^{n-1} j)/m + 2n - 1$$

$$= n^2/2m + O(n).$$

Again, since $n^2/m \leq 2n^2/\log n$, we have a total of $4n^2/\log n + O(n)$ gates in the M-circuit constructed.

□ Theorem 3.3

Section B. Asymptotic Bounds on $\wedge$-gates

In the remainder of this chapter, we consider the measures $QC_\wedge$ and $QL_\wedge$ which are the minimal number of $\wedge$-gates required in any

---

But

$$\int_2^n x/\log x \, dx \leq \sum_{i=2}^{n} i/\log i \leq \sum_{i=2}^{\lfloor n/2 \rfloor + 1} [i/\log i + (n-i+2)/\log(n-i+2)]$$

$$\leq \sum_{i=2}^{\lfloor n/2 \rfloor + 1} 2(\lceil n/2 \rceil + 1)/\log(n/2 + 1)$$

$$= 2(\lfloor n/2 \rfloor (\lceil n/2 \rceil + 1))/(2 \log (\lceil n/2 \rceil + 1))$$

$$= n^2/2 \log n + O(n/\log n).$$

circuit or formula respectively to express a quadratic m.b.f. of n variables. We do not consider the measures $QC_\vee$ and $QL_\vee$, and consider determination of their values to be an interesting open question. When arbitrary M-circuits are allowed, the bounds we obtain on $\wedge$-complexity are not as tight as those in the previous section, but tighter bounds are given in Section C when $\wedge$-gates are not allowed to have a path to another $\wedge$-gate in the circuit.

3.5 Lemma: Suppose $n \geq 2$. Then $QC_\wedge(n) \leq QL_\wedge(n) \leq n-1$.

Proof: Obviously $QC_\wedge(n) \leq QL_\wedge(n)$ since a formula may be considered a special kind of a circuit.

We prove that $QL_\wedge(n) \leq n-1$ by induction on n.

If $n = 2$, then clearly $QL_\wedge(2) = 1$ since the only quadratic m.b.f. of 2 variables is $T_2^2(x_1,x_2) = x_1 \wedge x_2$.

Now assume inductively that $QL_\wedge(n) \leq n-1$, and suppose that f is a quadratic m.b.f. of $n+1$ variables. We can express f as the factorization

$$f(x_1,\ldots,x_{n+1}) = (x_{n+1} \wedge g(x_1,\ldots,x_n)) \vee h(x_1,\ldots,x_n)$$

where $g = \bigvee\{x_i \mid x_i \cdot x_{n+1} \in PI(f)\}$ and

$$h = \bigvee\{x_i \cdot x_j \mid x_i \cdot x_j \in PI(f),\ i \neq n+1,\ j \neq n+1\}$$

Clearly $ML_\wedge(g) = 0$ since it is simply a disjunction of single variables. Also $ML_\wedge(h) \leq QL(n)$ since h is a quadratic m.b.f. of n variables. Inductively, we then know that $ML_\wedge(h) \leq n-1$, so $ML_\wedge(f) \leq ML_\wedge(g) + ML_\wedge(h) + 1 \leq n$.

□ Lemma 3.5

Remark: One can show that $QL_\wedge(4) = 2$ by examining all quadratic m.b.f.'s of 4 variables, and hence a similar proof demonstrates that $QL_\wedge(n) \leq n-2$ for $n \geq 4$.

We would now like to determine whether there are quadratic m.b.f.'s which require a number of $\wedge$-gates which is close to the upper bound provided in Lemma 3.5.† It is easy to demonstrate a quadratic m.b.f. in B_n for which $\Omega(n)$ $\wedge$-gates are required. For example, suppose n is even and consider the function $f_0(x_1,\ldots,x_n) = (x_1 \wedge x_2) \vee (x_3 \wedge x_4) \vee \ldots \vee (x_{n-1} \wedge x_n)$; the proof that

---

† We remark that the number of prime implicants of a quadratic m.b.f. correlates poorly with its $\wedge$-complexity since (as we show in Chapter 5) the function T_2^n, which has $\binom{n}{2}$ prime implicants, only requires $\lceil \log n \rceil$ $\wedge$-gates to compute.

f_0 requires $n/2$ $\wedge$-gates to compute is relatively straightforward. It is an open question whether $QC_\wedge(n)$ is asymptotic with n. Our objective is to show that the constant factor of the trivial lower bound of $n/2$ may be improved to $2/3$.

3.6. Theorem: Suppose $n \geq 2$. There is a quadratic m.b.f. $f \in B_n$ such that $MC_\wedge(f) = \lfloor 2n/3 \rfloor$.

Proof: We define f as follows: partition the n variables into $k = \lceil n/3 \rceil$ subsets $S_1, S_2, \ldots, S_k$, each containing 3 variables (except possibly S_k). We consider the function

$$f(x_1, \ldots, x_n) = T_2^3(S_1) \vee T_2^3(S_2) \vee \ldots \vee T_2^3(S_k) \qquad (2)$$

where $T_2^3(S_i)$ is the threshold 2 function of the variables in S_i (for the set S_k, if n is not evenly divisible by 3, we consider there to be 1 or 2 "dummy" variables set to the constant 0). We claim that $MC_\wedge(f) = \lfloor 2n/3 \rfloor$, and prove this result by induction on k in the event $n \equiv 0 \pmod 3$. The other cases $n \equiv 1 \pmod 3$ and $n \equiv 2 \pmod 3$ are proven similarly. We first show that $MC_\wedge(f) \geq 2n/3$.

Suppose $k = 1$ and $n = 3$. Then $f(x_1, x_2, x_3) = T_2^3(x_1, x_2, x_3)$. By Theorem 5.9 in which it is shown using other means that $MC_\wedge(T_2^n) = \lceil \log n \rceil$, we have $MC_\wedge(f) = \lfloor \log_2 3 \rfloor = 2$.

Now assume the statement true for k and $n = 3k$ -- namely that the function

$$f(x_1,\ldots,x_n) = T_2^3(S_1) \vee \ldots \vee T_2^3(S_k)$$

has $MC_\wedge(f) \geq 2k$. We are concerned with the function

$$f'(x_1,\ldots,x_n,y_1,y_2,y_3) = f(x_1,\ldots,x_n) \vee T_2^3(y_1,y_2,y_3)$$

where we have re-named the variables for ease of description.

Suppose N is an M-circuit computing f'. The general aim of the proof will be to show that there are constants a_1,a_2 and a_3 in $\{0,1\}$ with the following property: if y_i is set to a_i (for $i = 1,2$, and 3), then the resulting circuit N' computes f; moreover, it is possible to eliminate two unnecessary $\wedge$-gates from N' since each gate has at least one predecessor which computes the constant function $\underline{1}$. We proceed in a fashion similar to Paterson [1976].

Definition: Suppose N is a circuit which computes f', and suppose $i,j\epsilon\{1,2,3\}$. We say a node G of N has property $P_{i,j}$ iff $y_i\, y_j$ is a prime implicant of $Res(G,N)$.

Definition: If P is any property on nodes of N, we define the initial occurrences of P, denoted $I(P)$, to be

$$I(P) = \{G\epsilon Nodes(N) \mid G \text{ satisfies } P, \text{ but no predecessor of } G \text{ satisfies } P\}.$$

3.7 Lemma: Suppose N is an M-circuit which computes f', and $i,j \in \{1,2,3\}$, $i \neq j$. Then

(1) $I(P_{i,j}) \neq \emptyset$

and (2) $I(P_{i,j})$ consists solely of $\wedge$-gates.

Proof: Since the input variables do not satisfy $P_{i,j}$ and the output gate of N does, by passing up the circuit N from the output gate we must eventually find a gate which satisfies $P_{i,j}$ but none of whose predecessors do. This establishes statement (1). For statement (2), suppose that G is an $\vee$-gate which satisfies $P_{i,j}$, and let Pred(G,N) = {J,K}. By the remarks in Chapter 2 on prime implicants, we know that $PI(G,N) \subseteq PI(J,N) \cup PI(K,N)$, and since $y_i y_j \in PI(G,N)$, one of J or K must satisfy $P_{i,j}$. Hence $G \notin I(P_{i,j})$.

□ Lemma 3.7

3.8 Lemma: Suppose N,i, and j are as above, and $G \in I(P_{i,j})$. If Pred(G) = {J,K}, then either $y_i \in PI(J,N)$ and $y_j \in PI(K,N)$, or vice versa.

Proof: By Lemma 3.7, if $G \in I(P_{i,j})$, then G is an $\wedge$-gate. Every prime implicant of G is a product of a member of PI(J) and

a member of PI(K). Since neither PI(J) nor PI(K) contains $y_i \cdot y_j$, we must have either $y_i \in PI(J)$ and $Y_j \in PI(K)$, or vice versa.

□ Lemma 3.8.

3.9 Lemma: If N is as above, then $I(P_{1,2}) \cup I(P_{2,3}) \cup I(P_{1,3}) \geq 2$.

Proof: Suppose to the contrary that there is only a single gate G in $I(P_{1,2}) \cup I(P_{2,3}) \cup I(P_{1,3})$, and suppose $Pred(G,N) = \{J,K\}$. By Lemma 3.8, we may suppose w.l.o.g. that $y_1 \in PI(J,N)$ and $y_2 \in PI(K,N)$ since $G \in I(P_{1,2})$. Again by virtue of Lemma 3.8 and the fact that $G \in I(P_{2,3})$, we must have $y_3 \in PI(J,N)$ since $y_2 \in PI(K,N)$ implies that $y_2 \in PI(G,N)$, and hence that $y_1 y_2 \notin PI(G,N)$, contrary to the assumption that $G \in I(P_{1,2})$. But then $G \notin I(P_{1,3})$ since $y_1 y_3$ cannot be a prime implicant of $Res(G,N) = (y_1 \vee y_3 \vee h_1) \wedge (y_1 \vee h_2)$. This contradiction implies that the union contains at least 2 gates.

□ Lemma 3.9.

We are now in a position to prove the main theorem. Since there are at least two distinct $\wedge$-gates in $I(P_{1,2}) \cup I(P_{2,3}) \cup I(P_{1,3})$ and since none of these sets is empty by Lemma 3.7, we may w.l.o.g. assume that there exist distinct gates G and H in N such that $G \in I(P_{1,2})$ and $H \in I(P_{2,3})$. Assume using Lemma 3.8 that

$Pred(G,N) = \{K_1, K_2\}$ and $y_2 \in PI(K_1, N)$; and similarly that $Pred(H,N) = \{K_3, K_4\}$ and $y_2 \in PI(K_3, N)$. (It may be possible that some of the K_i nodes are not distinct.) The assignment of the variables $y_1 = 0$, $y_2 = 1$, and $y_3 = 0$ results in a circuit N' which computes the function f. Moreover, in N' the inputs K_1 and K_3 to G and H respectively now compute the constant function $\underline{1}$, and hence gates G and H may be eliminated in N' since $\underline{1} \wedge g = g$ for any Boolean function g. Hence if the resulting circuit is called N'', we have

$$MC_\wedge(N) \geq MC_\wedge(N'') + 2 \geq MC_\wedge(f) + 2 = 2n + 2$$

by induction, and the lower bound is complete.

To show that $C_\wedge(f) \leq \lfloor (2/3)n \rfloor$, it suffices to observe that

$$T_2^3(x_1, x_2, x_3) = ((x_1 \vee x_2) \wedge x_3) \vee (x_1 \wedge x_2)$$

and hence $C_\wedge(T_2^3) \leq 2$. Thus using the definition (2) one can compute f using $\lfloor (2/3)n \rfloor$ $\wedge$-gates.

□ Theorem 3.6

3.10 Corollary: Suppose $n \geq 4$. Then

$$\lfloor (2/3)n \rfloor \leq QC_\wedge(n) \leq QL_\wedge(n) \leq n-2.$$

Section C. A Graph Problem

We close this chapter by describing a graph-theoretic problem which is related to the $\wedge$-complexity of quadratic m.b.f.'s when they are computed with a restricted class of M-circuits. The research in the remainder of this chapter was done in conjunction with Ronald Rivest. We introduce notation similar to Harary [1969] for undirected graphs.

Definition: An undirected graph G is a pair (V,E), where V is a finite set whose members are called nodes or vertices, and $E \subseteq \{B \subseteq V \mid |B| = 2\}$ is a set whose members are called edges.

A bipartite graph is a graph $G = (V,E)$ whose vertices can be partitioned into two non-empty sets V_1 and V_2 such that if edge $\{v_1,v_2\} \in E$, then $v_1 \in V_1 \leftrightarrow v_2 \in V_2$; that is, each edge consists of one point in V_1 and one point in V_2.

A complete k-ℓ bipartite graph is a bipartite graph $(V = V_1 \cup V_2, E)$ such that $|V_1| = k$ $|V_2| = \ell$, and $E = \{\{v_1,v_2\} \mid v_1 \in V_1,\ v_2 \in V_2\}$.

Definition: If $G = \{(V_1,E_1),(V_2,E_2),\ldots,(V_k,E_k)\}$ is a collection of undirected graphs, and $G = (V,E)$ is an undirected graph, then an exact covering C of G by G is a collection of one-to-one mappings $h_i: V_i \to V$ for $i \in [1:k]$ such that $E = \bigcup_{i \in [1:k]} \{h(e) \mid e \in E_i\}$

where $h(\{a,b\}) = \{h(a),h(b)\}$ for any edge $\{a,b\}\in E_i$. The cost of the cover, denoted $\hat{D}(C)$, is k.

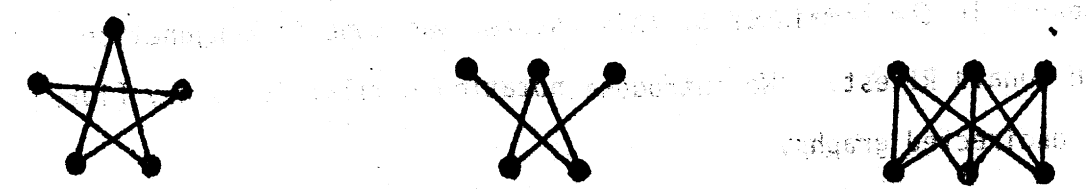

Fig. 3.1 Some Graphs
Points represent nodes and lines represent edges.

We can now define a measure on graphs.

Definition: Suppose $n\in\mathbb{N}$, and G is an n-node undirected graph. Then

$$D(G) = \min\{\hat{D}(C) \mid C \text{ is an exact cover of } G \text{ by } \{G_1,\ldots,G_k\}, \text{ where } G_i \text{ is a complete bipartite graph for } i\in[1:k]\}.$$

Define $H(n) = \max\{D(G) \mid G \text{ is an } n\text{-node undirected graph}\}$.

The connection between quadratic m.b.f.'s and graphs is as follows: Suppose f is an n-variable Boolean function. We can associate a corresponding n-node undirected graph $G_f = (V,E)$, where $V = \{1,2,\ldots,n\}$ and $E = \{\{i,j\} \mid x_i x_j \in PI(f)\}$. Conversely, suppose $G = (V,E)$ is an n-node undirected graph. By possibly renaming the vertices, we may assume $V = [1:n]$. Define the Boolean function $f_G:\{0,1\}^n \to \{0,1\}$ by

$$f_G(x_1,\ldots,x_n) = \vee\{x_i x_j \mid \{i,j\} \in E\} .$$

The functions $f \to G_f$ and $G \to f_G$ clearly demonstrate a one-to-one correspondance between the set of quadratic m.b.f.'s in B_n and the set of n-node undirected graphs with at least one edge. See Figure 3.2 for some examples.

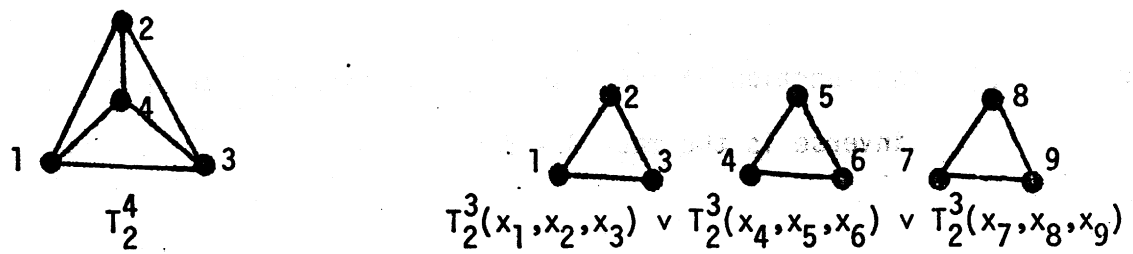

Fig. 3.2 Some Functions and their Graphs

In fact, we can get a more strict correspondence in mathematical terms.

Notation: $\hat{Q}_n = \{f \mid f$ is a quadratic m.b.f. of n variables$\} \cup$ $\{\underline{0}$, the constant function 0 of n variables$\}$.

$G_n = \{G = (V,E) \mid G$ is an undirected graph, and $V = [1{:}n]\}$.

Z_n is the graph $([1{:}n],\emptyset)$ in G_n.

If $G_1 = (V,E_1)$ and $G_2 = (V,E_2) \in G_n$, define $G_1 \cup G_2$ to be the graph $(V,E_1 \cup E_2)$.

If A is any set, and $f{:}A \to B$, then $f(A) = \{f(a) \mid a \in A\}$.

If $G = (V,E)$ is an undirected graph, and $h{:}V \to W$, then $h(G)$ is the graph $(h(V),h(E))$.

3.11 Lemma: Suppose $n \geq 0$. Then

(1) $(\hat{Q}_n, \vee, \underline{0})$ and $(G_n, \cup, Z_n)$ are monoids,

and (2) the function $f \to G_f$ is a monoid isomorphism whose inverse is the map $G \to f_G$.

Proof: Statement (1) is easily checked, as are the facts that the maps in (2) are one-to-one, onto, and inverses of each other. Moreover, if f_1 and $f_2 \in \hat{Q}_n$, then

$$PI(f_1 \vee f_2) = PI(f_1) \cup PI(f_2)$$

since each prime implicant is the product of two variables. Thus, if

$G_{f_1 \vee f_2} = ([1:n],E)$, $G_{f_1} = ([1:n],E_1)$, and $G_{f_2} = ([1:n],E_2)$, it is easy to check that $E = E_1 \cup E_2$, and the claim is verified.

□ Lemma 3.11

3.12 Corollary: Suppose $C = (h_1,\ldots,h_k)$ is a covering of the graph G by graphs $G_1,G_2,\ldots,G_k$. Then

$$f_G = f_{h_1(G_1)} \vee f_{h_2(G_2)} \vee \cdots \vee f_{h_k(G_k)}.$$

Proof: Proven inductively using Lemma 3.11.

□ Corollary 3.12

We now connect the complexity of functions and their graphs. We will say an M-circuit or M-formula N is single-level if there is no path between any pair of $\wedge$-gates in N. Obviously, any quadratic m.b.f. can be computed by a single-level M-formula and hence a single-level M-circuit. In a single-level M-circuit N, each immediate predecessor of every $\wedge$-gate consists of the disjunction of some set of input and constant nodes. Moreover, if N computes a quadratic m.b.f. f, then there is some subset α of

the $\wedge$-gates of N such that $f = \bigvee_{A \in \mathcal{A}} Res(A,N)$. Thus, given a single-level M-circuit N computing a quadratic m.b.f. f, one may construct a single-level M-formula F which also computes f and which has the same number of $\wedge$-gates as N since computing any number of disjunctions of variables takes no $\wedge$-gates.

If f is a quadratic m.b.f., define

$SLC_{\wedge}(f) = \min\{MC_{\wedge}(N) \mid N$ is a single-level M-circuit which computes $f\}$.

We shall prove

3.13 Theorem: Suppose f is a quadratic m.b.f.. Then $SLC_{\wedge}(f) = D(G_f)$.

Proof: Suppose f is a quadratic m.b.f., and that $D(G_f) = k$. Let $C = (h_1,\ldots,h_k)$ be an exact cover of G_f by complete bipartite graphs $G_1,\ldots,G_k$. We will construct a single-level M-formula for f which has one $\wedge$-gate for each graph in the cover of G_f. Suppose $G_i = (V_i = V_{i1} \cup V_{i2}, E_i)$ is the decomposition of the vertices of G_i as in the definition of bipartite graphs.

Note that $h_i(G_i)$ is also a complete bipartite graph. Since each graph is complete,

$$f_{h_i(G_i)}(\vec{x}) = \vee\{x_j x_k \mid \{j,k\}\in h_i(E_i)\}$$

$$= (\vee\{x_j \mid j\in h_i(V_{i1})\}) \wedge (\vee\{x_k \mid k\in h_i(V_{i2})\}).$$

Hence, using Corollary 3.12, we can express f as

$$f = f_{G_f} = f_{h_1(G_1)} \vee f_{h_2(G_2)} \vee \cdots \vee f_{h_k(G_k)}$$

$$= \bigvee_{i=1}^{k} [(\vee\{x_j \mid j\in h_i(V_{i1})\}) \wedge (\vee\{x_k \mid k\in h_i(V_{i2})\})]$$

which is a formula with k $\wedge$-gates. Thus $SLC_{\wedge}(f) \leq k$.

Conversely, suppose f is a quadratic m.b.f., and $SLC(f) = k$. By the remarks preceding the theorem, there is a single-level M-formula F for f such that $MC_{\wedge}(F) = k$. Since F is single-level, we may write

$$F = \bigvee_{i=1}^{k} [(\bigvee_{j\in A_i} x_j) \wedge (\bigvee_{k\in B_i} x_k)]$$

for some subsets A_i and B_i of $[1:n]$ for each $i\in[1:k]$. Since $PI(f)$ has no single variable terms, we know that $A_i \cap B_i = \emptyset$ for each $i\in[1:k]$. Thus

$$f(x) = \bigvee_{i=1}^{k} \{x_j x_k \mid j\in A_i \text{ and } k\in B_i\}.$$