Rivest

THE ROBUSTNESS OF COMBINATORIAL MEASURES OF BOOLEAN MATRIX COMPLEXITY

by

STEPHEN WAYNE BOYACK

Submitted to the Department of Mathematics
in Partial Fulfillment of the
Requirements for the
Degree of

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

3 June 1985

Signature of Author____

Department of Mathematics
May, 1985

Certified by____

Ronald L. Rivest
Thesis Supervisor

Accepted by____

Nesmith C. Ankeny
Chairman, Departmental Committee

THE ROBUSTNESS OF COMBINATORIAL MEASURES OF BOOLEAN MATRIX COMPLEXITY

by

STEPHEN W. BOYACK

Submitted to the Department of Mathematics
on May 4, 1985 in Partial Fulfillment of the
Requirements for the degree of Doctor of Philosophy

ABSTRACT

This paper studies the representation of linear and nonlinear Boolean functions by matrices, and presents the thesis that the circuit complexity of Boolean functions is *robust* in the sense that simple combinatorial or geometrical transformations to the matrix do not significantly change the complexity of the represented function. It is hypothesized that this fact may explain the difficulty of a long-standing open problem in complexity theory: the exhibition of concrete functions with nonlinear circuit complexity.

The complexity of a matrix with respect to a vectorial addition chain is shown to be equal to that of its transpose, and this result used to obtain bounds on the complexity of lopsided matrices. Measuring the complexity of Boolean functions over $\{\oplus\}$ or $\{\vee\}$ is shown to be $\mathcal{NP}$-hard. Asymptotically tight bounds are obtained on the circuit complexity of the hardest quadratic monotone functions and on the complexity of functions with respect to circuits whose width is equal to that of the input. Notions of conditional circuit complexity and Boolean line integrals are defined and characterized. The number and nature of Boolean polynomials whose functional values are identical to their coefficients is determined. It is shown that the complexity of a Boolean function equals that of its gradient if and only the complexity of the function $f(x_1, \ldots, x_n, y_1, \ldots, y_k)$ equals that of the set of functions $\{f(x_1, \ldots, x_n)\}_{y_1,\ldots,y_k}$, and that either property implies that the truth table of a Boolean function can be manipulated in various ways without substantially changing the function's circuit complexity. A function whose complexity differs from that of its inverse is demonstrated, and one-way functions are shown not to exist if the circuit complexity of a Boolean polynomial equals that of its coefficients.

In addition, the nonlinear lower bound problem is reduced to the question of whether certain simple monotone functions have nonlinear monotone complexity, and to the question of whether certain sets of subsets of an n-element set have complexity $> 2n + O(\log n)$. Lower bounds of $2n - O(\log n)$ are presented for these same sets.

Thesis Supervisor: Dr. Ronald L. Rivest

ACKNOWLEDGMENTS

I would like to express my appreciation to Ron Rivest for his patience and interest, and to Tom Leighton for his helpful discussions.

Kathy Houser provided great assistance by proofreading portions of the manuscript, as did Thomas Reinhardt by explaining the intricacies of the hardware and software with which it was prepared.

PREFACE

This paper reports various results concerning the circuit complexity of Boolean functions defined by matrices, and investigates the thesis that circuit complexity is a *robust* complexity measure — i.e., that it is insensitive to simple perturbations of the problem whose complexity is measured.

Chapter 1 reviews the basic facts about finite Boolean rings and algebras necessary for the rest of the paper. It includes tests for determining when elements of a finite Boolean algebra generate the entire algebra and for when they are algebraically independent. Chapter 2 defines the circuit complexity model of computation, and establishes upper and lower bounds on the circuit complexity of various single and multiple-output Boolean functions. *Conditional circuit complexity* is defined and characterized, and problems related to optimizing circuits over $\{\oplus\}$ and $\{\vee\}$ are shown to be $\mathcal{NP}$-complete or $\mathcal{NP}$-hard.

In Chapter 3 it is shown that the complexity of a linear Boolean function over the basis $\{\oplus\}$ is equal to that of its transpose. The resulting construction is applied to vectorial addition chains and to circuit synthesis problems, and is used to obtain sharp bounds on the complexity of functions with many inputs. The basic result is extended from linear functions to functions over arbitrary commutative associative operators and is used to produce a tight estimate of the complexity of the hardest quadratic monotone Boolean functions. In Chapter 4 asymptotically tight upper and lower bounds are obtained on the complexity of computing the hardest n-input Boolean function with a circuit of width n.

Chapter 5 starts by reviewing the calculation of Boolean derivatives and the Boolean analog of Taylor's Theorem. Boolean line integrals are then defined, and a characterization of functions whose integrals are path-independent is obtained that is similar to the analogous result for real functions. In Chapter 6 the terminology of Boolean derivatives is used to define the Δ-*transform* and Δ-*convolution* of a Boolean function. Calculating the Δ-transform is equivalent both to the problem of Boolean polynomial interpolation and the problem of Boolean polynomial evaluation. Those functions that are invariant under the Δ-transform are counted and characterized. It is demonstrated that computing the Δ-transform is complete with respect to Valiant's class $\#_2\mathcal{P}$, and the hypothesis that the circuit complexity of a Boolean function is approximately equal to that of its Δ-transform is discussed.

Chapter 7 defines the notion of the *set complexity* of a set of subsets of a finite set, and relates it to other measures of combinatorial complexity. The set complexity of a set of subsets is equivalent to the complexity of a Boolean matrix with respect to row operations and to the circuit complexity of the easiest member of a particular set of Boolean functions. Using set complexity as a tool, the important open problem of displaying a concrete Boolean function with nonlinear circuit complexity is shown to be as easy as demonstrating nonlinear monotone complexity within a narrow class of monotone functions or demonstrating set complexity $> 2n + O(\log n)$. Concrete functions with set complexity equal to $2n - O(\log n)$ are presented in the same chapter, showing that the gap that must be closed to show nonlinearity is numerically quite small. In addition, it is shown that polynomial set complexity corresponds to *exponential* circuit complexity, so that a relatively small nonlinearity in one model corresponds to a very large nonlinearity in the other. Finally, it is shown that matrices with nonlinear set complexity are likely to be very difficult to describe. In Chapter 8, the following two hypotheses concerning circuit complexity are shown to be equivalent: the complexity of $f(x_1, \ldots, x_n, y_1, \ldots, y_k)$ and that of $\{f(x_1, \ldots, x_n)\}_{y_1,\ldots,y_k}$ are equal up to a constant factor and the complexity of the gradient of a Boolean function and that of the function itself are the same up to a constant factor. In addition, it is shown that if either of these hypotheses are true, then the set complexity of a matrix has the same order as the set complexity of its transpose. Whether the hypotheses hold for arbitrary Boolean functions remains an open question, although in 1983 Baur and Strassen showed that the algebraic complexity of a real polynomial is within a constant factor of the algebraic complexity of its gradient.

In Chapter 9 a function whose circuit complexity differs from that of its inverse is constructed, and it is shown that the existence of small circuits for the Δ-transform of functions with small circuit complexity implies that one-way functions do not exist. Counting arguments are used to show that random sequences of functions are not one-way, and the set of all functions $B^3 \to B^3$ is examined exhaustively for one-way functions.

Chapter 10, the final chapter of the paper, summarizes its basic thesis: that circuit complexity may be robust in the sense that simple geometrical and combinatorial transformations of a finite function do not change the function's complexity. If true, this would place strong constraints on the structure of functions with nonlinear circuit complexity. Open problems related to topics addressed earlier are also listed.

TABLE OF CONTENTS

— Chapter One —

BOOLEAN RINGS AND ALGEBRAS

The purpose of this chapter is to establish notational conventions and to review some fundamental facts about finite Boolean rings and Boolean polynomials which will be used throughout the paper. All of the results in this chapter are elementary consequences of the definition of a Boolean ring.

Proposition 1.1

Given any Boolean algebra $\langle S, \wedge, \vee, -, 0, 1\rangle$, the operations $x \wedge y$ and $x \oplus y = (x \wedge \bar{y}) \vee (\bar{x} \wedge y)$ define a ring structure on S.

Proof:

$\wedge$ is associative and commutative with identity 1. $\oplus$ is associative and commutative with identity 0, and every element is its own inverse. $\wedge$ is distributive over $\oplus$, i.e., for any $a, b, c \in S$, $a \wedge (b \oplus c) = (a \wedge b) \oplus (a \wedge c)$. ∎

Identifying $\wedge$ with multiplication and $\oplus$ with addition, the resulting ring is a **Boolean ring,** i.e., a ring with 1 in which every element is idempotent ($a^2 = a$). Every Boolean ring is commutative and involutoric ($x + x = 0$), and the ring generated by a Boolean algebra clearly has both of these properties.

Proposition 1.2

Each Boolean ring corresponds to a Boolean algebra.

Proof:

The lattice operations $\wedge$, $\vee$, and complementation can be reconstructed from addition and multiplication by the rules

$$\begin{aligned} x \wedge y &= x \wedge y \\ x \vee y &= x \wedge y \oplus x \oplus y \\ \overline{x} &= x \oplus 1. \quad \blacksquare \end{aligned}$$

To emphasize that it represents a ring multiplication, $x \wedge y$ will normally be written xy. In what follows, we will primarily be concerned with the two-valued Boolean ring $\{0,1\}$, denoted B. B^n will denote the direct sum of n copies of B.

Proposition 1.3

Any function $f : B^n \to B$ can be expressed uniquely as a Boolean polynomial with coefficients in B.

Proof:

For any $f : B^n \to B$,

$$f(x_1,\ldots,x_n) = \bigoplus_{\substack{\text{all } n-\text{tuples} \\ (c_1,\ldots,c_n)}} [(x_1 \oplus c_1 \oplus 1)\cdots(x_n \oplus c_n \oplus 1)]\, f(c_1,\ldots,c_n),$$

where $(x_1,\ldots,x_n) \in B^n$ and $c_i \in B$. Note that in evaluating this sum at $(x_1,\ldots,x_n)$ all of the terms vanish except the one corresponding to $c_1 = x_1, c_2 = x_2, \ldots c_n = x_n$, which takes exactly the value $f(x_1,\ldots,x_n)$. When multiplied out the sum becomes a polynomial in $x_1,\ldots,x_n$, of degree $\leq n$.

This representation of f as a polynomial is unique. Since $x_i \wedge x_i = x_i$, there are just 2^n possible monomials of n variables, and 2^{2^n} polynomials, since each monomial may either appear or not appear. However, there are just 2^{2^n} distinct functions from B^n to B. $\blacksquare$

Boolean polynomials can be written like ordinary polynomials in n variables, except that because B is idempotent, it is never necessary to write any exponent greater than one, and because B is involutoric, it is never necessary to write any coefficient greater than one.

Since there are 2^{2^n} distinct functions from $B^n \to B$, every Boolean polynomial can be expressed uniquely in the form

$$\bigoplus_{\substack{\text{all } n\text{-tuples}\\ \alpha=(\alpha_1,\ldots,\alpha_n)}} a_\alpha(x_1^{\alpha_1}\cdots x_n^{\alpha_m}),$$

where a_α and α_i are $\in B$, $x_i^0 = 1$, and $x_i^1 = x_i$. An expression $x_1^{\alpha_1}x_2^{\alpha_2}\cdots x_n^{\alpha_n}$ is a **Boolean monomial**; it has degree k if exactly k of the α_i are 1. The **degree of a polynomial** is the highest degree of any of the monomials which it comprises.

A function $f : B^n \to B$ is **linear** if $f(x_1,\ldots,x_n) \oplus f(y_1,\ldots,y_n) = f(x_1 \oplus y_1,\ldots x_n \oplus y_n)$ for all $(x_1,\ldots,x_n)$ and $(y_1,\ldots,y_n) \in B^n$.

Proposition 1.4

A linear function is represented by a polynomial each term of which has degree one.

Proof:

If $f = a_1x_1 \oplus \ldots \oplus a_nx_n$, then $f(x_1 \oplus y_1,\ldots,x_n \oplus y_n) = a_1(x_1 \oplus y_1) \oplus \ldots \oplus a_n(x_n \oplus y_n) = f(x_1,\ldots,x_n) \oplus f(y_1,\ldots,y_n)$. On the other hand, if f is linear, $f(x_1,\ldots,x_n) = f(x_1,0,\ldots,0) \oplus f(0,x_2,\ldots,0) \oplus \ldots \oplus f(0,\ldots,0,x_n)$. Now $f(0,\ldots,x_i,\ldots,0)$ is a function of a single variable and must therefore be represented by a polynomial of degree ≤ 1, (i.e., x_i, $x_i \oplus 1$, 0, or 1). Hence, $f(x_1,\ldots,x_n) = a_0 \oplus \bigoplus_{i=1}^n a_ix_i$ for $a_i \in B$. However, a_0 cannot be 1; otherwise $f(0,\ldots,0) = a_0 = 1$ and $f(x_1 \oplus 0,\ldots,x_n \oplus 0) = \bigoplus_{i=1}^n a_ix_i \neq f(x_1,\ldots,x_n)$ — a contradiction. ∎

An affine function $B^n \to B$ is one whose polynomial representation has degree ≤ 1. Every affine function can be expressed as the sum of a linear function and a constant. A function $B^n \to B^m$ is linear if each of its coordinate functions is linear, and an function $B^n \to B^m$ is affine if each of its coordinate functions is affine.

In the literature of switching theory the polynomial representation of a function $f : B^n \to B$ is known as the **ring-sum expansion** of f or as the Reed-Muller canonic expansion of f (of polarity 0).

If $F : B^n \to B^m$, one can associate with F the m **coordinate functions** $f_i : B^n \to B$, defined by $f_i(x_1, \ldots, x_n) = \pi_i(F(x_1, \ldots, x_n))$, where $\pi_i : B^m \to B$ is the projection taking B^m onto the algebra generated by its i^{th} atom. For $F : B^n \to B^m$, the coordinate functions will be denoted $f_1, f_2, \ldots, f_m$. In general, Roman letters will denote multiple-output Boolean functions (functions from B^n to B^m), and lower-case letters single-output functions (functions from B^n to B).

Definitions 1.5

Let f be a function from B^n to B.

The **complement** of f is the function $g : B^n \to B$ taking $x \mapsto 1 \oplus f(x)$.

The **dual** of f is the function $g : B^n \to B$ taking $x \mapsto 1 \oplus f(1 \oplus x)$.

The **contradual** of f is the function $g : B^n \to B$ taking $x \mapsto f(x \oplus 1)$.

The dual of $x \wedge y$ is $x \vee y$, of $x \vee y$ is $x \wedge y$, of $x \oplus y$ is $x \oplus y \oplus 1$, and of $x \oplus 1$ is $x \oplus 1$. The dual of 0 is 1, and of 1 is 0. For every theorem or tautology in Boolean algebra, there is a corresponding dual theorem, produced by replacing 1 with 0, 0 with 1, $\wedge$ with $\vee$, $\vee$ with $\wedge$, and $\oplus$ with $\oplus 1$ in each predicate.

Definition 1.6

An **atom** of a Boolean algebra S is an element a such that for all $s \in S$, either $a \wedge s = a$ or $a \wedge s = 0$.

Definitions 1.7

A **homomorphism** between Boolean algebras [Boolean rings] S_1 and S_2 is a mapping which preserves $\{0, 1, \wedge, \vee, -\}[\{0, 1, \oplus, \wedge\}]$. An **isomorphism** is a bijective homomorphism, and an **automorphism** is an isomorphism from S_1 to itself.

Theorem 1.8

Let S be a finite Boolean algebra with at least one element. Then S is isomorphic to B^n for some integer n.

Proof:

(a) S has some finite non-zero number of atoms.

The relation $p \leq q \Leftrightarrow (p \wedge q = p)$ establishes a partial order on S, and since S is finite, each totally ordered chain in S has finite length. Consider the smallest non-zero element a in some chain of maximal length. If $s \neq 0$ is some element in this chain then $a \wedge s = a$. If $s \in S$ is not in the chain, then $a \wedge s \leq a$. In this last case, $a \wedge s = 0$, for otherwise a would not be minimal. Therefore, a is an atom. Let $\{a_1, \ldots, a_n\}$ be the set of atoms of S.

(b) Each pair of atoms are orthogonal, i.e., $a_i a_j = 0$.

If a_i and a_j are distinct atoms, $a_i a_j = a_i$ or 0 (since a_i is an atom). $a_i a_j$ also equals a_j or 0, since a_j is an atom, whence $a_i a_j = 0$ for all i, j.

(c) The atoms are linearly independent.

Suppose $\bigoplus a_i = 0$ for some subset of the a_i. If a is one of the atoms appearing in the sum, $0 = a \wedge 0 = a \wedge \bigoplus a_i = \bigoplus a a_i = a \oplus \bigoplus_{a_i \neq a} a a_i = a \oplus 0 = a$.

(d) The sum of the atoms is 1.

Suppose not. Then $1 \oplus \bigoplus_{i=1}^{n} a_i = a \neq 0$. Consider the set of $s \in S$ for which $s \leq a$. This set is finite and non-empty (since it contains a), so it must contain a minimal non-zero element with respect to $\leq$. Call this element b. Now, for any $s \in S$, $(b \wedge s) \wedge b = b \wedge s$, so $b \wedge s \leq b$. This means $b \wedge s \leq a$, whence $b \wedge s$ either equals 0 or b; or else b would not be minimal. Therefore b is an atom, and is equal to a_j for some j. Thus, $a_j \leq a$. But this is a contradiction, for $a_j = a_j \wedge a = a_j(1 \oplus \bigoplus_{i=1}^{n} a_i) = a_j \oplus a_j a_j \oplus \bigoplus_{i \neq j} a_j a_i = a_j \oplus a_j \oplus 0 = 0$. Therefore

$$\bigoplus_{i=1}^{n} a_j = 1.$$

(e) The atoms span S.

For any $s \in S$, $s = s \wedge 1 = s \wedge \bigoplus_{i=1}^{n} a_i = \bigoplus_{i=1}^{n} sa_i$. Since for each i, $sa_i = a_i$ or $sa_i = 0$,

$$S = \bigoplus_{\substack{\text{some set} \\ \text{of atoms}}} a_i.$$

(f) S is a vector space over B with basis $\{a_i\}$.

Every $s \in S$ has a unique representation as $\bigoplus_{i=1}^{n} x_i a_i$, for $x_i \in B$, and can be thought of as an n-tuple $(x_1, \ldots, x_n)$.

(g) S is isomorphic to B^n.

It needs only to be checked that the bijection taking $(x_1, \ldots, x_n)$ to $\bigoplus_{i=1}^{n} x_i a_i$ preserves $1, 0, \wedge$, and $\oplus$.

$$(0, \ldots, 0) \mapsto 0$$

$$(1, \ldots, 1) \mapsto 1$$

$$\begin{aligned}(x_1, \ldots, x_n) \wedge (y_1, \ldots y_n) &\mapsto (x_1 a_1 \oplus \ldots \oplus x_n a_n) \wedge (y_1 a_1 \oplus \ldots \oplus y_n a_n) \\ &= (x_1 \wedge y_1) a_1 \oplus \ldots \oplus (x_n \wedge y_n) a_n,\end{aligned}$$

since $a_i a_j = 0$.

$$\begin{aligned}(x_1, \ldots, x_n) \oplus (y_1, \ldots, y_n) &\mapsto (x_i a_i \oplus \ldots \oplus x_n a_n) \oplus (y_1 a_1 \oplus \ldots \oplus y_n a_n) \\ &= (x_1 \oplus y_1) a_1 \oplus \ldots \oplus (x_n \oplus y_n) a_n. \quad \blacksquare\end{aligned}$$

Thus, the structure of finite Boolean algebras is relatively simple. Infinite Boolean algebras can be much more complicated: although every Boolean algebra is isomorphic to a set of subsets of some set, the algebra need not be isomorphic to the set of all subsets of a set; nor need it have any atoms at all [HAL].

Definition 1.9.1

If $\langle S_1, 0, 1, \wedge, \vee, - \rangle$ is a Boolean algebra, and S_2 is a subset of S_1 containing 0 and 1 which is also a Boolean algebra with respect to $\wedge$ and $\vee$, then S_2 is a subalgebra of S_1.

The intersection of subalgebras of S_1 is a subalgebra of S_1.

Definition 1.9.2

If E is a subset S_1, the **subalgebra generated by** E is the intersection of all subalgebras of S_1 containing E.

Definition 1.9.3

If $\langle R_1, 0, 1, \oplus, \wedge \rangle$ is a Boolean ring, $R_2 \subseteq R_1$ is a **subring** of R_1 if it contains $0, 1$ and is itself a Boolean ring with respect to $\oplus$ and $\wedge$.

The intersection of subrings of R is a subring of R.

Definition 1.9.4

If R is a Boolean ring and E is a subset of R, then the **subring generated by** E is the intersection of the subrings of R which contain E.

The subring generated by E and the subalgebra generated by E represent the same set, which we will denote $\langle E \rangle$.

Proposition 1.10

Let R be a Boolean ring and $E = \{e_1, \ldots, e_m\}$ be a subset of R. The subring of R generated by E consists of all elements of the form:

$$\bigoplus_{\substack{\text{all } m\text{-tuples} \\ \alpha=(\alpha_1,\ldots,\alpha_m)}} a_\alpha \bigwedge_{i=1}^{m} e_i^{\alpha_i},$$

where $\alpha \in \{0,1\}$, $e_i^0 = 1$, and $e_i^1 = e_i$.

Proof:

The set described in the theorem is generated by finite combinations of e_i, so it is contained in any subring containing E. On the other hand, the same set is closed under $\oplus$ and $\wedge$, and contains 0 and 1, so it comprises a subring which contains E.

$$0 \bigwedge_{i=1}^{m} e_i^1 = 0 \in \langle E \rangle.$$

$$1 \bigwedge_{i=1}^{m} e_i^0 = 1 \in \langle E \rangle.$$

$$\left(\bigoplus_{\alpha} a_\alpha \bigwedge_{i=1}^{m} e_i^{\alpha_i}\right) \oplus \left(\bigoplus_{\alpha} b_\alpha \bigwedge_{i=1}^{m} e_i^{\alpha_i}\right) = \bigoplus_{\alpha} (a_\alpha \oplus b_\alpha) \bigwedge_{i=1}^{m} e_i^{\alpha_i} \in \langle E \rangle.$$

$$\begin{aligned}\left(\bigoplus a_\alpha \bigwedge_{i=1}^{m} e_i^{\alpha_i}\right) \wedge \left(\bigoplus b_\alpha \bigwedge_{i=1}^{m} e_i^{\alpha_i}\right) &= \bigoplus_{\alpha,\beta} \left(a_\alpha \bigwedge_{i=1}^{m} e_i^{\alpha_i}\right)\left(a_\beta \bigwedge_{i=1}^{m} e_i^{\beta_i}\right) \\ &= \bigoplus_{\alpha,\beta} (a_\alpha \wedge a_\beta) \bigwedge_{i=1}^{m} e_i^{\alpha_i \vee \beta_i} \\ &= \bigoplus_{\gamma} a_\gamma \bigwedge_{i=1}^{m} e_i^{\gamma_i} \\ &\in \langle E \rangle. \quad \blacksquare\end{aligned}$$

Example 1.10.1

If the atoms of B^6 are s_1, s_2, s_3, s_4, s_5, and s_6, the subring generated by $e_1 = s_1 \oplus s_5$ and $e_2 = s_1 \oplus s_2 \oplus s_3$ consists of the elements in B^6 of the form

$$a_0[1] \oplus a_1[s_1 \oplus s_5] \oplus a_2[s_1 \oplus s_2 \oplus s_3] \oplus a_3[s_1].$$

There are 16 distinct such elements, namely:

0	1
s_1	$1 \oplus s_1$
$s_1 \oplus s_2 \oplus s_3$	$1 \oplus s_1 \oplus s_2 \oplus s_3$
$s_2 \oplus s_3$	$1 \oplus s_2 \oplus s_3$
s_5	$1 \oplus s_5$
$s_1 \oplus s_5$	$1 \oplus s_1 \oplus s_5$
$s_2 \oplus s_3 \oplus s_5$	$1 \oplus s_2 \oplus s_3 \oplus s_5$
$s_1 \oplus s_2 \oplus s_3 \oplus s_5$	$1 \oplus s_1 \oplus s_2 \oplus s_3 \oplus s_5.$

Corollary 1.10.2

If $E = \{e_1, \ldots, e_m\}$, then $\langle E \rangle$ has at most 2^{2^m} distinct elements.

Proposition 1.11

If $E = \{e_1, \ldots, e_m\}$, then the atoms of $\langle E \rangle$ consist of those elements of the form $\bigwedge_{i=1}^m e_i'$, where $e_i' \in \{e_i, \bar{e}_i\}$.

Proof:

$$\left(\bigwedge_{i=1}^{m} e_i'\right) \wedge \left(\bigoplus_{\alpha} a_\alpha \bigwedge_{i=1}^{m} e_i^{\alpha_i}\right) = \bigoplus_{\alpha} a_\alpha \bigwedge_{i=1}^{m} e_i^{\alpha_i} e_i'$$
$$= \bigoplus_{\alpha} a_\alpha \bigwedge_{i=1}^{m} e_i'$$
$$= \bigwedge_{i=1}^{m} e_i' \quad \textit{or} \quad 0\,,$$

so $\bigwedge_{i=1}^m e_i'$ is an atom in $\langle E \rangle$. On the other hand, these are the only atoms. For suppose b is an atom in $\langle E \rangle$ not of the form $\bigwedge_{i=1}^m e_i'$. Since the product of two

distinct atoms is 0,

$$\begin{aligned}
b &= b \wedge 1 \\
&= b\left(\bigwedge_{i=1}^{m} (e_i \vee \bar{e}_i)\right) \\
&= b\left(\bigvee_{\substack{all\ possible \\ m-tuples}} \left(\bigwedge_{i=1}^{m} e_i'\right)\right) \\
&= \bigvee_{\substack{all\ possible \\ m-tuples}} \left(b \bigwedge_{i=1}^{m} e_i'\right) \\
&= \bigvee_{\substack{all\ possible \\ m-tuples}} (0) \\
&= 0 . \quad \blacksquare
\end{aligned}$$

Corollary 1.11.1

If $E = \{e_1, \ldots, e_m\}$, every element of $\langle E \rangle$ can be expressed in the form

$$\bigvee_{\alpha} \left(a_\alpha \bigwedge_{i=1}^{m} e_i'\right),$$

where $e_i' \in \{e_i, \bar{e}_i\}$, and $a_\alpha \in \{0, 1\}$.

Proposition 1.11.2

Let S be a finite Boolean ring. I is an ideal of S if and only if $I = \langle \{a_j\} \rangle$, where a_j are atoms of S.

Proof:

By Theorem 1.8, S is isomorphic to B^n for some n. Let $\{a_i\}$ be the atoms of S. Now, for $s \in S$, $s \bigoplus_{i=1}^{n} \alpha_i a_i = [\bigoplus_{i=1}^{n} \sigma_i a_i][\bigoplus_{j=1}^{m} \alpha_j a_j] = \bigoplus_{j=1}^{m} \sigma_i(\alpha_j a_j) \in \langle \{a_j\} \rangle$; hence $\langle \{a_i\} \rangle$ is an ideal. On the other hand, suppose that I is an ideal and that $\{b_j\}$ are all the atoms of S such that $b_j \leq c$ for some $c \in I$. $b_j c = b_j$, so $b_j \in I$, whence $\langle \{b_j\} \rangle \subseteq I$, and $I = \langle \{b_j\} \rangle$. $\blacksquare$

Theorem 1.12

Let S be a finite Boolean algebra with atoms $\{a_1, \ldots, a_n\}$ and let $E = \{e_1, \ldots, e_m\}$ be a set of elements from S with the atomic representations

$$e_i = \bigoplus_{j=1}^{n} \alpha_{ij} a_j.$$

Then the following conditions are equivalent:

(a) E generates S,

(b) the column vectors $(\alpha_{1j}, \alpha_{2j}, \ldots, \alpha_{mj})^T$ are distinct for $1 \leq j \leq n$.

(c) for every $1 \leq j' < j'' \leq n$, there exists some i such $\alpha_{ij'} \neq \alpha_{ij''}$.

Proof:

(c $\Rightarrow$ b) Immediate.

(a $\Rightarrow$ c) Suppose there are some j' and j'' such that $a_{ij'} = a_{ij''}$ for every i. Let S_1 be the subset of S whose j'^{th} and j''^{th} coordinates are equal, i.e., the set of elements $s \in S$ for which

$$s = \bigoplus_{j=1}^{n} \alpha_j a_j \Rightarrow \alpha_{j'} = \alpha_{j''}.$$

S_1 is a subalgebra of S, since it is evidently closed under $\bigoplus$ and $\bigwedge$, and contains 0 and 1. Now, $\langle E \rangle \subseteq S_1$, but $a_{j'}$, which is equal to $1 \bigwedge a_{j'} \bigoplus 0 \bigwedge a_{j''}$, is not contained in S, whence $\langle E \rangle \neq S$, a contradiction.

(b $\Rightarrow$ a) To show that E generates S, it suffices to show that E generates an arbitrary atom in S, say a_j. Consider

$$x = \left(\bigwedge_{a_j e_i = 1} e_i \right) \left(\bigwedge_{a_j e_i = 0} \bar{e}_i \right),$$

where the first product is taken over all e_i with $a_j e_i = a_j$ and the second over all e_i with $a_j e_i = 0$. Now $a_j x = a_j$. By (b), on the other hand, for any $k \neq j$ there exists some $e \in E$ with either $a_j e = a_j$ and $a_k e = 0$, or $a_j e = 0$ and $a_k e = a_k$. In either case, $a_k x = 0$. Thus, x is equal to a_j. ∎

Example 1.12.1

Suppose $a_1, a_2, \ldots, a_5$ are the elements of B^5. The elements

$$\begin{aligned}
e_1 &=(0\wedge a_1)\oplus(1\wedge a_2)\oplus(1\wedge a_3)\oplus(0\wedge a_4)\oplus(0\wedge a_5)\\
e_2 &=(1\wedge a_1)\oplus(1\wedge a_2)\oplus(1\wedge a_3)\oplus(1\wedge a_4)\oplus(0\wedge a_5)\\
e_3 &=(0\wedge a_1)\oplus(1\wedge a_2)\oplus(0\wedge a_3)\oplus(0\wedge a_4)\oplus(0\wedge a_5)\\
e_4 &=(1\wedge a_1)\oplus(1\wedge a_2)\oplus(1\wedge a_3)\oplus(1\wedge a_4)\oplus(1\wedge a_5)\\
e_5 &=(0\wedge a_1)\oplus(0\wedge a_2)\oplus(1\wedge a_3)\oplus(1\wedge a_4)\oplus(1\wedge a_5)
\end{aligned}$$

generate B^5, but the elements

$$\begin{aligned}
e_1 &=(0\wedge a_1)\oplus(1\wedge a_2)\oplus(1\wedge a_3)\oplus(0\wedge a_4)\oplus(0\wedge a_5)\\
e_2 &=(1\wedge a_1)\oplus(1\wedge a_2)\oplus(1\wedge a_3)\oplus(1\wedge a_4)\oplus(0\wedge a_5)\\
e_3 &=(0\wedge a_1)\oplus(1\wedge a_2)\oplus(0\wedge a_3)\oplus(0\wedge a_4)\oplus(0\wedge a_5)\\
e_4 &=(1\wedge a_1)\oplus(1\wedge a_2)\oplus(1\wedge a_3)\oplus(1\wedge a_4)\oplus(0\wedge a_5)\\
e_5 &=(0\wedge a_1)\oplus(0\wedge a_2)\oplus(1\wedge a_3)\oplus(0\wedge a_4)\oplus(1\wedge a_5)
\end{aligned}$$

do not, since in the latter case the coefficients of a_1 and a_4 are identical for each e_i.

Definition 1.13

Let S_1 be a Boolean algebra and $e_1, \ldots, e_m$ be elements of S_1. $E = \{e_1, \ldots, e_m\}$ is free on S_1 if for any Boolean algebra S_2, any mapping $g : E \to S_2$ can be extended to a homomorphism $f : S_1 \to S_2$ such that $f(e_i) = g(e_i)$ for all $e_i \in E$.

$$\begin{tikzcd}
E \arrow[rr, "1"] \arrow[dr, "g"'] & & S_1 \arrow[dl, dashed, "f"] \\
& S_2 &
\end{tikzcd}$$

The set of all functions $B^n \to B^m$ is a Boolean algebra, with the constant functions $f(x) = 0$ and $f(x) = 1$ representing 0 and 1 respectively, and meet and join given by $f \wedge g(x) = f(x) \wedge g(x)$ and $f \vee g(x) = f(x) \vee g(x)$.

Example 1.13.1

The set of functions $\{f(x_1, \ldots, x_n) = x_1, \ldots, f(x_1, \ldots, x_n) = x_n\}$ is a free set of generators for the Boolean algebra, A_n, of all functions from B^n to B. For any Boolean algebra S and map taking $x_i \mapsto g(x_i)$, the homomorphism $f : A_n \to S$ takes

$$F(x_1, \ldots, x_n) = \bigoplus_\alpha a_\alpha \bigwedge_{i=1}^{n} x_i^{\alpha_i}$$

to

$$\bigoplus_\alpha a_\alpha \bigwedge_{i=1}^{n} g^{\alpha_i}(x_i).$$

Proposition 1.14

If S_1 is a finite Boolean algebra and $E = \{e_1, \ldots, e_m\}$ is a set of elements in S_1, then E is free if and only if the elements $e_1, \ldots, e_m$ are algebraically independent in S_1, i.e., if and only if $a_\alpha \in \{0,1\}$ and

$$\bigoplus_\alpha a_\alpha \bigwedge_{i=1}^{m} e_i^{\alpha_i} = 0$$

implies $a_\alpha = 0$.

Proof:

Suppose E is free on S_1. For any Boolean algebra S_2 and any function $g : E \to S_2$, there must be a unique homomorphism $f : S_1 \to S_2$ such that $g(e_i) = f(e_i)$. But suppose there is also some algebraic relation between the e_i, say

$$P(e_1, \ldots, e_m) = \bigoplus_\alpha a_\alpha \bigwedge_{i=1}^{m} e_i^{\alpha_i} = 0.$$

Let $\bigwedge_{i \in D} e_i$ be a term of maximal degree in the polynomial P. Now,

$$\begin{aligned}
0 &= f(0) \\
&= f\left(\bigoplus_\alpha a_\alpha \bigwedge_{i=1}^{m} e_i^{\alpha_i}\right) \\
&= \bigoplus_\alpha f\left(a_\alpha \bigwedge_{i=1}^{m} e_i^{\alpha_i}\right) \\
&= \bigoplus_\alpha f(a_\alpha) f\left(\bigwedge_{i=1}^{m} e_i^{\alpha_i}\right) \\
&= \bigoplus_\alpha a_\alpha \bigwedge_{i=1}^{m} (f(e_i))^{\alpha_i} \\
&= \bigoplus_\alpha a_\alpha \bigwedge_{i=1}^{m} (g(e_i))^{\alpha_i} \quad .
\end{aligned}$$

However, if g is selected so that $g(e_i) = 1$ for $i \in D$ and $g(e_i) = 0$ for $i \notin D$, then this last quantity is just

$$\bigoplus_{i \in D} g(e_i) = 1.$$

This is a contradiction.

Conversely, suppose that $e_1, \ldots, e_m$ are algebraically independent. Every element in E must have a *unique* representation of the form

$$\bigoplus_{\alpha} a_\alpha \bigwedge_{i=1}^{m} e_i^{\alpha_i}.$$

Every element in E certainly has at least one representation in this form. However if it has two representations, say

$$\bigoplus_{\alpha} a_\alpha \bigwedge_{i=1}^{m} e_i^{\alpha_i} = \bigoplus_{\alpha} b_\alpha \bigwedge_{i=1}^{m} e_i^{\alpha_i},$$

they must be identical since by Proposition 1.3

$$\bigoplus_{\alpha} (a_\alpha \oplus b_\alpha) \bigwedge_{i=1}^{m} e_i^{\alpha_i} = 0$$

implies $a_\alpha \oplus b_\alpha = 0$ and $a_\alpha = b_\alpha$.

Next, let S_2 be an arbitrary Boolean algebra and g be any function $E \to S_2$. The function $f : \langle E \rangle \to S$ defined by

$$\bigoplus_{\alpha} a_\alpha \bigwedge_{i=1}^{m} e_i^{\alpha_i} \mapsto \bigoplus_{\alpha} a_\alpha \bigwedge_{i=1}^{m} g^{\alpha_i}(e_i)$$

is a homorphism of Boolean algebras, since

$$f(0) = 1,$$

$$f(1) = 1,$$

$$f\left(\bigoplus_\alpha a_\alpha \bigwedge_{i=1}^m e_i^{\alpha_i} \oplus \bigoplus_\alpha b_\alpha \bigwedge_{i=1}^m e_i^{\alpha_i}\right) = f\left(\bigoplus_\alpha (a_\alpha \oplus b_\alpha) \bigwedge_{i=1}^m e_i^{\alpha_i}\right)$$
$$= \bigoplus_\alpha (a_\alpha \oplus b_\alpha) \bigwedge_{i=1}^m g^{\alpha_i}(e_i)$$
$$= \left(\bigoplus_\alpha a_\alpha \bigwedge_{i=1}^m g^{\alpha_i}(e_i)\right) \oplus \left(\bigoplus_\alpha b_\alpha \bigwedge_{i=1}^m g^{\alpha_i}(e_i)\right)$$
$$= f\left(\bigoplus_\alpha a_\alpha \bigwedge_{i=1}^m e_i\right) \oplus f\left(\bigoplus_\alpha b_\alpha \bigwedge_{i=1}^m e_i\right),$$

and

$$f\left(\bigoplus_\alpha (a_\alpha \bigwedge_{i=1}^m e_i^{\alpha_i}) \wedge \bigoplus_\alpha (b_\alpha \bigwedge_{i=1}^m e_i^{\alpha_i})\right) = f\left(\bigoplus_\alpha (a_\alpha \wedge b_\alpha) \bigwedge_{i=1}^m e_i^{\alpha_i \vee \beta_i}\right)$$
$$= \bigoplus_\alpha (a_\alpha \wedge b_\alpha) \bigwedge_{i=1}^m g^{\alpha_i \vee \beta_i}(e_i)$$
$$= \left(\bigoplus_\alpha a_\alpha \bigwedge_{i=1}^m g^{\alpha_i}(e_i)\right) \wedge \left(\bigoplus_\alpha b_\alpha \bigwedge_{i=1}^m g^{\beta_i}(e_i)\right)$$
$$= f\left(\bigoplus_\alpha a_\alpha \bigwedge_{i=1}^m e_i^{\alpha_i}\right) \wedge f\left(\bigoplus_\alpha b_\alpha \bigwedge_{i=1}^m e_i^{\beta_i}\right).$$

In addition, f agrees with g on the set E. Now extend the domain of f from E to S_1. Since S_1 is finite, it is atomic. Denote the atoms of S by $\{a_1, \ldots, a_n\}$ and the atomic representation of e_i by

$$e_i = \bigoplus_{j=1}^n \alpha_{ij} a_j.$$

Consider the set of vectors $\{(\alpha_{1j}, \ldots \alpha_{mj})\}_{j=1,\ldots,n}$. If these are all distinct, $\langle E \rangle = S$, and the proof is done. If not, let $T : \{1, \ldots, n\} \to \{1, \ldots, n\}$ be the map which takes each integer j to $T(j)$, the smallest integer such that

$$(\alpha_{1T(j)}, \ldots, \alpha_{mT(j)}) = (\alpha_{1j}, \ldots, \alpha_{mj}).$$

Next, define $P : S_1 \to \langle E \rangle$ by

$$\bigoplus_{j=1}^n \alpha_j a_j \mapsto \bigoplus_{j=1}^n \alpha_T(j) a_j.$$

P is constant on E, since

$$\bigoplus_{j=1}^n \alpha_{ij} a_j \mapsto \bigoplus_{j=1}^n \alpha_{iT(j)} a_j = \bigoplus_{j=1}^n \alpha_{ij} a_j.$$

P therefore preserves 0 and 1. P also preserves sums and products, e.g.,

$$\begin{aligned}
P\left(\bigoplus_{j=1}^{n} \alpha_j a_j \wedge \bigoplus_{j=1}^{n} \beta_j a_j\right) &= P\left(\bigoplus_{j=1}^{n} (\alpha_j \wedge \beta_j) a_j\right) \\
&= \bigoplus_{j=1}^{n} \left((\alpha_{T(j)} \wedge \beta_{T(j)}) a_j\right) \\
&= P\left(\bigoplus_{j=1}^{n} \alpha_j a_j\right) \wedge P\left(\bigoplus_{j=1}^{n} \alpha_j a_j\right).
\end{aligned}$$

Therefore P is a homomorphism, and f extends g to S_1. ∎

Corollary 1.14.1

If $E = \{e_1, \ldots, e_m\}$ freely generates S_1, then every element in S has a unique representation as a polynomial in E. In addition, any map from E to S_2 has a unique extension to a homomorphism from S_1 to S_2.

Example 1.13.1 shows that for every n, there exists a freely-generated Boolean algebra with 2^{2^n} elements. The converse is also true: every finite freely-generated Boolean algebra has 2^{2^n} elements for some integer n, since every finite algebra is finitely-generated and every polynomial in the generators must be distinct. Thus, up to isomorphism, the only finite free Boolean algebra is B^{2^n}.

Theorem 1.15

Suppose $e_1, \ldots, e_m$ are elements of B^n, where e_i has atomic representation

$$e_i = \bigoplus_{j=1}^{n} \alpha_{ij} a_j.$$

Then $\{e_1, \ldots, e_m\}$ is free if and only if every possible Boolean m-tuple appears as $(\alpha_{1j}, \ldots, \alpha_{mj})$ for some j.

Proof:

Suppose there exists some Boolean m-tuple $(c_1, \ldots, c_m)$ which is not equal to $(\alpha_{1j}, \ldots, \alpha_{mj})$ for any j. Consider the function $P_1 : B^m \to B$ given by

$$P_1(x_1, \ldots, x_m) = \begin{cases} 1, & \text{if } (x_1, \ldots, x_m) = (c_1, \ldots, c_m) \\ 0, & \text{otherwise.} \end{cases}$$

By Proposition 1.3, $P_1(x_1, \ldots, x_m)$ can be expressed as a polynomial

$$\bigoplus_{\beta \in B^m} p_\beta x_1^{\beta_1} x_2^{\beta_2} \cdots x_m^{\beta_m}$$

where $p_\beta \in \{0, 1\}$. $P_1(x_1, \ldots, x_m)$ is not uniformly zero, and since each function $B^n \to B$ is represented by a unique polynomial, $p_\beta \neq 0$ for some $\beta \in B^n$. Consider the expression

$$\bigoplus_{\beta \in B^m} p_\beta e_1^{\beta_1} e_2^{\beta_2} \cdots e_m^{\beta_m} = P_2(e_1, \ldots, e_m).$$

For any atom $a_j \in B^n$

$$\begin{aligned} P_2(e_1, \ldots, e_m)\, a_j &= \bigoplus_{\beta \in B^m} p_\beta e_1^{\beta_1} a_j e_2^{\beta_2} a_j \cdots e_m^{\beta_m} a_j \\ &= a_j \bigoplus_{\beta \in B^m} p_\beta \alpha_{1j}^{\beta_1} \alpha_{2j}^{\beta_2} \cdots \alpha_{mj}^{\beta_m} \\ &= a_j\, P_1(\alpha_{1j}, \alpha_{2j}, \ldots, \alpha_{mj}) \\ &= 0. \end{aligned}$$

Since $P_2(e_1, \ldots, e_m) = 0$ for a non-trivial polynomial P_2, $e_1, \ldots, e_m$ are not algebraically independent and, by Proposition 1.14, are not free.

Conversely, suppose every possible Boolean m-tuple appears as $(\alpha_{1j}, \ldots, \alpha_{mj})$ for some j. If

$$\begin{aligned} Q_1(e_1, \ldots, e_m) &= \bigoplus_{\beta \in B^m} q_\beta e_1^{\beta_1} e_2^{\beta_2} \cdots e_m^{\beta_m} \\ &= 0 \end{aligned}$$

for $q \in \{0, 1\}$, then for any atom $a_j \in B^n$

$$\begin{aligned} Q_1(e_1, \ldots, e_m)\, a_j &= \bigoplus_{\beta \in B^m} q_\beta e_1^{\beta_1} a_j e_2^{\beta_2} a_j \cdots e_m^{\beta_m} a_j \\ &= a_j \bigoplus_{\beta \in B^m} q_\beta \alpha_{1j}^{\beta_1} \alpha_{2j}^{\beta_2} \cdots \alpha_{mj}^{\beta_m} \\ &= a_j\, Q_2(\alpha_{1j}, \alpha_{2j}, \ldots, \alpha_{mj}) \\ &= 0. \end{aligned}$$

That is, the polynomial $Q_2(\alpha_{1j}, \alpha_{2j}, \ldots, \alpha_{mj})$ equals 0 for every one of the 2^m possible values of its arguments. By Proposition 1.3, Q_2 is the zero polynomial, i.e., $q_\beta = 0$ for all $\beta \in B^m$. Therefore $Q_1(e_1, \ldots, e_m) = 0$ and the $e_1, \ldots, e_m$ are algebraically independent. By Proposition 1.14, $\{e_1, \ldots, e_m\}$ is free. ∎

Example 1.15.1

$e_1 = (1,0,0,1,0,1)$ and $e_2 = (1,1,0,1,1,0)$ are free. $e_3 = (1,0,0,1,0,0)$ and $e_4 = (1,1,0,1,1,1)$ are not. (Note that $e_3e_4 \oplus e_3 = 0$.)

Corollary 1.15.2

Suppose $\{e_1, \ldots, e_m\}$ is a free set of elements from B^n and that e_i has atomic representation

$$e_i = \bigoplus_{j=1}^{n} \alpha_{ij} a_j.$$

$\{e_1, \ldots, e_m\}$ can be extended to a free set of generators for B^n if and only if $n = 2^k$ for some integer k and if $(\alpha_{1j}, \ldots, \alpha_{mj})$ equals each possible Boolean m-tuple for exactly 2^{k-m} values of j.

Example 1.15.2.1

$\{e_1 = (1,0,0,1,1,0,1,0), e_2 = (1,1,1,0,1,0,0,0)\}$ can be extended to a free set of generators for B^8 by adjoining $e_3 = (0,1,0,0,1,1,1,0)$. By contrast, $\{e_1 = (1,0,0,1,1,0,1,1), e_2 = (1,1,1,0,1,0,0,1)\}$ cannot be so extended.

Proposition 1.16

An automorphism on a finite Boolean algebra takes atoms to atoms.

Proof:

Let ϕ be an automorphism on some finite Boolean algebra S. By Theorem 1.8, S has at least one atom a. Since S is finite, ϕ is invertible, and $\phi(a) \wedge x = \phi(a) \wedge \phi(\phi^{-1}(x)) = \phi(a \wedge \phi^{-1}(x))$. Since a is an atom, $a \wedge \phi^{-1}(x)$ is equal either to a or to 0, and $\phi(a \wedge \phi^{-1}(x))$ is equal to either $\phi(a)$ or $\phi(0) = 0$. ∎

Proposition 1.17

Let A_n be the Boolean algebra of all functions from $B^n \to B$. There is a one-to-one correspondence between:

(a) automorphisms on A_n,

(b) free sequences of generators for A_n,

(c) permutations of the atoms of A_n, and

(d) invertible functions $B^n \to B^n$.

Proof:

First, note that A_n is isomorphic to B^{2^n} and to any free Boolean algebra on n generators. The atoms of A_n are the functions $\{a_j\}_{j \in B^n}$ for which

$$a_j(x_1, \ldots, x_n) = \begin{cases} 1, & \text{if } (x_1, \ldots, x_n) = j \\ 0, & \text{otherwise.} \end{cases}$$

(a$\leftrightarrow$c) Every automorphism on A_n permutes atoms, and every permutation of atoms $\pi : \{a_j\} \to \{a_j\}$ defines an automorphism:

$$\bigoplus_{j \in B^n} \alpha_j a_j \mapsto \bigoplus_{j \in B^n} \alpha_j \pi(a_j).$$

(c$\leftrightarrow$b) Given a free sequence of generators

$$e_i = \bigoplus_{j \in B^n} \alpha_{ij} a_j,$$

one can associate with any permutation $\pi : \{a_j\} \to \{a_j\}$ a new free sequence of generators

$$\pi(e_i) = \bigoplus_{j \in B^n} \alpha_{ij} \pi(a_j).$$

Conversely, if $\{h_i\}$ is a free set of generators with atomic representations

$$h_i = \bigoplus_{j \in B^n} \beta_{ij} a_j,$$

the vectors $\{(\beta_{1j}, \ldots, \beta_{mj})\}_{j \in B^n}$ are distinct, whence

$$h_i = \bigoplus_{j \in B^n} \alpha_{i\phi(j)} a_j$$
$$= \bigoplus_{j \in B^n} \alpha_{ij} a_{\phi^{-1}(j)}$$

for some invertible function $\phi : B^n \to B^n$. $a_j \mapsto a_{\phi^{-1}(j)}$ defines a permutation $\pi : \{a_j\} \to \{a_j\}$.

(d↔b) Given a free sequence of generators

$$e_i = \bigoplus_{j \in B^n} \alpha_{ij} a_j,$$

with each invertible function $\phi : B^n \to B^n$ one can associate a new sequence of generators

$$g_i = \bigoplus_{j \in B^n} \alpha_{i\phi(j)} a_j.$$

Since the column vectors $(\alpha_{i\phi(j)}, \ldots, \alpha_{m\phi(j)})^T$ are distinct, the g_i are free. Conversely, for each sequence of free generators

$$h_i = \bigoplus_{j \in B^n} \beta_{ij} a_j$$

the map $(\beta_{1j}, \ldots, \beta_{nj}) \mapsto (\alpha_{1j}, \ldots, \alpha_{nj})$ defines an invertible function on B^n. ∎

Example 1.17.1

B^4, which consists of the 16 elements whose atomic representations are

(0000)	(0001)	(0010)	(0011)
(0100)	(0101)	(0110)	(0111)
(1000)	(1001)	(1010)	(1011)
(1100)	(1101)	(1110)	(1111),

is generated by the free set $\{(0011), (0101)\}$. B^4 also has 23 other free sets of generators, namely:

$$\begin{array}{ll}
\{(0011)(1001)\} & \{(1001)(0011)\} \\
\{(1001)(1010)\} & \{(0011)(0110)\} \\
\{(0011)(1010)\} & \{(1010)(0011)\} \\
\{(1010)(1001)\} & \{(0110)(0011)\} \\
\{(0101)(1001)\} & \{(1001)(0101)\} \\
\{(1001)(1100)\} & \{(0101)(0011)\} \\
\{(0110)(1010)\} & \{(1010)(0110)\} \\
\{(1010)(1100)\} & \{(0110)(0101)\} \\
\{(0101)(1100)\} & \{(1100)(0101)\} \\
\{(1100)(1001)\} & \{(0101)(0110)\} \\
\{(0110)(1100)\} & \{(1100)(0110)\} \\
\{(1100)(1010)\}. &
\end{array}$$

There are 24 automorphisms on B^4, including the identity, and under composition they form a group that is isomorphic to the symmetric group on four letters. A typical automorphism is

$$\begin{array}{ll}
(0000) \mapsto (0000) & (0001) \mapsto (0100) \\
(0010) \mapsto (1000) & (0011) \mapsto (1100) \\
(0100) \mapsto (0010) & (0101) \mapsto (0110) \\
(0110) \mapsto (1010) & (0111) \mapsto (1110) \\
(1000) \mapsto (0001) & (1001) \mapsto (0101) \\
(1010) \mapsto (1001) & (1011) \mapsto (1101) \\
(1100) \mapsto (0011) & (1101) \mapsto (0111) \\
(1110) \mapsto (1011) & (1111) \mapsto (1111).
\end{array}$$

There are 23 more.

— Chapter Two —

CIRCUIT COMPLEXITY

The model of complexity with which this paper is primarily concerned is *circuit complexity,* also known in the literature as *network complexity* or *combinatorial* complexity ([SAV], [HOP], et al). This model measures the complexity of a finite Boolean function by the minimum number of logical operations ("gates") that are required for an idealized electronic circuit to compute it.

This chapter gives a formal definition of circuit complexity and demonstrates fundamental upper and lower bounds on the complexity of sets of Boolean functions.

Definition 2.1

Let S and T be sets, and let Ω be a **basis** of functions $T \times T \rightarrow T$. Let $G : S \rightarrow T^k$ and $F : S \rightarrow T^m$ have coordinate functions $\{g_i\} : S \rightarrow T$ and $\{f_j\} : S \rightarrow T$. A **combinatorial circuit** computing F from G consists of a set of input vertices, a set of output vertices, a set of gates, and a set of wires. Each input vertex is associated with a function g_i, as are all wires leaving that vertex. Wires are directed edges which lead from input vertices or from the outputs of gates to output vertices or the inputs of other gates. Each gate has two inputs, each of which has at most one incoming wire. Each gate also has one output. Every gate is associated with some operation $\omega \in \Omega$ such that if h_j and h_l are the functions belonging to the wires feeding into the gate, then the function $\omega(h_j, h_l)$ is associated with all of the wires leaving the output of the gate. An output vertex has at most one incoming wire, and is associated with the same function as that wire. As a convention, if T contains 0, a vertex or gate which has no incoming wires at all is assumed to be associated with the function which maps all of

S to 0. Finally, a circuit is assumed to be *acyclic* in the sense that no closed loops are formed by the gates and wires. This condition ensures that the function associated with each wire is well-defined, since its value can be calculated uniquely by tracing through the circuit starting from the input terminals and following the wires toward the output terminals.

The **circuit complexity of F over Ω given G**, $C_\Omega(F \mid G)$, is the minimum number of nodes in a circuit with inputs $\{g_i\}$ and outputs $\{f_j\}$, if such a circuit exists. When no circuit with nodes from Ω computes F from G, then $C_\Omega(F \mid G) = \infty$. When T is the 2-element Boolean algebra B, S is the 2^n-element Boolean algebra B^n, and Ω is the set of all sixteen functions $B^2 \rightarrow B$, $C(F \mid G)$ denotes the **conditional circuit complexity of F given G**. When G is the identity function $B^n \rightarrow B^n$, so that $\{g_i\}$ is the set of functions $\{x_1, x_2, \ldots, x_n\}$, $C(F \mid G)$ is the **circuit complexity of F**, denoted $C(F)$.

Proposition 2.2

Conditional circuit complexity satisfies the "triangle" inequality

$$C_\Omega(F \mid G) \leq C_\Omega(F \mid H) + C_\Omega(H \mid G).$$

Proof:

A combinatorial circuit which computes H from G, connected with a circuit which computes F from H, forms a circuit which computes F from G. ▮

Proposition 2.2.1

If G generates F, then

$$C_\Omega(F \mid G) = \min_{\substack{GG'G=G \\ G':Image(G)\rightarrow S}} C_\Omega(FG').$$

Proof:

$GG'G = G$ if and only if G' has the property that $GG'(x_1, \ldots, x_k) = (x_1, \ldots, x_k)$ for all $x \in \text{Image}(G)$. If G generates F, there exists a circuit $\mathcal{N}_1$ which computes

F given $\{g_1, g_2, \ldots, g_k\}$. Let X be the function computed by $\mathcal{N}_1$ when given inputs $\{x_1, x_2, \ldots, x_k\}$. $XG = F$, and since G is surjective as a function from S to the image of G, there exists at least one function G' with $GG' = I$ on the image of G. For any such G',

$$X = XGG' = FG'$$

on any $x \in \text{Image}(G)$.

Conversely, if for some G', $\mathcal{N}_2$ computes FG' when given inputs $x_1, x_2, \ldots, x_k$, then when $\mathcal{N}_2$ is given inputs $g_1, g_2, \ldots, g_k$ it computes

$$FG'G = XGG'G = XG = F. \blacksquare$$

Note that the fact that G generates F is an essential condition in Theorem 2.2.1. Some procedures for checking if one set of functions generates another are described in chapter 7.

Corollary 2.2.1.1

If G is injective, and if G generates F, then

$$C_\Omega(F \mid G) = \min_{\substack{G^lG = I \\ G^l : Image(G) \to S}} C_\Omega(FG^l).$$

Proof:

Since G is injective, it has at least one left inverse G^l. G' can have only one value for each $x \in \text{Image}(G)$. $G' = G^l$ on $\text{Image}(G)$. $\blacksquare$

Corollary 2.2.1.2

If G is surjective, and if G generates F, then

$$C_\Omega(F \mid G) = C_\Omega(FG^r)$$

for any $G^r : T^k \to S$ such that $GG^r = I$.

(6) Use CIRCUIT COMPLEXITY OVER $\{\vee\}$ to check if there is a circuit of complexity $\leq W$ which computes F. If so, then by Lemma 2.6.2

Proposition 2.7.1

There exists a linear function $F : B^n \rightarrow B^n$ such that

$$C_{\oplus}(F^{(k)}) < k\,C_{\oplus}(F).$$

Proof:

By Corollary 2.5.5, there exists a linear function $F : B^n \rightarrow B^n$ with $C_{\oplus}(F) \sim n^2/2\log n$. Thus, for $k = n$, $kC_{\oplus}(F) \sim \frac{n^3}{2\log n}$. On the other hand, evaluating an $n \times n$ linear function is equivalent to multiplying a column vector of indeterminates by a $n \times n$ matrix of constants; evaluating the same function on n disjoint sets of variables is equivalent to multiplying an $n \times n$ matrix of indeterminates by the same matrix of constants. The recursive Strassen matrix multiplication algorithm [STA] gives a circuit for this computation [Appendix C] that has $O(n^{\log 7})$ additions and no multiplications, so $C_{\oplus}(F^{(k)}) = O(n^{2.808})$. ∎

The fact that savings can accrue when computing identical or similar functions on disjoint sets of variables over the complexity measures $C(F)$ and $C_{\oplus}(F)$, but cannot over the measures $C_{monotone}(F)$ or $C_{\{\vee\}}(F)$, is an important one and suggests that $C_{\oplus}(F)$ models $C(F)$ much more accurately than does $C_{monotone}(F)$. Concrete examples of finite Boolean functions $B^n \rightarrow B^n$ with nonlinear monotone complexity are well known: $O(n\log n)$ [NE3]; $O(n^{3/2})$ [PIP]; $O(n^{2-\epsilon})$ [WEG]. However, other than functions whose proof of complexity is based on diagonalization arguments, no concrete example of a finite Boolean function with either non-linear $C(F)$ or nonlinear $C_{\oplus}(F)$ is known, despite efforts by a variety of researchers over many years. The demonstrations of

— Chapter Three —

TRANSPOSE THEOREM AND APPLICATIONS

This chapter begins with a useful new result – that the number of additions necessary to compute a matrix is, with an adjustment for the dimensions, equal to the number of additions to compute the transpose of the matrix. This result complements but is distinct from that of Hopcroft and Musinski [HOM], which relates the number of nonscalar multiplications required to compute a matrix product with the number required for the transposed product. This theorem provides a good example of a combinatorially simple transformation to the matrix representation for a function which makes a non-trivial change to the function but does not change its complexity.

Theorem 3.1 (Transpose Theorem)

Let F be an m-input, n-output linear Boolean function represented in the usual way by a matrix, i.e.,

$$F(x_1, \ldots, x_m) = (f_1(x_1, \ldots, x_m), \ldots, f_n(x_1, \ldots, x_m))$$

$$f_i(x_1, \ldots, x_m) = \bigoplus_{j=1}^{m} a_{ij}x_j$$

$$[F] = [a_{ij}].$$

Assume $[F]$ has no trivial columns, i.e., for every j there is some i such that $a_{ij} \neq 0$. Let F^T be the n-input, m-output linear function whose matrix is the transpose of $[F]$. Then

$$C_{\oplus}(F^T) = C_{\oplus}(F) + n - m.$$

Proof:

A circuit for F over $\{\oplus\}$ can be associated with a straight-line program; at the i^{th} step of the computation, the program calculates $g_i = g_j \oplus g_k$, where j and k are less than i, and $g_1 = x_1$, $g_2 = x_2$, $\ldots$, $g_m = x_m$. If an optimal program requires c steps, its computation can be simulated by the construction of a lower triangular $(c+m) \times (c+m)$ B-valued matrix. Starting with a $(c+m) \times (c+m)$ identity matrix, each operation $g_s = g_p \oplus g_r$ is simulated by adding the p^{th} and r^{th} rows of the matrix to the s^{th} row. Denote the resulting matrix by $[\mathcal{F}]$. Since the existence of the operation $g_s = g_p \oplus g_r$ implies that $p < s$ and $r < s$, $[\mathcal{F}]$ is lower triangular. In general, $[\mathcal{F}]$ has the form

$$\begin{bmatrix} I & 0 \\ intermediates & garbage \\ F & garbage \end{bmatrix}.$$

Since each new row can be computed with two additions, $C_{\oplus}(\mathcal{F}) \leq 2c$.

Every step in the synthesis of $\mathcal{F}$ corresponds to multiplication on the left by a lower-triangular matrix that is equal to the identity matrix except that in one of its rows there are two extra 1's. The columns of the matrix which contain the extra 1's correspond to the rows of $[\mathcal{F}]$ which have been added to each other. For example, the operation $g_s = g_p \oplus g_r$ corresponds to multiplication on the left by a $(c+m) \times (c+m)$ matrix whose sp^{th} element, sr^{th} element, and each principal diagonal element are 1 and whose other entries are 0. Thus in synthesizing $\mathcal{F}$, the matrix $[\mathcal{F}]$ has been decomposed into a product $[T_{c+m}][T_{c+m-1}]\cdots[T_{m+1}]$, where each $[T_j]$ is a matrix with 0's everywhere, except for 1's on the principal diagonal and two additional 1's on the same row somewhere below the diagonal.

If $[\mathcal{F}] = [T_{c+m}][T_{c+m-1}]\cdots[T_{m+1}]$ then $[\mathcal{F}]^T = ([T_{c+m}][T_{c+m-1}]\cdots[T_{m+1}])^T = [T_{m+1}]^T[T_{m+2}]^T\cdots[T_{c+m}]^T$. But note that the transpose of a matrix of the form $[T_j]^T$ is an upper triangular matrix equal to the identity except that it has two extra 1's in one of its columns. That is, it is a matrix representing the addition of one row to two others. Two such additions can be done by a straight-line program in two steps or by a network with two gates. Thus,

$$[\mathcal{F}]^T = \begin{bmatrix} I & \textit{intermediates} & F^T \\ 0 & \textit{garbage} & \textit{garbage} \end{bmatrix}$$

and $C_{\oplus}(\mathcal{F}^T) \leq 2c$.

Finally, note that since the $(c+m) \times (c+m)$ matrix $[\mathcal{F}]^T$ contains the $m \times n$ submatrix $[F]^T$, a circuit for F^T can be obtained from a circuit for $\mathcal{F}^T$ by ignoring the last c outputs, assigning the variables $x_1, \ldots, x_n$ to the desired inputs, and setting the remaining variables to zero. Consider what happens when the first $c+m-n$ inputs are set to 0 in turn. $[\mathcal{F}]^T$ has $c+m$ columns, and $\mathcal{F}$ depends on all its variables; so the first input vertex is connected directly to at least one $\oplus$ gate. If this input is set to equal to 0 and the gate to which it is connected is removed from the circuit, then the circuit that remains computes a linear function of the remaining inputs. The matrix of this new function is identical to $[\mathcal{F}]^T$ except for the deletion of the leftmost column. Since $\mathcal{F}^T$ is linear, the new function also depends on all of its variables. Continuing to set variables one by one to 0 eventually results in the deletion of $c+m-n$ $\oplus$ gates, leaving a circuit for F^T. Therefore $C_{\oplus}(F^T) \leq C_{\oplus}(\mathcal{F}^T) - (c+m-n) = c+n-m = C_{\oplus}(F) + n - m$.

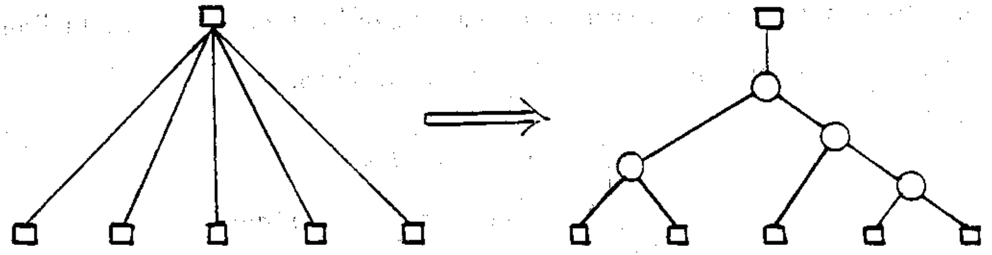
To complete the proof, exchange the roles of F and F^T, and repeat the argument above to obtain $C_{\oplus}(F) \leq C_{\oplus}(F^T) - n + m$. ∎

Corollary 3.1.1

If $[F]$ is square,

$$C_{\oplus}(F) = C_{\oplus}(F^T).$$

Since in the proof of Theorem 3.1 the circuit for F^T really comes from the circuit for F and not from the straight-line program, it is possible to construct a circuit for the transpose function directly from the original circuit without having to use straight-line programs at all.



Fanout nodes

Theorem 3.2

Consider a circuit of $\oplus$-gates which computes a linear Boolean function $F : B^m \to B^n$. The fan-in of each gate is either one or two, while there may be no limit to the fanout of a gate. As a convention, assume that the circuit is redrawn in such a way that no more than three wires join at a single point, i.e., replace the fanout wires of each node of the circuit that has outdegree $k \geq 2$ by a binary tree with $k-1$ vertices. Call the auxiliary vertices, each of which has indegree 1 and outdegree 2, *fanout nodes*. The two descendants of each fanout node are associated with the same function as the node itself, and a fanout node is assumed to have zero cost, i.e., it is not counted in computing $C_\oplus$.

Suppose the following modifications are made to the circuit for F:

(a) change inputs to outputs;

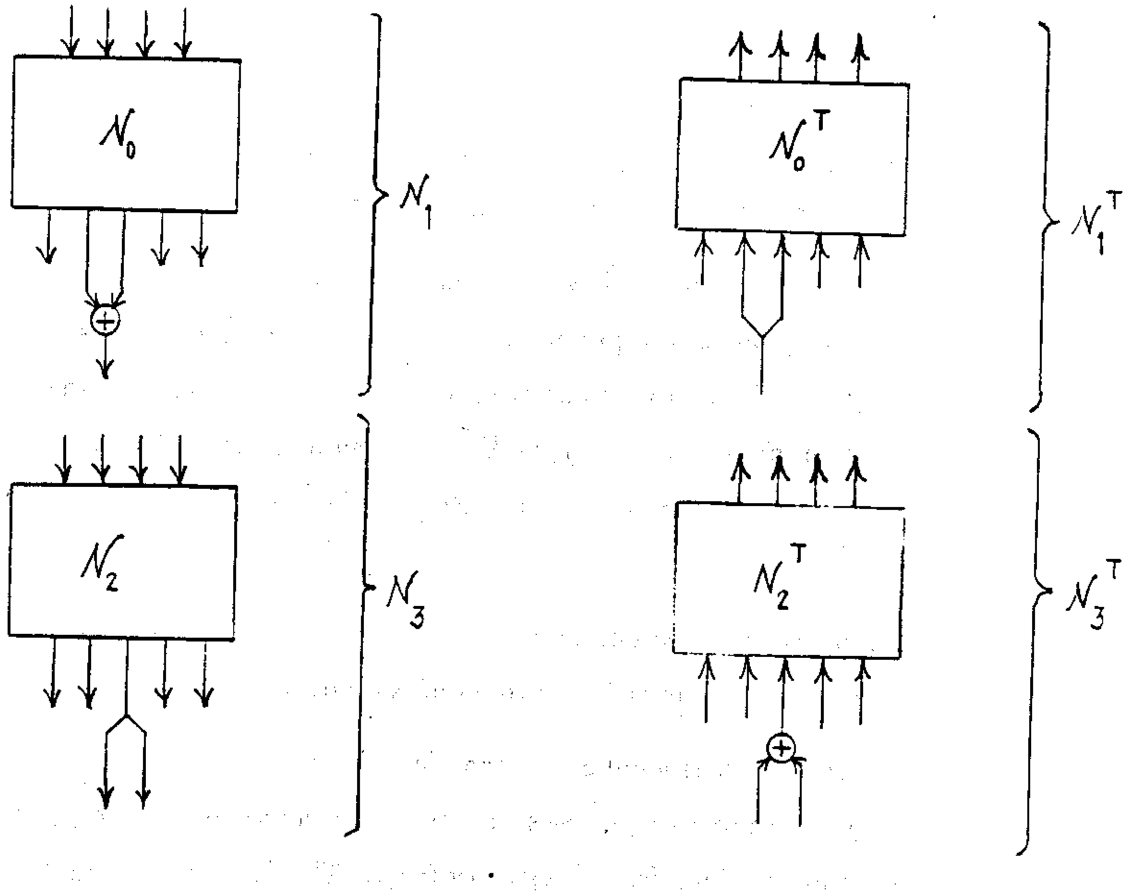
(b) change outputs to inputs;

(c) reverse the direction of all wires;

(d) change $\oplus$ gates to fanout nodes;

(e) change fanout nodes to $\oplus$ gates.

Then the resulting circuit computes F^T and has $C_\oplus(F) + n - m$ gates.



Transpose Circuits

Proof:

The proof proceeds by induction on the size of the circuit for F.

First note that inputs which are not connected to any outputs correspond, in both the transpose matrix and the transpose circuit, to outputs that are not connected to any inputs. Therefore, one may as well assume that every input is connected to some output. Next, observe that, for any integer m, the theorem holds (in a trivial way) for the gateless circuit which computes the identity function on m inputs.

Now suppose the theorem holds for a circuit $\mathcal{N}_0$, two of whose outputs are designated y_1 and y_2. Then the theorem must also hold for the circuit $\mathcal{N}_1$ which is identical to $\mathcal{N}_0$ except that y_1 and y_2 are added to each other at the very bottom of the circuit.

The matrix for $\mathcal{N}_1$ is identical to that for $\mathcal{N}_0$, except that two of the rows of $\mathcal{N}_0$ have been removed and replaced by a single row equal to the vector sum of the two missing rows. The transpose of the matrix for $\mathcal{N}_1$ is identical to that of $\mathcal{N}_0$, except that two columns have been removed and replaced by a single column equal to the vector sum of the two missing columns.

The corresponding network $\mathcal{N}_1{}^T$ has one input where $\mathcal{N}_0{}^T$ has two (corresponding to y_1 and y_2), and whose projection on each of the output vectors is 1 if and only if exactly one of the projections of y_1 and y_2 in $\mathcal{N}_1{}^T$ is 1. That is, the projection of the common input to y_1 and y_2 in $\mathcal{N}_1{}^T$ is the sum of the projections corresponding to y_1 and y_2, and the matrix representing $\mathcal{N}_1{}^T$ is the transpose of the matrix representing $\mathcal{N}_1$.

Next, suppose the theorem holds for a circuit $\mathcal{N}_2$ and a new circuit $\mathcal{N}_3$ is formed by fanning out y_3, one of the outputs of $\mathcal{N}_2$, into two outputs y_4 and y_5.

The matrix for $\mathcal{N}_3$ is identical to that for $\mathcal{N}_2$, except that where $\mathcal{N}_2$'s matrix has a row corresponding to y_3, $\mathcal{N}_3$'s has two rows, corresponding to y_4 and y_5. Each of these rows are identical to $\mathcal{N}_2$'s row for y_3. The function computed by $\mathcal{N}_3^T$ acts on inputs y_4 and y_5 exactly like $\mathcal{N}_2^T$ acts on input y_3, so $\mathcal{N}_3{}^T$'s matrix has two identical columns, each identical to the column vector for y_3 in $\mathcal{N}_2{}^T$. Again, the matrix for $\mathcal{N}_3{}^T$ is the transpose of the matrix for $\mathcal{N}_3$.

Finally, note that since any acyclic network can be ranked, every network observing the fanout conventions previously described can be formed from a smaller network which observes the conventions, either by adding a gate or adding a fanout.

An optimal network for an m-input, n-output function has $C_{\oplus}(F)$ gates. If f is the number of fanout nodes, then the network has $m + C_{\oplus}(F) + 2f$ sources and $n + 2C_{\oplus}(F) + f$ sinks. Since these two quantities must be equal to each other, the number of fanout nodes must equal $C_{\oplus}(F) + n - m$. But under the transformation specified in Theorem 3.2, the number of gates in the circuit for F^T is exactly equal to the number of fanout nodes in the circuit for F. Since converting an arbitrary network to one which observes the fanout conventions does not increase the number of gates, we have $C_{\oplus}(F^T) = C_{\oplus}(F) + n - m$. ∎

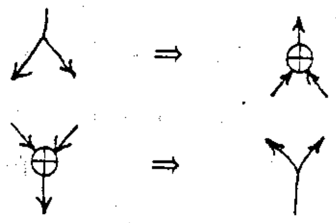
The *adjacency matrix* of a directed graph is the matrix $[a_{ij}]$, where $a_{ij} = 1$ if and only if an edge runs from node i to to node j. Considering a circuit computing

$\{f_1, \ldots, f_m\}$ over $\{\ominus\}$ as a directed graph whose vertices correspond to the gates, the transpose of the adjacency matrix of the circuit is the adjacency matrix of the transpose circuit.

Next, observe that the transpose theorem has a more general application than just to circuits over the basis $\{\ominus\}$ — it holds equally for multiple-output functions over a basis consisting of any single two-input function that is both associative and commutative, including *min*, *max*, $\wedge$, $\vee$, $\cup$, $\cap$, real addition, real multiplication, or convolution. Thus, Theorems 3.1 and 3.2 can be generalized as follows:

Proposition 3.3

Suppose there exists a circuit in the basis $\{+\}$ (ordinary addition of integers) which observes the fanout conventions in Theorem 3.2, has C gates, and which computes a function $F : \mathcal{N}^m \rightarrow \mathcal{N}^n$. F is associated naturally with the matrix of nonnegative integers $F = [a_{ij}]$, where a_{ij} is the number of times the j^{th} input is added to the i^{th} output. If the circuit is transformed by the rules



inputs $\Rightarrow$ *outputs*,

then the transformed circuit has $C+n-m$ gates and computes a linear function whose matrix representation is the transpose of the matrix for F.

Proof:

The inductive proof of Theorem 3.2 applies. Note that a_{ij} is equal to the number of distinct paths from input j to output i along the wires of the circuit. ∎

Let $\circ$ denote a commutative, associative binary operation. The cyclic monoid generated by a single indeterminate x under the operation $\circ$ is a homomorphic image of the cyclic monoid $(\mathcal{N}, +, 0)$. More precisely, $(\langle x\rangle, \circ, 1)$ is either isomorphic to $(\mathcal{N},+,0)$, or it is equal to $\{1,x,x^2,\ldots,x^{r+p-1}\}$ for some integers r and p, where $\{x^r,x^{r+1},\ldots,x^{r+p-1}\}$ is a cyclic group of order p. The quantity r is called the *index*, and p the *period*. If $\circ$ is $\bigoplus$, then $p=2$ and $r=0$.

Denote the homomorphism taking $(\mathcal{N},+,0)$ to $(\langle x\rangle,\circ,1)$ by $\phi_\circ$. The commutative monoid generated by the independent indeterminates $x_1,x_2,\ldots,x_n$ is isomorphic to the direct sum of n copies of $(\langle x\rangle,\circ,1)$, and any function of $x_1,x_2,\ldots,x_n$ computed by means of the $\circ$ operation can be represented by an n-tuple of nonnegative integers under the map

$$(a_1,a_2,\ldots,a_n)\mapsto x_1^{\phi_\circ(a_1)}\circ x_2^{\phi_\circ(a_2)}\circ\ldots\circ x_n^{\phi_\circ(a_n)}.$$

Therefore any m-output function over $\circ$ can be represented either by an $m\times n$ matrix of non-negative integers, which need not be unique, or by an $m\times n$ matrix of elements from $(\langle x\rangle,\circ,1)$.

Theorem 3.4

Suppose that $S=(\langle x\rangle,\circ,1)$ is a cyclic monoid. Let F be a function $S^m\to S^n$ generated by $\circ$ and represented by $n\times m$ matrix of coefficients $[a_{ij}]$. Consider a circuit for F in the basis $\{\circ\}$ that has C gates. Then the circuit created by the transformation in Proposition 3.3 has $C+n-m$ gates and the function computed by the transformed circuit is represented by the matrix $[a_{ij}]^T$.

Proof:

Denote by b_{ij} the number of times that the j^{th} input is $\circ$'d to the i^{th} output in the original circuit, by c_{ij} the number of times that the j^{th} input is $\circ$'d to the i^{th} output in the transformed circuit, and by $\phi_\circ$ the homomorphism taking $(\mathcal{N},+,0)$ to $(\langle x\rangle,\circ,1)$. Since $[a_{ij}]$ represents F, $\phi_\circ(a_{ij})=\phi_\circ(b_{ij})$. By Proposition 3.3, $b_{ij}=c_{ji}$. This gives $\phi_\circ(a_{ij})=\phi_\circ(c_{ji})$, or by reversing indices, $\phi_\circ(a_{ji})=\phi_\circ(c_{ij})$. That is, $[a_{ij}]^T$ represents the function computed by the transpose circuit. ∎

Theorem 3.1 has many applications. It can be directly interpreted as a powerful theorem about addition chains, or it can be used to study both upper and lower bounds on circuit complexity (or straight-line program complexity) whenever the basis for computation consists of a commutative, associative operation. Every circuit in this model has a "transpose," which solves a "transposed" problem; and every lower bound or upper bound corresponds to a lower or an upper bound on a "transposed" circuit.

These applications will be discussed in turn, starting with *addition chains* [KNU][PI2].

Consider the following problem: given a set of indeterminates $\{x_1, x_2, \dots, x_n\}$, generate the following set of monomials

$$\begin{gathered} x_1^{a_{11}} x_2^{a_{12}} \cdots x_n^{a_{1n}}, \\ x_1^{a_{21}} x_2^{a_{22}} \cdots x_n^{a_{2n}}, \\ \vdots \\ x_1^{a_{m1}} x_2^{a_{m2}} \cdots x_n^{a_{mn}}, \end{gathered}$$

using only the operation of multiplication. Formally, this is identical to the problem of synthesizing the integer matrix $[a_{ij}]$ from the set of elementary row vectors $\{(1,0,0,\dots,0),\ (0,1,0,\dots,0),\dots,(0,0,0,\dots,1)\}$ using ordinary vector addition. The number of steps in this computation is called the length of the addition chain for the set of monomials or the matrix $[a_{ij}]$.

For example, there is an addition chain of length 9 which computes the set of monomials $\{x_1^2 x_2^2 x_3, x_1^{14} x_2^{12} x_3^7 x_4^2, x_1^4 x_2^4 x_3^2 x_4\}$, namely

$$\begin{array}{ll} x_1^2 & (1) \\ x_1^2 x_3 & (2) \\ x_2^2 & (3) \\ x_1^2 x_2^2 x_3 & (4) \\ x_1^4 x_2^4 x_3^2 & (5) \\ x_1^4 x_2^4 x_3^2 x_4 & (6) \\ x_1^8 x_2^8 x_3^4 x_4^2 & (7) \\ x_1^{12} x_2^{12} x_3^6 x_4^2 & (8) \\ x_1^{14} x_2^{12} x_3^7 x_4^2 & (9). \end{array}$$

Since every addition chain can be implemented as a circuit in the basis $\{+\}$ and every circuit in $\{+\}$ corresponds to an addition chain, the transpose theorem can be applied:

Corollary 3.5

If a set of n monomials in $\{x_1, \ldots, x_m\}$ with exponents $[a_{ij}]$ can be generated by an addition chain of length l, then the set of m monomials in $\{x_1, \ldots, x_n\}$ with exponents $[a_{ij}]^T$ can be generated by an addition chain of length $l + n - m$.

For example, according to the corollary, there is some addition chain of length $8 = 9 + 3 - 4$ which computes the set $\{x_1^2 x_2^{14} x_3^4, x_1^2 x_2^{12} x_3^4, x_1 x_2^7 x_3^2, x_2^2 x_3\}$.

The transpose theorem does more than simply show the existence of an addition chain for the transpose set of monomials — it also shows how to construct this chain. A direct application of the theorem yields

$$\begin{array}{ll}
x_1 & (1) \\
x_2 & (2) \\
x_3 & (3) \\
x_1 x_2 & (4) \\
x_2 x_3 & (5) \\
x_2^2 x_3 & (6) \\
x_2^3 x_3 & (7) \\
x_2^6 x_3^2 & (8) \\
x_1 x_2^7 x_3^2 & (9) \\
x_1^2 x_2^{14} x_3^4 & (10) \\
x_1^2 x_2^{12} x_3^4 & (11).
\end{array}$$

In 1981, J. Olivos published a lengthy proof of Corollary 3.5 for the special case $m = 1$. Using an example from his paper [OLI], the length of the shortest addition chain computing $\{x^7, x^{15}, x^{23}\}$ is 7, e.g.,

x^2	(1)
x^3	(2)
x^4	(3)
x^7	(4)
x^8	(5)
x^{15}	(6)
x^{23}	(7).

From this, one can conclude that the length of the shortest addition chain for the monomial $x_1^{23}x_2^{15}x_3^7$ is $7+3-1=9$. Again, Theorem 3.1 [or 3.2] gives the construction

x_1x_2	(1)
$x_1x_2x_3$	(2)
$x_1^2x_2$	(3)
$x_1^3x_2^2x_3$	(4)
$x_1^5x_2^3x_3$	(5)
$x_1^6x_2^4x_3^2$	(6)
$x_1^{11}x_2^7x_3^3$	(7)
$x_1^{12}x_2^8x_3^4$	(8)
$x_1^{23}x_2^{15}x_3^7$	(9)

Because the transpose theorem holds for the operations $\vee$ and $\cup$, as well as for $+$, Corollary 3.5 holds with equal validity for addition chains of Boolean variables, whose monomials obey the multiplication law

$$(x_1^{\alpha_1}\cdots x_n^{\alpha_n})(x_1^{\beta_1}\cdots x_n^{\beta_n}) = x_1^{\alpha_1\vee\beta_1}\cdots x_n^{\alpha_n\vee\beta_n}.$$

Corollary 3.5 can be also be generalized in a slightly different way. Define an **addition – scalar multiplication chain** to be a chain of operations on monomials in which the multiplication of two monomials has cost c and the exponentiation of a monomial to the power λ has cost $k(\lambda)$. Then the following result holds, regardless of the relative costs c and k.

Corollary 3.6

If a set of n monomials in $\{x_1, \ldots, x_m\}$ with exponents $[a_{ij}]$ can be generated by an addition – scalar multiplication chain with total cost l, then the set of m monomials in $\{x_1, \ldots, x_n\}$ with exponents $[a_{ij}]^T$ can be generated by a chain with cost $l + c(n - m)$.

Proof:

The computations of the chain can be thought of as being performed by a circuit whose inputs are the elementary vectors $(1, 0, 0, \ldots, 0)$, $(0, 1, 0, \ldots, 0)$, $\ldots$, $(0, 0, 0, \ldots, 1)$ and whose gates compute vector addition and scalar multiplication. If $[a_{ij}]$ is the matrix associated with such a circuit then a_{ij} equals the sum over all directed paths from input j to output i of the product of the scalars encountered along the path. Reversing the direction of all wires in the network computes a function represented by a matrix whose ijth element is the sum over all directed paths from input i to output j of the product of scalars encountered along the path. That is, the matrix computed by the 'transposed' circuit is $[a_{ji}] = [a_{ij}]^T$. If the original circuit has C additions and S scalar multiplications then the transposed circuit has $C + n - m$ additions and S multiplications. Each multiplication by λ in the original circuit corresponds to one multiplication by λ in the transposed circuit. ∎

An **addition-subtraction chain** is a series of operations on monomials in which two monomials can be multiplied or divided at unit cost.

Theorem 3.7

Let $l_\pm([a_{ij}])$ denote the length of the shortest addition-subtraction chain for the set of monomials with coefficients $[a_{ij}]$. Then

$$l_\pm([a_{ij}]^T) \leq l_\pm([a_{ij}]) + n - m + \min\{n, m\}.$$

Proof:

An addition-subtraction chain can be viewed as a circuit whose gates compute addition or subtraction. Each subtraction operation can in turn be viewed as an addition operation in which one of the two inputs has been multiplied by the scalar -1. By Corollary 3.6, the set of monomials with coefficients $[a_{ij}]^T$ can be computed with $n - m$ additions plus the number of additions necessary to synthesize the set of monomials with coefficients $[a_{ij}]$ plus a certain number of multiplications by -1.

By repeated use of the identities

$$\begin{aligned} h_i(-1)(-1) &= h_i \\ h_j + h_k(-1) &= h_j - h_k \\ h_j - h_k(-1) &= h_j + h_k \\ h_j(-1) - h_k &= (-1)(h_j + h_k) \end{aligned}$$

every multiplication by (-1) can be either be absorbed into a $(+)$ or $(-)$ gate or be pushed lower in the circuit. After a finite number of applications of these identities, all scalar multiplications which have not been absorbed have arrived at the bottom of the circuit. Since the circuit for $[a_{ij}]^T$ has only m outputs, at most m such operations can remain unabsorbed. If we adopt the convention that a subtraction gate whose positive input is missing computes multiplication by (-1), then m subtraction gates suffice to perform the required multiplications.

Similarly, using the identities

$$\begin{aligned} (-1)(-1)h_i &= h_i \\ (-1)(h_j + h_k) &= h_j(-1) + h_k(-1) \\ (-1)(h_j - h_k) &= h_k - h_j, \end{aligned}$$

each scalar multiplication can be moved higher in the circuit until it is either absorbed into a gate or until it reaches the very top. Since the circuit for $[a_{ij}]^T$ has only n inputs, at most n multiplications remain when each multiplication has been moved as far up as it can go. n subtraction gates suffice to compute these n multiplications. ▮

Next, the transpose theorem is used to analyze the complexity of some sets of linear functions.

Proposition 3.8.1

Let $F : B^n \rightarrow B^3$ have coordinate functions $f_i = \bigoplus_{j=1}^{n} a_{ij}x_i$, and assume that F depends on all its variables. Then

$$n - 3 \leq C_{\oplus}(F) \leq n + 1.$$

Proof:

F^T is a linear function $B^3 \rightarrow B^n$, none of whose coordinate functions is constantly zero. All seven non-zero linear functions $B^3 \rightarrow B$ can be computed from $\{x_1, x_2, x_3\}$ with just four gates, so $C_{\oplus}(F^T) \leq 4$. By Theorem 3.1, $C_{\oplus}(F) \leq n + 1$. On the other hand, $0 \leq C_{\oplus}(F^T)$, whence $n - 3 \leq C_{\oplus}(F)$. ∎

Examples of functions attaining these upper and lower bounds follow:

Examples 3.8.1.1

$n = 4$ and $C_{\oplus}(f) = 5$:

$$f_1 = x_1 \oplus x_2 \oplus x_4$$

$$f_2 = x_1 \oplus x_3 \oplus x_4$$

$$f_3 = x_2 \oplus x_3 \oplus x_4.$$

$n = 6$ and $C_{\oplus}(f) = 3$:

$$f_1 = x_1 \oplus x_2$$

$$f_2 = x_3 \oplus x_4$$

$$f_3 = x_5 \oplus x_6.$$

This result can be generalized to functions with an arbitrary number of outputs.

Theorem 3.8.2

Let $F : B^n \to B^m$ have coordinate functions $f_i = \bigoplus_{j=1}^{n} a_{ij}x_j$, and suppose that F depends on all its variables. Then

$$n - m \leq C_{\oplus}(F) \leq n + 2^m - 2m - 1.$$

Proof:

F^T is a linear function $B^m \to B^n$, none of whose coordinate functions are 0. By Lemma 2.5.2, $C_{\oplus}(F^T) \leq 2^m - m - 1$. By Theorem 3.1, $C_{\oplus}(F) \leq n + 2^m - 2m - 1$. On the other hand, $0 \leq C_{\oplus}(F^T)$, so by Theorem 3.1, $n - m \leq C_{\oplus}(F)$. ∎

Therefore the hardest linear function $B^n \to B^m$ requires just one gate per input in the limit as $n \to \infty$ for fixed m.

Definition 3.9

Let $m \geq 2$. $\mathcal{F}ull_m : B^{2^m-m-1} \to B^m$ is the linear function whose matrix contains as column vectors all possible m-tuples with weight ≥ 2, arranged in lexicographic order.

Example 3.9.1

$\mathcal{F}ull_3$:

$$f_1 = x_2 \oplus x_3 \oplus x_4$$
$$f_2 = x_1 \oplus x_3 \oplus x_4$$
$$f_3 = x_1 \oplus x_2 \oplus x_4$$

Example 3.9.2

$\mathcal{F}ull_4$:

$$f_1 = x_5 \oplus x_6 \oplus x_7 \oplus x_8 \oplus x_9 \oplus x_{10} \oplus x_{11}$$
$$f_2 = x_2 \oplus x_3 \oplus x_4 \oplus x_8 \oplus x_9 \oplus x_{10} \oplus x_{11}$$
$$f_3 = x_1 \oplus x_3 \oplus x_4 \oplus x_6 \oplus x_7 \oplus x_{10} \oplus x_{11}$$
$$f_4 = x_1 \oplus x_2 \oplus x_4 \oplus x_5 \oplus x_7 \oplus x_9 \oplus x_{11}$$

Theorem 3.10

Suppose $m > 2$ and $n \geq 2^m - m - 1$, and let G be a linear function $B^n \to B^m$. Then

$$C_\oplus(G) \leq C_\oplus(\mathcal{F}ull_m) + n - 2^m + m + 1.$$

Proof:

By Lemma 2.5.2, $C_\oplus(\mathcal{F}ull_m^T) = 2^m - m - 1$. A circuit for G^T can be synthesized from a circuit for $\mathcal{F}ull_m^T$ by ignoring coordinate functions which are not coordinate functions of G^T, and duplicating (fanning out) coordinate functions which appear more than once in G^T. Applying Theorem 3.1 gives the result. ∎

Corollary 3.10.1

$\mathcal{F}ull_m$ is maximally hard with respect to $C_\oplus$ among all Boolean functions $B^{2^m-m-1} \to B^m$ and has complexity $2^{m+1} - 3m - 2$.

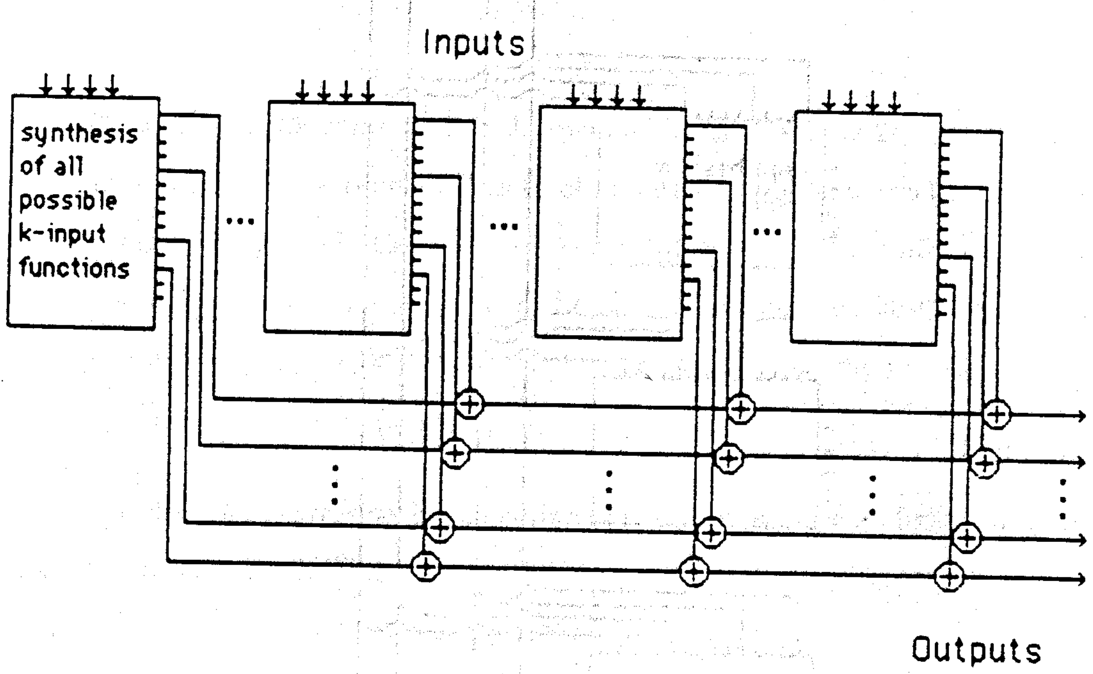
Corollary 3.10.2

$$\max_{F:B^n \to B^{\log n}} C_\oplus(F) = 2n - 2\log n - 1.$$

This maximum is attained if and only if $[F]$ contains $[\mathcal{F}ull_{\log n}]$ as a submatrix and $[F]$ has no columns consisting entirely of zeros. Note that the upper bound in Theorem 3.8.2 is tight for $m \leq \log n$, i.e., there exist linear functions $F : B^n \to B^m$ with $C_\oplus(F)$ equal to $n + 2^m - 2m - 1$. The lower bound in Theorem 3.8.2 is tight for all n and m.

Now, let us look at how the transpose construction in Theorem 3.2 transforms a combinatorial algorithm, using as an example the Four Russians' procedure [ADK] from the proof of Lemma 2.5.3.

Recall that transposing a circuit takes additions (gates) to fanouts and fanouts to additions (gates). The Four Russians' algorithm subdivides the set of input variables,



Four Russians' Algorithm

computes all possible functions within each subset of the inputs, and then combines the resulting functions to get the final result. The transpose of a circuit implementing the Four Russians' algorithm would:

(a) have one input for each output of the original circuit;

(b) have one output for each input in the original circuit;

(c) fanout k copies of each input;

(d) using each complete set of inputs independently, synthesize k different m-input, n-output functions.

Such a procedure could be called "band synthesis" since it computes horizontal bands of the matrix independently.

This fundamental result has several applications. For example:

Theorem 3.14

If $F : B^n \to B$ is a quadratic monotone function, then

$$C(F) \leq C_{monotone}(F) \leq \frac{n^2}{4\log n} + o\left(\frac{n^2}{\log n}\right).$$

Proof:

Every quadratic monotone function has the form

$$\bigvee_{1 \leq i < j \leq n} a_{ij} x_i x_j.$$

If it can be shown that the "upper triangular" set of disjunctions

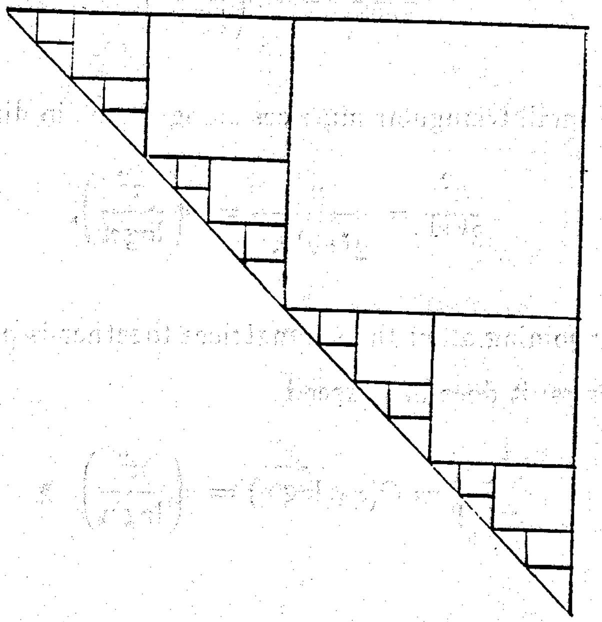
$$\left\{ \bigvee_{i < j \leq n} a_{ij} x_j \right\}_{1 \leq i < n}$$

can be synthesized with $\sim (n^2/4\log n) \vee$ operations, then the result will follow immediately, since

$$\bigvee_{1 \leq i < j \leq n} a_{ij} x_i x_j = \bigvee_{i=1}^{n} x_i \left(\bigvee_{j=i+1}^{n} a_{ij} x_j \right).$$

Since by Theorem 3.2 the transpose theorem holds for an arbitrary commutative, associative operation, the construction in Theorem 3.12 remains valid over the basis $\{\vee\}$. Therefore any $n \times n$ B-valued matrix can be synthesized from the identity matrix with $\sim n^2/4\log n$ disjunctions.

Now consider the scheme for subdividing an upper triangular matrix that is shown in the figure. The smallest square submatrix has dimensions $n/2^k \times n/2^k$, where $k \sim \sqrt{\log n}$, and n is taken sufficiently large so that $C_{\{\vee\}}$ of any $n/2^k$-input, $n/2^k$-output disjunction is less than



Subdivision Scheme for the Matrix Associated with a Quadratic Monotone Function

There are 2^{i-1} square submatrices with dimension $(n/2^i) \times (n/2^i)$ plus or minus 1, so the disjunctive complexity of the set of square submatrices does not exceed

$$\begin{aligned}
&C_{\{\vee\}}\left(\frac{n}{2}\right) + 2C_{\{\vee\}}\left(\frac{n}{4}\right) + \cdots + 2^{k-1}C_{\{\vee\}}\left(\frac{n}{2^k}\right) + o\left(\frac{n^2}{\log n}\right) \\
&\leq \frac{(1+\epsilon)n^2}{8(\log n - 1)} + \frac{(1+\epsilon)n^2}{16(\log n - 2)} + \cdots + \frac{(1+\epsilon)n^2}{2^{k+2}(\log n - k)} + o\left(\frac{n^2}{\log n}\right) \\
&\leq \frac{(1+\epsilon)n^2}{4(\log n - k)}\left(\sum_{i=2}^{k} 2^{-k}\right) + o\left(\frac{n^2}{\log n}\right) \\
&\leq \frac{(1+\epsilon)n^2}{4(\log n - k)} + o\left(\frac{n^2}{\log n}\right) \\
&\leq \left(\frac{1}{\log n} + \frac{k}{\log^2 n - k\log n}\right)\frac{(1+\epsilon)n^2}{4} + o\left(\frac{n^2}{\log n}\right).
\end{aligned}$$

Since $k \sim \sqrt{\log n}$, this last quantity is bounded above by

$$\frac{(1+\epsilon)n^2}{4\log n} + o\left(\frac{n^2}{\log n}\right).$$

The cost of the small triangular matrices along the main diagonal does not exceed

$$\frac{n^2}{2^{k+1}} = \frac{n^2}{2^{1+\sqrt{\log n}}} = o\left(\frac{n^2}{\log n}\right),$$

and the cost for joining all of the submatrices together is proportional to the sum of their perimeters: it does not exceed

$$\sum_{i=1}^{k} \frac{n}{2} = O(n\sqrt{\log n}) = o\left(\frac{n^2}{\log n}\right). \blacksquare$$

This construction improves by a factor of 16 a result by P. Bloniarz [BLO], and gives an asymptotically tight upper bound. Since there are $2^{n(n-1)/2}$ monotone quadratic functions, Lemma 2.2.2 implies that for any ϵ, all but a vanishingly small proportion of these functions have circuit complexity in excess of $(1-\epsilon)n^2/4\log n$. Thus, if QM_n is the class of quadratic monotone functions $B^n \to B$, then

$$\max_{F \in QM_n} C(F) \sim \max_{F \in QM_n} C_{monotone}(F) \sim \frac{n^2}{4\log n}.$$

Corollary 3.13.1

If a quadratic Boolean function is a function $B^n \to B$ with the form

$$\bigoplus_{1\le i<j\le n} a_{ij}x_ix_j,$$

then

$$\max_{\substack{F:B^n\to B \\ F \text{ quadratic}}} C(F) \sim \frac{n^2}{4\log n}.$$

Theorem 3.12 gives the asymptotic complexity of the hardest $n \times n$ matrix. E. Nechiporuk [NEC] and N. Pippenger [PI4] have generalized this result to rectangular matrices whose dimensions are not exponentially lopsided, producing the following result.

Theorem 3.14

Let $F : B^n \to B^m$ be a linear Boolean function, with $\log m = o(n)$ and $\log n = o(m)$. Then

$$C_{\oplus}(F) \leq \frac{nm}{\log nm} + o\left(\frac{nm}{\log nm}\right).$$

This result has applications similar to those of Theorem 3.12. For example:

Corollary 3.14.1

Consider the class of **monotone bilinear** functions $B^{n+m} \to B$, i.e., the set of functions of the form

$$\bigvee_{\substack{1 \leq i \leq m \\ 1 \leq j \leq n}} a_{ij} x_i x_j.$$

If $\log m = o(n)$ and $\log n = o(m)$, then

$$\max_{\substack{F:B^{n+m}\to B \\ \text{monotone bilinear}}} C(F) \sim \max_{\substack{F:B^{n+m}\to B \\ \text{monotone bilinear}}} C_{monotone}(F) \sim \frac{mn}{\log mn}.$$

Corollary 3.14.2

Consider the class of **bilinear** Boolean functions $B^{n+m} \to B$, i.e., the class of functions with the form

$$\bigoplus_{\substack{1 \leq i \leq m \\ 1 \leq j \leq n}} a_{ij} x_i x_j.$$

If $\log m = o(n)$ and $\log n = o(m)$, then

$$\max_{\substack{F:B^{m+n}\to B \\ F \text{ bilinear}}} C(F) \sim \frac{mn}{\log mn}.$$

What about lopsided matrices? All of these matrices have complexity bounded by the order of their longest dimension. The transpose theorem provides a useful tool for accurately estimating this complexity.

For example, consider the complexity over $\{\vee\}$ of the hardest function $B^n \to B^m$, where m is approximately $\log n$. Each such function can be represented by an $m \times n$ B-valued matrix. By Corollary 3.10.2,

$$\max_{F:B^n \to B^{\log n}} C_{\{\vee\}}(F) = 2n + O(\log n).$$

Slightly reducing the number of outputs reduces the complexity of the hardest function. By Theorem 3.8.2,

$$\max_{F:B^n \to B^{\log n - \log\log n}} C_{\{\vee\}}(F) \leq n + \frac{n}{\log n} - 2\log n + 2\log\log n - 1.$$

On the other hand, if F has coordinate functions $f_i(x_1, \ldots, x_n) = \bigvee_{j=1}^{n} a_{ij}x_j$ then $[a_{ij}]^T$ has dimensions $n \times (\log n - \log\log n)$, so its set of row vectors can comprise at most $\frac{n}{\log n} - \log n + \log\log n - 1$ distinct vectors with weight greater than 1. Consequently, if $G : B^{\log n - \log\log n} \to B^n$ has coordinate functions $g_j(x_1, \ldots, x_{\log n - \log\log n}) = \bigvee_{i=1}^{\log n - \log\log n} a_{ij}x_i$ then $C_{\{\vee\}}(G) \geq \frac{n}{\log n}$. By the transpose theorem,

$$\max_{F:B^n \to B^{\log n - \log\log n}} C_{\{\vee\}}(F) \geq n + \frac{n}{\log n} - 2\log n + 2\log\log n - 1.$$

That is,

$$\max_{F:B^n \to B^{\log n - \log\log n}} C_{\{\vee\}}(F) = n + \frac{n}{\log n} + O(\log n).$$

Slightly increasing the number of inputs increases the complexity of the hardest function, but not by nearly as much as decreasing the number of inputs reduced it. $[F]^T$ has dimensions $n \times (\log n + \log\log n)$. Dividing $[F]^T$ into two vertical strips of dimension $n \times \frac{1}{2}(\log n + \log\log n)$, synthesizing all possible $\frac{1}{2}(\log n + \log\log n)$-tuples, and joining them together with n disjunctions computes the n rows of $[F]^T$ with no more than $n + (\sqrt{2} + \frac{1}{\sqrt{2}})\sqrt{n\log n}$ operations. By the transpose theorem, F can be computed with no more than $2n + (\sqrt{2} + \frac{1}{\sqrt{2}})\sqrt{n\log n} - \log n - \log\log n$ $\vee$ gates. This gives the following upper bound on the complexity of the hardest function:

$$\max_{F:B^n \to B^{\log n + \log\log n}} C_{\{\vee\}}(F) \leq 2n + (2.13)\sqrt{n\log n} + O(\log n).$$

On the other hand, the next argument provides a lower bound on the complexity of the hardest function with these dimensions.

Proposition 3.15

For arbitrarily large n, there exists a function $F : B^n \to B^{\log n + \log\log n}$ with

$$C_{\{\vee\}}(F) \geq 2n + \sqrt{2n} - O(\log n).$$

Proof:

Let k be an even integer and let $n = 2^{2^k}$ and $m = 2^k + k$. By Stirling's formula

$$\binom{m}{\frac{m}{2}} > e^{\frac{-1}{12m}}(2\pi m)^{-\frac{1}{2}}\, 2^m,$$

so the number of Boolean m-tuples whose weight is $m/2$ exceeds n. Let $G : B^m \to B^n$ be some function whose coordinate functions are distinct and which are given by

$$g_i(x_1, \ldots, x_m) = \bigvee_{j=1}^{m} a_{ij}x_j,$$

where $(a_{i1}, \ldots, a_{im})$ has weight $m/2$ for every i. Consider some optimal circuit for G. Let $START$ be the set of functions $\{x_1, \ldots, x_m\}$, let $FINISH$ be the set of coordinate functions $\{g_1, \ldots, g_n\}$, and let $INTER$ be the set of functions which are produced by the gates of the circuit but which are not in $FINISH$. Each function in $FINISH$ is the output of some gate in the circuit; let us count the number of such functions according to whether the inputs of the gate which produced it are in $START$, $INTER$, or $FINISH$.

(a) The join of two elements from $START$ has weight 1 or 2, so such an element cannot be in $FINISH$.

(b) At most $m|INTER|$ pairs of elements from $START$ and $INTER$ can have a join which is in $FINISH$.

(c) The join of an element x_j from $START$ and an element f_i from $FINISH$ cannot be in $FINISH$ unless $x_j \leq f_i$ in the Boolean algebra of functions $B^n \to B$. However in this case, $x_j \vee f_i = f_i$, so the gate produces nothing new and a circuit which contains it cannot be optimal.

(d) At most $|INTER|(|INTER|-1)/2$ pairs of elements from $INTER$ can have a join in $FINISH$.

(e) Since all the functions in $FINISH$ have the same weight, the join of a function from $INTER$ and a function f_i from $FINISH$ cannot be in $FINISH$ unless the join of the intermediate function with f_i is f_i itself. A circuit containing a gate which performs such a computation cannot be optimal.

(f) The join of two elements from $FINISH$ cannot be in $FINISH$ unless the two elements are identical.

Thus every function in $FINISH$ is the join of a function from $INTER$ with one from $INTER \cup START$. Since the n functions in $FINISH$ are all distinct, $m|INTER| + |INTER|(|INTER|-1)/2 > n$. Solving for $|INTER|$ in terms of m and n and taking the limit as $n \to \infty$ while $m = O(\log n)$ gives $|INTER| > \sqrt{2n} - O(\log n)$. That is, at least $\sqrt{2n} - O(\log n)$ intermediate functions must be computed by any circuit which computes G. Therefore $C_{\{\vee\}}(G) \geq n + \sqrt{2n} - O(\log n)$. Let $F : B^n \to B^m$ have coordinate functions $f_j(x_1, \ldots, x_n) = \bigvee_{i=1}^{n} a_{ij}x_i$. By the transpose theorem, $C_{\{\vee\}}(F) \geq 2n + \sqrt{2n} - O(\log n)$. ∎

To give one final example of how the transpose theorem can be applied to circuit synthesis, consider the complexity over $\{\oplus\}$ of a set of linear functions with the property that each input affects only a limited number of outputs. If F has n inputs and m outputs, and if each input appears in only k of the outputs, then F^T has m inputs and n outputs, and each row of $[F]^T$ has weight not exceeding k. Dividing $[F]^T$ into s vertical bands of width m/s, synthesizing all possible functions of weight $k-s+1$ within each band, and patching them together with the functions $x_1, \ldots, x_m$ gives a construction for $[F]^T$ with no more than $(s-1)n + s\sum_{i=2}^{k-s+1}\binom{m/s}{i}$ operations. The transpose theorem gives a construction for $[F]$ with $C_{\oplus}(F) \leq sn - m + s\sum_{i=2}^{k-s+1}\binom{m/s}{i}$.

For example, if F is an n-input, m-output linear Boolean function in which each input is connected to no more than k outputs, each row of $[F]^T$ has weight $\leq k$, whence $C_{\oplus}(F) \leq n - m + \sum_{i=2}^{k}\binom{m}{i}$. Therefore any 1024-input, 10-output Boolean function over $\{\vee\}$, each of whose inputs is connected to at most 5 outputs, can be computed by a circuit with no more than 1641 two-input $\vee$ gates. By contrast, 5110 gates would be required if each of the 10 output functions were synthesized independently. Similarly, $C_{\oplus}(F) \leq 2n - m + 2\sum_{i=2}^{k-1}\binom{m/2}{i}$, so any 1024-input, 20-output Boolean function over $\{\vee\}$, each of whose inputs is connected to no more than 5 outputs, can be computed with no more than 2778 gates.

The worst-case complexity of Boolean functions over various bases is summarized in the following chart:

	$\{B^2 \rightarrow B\}$	$\{\oplus\}$	$\{\vee\}$	$\{\vee, \wedge\}$
functions : $B^n \rightarrow B$	2^{2^n}	2^n	2^n	$2^{2^n\sqrt{2/\pi n}\,U\left(\frac{\log n}{n}\right)}$
functions : $B^n \rightarrow B^n$	2^{n2^n}	2^{n^2}	2^{n^2}	$2^{2^n\sqrt{2n/\pi}\,U\left(\frac{\log n}{n}\right)}$
max complexity : $B^n \rightarrow B$	$\sim 2^n/n$	$n-1$	$n-1$	$\sim\left(\sqrt{\frac{2}{\pi n}}\right)\frac{2^n}{n}$?
max complexity : $B^n \rightarrow B^n$	$\sim 2^n$	$\sim \frac{n^2}{2\log n}$	$\sim \frac{n^2}{2\log n}$	$\sim\left(\sqrt{\frac{2n}{\pi}}\right)\frac{2^n}{\log n2^n}$?
$C_\Omega(F \times G) \leq C_\Omega(F) + C_\Omega(G)$	yes	yes	no	no

Chapter Four

MEMORYLESS CIRCUITS

The complexity of linear Boolean functions with respect to the size of arbitrary linear circuits was discussed in Chapters 2 and 3. Next, the computation of n-input, n-output linear functions in a slightly more restricted model is considered, and an asymptotically tight formula is developed for the complexity of the hardest functions in the new model.

Since the graph associated with a circuit is acyclic, it can be *ranked*, i.e., a mapping N made from the natural numbers to the gates of the circuit such that for any gate g, the gates g_1 and g_2 which feed the inputs of g satisfy $N(g_1) < N(g)$ and $N(g_2) < N(g)$. If $N(g_i) > N(g_j)$, then g_i is **above** g_j, and if $N(g_i) < N(g_j)$, then g_i is **below** g_j. By convention, input nodes rank above all gates, and output nodes rank below all gates.

Definitions 4.1

The **width of a circuit at a gate** g is the minimum, over all rankings, of the number of nodes below g which have inputs above g. The **width of a circuit** is the maximum width at any gate. We call an n-input, n-output circuit of width n **memoryless**, since it corresponds to a random access machine program which uses no more memory than that required to hold the input data.

Proposition 4.2

If $F : B^n \rightarrow B^n$ is invertible and is computed by a memoryless circuit, then F is affine.

Proof:

Let N be a ranking of gates in an optimal circuit for F, and let g be the highest gate. Consider the width of the circuit at g. Each of the $n-2$ inputs to the portion of the circuit below g which are not also inputs to g must ultimately be connected to some output node, for otherwise F would not be invertible. If the circuit is optimal, the output of g must also ultimately be connected to some output node. Finally, since F is invertible, exactly one of the two inputs to g must be connected to some output without passing through g. Since either $(g(x_1,x_2),x_1)$ or $(g(x_1,x_2),x_2)$ is an invertible function $B^2 \to B^2$, $g(x_1,x_2)$ must equal $x_1 \bigoplus x_2$, $x_1 \bigoplus x_2 \bigoplus 1$, $x_1 \bigoplus 1$, or $x_2 \bigoplus 1$. ∎

A circuit of width n for an n-input, n-output invertible Boolean function F can be described as a sequence of operations of the form

$$x_j := x_j \bigoplus a x_i \bigoplus b,$$

where $i \neq j$ and $a, b \in \{0,1\}$.

Definition 4.2.1

Let F be an invertible function $B^n \to B^n$. $\mathcal{M}_\oplus(F)$ is the minimum number of gates to compute F with a circuit of width n.

Lemma 4.2.2

Suppose that F is an affine function $B^n \to B^n$ with coordinate functions $f_i(x_1,\ldots,x_n) = \bigoplus_{j=1}^n a_{ij}x_j \bigoplus b_i$. Then $C \leq \mathcal{M}_\oplus(F) \leq C+n$, where C is the number of operations of the form $(x_j := x_j \bigoplus x_i)$ necessary to synthesize the linear functions $f_i(x_1,\ldots,x_n) - f_i(0,\ldots,0) = \bigoplus_{j=1}^n a_{ij}x_j$. If F is linear, $\mathcal{M}_\oplus(F)$ is equal to the number of operations of the form $(x_j := x_j \bigoplus x_i)$ necessary to synthesize F.

Proof:

If F is computed by a sequence of operations of the form $(x_j := x_j \bigoplus a x_i \bigoplus b)$ then deleting the constant term from each operation gives a sequence of linear operations

$(x_j := x_j \oplus ax_i)$ which computes $F(x_1, \ldots, x_n) - F(0, \ldots, 0)$. An operation of the form $(x_j := x_j \oplus 0 \wedge x_j)$ has no cost. On the other hand, $f_i(0, \ldots, 0)$ is equal to either 0 or 1, so $\{f_i(x_1, \ldots, x_n)\}$ can be computed from $\{f_i(x_1, \ldots, x_n) - f_i(0, \ldots, 0)\}$ with no more than n operations of the form $(x_j := x_j \oplus 1)$. ∎

It is assumed in this model that permutation of output functions can be done for free; if this is not permitted, denote the resulting circuit complexity by $M'_{\oplus}$. Two output functions can always be transposed with three successive additions in the following manner:

$$\begin{array}{cc}
a & b \\
\\
a \oplus b & b \\
\\
a \oplus b & a \\
\\
b & a
\end{array}$$

Therefore, $M'_{\oplus}(F) \leq M_{\oplus}(F) + 3(n-1)$. In general, $C_{\oplus}(F) \leq M_{\oplus}(F) \leq M'_{\oplus}(F)$, and it is not too difficult to find functions for which these inequalities are strict. In particular, the linear function $B^4 \rightarrow B^4$ defined by the matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$

is easier with auxiliary memory than without it. In the former case, one can compute the function with 3 gates by computing the intermediate function $x_1 \oplus x_2$, while without memory, 4 gates are required.

The measures $M_{\oplus}$ and $M'_{\oplus}$ are of particular interest in the context of Chapter 9. The complexity of an invertible Boolean function with respect to either measure is exactly equal to that of its inverse, since the operation $(x_j := x_j \oplus ax_i \oplus b)$ is involutoric and since $(AB)^{-1} = B^{-1}A^{-1}$.

Proposition 4.3

If F is linear,

$$\mathcal{M}_{\oplus}(F^T) = \mathcal{M}_{\oplus}(F)$$

and

$$\mathcal{M}'_{\oplus}(F^T) = \mathcal{M}'_{\oplus}(F).$$

Proof:

By Lemma 4.2.2, $\mathcal{M}_{\oplus}(F)$ is the length of the shortest sequence $(x_i := x_i \oplus x_j), \ldots, (x_k := x_k \oplus x_l), \ldots, (x_r := x_r \oplus x_s)$ which computes F. By Theorem 3.1, the sequence $(x_s := x_s \oplus x_r), \ldots, (x_l := x_l \oplus x_k), \ldots, (x_j := x_j \oplus x_i)$ computes F^T. ∎

The following upper bound holds for both $\mathcal{M}_{\oplus}$ and $\mathcal{M}'_{\oplus}$.

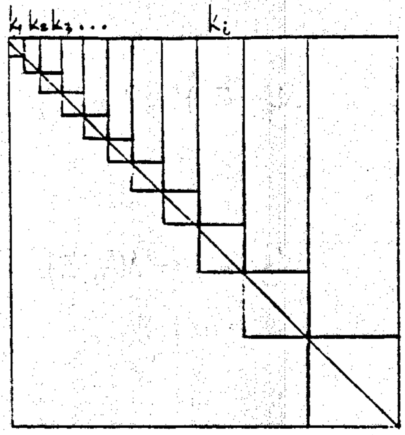
Theorem 4.4

For any function $F : B^n \to B^n$,

$$\mathcal{M}_{\oplus}(F) \leq \mathcal{M}'_{\oplus}(F) \leq \frac{n^2}{\log n} + o\left(\frac{n^2}{\log n}\right).$$

Proof:

By Lemma 4.2.2, $\mathcal{M}_{\oplus}(F) \leq C + O(n)$, where C is the number of operations $(x_j := x_j \oplus x_i)$ required to synthesize the linear function $F(x_1, \ldots, x_n) - F(0, \ldots, 0)$. The matrix $[F(x_1, \ldots, x_n) - F(0, \ldots, 0)]$ can be decomposed by Gaussian elimination into the product of three matrices, $[L][U][P]$, where $[L]$ is lower triangular, $[U]$ is upper triangular, and $[P]$ is a permutation matrix. If there exist circuits of width n for P, U, and L, they can be cascaded to produce a circuit for the function LUP. If transpositions are free, P takes no additions at all; while if they are not, P can be synthesized with no more than $3n - 3$ gates. By Theorem 4.3, the complexity of the hardest lower triangular matrix is the same as that of the hardest upper triangular matrix. Therefore, to prove the result it suffices to show that any $n \times n$



Layout for Modified Four Russians' Construction

upper triangular matrix can be synthesized from the identity by $\frac{(1+\epsilon)n^2}{2\log n}$ additions of one row into another.

This can be done with a modified version of the Four Russians' algorithm: divide the matrix into vertical strips of width $k_1, k_2, \ldots$, and synthesize the shortest [leftmost] strip, then the next shortest, and so on, adding the various k_i-length pieces together to complete the matrix. The square $k_i \times k_i$ submatrices along the principal diagonal are used as work spaces to successively generate all possible k_i-tuples, which are then added to the rows of the strip above the submatrix as many times as necessary to synthesize the strip. When all the entries in a strip are correct, the final form of the upper triangular submatrix itself can be obtained with no more than $\frac{1}{2}k_i^2$ additional operations.

The following procedure generates all possible k_i-tuples within the i^{th} submatrix:

(a) build in the first row, from the smallest weight vectors to the largest, all k_i-tuples whose first component is 1;

(b) build in the second row, from the smallest weight vectors to the largest, all k_i-tuples whose first component is 0 and whose second component is 1;

(c) continue in this fashion, building up in the j^{th} row of the submatrix all k_i-tuples whose leftmost 1 appears in the j^{th} component.

All 2^{k_i} possible k_i-tuples are created in $2^{k_i} - k_i - 1$ steps, and the i^{th} submatrix remains upper triangular at all times. To choose k_i for a given n, set $k_1 = 2$, and set

$$k_i = \lfloor \log \sum_{j=1}^{i-1} k_i - \log\log \sum_{j=1}^{i-1} k_i \rfloor,$$

for i such that $\sum_{j=1}^{i} k_j \leq n$. If the rightmost strip is permitted to have width less than $\lfloor \log \sum_{j=1}^{i-1} k_i - \log\log \sum_{j=1}^{i-1} k_i \rfloor$ for certain values of n, then $\sum k_i = n$. If $M(n)$ is the cost of an $n \times n$ upper triangular matrix, less any incomplete rightmost strip, then

$$M(n + \lfloor \log n - \log\log n \rfloor) \leq M(n) + n + \frac{n}{\log n} + \frac{1}{2}\log^2 n.$$

For any $\epsilon > 0$,

$$\frac{d}{dn}\left(\frac{(1+\epsilon)n^2}{2\log n}\right) - \frac{\frac{n}{\log n} + n + \frac{1}{2}\log^2 n}{\lfloor \log n - \log\log n \rfloor} \rightarrow \infty$$

as $n \rightarrow \infty$, so $\frac{(1+\epsilon)n^2}{2\log n}$ exceeds $M(n)$ for n sufficiently large. Any partial strip on the right edge of the matrix with width less than $\lfloor \log \sum_{j=1}^{i-1} k_i - \log\log \sum_{j=1}^{i-1} k_i \rfloor$ has no more than $n \log n$ entries, so its synthesis can contribute at most $o(n^2/\log n)$ steps to the overall cost.

Thus, for any $n \times n$ triangular matrix

$$\mathcal{M}_{\oplus}(F) \leq \mathcal{M}'_{\oplus}(F) \leq \frac{n^2}{2\log n} + o\left(\frac{n^2}{\log n}\right),$$

and for any $n \times n$ square matrix

$$\mathcal{M}_{\oplus}(F) \leq \mathcal{M}'_{\oplus}(F) \leq \frac{n^2}{\log n} + o\left(\frac{n^2}{\log n}\right). \quad \blacksquare$$

Corollary 4.4.1

Every linear invertible function $B^n \to B^n$ can be computed by a memoryless circuit.

Next, a lower bound for $\mathcal{M}_\oplus$ and $\mathcal{M}'_\oplus$ is presented. Under either model of computation, at most $(n(n-1))^C$ different matrices can be produced with C operations. By the following proposition, there are at least $(.28)2^{n^2}$ nonsingular $n \times n$ B-valued matrices, so there exist invertible linear functions $F : B^n \to B^n$ for which $\mathcal{M}'_\oplus(F) \geq \mathcal{M}_\oplus(F) \geq \frac{n^2}{2\log n} - o\left(\frac{n^2}{\log n}\right)$.

Proposition 4.5

For large n, a constant fraction of the linear Boolean functions from $B^n \to B^n$ are invertible.

Proof:

B^n has 2^n elements. If $F : B^n \to B^n$ is non-singular, then given the value of F for k linearly independent vectors, and $(x_1, \ldots, x_n) \in B^n$ not in the k-dimensional subspace generated by these vectors, $F(x_1, \ldots, x_n)$ can take on at most $2^n - 2^k$ different values. Therefore, the total number of invertible linear functions $B^n \to B^n$ is

$$(2^n - 1)(2^n - 2)(2^n - 4)\cdots\left(2^n - 2^{n-1}\right) = 2^{n^2} \prod_{k=1}^{n} \left(1 - \frac{1}{2^k}\right).$$

There are $(2^n)^n = 2^{n^2}$ linear functions $B^n \to B^n$, so the proportion of these functions which are invertible is:

$$\prod_{k=1}^{n} \left(1 - \frac{1}{2^k}\right).$$

This quantity is a monotonically decreasing function of n, and

$$
\begin{aligned}
1 &> \prod_{k=1}^{n}\left(1-\frac{1}{2^k}\right) = \exp\left(\sum_{k=1}^{n}\ln\left(1-\frac{1}{2^k}\right)\right) \\
&> \exp\left(-\sum_{k=1}^{n}\frac{\left(\frac{1}{2^k}\right)}{1-\left(\frac{1}{2^k}\right)}\right) = \exp\left(-\sum_{k=1}^{n}\frac{1}{2^k-1}\right) \\
&> \exp\left(-2\sum_{k=1}^{n}\frac{1}{2^k}\right) \\
&> \exp\left(-2+\frac{1}{2^{n-1}}\right) \\
&> 0,
\end{aligned}
$$

so the infinite product $\prod_{k=1}^{\infty}\left(1-\frac{1}{2^k}\right)$ converges to a nonzero limit. By a combinatorial argument due to Euler [NIV],

$$
\begin{aligned}
\prod_{k=1}^{\infty}\left(1-\frac{1}{2^k}\right) &= 1+\sum_{j=1}^{\infty}(-1)^j\left(\frac{1}{2^{(3j^2+j)/2}}+\frac{1}{2^{(3j^2-j)/2}}\right) \\
&= 1-\frac{1}{2}-\frac{1}{2^2}+\frac{1}{2^5}+\frac{1}{2^7}-\frac{1}{2^{12}}-\frac{1}{2^{15}}+\frac{1}{2^{22}}+\frac{1}{2^{26}}+\cdots \\
&= .288788103\ldots \quad \blacksquare
\end{aligned}
$$

However the bound of $\frac{n^2}{2\log n} - o\left(\frac{n^2}{\log n}\right)$ on the complexity of the hardest function can be improved by counting more carefully the number of functions with a given memoryless complexity.

Theorem 4.6

For all but a vanishing proportion of invertible $F : B^n \to B^n$,

$$\mathcal{M}_{\oplus}(F) \geq (1-\epsilon)\left(\frac{n^2}{\log n}\right).$$

Proof:

A memoryless circuit for the invertible linear Boolean function $F - F(0)$ can be described as a sequence of operations

$$x_j := x_j \oplus x_i,$$

with $i \neq j$. The strategy of the proof will be to define a canonical form for sequences of length C which has the property that any two sequences with the same canonical form compute the same function. To obtain an upper bound on the number of functions with complexity not exceeding C, the number of distinct canonical forms will be counted.

If i, j, k, l are all distinct, two successive operations

$$x_j := x_j \bigoplus x_i$$

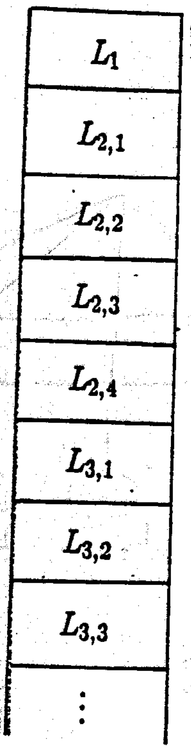
$$x_k := x_k \bigoplus x_l$$

can be transposed without changing the function that is computed by the sequence, so two such operations may be called *exchangeable*. Exchangeability is a reflexive relation on operations.

If the r^{th} operation in a sequence precedes the s^{th} operation, and if these two operations would not be exchangeable if they occurred in immediate succession, the r^{th} operation *blocks* the s^{th} operation.

Finally, the operation $x_j := x_j \bigoplus x_i$ *lexicographically precedes* the operation $x_k := x_k \bigoplus x_l$ if $j < k$ or if $j = k$ and $i < l$. Lexicographic precedence defines a linear order on the set of operations.

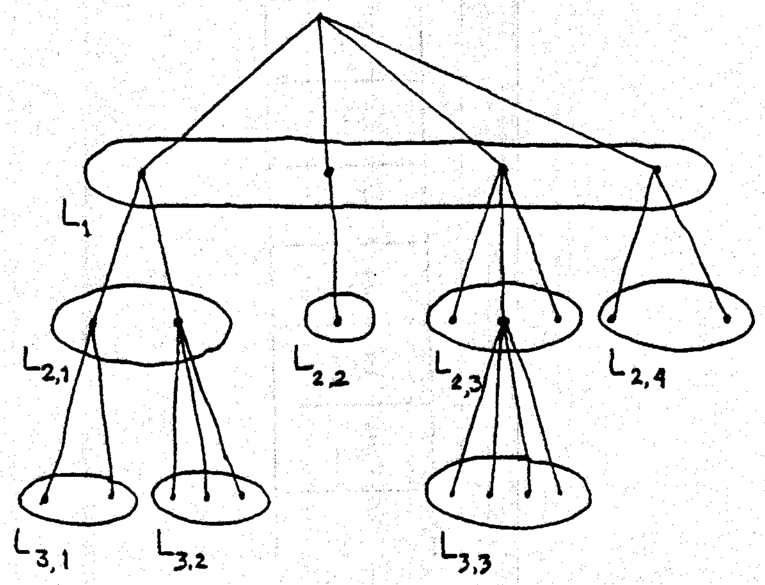
Next, the canonical form of a sequence of length C is defined by a series of transpositions, each of which does not change either the length of the sequence or the function which it computes. First, consider L_1, the set of all operations in the sequence which are not blocked at all. Each operation in L_1 is exchangeable with every operation which precedes it and which is not in L_1, for otherwise it would be blocked itself. Each operation in L_1 is also exchangeable with each other operation in L_1, since otherwise the earlier of the two would block the later one. Thus, these operations can be moved to the top of the sequence, and placed in lexicographic order, without changing the output function which is computed by the circuit. Every operation that was blocked in the original sequence is still blocked, since the blocking operation could not have been moved below it. L_1 comprises the first *level* of the sequence.



Reordered Sequence

Next, build the second level of the sequence in $|L_1|$ *sections*, where $|L_1|$ is the the number of operations at the first level. Define $L_{2,1}$, the set of operations below L_1 that are blocked by the first operation in L_1. By the same argument as before, every pair of operations in $L_{2,1}$ are mutually exchangeable, and each is exchangeable with every operation which precedes it and is below L_1. Using a series of transpositions, move every operation in $L_{2,1}$ to the top of the second level, and place them in lexicographic order. As before, all the remaining operations below $L_{2,1}$ are blocked by some operation after the first one in L_1. This completes the first section of the second layer. To build the second section of the second layer, find $L_{2,2}$, the set of operations below $L_{2,1}$ that are blocked by the second operation in L_1, move these up beneath $L_{2,1}$, and place them in lexicographic order. Continue this process for each of the $|L_1|$ operations in L_1. This completes the second level.

To build the third level, and all succeeding levels, continue inductively. The $j+1$th level, comprising $\{L_{j+1,i}\}_{1 \le i \le |L_j|}$, is constructed by sequentially examining each jth



A Blocking Tree

level operation in turn, and appending in lexicographic order all of the operations beneath the j^{th} level which are blocked by the operation being examined.

Because all operations below the j^{th} level are blocked by operations in the j^{th} level, this inductive process must eventually terminate by exhausting all the operations in the original sequence.

The reordered sequence computes the same function as the original one. Moreover, the reordered sequence has a natural tree structure, the *blocking tree*, defined as follows. Consider a graph G with $C+1$ vertices, one of which is distinguished as the root. Each of the remaining C vertices is associated with a single operation in the circuit. Draw an edge between vertex r and vertex s if and only if the r^{th} operation in the circuit is the last [or lowest] operation which blocks the s^{th} operation. If the r^{th} operation is not blocked by any operation in the sequence, draw an edge between vertex r and the root. G is connected, since if one begins at any vertex r, and traces the path $r \rightarrow$(the lowest vertex blocking r)$\rightarrow$(the lowest vertex blocking (the lowest vertex blocking r))$\rightarrow \cdots$, and so on, he will eventually end at the root. The analogous path starting at r' will also end at the root; hence r and r' are connected to each other. G has exactly C edges, since we constructed it in C steps, adding one edge at each step. Since it is connected and has one fewer edge than vertices, G is a tree.

If G is the blocking tree of some canonical sequence, the maximum degree of G is $n/2$, since each section of a reordered sequence consists of pairwise exchangeable elements, and at most $n/2$ operations can be disjoint in all their variables.

The canonical sequence associated with with a blocking tree can be recovered by working down and across, first computing all first level operations in lexicographic order; then all second level operations in lexicographic order of their parent nodes, with operations with the same parent node computed in lexicographic order; then all third level operations, in lexicographic order of their parents, with operations with the same parent computed in lexicographic order; and so on. Thus, a canonical sequence is uniquely associated with a tree in which every node besides the root is labeled with an operation in such a way that adjacent nodes have nonexchangeable operations, and nodes with a common parent have pairwise exchangeable operations. Conversely, every such labeled tree gives a canonical sequence.

An upper bound on the number of canonical sequences of length C, and hence on the number of distinct functions that can be computed with C operations, can be obtained by counting the number of rooted trees and multiplying by the maximum number of labelings that can be given to any one tree in a way that permits it to be the blocking tree of a canonical sequence.

R. Otter [OTT] has shown that there exist constants $k \simeq .4399$ and $\varsigma \simeq 2.956$ such that the number of rooted trees with n vertices equals

$$\frac{k\varsigma^n}{n^{3/2}} + O\left(\frac{\varsigma^n}{n^{5/2}}\right).$$

There are at most $n/2$ immediate descendents of the root, so operations can be assigned to these nodes in no more than $(n(n-1))^{n/2}$ ways. If operation $x_i :=$

$x_i \oplus x_j$ is associated with a vertex of G other than the root, then each daughter of this vertex must be associated with an operation of the form $x_i := x_i \oplus x_l$, $x_j := x_j \oplus x_l$, $x_l := x_l \oplus x_i$, or $x_l := x_l \oplus x_j$. Therefore, traversing the tree from the immediate descendants of its root to its leaves, counting the number of permissible assignments of operations to vertices, one has no more than $4n$ choices at each node. The total number of assignments does not exceed $(4n)^C n^n$.

Since the number of rooted trees with $C+1$ vertices is $o(3^C)$, and since the number of labellings of a given tree does not exceed $4^C n^n n^C$, the total number of functions with complexity C is $o(12^C n^n n^C)$. By Proposition 4.5 there are $O(2^{n^2})$ invertible linear functions $B^n \to B^n$ and, by Corollary 4.4.1, they can all be computed over $\{\oplus\}$. If $C \leq \frac{(1-\epsilon)n^2}{\log n}$, only a vanishingly small proportion of the possible invertible functions could have complexity less than or equal to C. ∎

Thus, the lower bound in Theorem 4.3 is asymptotically tight:

$$\max_{F:B^n \to B^n} M_{\oplus}(F) \sim \max_{F:B^n \to B^n} M'_{\oplus}(F) \sim \frac{n^2}{\log n}.$$

That is, the complexity of the n-input, n-output linear Boolean function that is hardest with respect to memoryless circuits is just twice that of the n-input, n-output linear Boolean function that is hardest with respect to circuits of unbounded width.

— Chapter Five —

BOOLEAN DERIVATIVES

This chapter reviews the properties of the *derivative* of a Boolean function, a concept that will be used in Chapters 6 and 8. Although finite Boolean algebras are discrete spaces, a number of the classical theorems of differential calculus remain valid in this setting. The concept of a Boolean derivative and the Boolean version of Taylor's theorem are apparently due to I. Reed [RED], D. Muller [MUL], and S. Akers [AKE]. These ideas have been further generalized by André Thayse and Marc Davio [DAV]. Theorem 5.20 is a new result.

Definition 5.1

Given a function $f : B^n \to B$, the **partial derivative of f with respect to x_i** is the function from B^n to B defined by

$$\frac{\partial f(x_1,\ldots,x_n)}{\partial x_i} = f(x_1,\ldots,x_i,0,x_{i+1},\ldots,x_n) \oplus f(x_1,\ldots,x_{i-1},1,x_{i+1},\ldots,x_n).$$

That is, $\partial f/\partial x_i$ is 1 or 0, according to whether the value of f changes as x_i changes while $x_1,\ldots,x_{i-1}$ and $x_{i+1},\ldots,x_n$ remain fixed.

Definition 5.2

Given $f : B^n \to B$, the **gradient** of f is the function $\nabla f : B^n \to B^n$ given by

$$\nabla f(x_1,\ldots,x_n) = \left(\frac{\partial f}{\partial x_1},\ldots,\frac{\partial f}{\partial x_n}\right).$$

Definition 5.3

Given $f : B^n \to B$, the **tangent map** to f at $(\alpha_1, \ldots, \alpha_n)$ is the affine function $T_f : B^n \to B$ defined by

$$\begin{aligned} T_f(x_1, \ldots, x_n) = f(\alpha_1, \ldots, \alpha_n) & \\ & \oplus (x_1 \oplus \alpha_1)\frac{\partial f}{\partial x_1}(\alpha_1, \ldots, \alpha_n) \\ & \oplus \cdots \oplus (x_n \oplus \alpha_n)\frac{\partial f}{\partial x_n}(\alpha_1, \ldots, \alpha_n). \end{aligned}$$

Definition 5.4

Given $f : B^n \to B$, the **derivative** of f at $(\alpha_1, \ldots, \alpha_n)$ is the linear functional

$$f'(x_1, \ldots, x_n) = \bigoplus_{i=1}^{n} x_i \frac{\partial f(\alpha_1, \ldots, \alpha_n)}{\partial x_i}.$$

Proposition 5.5

Partial differentiation is linear, i.e.,

$$\frac{\partial(f \oplus g)}{\partial x_i} = \frac{\partial f}{\partial x_i} \oplus \frac{\partial g}{\partial x_i}.$$

Proof:

$$\begin{aligned} \frac{\partial(f \oplus g)}{\partial x_i} &= (f \oplus g)(x_1, \ldots, 0, \ldots, x_n) \oplus (f \oplus g)(x_1, \ldots, 1, \ldots, x_n) \\ &= f(x_1, \ldots, 0, \ldots, x_n) \oplus g(x_1, \ldots, 0, \ldots, x_n) \\ &\quad \oplus f(x_1, \ldots, 1, \ldots, x_n) \oplus g(x_1, \ldots, 1, \ldots, x_n) \\ &= f(x_1, \ldots, 0, \ldots, x_n) \oplus f(x_1, \ldots, 1, \ldots, x_n) \\ &\quad \oplus g(x_1, \ldots, 0, \ldots, x_n) \oplus g(x_1, \ldots, 1, \ldots, x_n) \\ &= \frac{\partial f}{\partial x_i} \oplus \frac{\partial g}{\partial x_i}. \quad \blacksquare \end{aligned}$$

Proposition 5.6

For $a \in B$,

$$\begin{aligned} a\frac{\partial f(x_1,\ldots,x_n)}{\partial x_i} &= af(x_1,\ldots,0,\ldots,x_n) \oplus af(x_1,\ldots,1,\ldots,x_n) \\ &= a\frac{\partial f}{\partial x_i}. \end{aligned}$$

Proposition 5.7

Differentiation is linear, i.e., given $f, g : B^n \to B$,

$$f'(x_1,\ldots,x_n) \oplus g'(x_1,\ldots,x_n) = (f \oplus g)'(x_1,\ldots,x_n).$$

Proof:

$$\begin{aligned} (f \oplus g)' &= \bigoplus_{i=1}^{n} \frac{\partial(f \oplus g)}{\partial x_i} \\ &= \bigoplus_{i=1}^{n} \left[x_i\frac{\partial f}{\partial x_i} \oplus x_i\frac{\partial g}{\partial x_i} \right] \\ &= \bigoplus_{i=1}^{n} x_i\frac{\partial f}{\partial x_i} \oplus \bigoplus_{i=1}^{n} x_i\frac{\partial g}{\partial x_i} \\ &= f' \oplus g'. \quad \blacksquare \end{aligned}$$

Proposition 5.8

The partial derivative satisfies a version of Leibnitz's rule:

$$\frac{\partial(fg)}{\partial x_i} = f(x_1,\ldots,1,\ldots,x_n)\frac{\partial g}{\partial x_i} \oplus g(x_1,\ldots,0,\ldots,x_n)\frac{\partial f}{\partial x_i}.$$

Proof:

$$\begin{aligned}
\frac{\partial(fg)}{\partial x_i} &= fg(x_1,\ldots,0,\ldots,x_n) \oplus fg(x_1,\ldots,1,\ldots,x_n) \\
&= f(x_1,\ldots,0,\ldots,x_n)g(x_1,\ldots,0,\ldots,x_n) \\
&\qquad \oplus f(x_1,\ldots,1,\ldots,x_n)g(x_1,\ldots,1,\ldots,x_n) \\
&= f(x_1,\ldots,0,\ldots,x_n)g(x_1,\ldots,0,\ldots,x_n) \\
&\qquad \oplus f(x_1,\ldots,0,\ldots,x_n)g(x_1,\ldots,1,\ldots,x_n) \\
&\qquad \oplus f(x_1,\ldots,1,\ldots,x_n)g(x_1,\ldots,0,\ldots,x_n) \\
&\qquad \oplus f(x_1,\ldots,1,\ldots,x_n)g(x_1,\ldots,1,\ldots,x_n) \\
&= f(x_1,\ldots,0,\ldots,x_n)\frac{\partial g}{\partial x_i} \oplus g(x_1,\ldots,1,\ldots,x_n)\frac{\partial f}{\partial x_i} \\
&= f(x_1,\ldots,1,\ldots,x_n)\frac{\partial g}{\partial x_i} \oplus g(x_1,\ldots,0,\ldots,x_n)\frac{\partial f}{\partial x_i}. \quad \blacksquare
\end{aligned}$$

Proposition 5.9

The partial derivative of f with respect to x_i is independent of [the value of] x_i.

Proof:

$$\begin{aligned}
\left.\frac{\partial f(x_1,\ldots,x_n)}{\partial x_i}\right|_{(c_1,\ldots,0,\ldots,c_n)} &= f(c_1,\ldots,0,\ldots,c_n) \oplus f(c_1,\ldots,1,\ldots,c_n) \\
&= \left.\frac{\partial f(x_1,\ldots,x_n)}{\partial x_i}\right|_{(c_1,\ldots,1,\ldots,c_n)}. \quad \blacksquare
\end{aligned}$$

This means that the product rule for partial derivatives can be written in a more symmetric form.

Proposition 5.10

$$\begin{aligned}\frac{\partial(fg)}{\partial x_i} &= f(x_1,\ldots,x_i\oplus 1,\ldots,x_n)\frac{\partial g}{\partial x_i} \\ &\quad \oplus g(x_1,\ldots,x_i,\ldots,x_n)\frac{\partial f}{\partial x_i} \\ &= \left[f(x_1,\ldots,x_i,\ldots,x_n)\oplus\frac{\partial f}{\partial x_i}\right]\frac{\partial g}{\partial x_i} \\ &\quad \oplus g(x_1,\ldots,x_n)\frac{\partial f}{\partial x_i} \\ &= f\left[\frac{\partial g}{\partial x_i}\right]\oplus g\left[\frac{\partial f}{\partial x_i}\right]\oplus\left[\frac{\partial f}{\partial x_i}\right]\left[\frac{\partial g}{\partial x_i}\right].\end{aligned}$$

Corollary 5.10.1

If f is independent of x_i, then $\frac{\partial f}{\partial x_i}$ is a constant function.

Corollary 5.10.2

The derivative of a constant function is 0.

Corollary 5.10.3

All second and higher partials with respect to the same variable are zero, i.e.,

$$\frac{\partial^2 f}{\partial x_i^2} = 0.$$

Corollary 5.10.4

If $g(x_1,\ldots,x_n)$ is independent of x_i, then

$$\frac{\partial fg}{\partial x_i} = g(x_1,\ldots,x_n)\left[\frac{\partial f}{\partial x_i}\right].$$

Proposition 5.11

All mixed partials are equal.

Proof:

$$\begin{aligned}
\frac{\partial}{\partial x_k}\left[\frac{\partial}{\partial x_j}\left[\frac{\partial f}{\partial x_i}\right]\right] &= [[[f(x_1,\ldots,0,\ldots,0,\ldots,0,\ldots,x_n)\\
&\quad \oplus f(x_1,\ldots,1,\ldots,0,\ldots,0,\ldots,x_n)]\\
&\quad \oplus [f(x_1,\ldots,0,\ldots,1,\ldots,0,\ldots,x_n)\\
&\quad \oplus f(x_1,\ldots,1,\ldots,1,\ldots,0,\ldots,x_n)]]\\
&\quad \oplus [[f(x_1,\ldots,0,\ldots,0,\ldots,1,\ldots,x_n)\\
&\quad \oplus f(x_1,\ldots,1,\ldots,0,\ldots,1,\ldots,x_n)]\\
&\quad \oplus [f(x_1,\ldots,0,\ldots,1,\ldots,1,\ldots,x_n)\\
&\quad \oplus f(x_1,\ldots,1,\ldots,1,\ldots,1,\ldots,x_n)]]]\\
&= \left[\frac{\partial^2}{\partial x_k \partial x_j}\right]\left[\frac{\partial f}{\partial x_i}\right]\\
&= [[[f(x_1,\ldots,0,\ldots,0,\ldots,0,\ldots,x_n)\\
&\quad \oplus f(x_1,\ldots,0,\ldots,1,\ldots,0,\ldots,x_n)]\\
&\quad \oplus [f(x_1,\ldots,1,\ldots,0,\ldots,0,\ldots,x_n)\\
&\quad \oplus f(x_1,\ldots,1,\ldots,1,\ldots,0,\ldots,x_n)]]\\
&\quad \oplus [[f(x_1,\ldots,0,\ldots,0,\ldots,1,\ldots,x_n)\\
&\quad \oplus f(x_1,\ldots,0,\ldots,1,\ldots,1,\ldots,x_n)]\\
&\quad \oplus [f(x_1,\ldots,1,\ldots,0,\ldots,1,\ldots,x_n)\\
&\quad \oplus f(x_1,\ldots,1,\ldots,1,\ldots,1,\ldots,x_n)]]]\\
&= \frac{\partial}{\partial x_k}\left[\frac{\partial}{\partial x_i}\left[\frac{\partial f}{\partial x_j}\right]\right]. \quad \blacksquare
\end{aligned}$$

Recall from Chapter 1 that for $x_1,\ldots,x_n, y_1,\ldots,y_n \in B$, $(x_1,\ldots,x_n) \le (y_1,\ldots,y_n)$ when $x_i \le y_i$ for all i; $x_i^0 = 1$; and $x_i^1 = x_i$. With this notation, we have:

Corollary 5.11.1

$$\frac{\partial^k(x_1^{\alpha_1}x_2^{\alpha_2}\cdots x_n^{\alpha_n})}{\partial x_1^{\beta_1}\partial x_2^{\beta_2}\cdots\partial x_n^{\beta_n}} = \begin{cases} 0, & \text{if } (\alpha_1,\ldots,\alpha_n) \not\leq (\beta_1,\ldots,\beta_n) \\ x_1^{\alpha_1\oplus\beta_1}\cdots x_n^{\alpha_n\oplus\beta_n}, & \text{if } (\alpha_1,\ldots,\alpha_n) < (\beta_1,\ldots,\beta_n) \\ 1, & \text{if } (\alpha_1,\ldots,\alpha_n) = (\beta_1,\ldots,\beta_n) \end{cases}$$

Examples 5.11.1.1

$$\frac{\partial(x_1x_3x_4)}{\partial x_1} = x_2x_3$$

$$\frac{\partial^3(x_1x_3x_4)}{\partial x_1\partial x_3\partial x_4} = 1$$

$$\frac{\partial^3(x_1x_3x_4)}{\partial x_1\partial x_2\partial x_3} = 0.$$

Corollary 5.11.2

The derivative of a Boolean monomial $x_1^{\alpha_1}x_2^{\alpha_2}\cdots x_n^{\alpha_n}$ with respect to the variables $x_1^{\beta_1},\ldots,x_n^{\beta_n}$, when evaluated at $x_1 = x_2 = \ldots = x_n = 0$, is 1 if $(\alpha_1,\ldots,\alpha_n) = (\beta_1,\ldots,\beta_n)$, and is 0 otherwise.

Example 5.11.2.1

$$\frac{\partial^2(a_1x_1x_3x_4 \oplus a_2x_2x_3 \oplus a_3x_2x_4 \oplus a_4x_1x_4 \oplus a_5x_1 \oplus a_6x_2 \oplus a_7x_3 \oplus a_8)}{\partial x_1\partial x_3},$$

when evaluated at $x_1 = x_2 = x_3 = x_4 = 0$, equals a_2.

Proposition 5.12

$$\left.\frac{\partial^m f(x_1,\ldots,x_n)}{\partial x_1^{\beta_1}\cdots\partial x_n^{\beta_n}}\right|_{(0,\ldots,0)} = \bigoplus_{\substack{(x_1,\ldots,x_n)\\ \leq(\beta_1,\ldots,\beta_n)}} f(x_1,\ldots,x_n).$$

Proof:

Let x'_i be those x_i with $\beta_i = 1$, i.e., those variables with respect to which the derivative is to be taken, and x''_j be the remaining variables. The proposition now reads

$$\left.\frac{\partial^m f(x_1,\ldots,x_n)}{\partial x'_1\cdots\partial x'_n}\right|_{(0,\ldots,0)} = \bigoplus_{all\,(x'_1,\ldots,x'_n)} \left. f(x_1,\ldots,x_n)\right|_{x''_i=0} .$$

The proof is by induction on m:

$$\left.\frac{\partial f(x_1,\ldots,x_n)}{\partial x'_1}\right|_{(0,\ldots,0)} = f(0,\ldots,0,\ldots,0,)\oplus f(0,\ldots,1,\ldots,0).$$

If the statement is true for $m-1$, then

$$\frac{\partial}{\partial x'_m}\left[\frac{\partial^{m-1} f(x_1,\ldots,x_n)}{\partial x'_1\ldots\partial x'_{m-1}}\right] = \bigoplus_{\substack{all\ n-tuples\\ (x'_1,\ldots,x'_{m-1},0)}} \left. f(x_1,\ldots,x_n)\right|_{x''_i=0}$$

$$\oplus \bigoplus_{\substack{all\ n-tuples\\ (x'_1,\ldots,x'_{m-1},1)}} \left. f(x_1,\ldots,x_n)\right|_{x''_i=0},$$

whence it is true for m. ∎

Corollary 5.12.1

Applying the proposition to the function $g(x_1,\ldots,x_n) = f(x_1\oplus c_1,\ldots,x_n\oplus c_n)$,

$$\left.\frac{\partial^m g(x_1,\ldots,x_n)}{\partial x_1^{\beta_1}\ldots\partial x_n^{\beta_n}}\right|_{(c_1,\ldots,c_n)} \bigoplus_{\substack{(x_1,\ldots,x_n)\\ \le(\beta_1,\ldots,\beta_n)}} f(x_1\oplus c_1,\ldots,x_n\oplus c_n).$$

Example 5.12.1.1

$$\frac{\partial^2 f(0,1,1,0,1)}{\partial x_2\partial x_4}$$

$$= f(0,1,1,0,1)\oplus f(0,1,1,1,1)\oplus f(0,0,1,0,1)\oplus f(0,0,1,1,1).$$

Proposition 5.13

If f is an affine function, then each of its partial derivatives is a constant function. Conversely, if each of the partial derivatives of g is a constant, then G is affine.

Proof:

Let

$$f(x_1,\ldots,x_n) = a_0 \oplus \bigoplus_{i=1}^{n} a_i x_i.$$

Then

$$\begin{aligned}\frac{\partial f(x_1,\ldots,x_n)}{\partial x_j} &= f(x_1,\ldots,0,\ldots,x_n) \oplus f(x_1,\ldots,1,\ldots,x_n)\\ &= \left[a_0 \oplus \bigoplus_{i\neq j} a_i x_i\right] \oplus \left[a_0 \oplus a_j \oplus \bigoplus_{i\neq j} a_i x_i\right]\\ &= a_j,\end{aligned}$$

a constant independent of $(x_1,\ldots,x_n)$. Now suppose $\frac{\partial g}{\partial x_i} = a_i$ for all i, and consider $e : B^n \to B$, defined by

$$\begin{aligned}e(x_1,\ldots,x_n) &= g(x_1,\ldots,x_n) \oplus \bigoplus_{i=1}^{n} \frac{\partial g}{\partial x_i} x_i\\ &= g(x_1,\ldots,x_n) \oplus \bigoplus_{i=1}^{n} a_i x_i.\end{aligned}$$

But

$$\begin{aligned}\frac{\partial e(x_1,\ldots,x_n)}{\partial x_i} &= \frac{\partial g(x_1,\ldots,x_n)}{\partial x_i} \oplus a_i\\ &= 0,\end{aligned}$$

for all $1 \leq i \leq n$.

If every partial derivative of e is 0 everywhere, $e(0,0,0,\ldots,0) = e(1,0,0,\ldots,0) = e(0,1,0,\ldots,0) = e(1,1,0,\ldots,0) = \ldots = e(1,1,1,\ldots,1)$. That is, $e(x_1,x_2,x_3,\ldots,x_n)$ takes on but a single value, $e(0,0,0,\ldots,0)$, independent of $(x_1,x_2,x_3,\ldots,x_n)$. Therefore

$$g(x_1,\ldots,x_n) = e(x_1,\ldots,x_n) \oplus \bigoplus_{i=1}^{n} a_i x_i$$
$$= e(0,\ldots 0) \oplus \bigoplus_{i=1}^{n} a_i x_i. \quad \blacksquare$$

The Boolean derivative satisfies a close analog of the multivariate Taylor's theorem of ordinary calculus—given knowledge of all of a function's derivatives at a single point, it is possible to predict the value of the function everywhere.

Theorem 5.14

Let f be a function from $B^n \to B$, and let $Y = (y_1,\ldots,y_n)$ be some element in the domain of f. Then,

$$f(Y) = \bigoplus_{\substack{(\beta_1,\ldots,\beta_n)\\ \leq(y_1,\ldots,y_n)}} \left.\frac{\partial f(x_1,\ldots,x_n)}{\partial x_1^{\beta_1}\cdots\partial x_n^{\beta_n}}\right|_{(0,0,\ldots,0)}$$
$$= \bigoplus_{\beta\in B^n} \left(\left.\frac{\partial f(x_1,\ldots,x_n)}{\partial x_1^{\beta_1}\cdots\partial x_n^{\beta_n}}\right|_{(0,0,\ldots,0)} \right) y_1^{\beta_1}\cdots y_n^{\beta_n}.$$

Proof:

Let

$$f(X) = \bigoplus_{\beta\in B^n} a_\beta x_1^{\beta_1} x_2^{\beta_2}\cdots x_n^{\beta_n}$$

be the ring-sum expansion of f. Observe that Corollary 5.11.2 implies that

$$\left.\frac{\partial f(x_1,\ldots,x_n)}{\partial x_1^{\beta_1}\cdots\partial x_n^{\beta_n}}\right|_{(0,0,\ldots,0)} = a_\beta,$$

since all terms of lower order than β disappear in the differentiation step, while all higher order terms disappear in the evaluation step.

Thus,

$$\begin{aligned} f(Y) &= \bigoplus_{\beta \in B^n} a_\beta y_1^{\beta_1} \cdots y_n^{\beta_n} \\ &= \bigoplus_{\beta \leq Y} a_\beta, \end{aligned}$$

since the product $y_1^{\beta_1} \cdots y_n^{\beta_n}$ vanishes unless $\beta \leq Y$. ∎

This formula can also be derived from Proposition 5.12 by Möbius inversion. Theorem 5.14 gives the Maclaurin expansion of f; a change of variables gives the more general Taylor's expansion:

Corollary 5.14.1

$$\begin{aligned} f(Z) &= \bigoplus_{\beta \leq Z \oplus C} \left. \frac{\partial f(x_1, \ldots, x_n)}{\partial x_1^{\beta_1} \cdots \partial x_n^{\beta_n}} \right|_{(c_1, \ldots, c_n)} \\ &= \bigoplus_{\beta \in B^n} \left(\left. \frac{\partial f(x_1, \ldots, x_n)}{\partial x_1^{\beta_1} \cdots \partial x_n^{\beta_n}} \right|_{(c_1, \ldots, c_n)} \right) (z_1 \oplus c_1)^{\beta_1} \cdots (z_n \oplus c_n)^{\beta_n}. \end{aligned}$$

Example 5.14.1.1

Let $f(x_1, x_2, x_3) = (x_1 \wedge x_2) \vee x_3$.

$$\begin{aligned} f(x_1, x_2, x_3) &= x_1 x_2 x_3 \oplus x_1 x_2 \oplus x_3 \\ \frac{\partial f}{\partial x_3}(x_1, x_2, x_3) &= x_1 x_2 \oplus 1 \\ \frac{\partial f}{\partial x_2}(x_1, x_2, x_3) &= x_1 x_3 \oplus x_1 \\ \frac{\partial^2 f}{\partial x_2 \partial x_3}(x_1, x_2, x_3) &= x_1 \\ \frac{\partial f}{\partial x_1}(x_1, x_2, x_3) &= x_2 x_3 \oplus x_2 \\ \frac{\partial^2 f}{\partial x_1 \partial x_3}(x_1, x_2, x_3) &= x_2 \\ \frac{\partial^2 f}{\partial x_1 \partial x_2}(x_1, x_2, x_3) &= x_3 \oplus 1 \\ \frac{\partial^3 f}{\partial x_1 \partial x_2 \partial x_3}(x_1, x_2, x_3) &= 1 \end{aligned}$$

$x_1x_2x_3$	f	$\Delta_{(0,0,0)}f$	$\Delta_{(1,0,1)}f$
0 0 0	0	0	1
0 0 1	1	1	0
0 1 0	0	0	0
0 1 1	1	0	1
1 0 0	0	0	0
1 0 1	1	0	0
1 1 0	1	1	0
1 1 1	1	1	1

$$\begin{aligned} f(1,1,0) &= f(0,0,0) \oplus \frac{\partial f}{\partial x_1}(0,0,0) \oplus \frac{\partial f}{\partial x_2}(0,0,0) \oplus \frac{\partial^2 f}{\partial x_1 \partial x_2}(0,0,0) \\ &= f(1,0,1) \oplus \frac{\partial f}{\partial x_2}(1,0,1) \oplus \frac{\partial f}{\partial x_3}(1,0,1) \oplus \frac{\partial^2 f}{\partial x_2 \partial x_3}(1,0,1) \\ &= 1. \end{aligned}$$

Definition 5.15

If $F : B^n \to B^m$ has coordinate functions $f_1, \ldots, f_m$, the **Jacobian matrix of** F is the $m \times n$ matrix $[J_F]$ whose ijth entry is the function $\partial f_i / \partial x_j$.

Proposition 5.15.1

If $[J_F] = [J_G]$, then $F - G = K$, where $K : B^n \to B^m$ is a constant function.

Proof:

Once $F(0,0,\ldots,0)$ is known, $F(x_1,\ldots,x_n)$ is determined by $[J_F]$.

$$\begin{aligned} f_j(x_1,\ldots,x_n) =& f_j(0,0,\ldots,0) \oplus \frac{\partial f_j}{\partial x_1}(x_1,0,\ldots,0) \\ & \oplus \frac{\partial f_j}{\partial x_2}(X_1,X_2,0,\ldots,0) \oplus \frac{\partial f_j}{\partial x_3}(x_1,x_2,x_3,\ldots,0) \\ & \oplus \cdots \oplus \frac{\partial f_j}{\partial x_n}(x_1,x_2,\ldots,x_{n-1},x_n). \quad \blacksquare \end{aligned}$$

F is affine if and only if $[J_F]$ is constant.

Definition 5.16

If $f : B^n \to B$, the **Hessian matrix** of f is the $n \times n$ matrix whose ij^{th} entry is $\partial f/\partial x_i \partial x_j$.

A Hessian matrix is symmetric and has zeros on the main diagonal, and its ij^{th} entry is independent of x_i and x_j.

Proposition 5.16.1

If the Hessian matrix of f is identical to the Hessian matrix of g then $f - g = l$, where l is affine.

Proof:

$$\begin{aligned} f(x_1, \ldots, x_n) = {} & f(0,0,\ldots,0) \oplus \frac{\partial f}{\partial x_1}(x_1, 0, \ldots, 0) \\ & \oplus \frac{\partial f}{\partial x_2}(x_1, x_2, 0, \ldots, 0) \oplus \frac{\partial f}{\partial x_3}(x_1, x_2, x_3, \ldots, 0) \\ & \oplus \cdots \oplus \frac{\partial f}{\partial x_n}(x_1, x_2, \ldots, x_{n-1}, x_n). \end{aligned}$$

On the other hand,

$$\begin{aligned} \frac{\partial f}{\partial x_i}(x_1, \ldots, x_n) = {} & \frac{\partial f}{\partial x_i}(0,0,\ldots,0) \oplus \frac{\partial f}{\partial x_i \partial x_1}(x_1, 0, \ldots, 0) \oplus \frac{\partial f}{\partial x_i \partial x_2}(x_1, x_2, \ldots, 0) \\ & \oplus \cdots \oplus \frac{\partial f}{\partial x_i \partial x_n}(x_1, x_2, \ldots, x_{n-1}, x_n). \end{aligned}$$

Thus, f is determined by its Hessian matrix, $f(0,0,\ldots,0)$ and $\{\frac{\partial f}{\partial x_j}(0,0,\ldots,0)\}_{j=1,\ldots,n}$. In fact, by Theorem 5.14, if f and g have the same Hessian matrix they differ only by

$$f(0,\ldots,0) \oplus \bigoplus_{i=1}^{n} \frac{\partial f}{\partial x_i}(0,\ldots,0)x_i. \quad \blacksquare$$

The expansion in Proposition 5.16.1 is equally valid when based at any fixed point in B^n; i.e., for any point $(\alpha_1, \ldots, \alpha_n) \in B^n$, f is completely determined by its Hessian matrix and by $[T_f(\alpha_1, \ldots, \alpha_n)]$.

The Boolean analog of line integrals can now be defined.

Definition 5.17

A parameterized path in B^n is a function ϕ from the integers $0, 1, 2, \ldots, T$ to B^n with the property that $d\phi(t)/dt = \phi(t) \oplus \phi(t+1)$ is an atom in B^n for every t between 0 and $T-1$. That is, the successive points $\phi(0), \phi(1), \phi(2), \ldots, \phi(T)$ trace out a path along the edges of the n-cube B^n.

Definition 5.18

Let F be a function $B^n \to B^m$, and let f_i be the i^{th} component of F. Let P be some path in B^n with parameterization $\phi : \{0, 1, \ldots, T\} \to B^n$, and let $\phi_i : \{0, 1, \ldots, T\}$ be the i^{th} component of ϕ. Then the **line integral of F along the path P** is

$$\begin{aligned}\int_P F(x) \cdot dx &= \bigoplus_{t=0}^{T-1} \left(f_1 \frac{d\phi_1}{dt} \oplus f_2 \frac{d\phi_2}{dt} \oplus \cdots \oplus f_n \frac{d\phi_n}{dt} \right) \\ &= \bigoplus_{t=0}^{T-1} \bigoplus_{i=1}^{n} f_i(\phi(t+1)) \frac{d\phi_i(t)}{dt}.\end{aligned}$$

Proposition 5.19

Integrals add along paths.

Proof:

$$\begin{aligned}\int_{P_1} F \cdot dx \oplus \int_{P_2} F \cdot dx = &\left[\bigoplus_{i=1}^{n} f_i(\phi(1))(\phi_i(0) \oplus \phi_i(1)) \right] \oplus \cdots \\ &\oplus \left[\bigoplus_{i=1}^{n} f_i(\phi_i(T))(\phi_i(T-1) \oplus \phi_i(T)) \right] \oplus \cdots \\ &\oplus \left[\bigoplus_{i=1}^{n} f_i(\psi_i(1))(\psi_i(0) \oplus \psi_i(1)) \right] \oplus \cdots \\ &\oplus \left[\bigoplus_{i=1}^{n} f_i(\psi_i(T'))(\psi_i(T'-1) \oplus \psi(T')) \right]\end{aligned}$$

$$= \int_{P_1+P_2} F \cdot dx,$$

where $P_1 + P_2$ is the path traced by $\phi(1), \phi(2), \ldots, \phi(T-1)$, $\phi(T) = \psi(0)$, $\psi(1)$, $\ldots$, $\psi(T')$. ∎

The value of the line integral depends only on the path P and the direction in which P is travelled, but not on the parameterization. The direction *is* important—in general:

$$\int_P F \cdot dx \neq -\int_{-P} F \cdot dx.$$

The next result is a Boolean version of one of the key theorems of real multivariate calculus:

Theorem 5.20

Consider $F : B^n \to B^n$, with coordinate functions $f_i : B^n \to B$. Then the following conditions are equivalent:

(a) there exists a potential function $f : B^n \to B$ with $F = \nabla f$. That is, there exists an f such that $f_j = \frac{\partial f}{\partial x_j}$ for all $1 \leq j \leq n$.

(b) F is conservative. That is, if P_1 and P_2 are any two paths in B^n with the same endpoints, then

$$\int_{P_1} F \cdot dx = \int_{P_2} F \cdot dx.$$

(c)

$$\int_P F(x) \cdot dx = 0,$$

for any closed loop P.

(d)

$$\frac{\partial f_i}{\partial x_j} = \begin{cases} \frac{\partial f_j}{\partial x_i}, & \text{if } i \neq j \\ 0, & \text{if } i = j. \end{cases}$$

Proof:

$\underline{(a) \Rightarrow (d)}$

For any $f : B^n \to B$, the mixed partials of f are equal by Proposition 5.11 and the second partials are zero by Corollary 5.10.3.

$\underline{(c) \Rightarrow (b)}$

Let P_1 be the path traced by $\phi(0), \ldots, \phi(T)$ and P_2 be the path traced out by $\psi(0), \ldots, \psi(T')$. Suppose $\phi(0) = \psi(0)$ and $\phi(T) = \psi(T')$. The path $P_1 - P_2$ traced by $\phi(0), \ldots, \phi(T), \psi(T'-1), \ldots, \psi(0)$ is a closed loop, and so, by hypothesis

$$\int_{P_1 - P_2} F(x) \cdot dx = 0.$$

Thus,

$$\int_{P_1} F \cdot dx = \int_{-P_2} F \cdot dx.$$

Finally, since the path $P_2 - P_2$ traced by $\psi(0), \ldots, \psi(T'), \ldots, \psi(0)$ is a closed loop,

$$\int_{P_2} F dx = \int_{-P_2} F dx,$$

whence

$$\int_{P_1} F \cdot dx = \int_{P_2} F \cdot dx.$$

$\underline{(b) \Rightarrow (a)}$

Define $f : B^n \to B$ by

$$f(X) = \int_P F(y) \cdot dy,$$

where p is any path from 0 to X. Since F is conservative, one may assume that the path of integration does not move along the j^{th} dimension until its very last step. That is, $d\phi_j(t)/dt = 0$ for $0 \leq t < T - 1$, $d\phi_j(T-1)/dt = 1$, and $d\phi_i(T-1)/dt = 0$ for $i \neq j$. Then,

$$\begin{aligned}\frac{\partial f}{\partial x_j} &= \frac{\partial}{\partial x_j}\int_P F(y)\cdot dy \\ &= \sum_{t=0}^{T-1}\sum_{i=1}^{n} f_i(\phi(t+1))\frac{d\phi_i(t)}{dt} \oplus \sum_{t=0}^{T-2}\sum_{i=1}^{n} f_i(\phi(t+1))\frac{d\phi_i(t)}{dt} \\ &= \sum_{i=1}^{n} f_i(\phi(T))\frac{d\phi_i(T-1)}{dt} \\ &= f_j(\phi(T)) \\ &= f_j(X).\end{aligned}$$

<u>$(d) \Rightarrow (c)$</u>

Assume the implication is false, i.e., that there exist closed paths around which the line integral of F is non-zero. Without loss of generality, it can be assumed that some such path passes through the origin. Since the integral around a closed loop is independent of the starting point of the parameterization (but possibly dependent on the direction), assume that loops passing through the origin start and end there.

Define the *weight of a path* to be the sum of the Hamming weights of the successive points along the path, i.e.,

$$weight\,(P) = \sum_{t=0}^{T} weight\,\phi(t).$$

Every path has some weight; assume that P is a path with *minimal weight* among all closed paths along which F has a non-zero integral. Let P be traced by the parameter ϕ, with $\phi(0) = \phi(T) = (0,0,\ldots,0)$.

First, we wish to show that P has no *reversals*, i.e., points where it turns back on itself. Suppose there is some i and t such that $\phi_i(t-1) = c$, $\phi_i(t) = c \oplus 1$, and $\phi_i(t+1) = c$. Then $\frac{d\phi_i(t-1)}{dt} = \frac{d\phi_i(t)}{dt} = 1$ while $\frac{d\phi_j(t-1)}{dt} = \frac{d\phi_j(t)}{dt} = 0$, so that the terms contributed to the integral by these two steps are $f_i(\phi(t))$ and $f_i(\phi(t))$, respectively.

However,

$$f_i(\phi(t)) \oplus f_i(\phi(t+1)) = \left.\frac{\partial f_i(X)}{\partial x_i}\right|_{X=\phi(t)}.$$

But by the hypothesis (d), $\frac{\partial f_i(x)}{\partial x_i}$ is uniformly zero. Thus the two steps from $\phi(t-1)$ to $\phi(t)$ and from $\phi(t)$ to $\phi(t+1)$ together contribute nothing to the integral, and their removal from the path P produces a new path with smaller weight and a non-zero integral, a contradiction. Thus, P has no reversals.

The shortest possible candidate for P would be a path of length two, i.e., one which started at the origin, took one step out, and immediately returned. This would constitute a reversal, however; so if P exists at all, it must have length ≥ 4 and weight ≥ 4.

Now, consider some time t for which the weight of $\phi(t)$ is *maximal.*

$$\max|\phi(t)| \geq 2,$$

so we can write $|\phi(t-1)|+1 = |\phi(t)| = |\phi(t+1)|+1$. If there is no reversal, there must exist i, j with

$$\begin{aligned}
\phi_i(t-1) &= 0 \\
\phi_i(t) &= 1 \\
\phi_i(t+1) &= 1 \\
\phi_j(t-1) &= 1 \\
\phi_j(t) &= 1 \\
\phi_j(t+1) &= 0,
\end{aligned}$$

while $\phi_k(t-1) = \phi_k(t) = \phi_k(t+1)$ for all $k \neq i, j$. The contribution of these two steps to the integral is $f_i(\phi(t)) \oplus f_j(\phi(t+1))$. Now consider the modified path P', identical to P except that

$$\begin{aligned}
\phi'_i(t-1) &= 0 \\
\phi'_i(t) &= 0 \\
\phi'_i(t+1) &= 1 \\
\phi'_j(t-1) &= 1 \\
\phi'_j(t) &= 0 \\
\phi'_j(t+1) &= 0.
\end{aligned}$$

The contribution of *these* two steps is $f_j(\phi'(t)) \oplus f_i(\phi'(t+1))$. What is the difference between the two integrals?

$$\begin{aligned}\int_P F(x)\cdot dx - \int_{P'} F(x)\cdot dx &= f_i(\phi(t)) \oplus f_i(\phi'(t+1)) \\ &\quad \oplus f_j(\phi'(t)) \oplus f_j(\phi(t+1)) \\ &= \frac{\partial f_i(\phi(t+1))}{\partial x_j} \oplus \frac{\partial f_j(\phi'(t+1))}{\partial x_i} \\ &= \frac{\partial f_i(\phi(t+1))}{\partial x_j} \oplus \frac{\partial f_j(\phi(t+1))}{\partial x_i}.\end{aligned}$$

But by hypothesis (d), this last quantity is zero. Therefore, $\int_P F(x)\cdot dx$ is non-zero, while the weight of P' is less than that of P, a contradiction. One concludes that there could have been no such counterexample P. ∎

– Chapter Six –

THE Δ-TRANSFORM

In this chapter, a linear transformation of a Boolean functional is defined which is equivalent to both Boolean polynomial evaluation and to Boolean polynomial interpolation. The set of Boolean functions invariant under this transform is enumerated and characterized.

Definition 6.1

Given a function $f : B^n \to B$ and $Y \in B^n$, the **Δ-transform** of f at $Y = (y_1, \ldots, y_n)$ is the function $\Delta_Y f : B^n \to B$ given by

$$\Delta_Y f(x_1, \ldots, x_n) = \begin{cases} f(Y), & \text{if} \quad (x_1, \ldots, x_n) = 0 \\ \\ \frac{\partial^k f}{\partial x_{\alpha_1} \cdots \partial x_{\alpha_k}}(Y), & \text{if} \quad (x_1, \ldots, x_n) \neq 0, \end{cases}$$

where the partial derivative is taken with respect to each variable x_i in $(x_1, \ldots, x_n)$ that has the value 1. When no confusion will result, write Δf for $\Delta_0 f$.

Example 6.1.1

$$\Delta_{1100} f(1,0,1,1) = \frac{\partial^3 f(1,1,0,0)}{\partial x_1 \partial x_3 \partial x_4}.$$

Example 6.1.2

The Δ_Y-transform of the zero function is the zero function, since $f(Y) = 0$ for all Y, and all partial derivatives are zero.

Example 6.1.3

Suppose $f : B^n \to B$ is the constant function $f(x_1, \ldots, x_n) = 1$. Then

$$\begin{aligned}\Delta_X f(x_1, \ldots, x_n) &= \begin{cases} 1, & \text{if } (x_1, \ldots, x_n) = 0 \\ 0, & \text{otherwise,} \end{cases} \\ &= NOR(x_1, \ldots, x_n).\end{aligned}$$

Example 6.1.4

Appendix B displays a table of f and Δf for each of the 256 distinct functions from B^3 to B.

Proposition 6.1.5

The Δ_Y-transform is linear.

Proof:

$$\begin{aligned}\Delta_Y(f \oplus g)(x_1, \ldots, x_n) &= \frac{\partial(f \oplus g)}{\partial x_{\alpha_1} \cdots}(Y) \\ &= \frac{\partial f}{\partial x_{\alpha_1} \cdots}(Y) \oplus \frac{\partial g}{\partial x_{\alpha_1} \cdots}(Y) \\ &= \Delta_Y f(x_1, \ldots, x_n) \oplus \Delta_Y g(x_1, \ldots, x_n). \quad \blacksquare\end{aligned}$$

The Δ_Y-transform represents a linear transformation on the space of all functions from B^n to B. For example, the Δ-transform is represented by the matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

with respect to the following basis for the vector space of functions $B^3 \to B$:

$$\begin{aligned}
a_1(x_1,x_2,x_3) &= 1 \text{ if and only if } (x_1,x_2,x_3) = (0,0,0)\\
a_2(x_1,x_2,x_3) &= 1 \text{ if and only if } (x_1,x_2,x_3) = (0,0,1)\\
a_3(x_1,x_2,x_3) &= 1 \text{ if and only if } (x_1,x_2,x_3) = (0,1,0)\\
a_4(x_1,x_2,x_3) &= 1 \text{ if and only if } (x_1,x_2,x_3) = (0,1,1)\\
a_5(x_1,x_2,x_3) &= 1 \text{ if and only if } (x_1,x_2,x_3) = (1,0,0)\\
a_6(x_1,x_2,x_3) &= 1 \text{ if and only if } (x_1,x_2,x_3) = (1,0,1)\\
a_7(x_1,x_2,x_3) &= 1 \text{ if and only if } (x_1,x_2,x_3) = (1,1,0)\\
a_8(x_1,x_2,x_3) &= 1 \text{ if and only if } (x_1,x_2,x_3) = (1,1,1).
\end{aligned}$$

Theorem 6.2

For any $f : B^n \to B$, $\Delta f(a_1,\ldots,a_n) = \Delta_0 f(a_1,\ldots,a_n)$ equals the coefficient of the term $x_1^{a_1}x_2^{a_2}\cdots x_n^{a_n}$ in the ring-sum expansion of f.

Proof:

Every function is expressible as the sum of monomials in the variables $x_1,\ldots,x_n$. The partial derivative of a function with respect to any set of variables is the sum of the partial derivatives of each monomial with respect to that same set of variables. Therefore, the partial derivative of $\bigwedge_{i=1}^{n} x_i^{\alpha_i}$ with respect to the variables $x_i^{\beta i}$ is 0 when evaluated at 0, unless $\alpha_i = \beta_i$. ∎

Example 6.2.1

Let $f(x_1,x_2) = x_1 \bigvee x_2 = x_1 \bigoplus x_2 \bigoplus x_1x_2 = a_{00} \bigoplus a_{10}x_1 \bigoplus a_{01}x_2 \bigoplus a_{11}x_2$, where $a_{00} = 0$, $a_{10} = 1$, $a_{01} = 1$, and $a_{11} = 1$. $\Delta f(0,0) = 1$, $\Delta f(1,0) = 1$, $\Delta f(0,1) = 1$, and $\Delta f(1,1) = 1$, whence $\Delta f(x_1,x_2) = x_1 \bigvee x_2$.

Each function $B^n \to B$ is associated with a unique polynomial, and $\Delta(f-g) = 0$ if and only if $f - g = 0$, so the Δ-transform is one-to-one. The set of functions $B^n \to B$ is finite (of order 2^{2^n}), so the Δ-transform is invertible.

Theorem 6.3

The Δ-tranform is involutoric. That is,

$$\Delta(\Delta f) = f.$$

Proof:

This follows directly from Proposition 5.12:

$$\Delta f(\alpha_1, \ldots, \alpha_n) = \bigoplus_{\substack{(x_1,\ldots,x_n) \\ \leq(\alpha_1,\ldots,\alpha_n)}} f(x_1, \ldots, x_n),$$

and from Theorem 5.14:

$$f(y_1, \ldots, y_n) = \bigoplus_{\substack{(\beta_1,\ldots,\beta_n) \\ \leq(y_1,\ldots,y_n)}} \Delta f(\beta_1, \ldots, \beta_n). \quad \blacksquare$$

Definition 6.4

Given two functions $f, g : B^n \to B$, the Δ-convolution of f and g is

$$f * g = \Delta(\Delta f \wedge \Delta g).$$

Proposition 6.5

Δ-convolution is commutative, associative, and idempotent, and is distributive over $\bigoplus$.

Proof:

$$\begin{aligned} f * g &= \Delta(\Delta f \wedge \Delta g) \\ &= \Delta(\Delta g \wedge \Delta f) \\ &= f * g. \end{aligned}$$

$$\begin{aligned} f * (g * h) &= \Delta(\Delta f \wedge (\Delta(\Delta(\Delta g \wedge \Delta h)))) \\ &= \Delta(\Delta f \wedge \Delta g \wedge \Delta h) \\ &= \Delta(\Delta(\Delta(\Delta f \wedge \Delta g)) \wedge \Delta h) \\ &= (f * g) * h. \end{aligned}$$

$$\begin{aligned} f * f &= \Delta(\Delta f \wedge \Delta f) \\ &= \Delta(\Delta f) \\ &= f. \end{aligned}$$

$$\begin{aligned} f * (g \oplus h) &= \Delta(\Delta f \wedge \Delta(g \oplus h)) \\ &= \Delta(\Delta f \wedge (\Delta g \oplus \Delta h)) \\ &= \Delta((\Delta f \wedge \Delta g) \oplus (\Delta f \wedge \Delta h)) \\ &= (f * g) \oplus (f * h). \quad ∎ \end{aligned}$$

Δ-convolution is identical to ordinary convolution of integer polynomials, modulo the Boolean relations $x^2 = x$ and $x + x = 0$.

A_n, the set of functions $B^n \to B$, is a Boolean ring of order 2^{2^n} with respect to $\wedge$, $\oplus$, 0, and 1. It is also a Boolean algebra with respect to $\wedge$, $\vee$, 0, and 1. The same set is a ring with respect to $*$, $\oplus$, 0, and $\Delta(1)$, and an algebra with respect to $*$, $[(x * y) \oplus x \oplus y]$, 0, and $\Delta(1)$. The map taking $\wedge$ to $*$, $\oplus$ to $\oplus$, 0 to 0, and 1 to $\Delta(1)$ induces an isomorphism of algebras and rings. Finally, to any circuit computing $f(x_1, \ldots, x_n)$ over the basis $\{\wedge, \oplus, -\}$ from the elementary functions $x_1, \ldots, x_n$ there corresponds a circuit computing Δf over $\{*, \oplus, \oplus\Delta(1)\}$ from the elementary functions $\Delta(x_1), \ldots, \Delta(x_n)$. Note that convolution is an operator in the function space, but not in B^n.

Proposition 6.6

Let $\alpha, \beta, \gamma \in B^n$. Then

$$(f * g)(\alpha) = \bigoplus_{\gamma \wedge \beta = \alpha} f(\beta) g(\gamma).$$

Proof:

$$\begin{aligned}(f * g)(\alpha) &= \Delta(\Delta f \wedge \Delta g)(\alpha)\\ &= \bigoplus_{x\leq\alpha}(\Delta f \wedge \Delta g)(x)\\ &= \bigoplus_{x\leq\alpha}(\Delta f(x) \wedge \Delta g(x))\\ &= \bigoplus_{x\leq\alpha}\left[\bigoplus_{\beta\leq x} f(\beta) \wedge \bigoplus_{\gamma\leq x} g(\gamma)\right]\\ &= \bigoplus_{\beta,\gamma\leq\alpha} a_{\beta\gamma} f(\beta) g(\gamma),\end{aligned}$$

where $a_{\beta\gamma}$ is the parity of the number of x satisfying $\beta \vee \gamma \leq x \leq \alpha$. The number of such x is an integral power of two; $a_{\beta\gamma}$ is zero if $\beta \vee \gamma < \alpha$, and one if $\beta \vee \gamma = \alpha$. Thus,

$$f * g = \bigoplus_{\gamma\vee\beta=\alpha} f(\beta)g(\gamma). \quad ∎$$

Definition 6.7

A function $f : B^n \to B$ is **Δ-invariant** if $\Delta f = f$.

Theorem 6.8

f is Δ-invariant if and only if $f = \Delta g \oplus g$, for some $g : B^n \to B$.

Proof:

(a) If $f = \Delta g \oplus g$, then

$$\begin{aligned}\Delta f &= \Delta(\Delta g \oplus g)\\ &= \Delta(\Delta g) \oplus \Delta g\\ &= g \oplus \Delta g\\ &= f,\end{aligned}$$

so f is invariant under Δ.

(b) The set of all functions from $B^n \to B$ is a vector space over B of dimension 2^n. Consider the linear function ϕ, which takes $f \mapsto f \oplus \Delta f$. By part (a), the image of ϕ consists entirely of functions which are invariant under Δ. On the other hand, if f is invariant under Δ, $f \bigoplus \Delta f = 0$, and f is in the kernel of ϕ. In fact, the kernel of ϕ consists of exactly those functions that are Δ-invariant, since $\Delta f = f \Rightarrow \phi(f) = 0$. Thus, Image $(\phi) \subseteq$ Kernel (ϕ), whence rank $\phi \leq$ nullity ϕ. Since the rank of ϕ plus the nullity of ϕ is 2^n, rank $(\phi) \leq 2^{n-1}$. In what follows, we will see that the rank of ϕ is greater than or equal to 2^{n-1}; whence, the two quantities must be equal. It will therefore follow that Image $(\phi) =$ Kernel (ϕ), and that every Δ-invariant function is of the form $\Delta f \bigoplus f$.

For $\beta \in B^n$, the functions $a_\beta : B^n \to B$ defined by

$$a_\beta(x) = \begin{cases} 1, & x = \beta \\ 0, & \text{otherwise,} \end{cases}$$

form a basis for the vector space of functions $B^n \to B$.

$$\begin{aligned} \Delta a_\beta(y) &= \bigoplus_{x \leq y} a_\beta(x) \\ &= \begin{cases} 1, & \beta \leq y \\ 0, & \text{otherwise.} \end{cases} \end{aligned}$$

$$\begin{aligned} \phi(a_\beta(y)) &= \Delta a_\beta(y) \bigoplus a_\beta(y) \\ &= \bigoplus_{x < y} a_\beta(x) \\ &= \begin{cases} 1, & \beta < y \\ 0, & \text{otherwise.} \end{cases} \end{aligned}$$

Let b be a fixed atom in B^n, and consider the set of functions $\{\phi(a_\beta)\}$ for which $\beta \wedge b = 0$, i.e., the set of $\phi(a_\beta)$ for which β has a zero b^{th} component. There are 2^{n-1} such functions; we have only to check that they are linearly independent.

Suppose some subset of these functions sums to zero, i.e., suppose there is a subset $S \subseteq B^n$ such that $\beta \wedge b = 0$ for all $\beta \in S$, and

$$\bigoplus_{\beta \in S} \phi(a_\beta(y)) = 0$$

for all $y \in B^n$. Let β' be some minimal element in S, i.e., an element such that $\beta \not< \beta'$ for any $\beta \in S$. Now consider the element $\beta' \vee b \in B^n$. Since $\beta' \wedge b = 0$,

$$\beta' < \beta' \vee b$$

and

$$\phi a_{\beta'}(\beta' \vee b) = 1.$$

On the other hand, suppose β is some element of S not equal to β'. If $\beta < \beta' \vee b$, then

$$\begin{aligned} \beta \vee (\beta' \vee b) &= \beta' \vee b \\ \bar{b}(\beta \vee \beta' \vee b) &= \bar{b}(\beta' \vee b) \\ \beta \vee \beta' &= \beta' \\ \beta &\leq \beta', \end{aligned}$$

a contradiction to the minimality of β'. Therefore, $\beta \not< \beta' \vee b$, and

$$\phi a_\beta(\beta' \vee b) = 0.$$

Thus,

$$\bigoplus_{\beta \in S} \phi a_\beta(\beta' \vee b) = 1.$$

But this is contradicts the assumption that the ϕa_β sum to the zero function. Thus, the ϕa_β are independent. ∎

Corollary 6.8.1

The number of Δ-invariant functions $B^n \to B$ is $2^{2^{n-1}}$.

Example 6.8.2

There are exactly $2^{2^3} = 16$ functions $B^3 \to B$ that are invariant under Δ, namely:

$$0$$

$$x_1 x_2 x_3$$

$$x_1x_3 \oplus x_2x_3$$

$$x_1x_2 \oplus x_2x_3$$

$$x_1x_2 \oplus x_1x_3$$

$$x_1x_3 \oplus x_2x_3 \oplus x_1x_2x_3$$

$$x_1x_2 \oplus x_2x_3 \oplus x_1x_2x_3$$

$$x_1x_2 \oplus x_1x_3 \oplus x_1x_2x_3$$

$$x_1 \oplus x_2 \oplus x_3 \oplus x_2x_3$$

$$x_1 \oplus x_2 \oplus x_3 \oplus x_1x_3$$

$$x_1 \oplus x_2 \oplus x_3 \oplus x_1x_2$$

$$x_1 \oplus x_2 \oplus x_3 \oplus x_2x_3 \oplus x_1x_2x_3$$

$$x_1 \oplus x_2 \oplus x_3 \oplus x_1x_3 \oplus x_1x_2x_3$$

$$x_1 \oplus x_2 \oplus x_3 \oplus x_1x_2 \oplus x_1x_2x_3$$

$$x_1 \oplus x_2 \oplus x_3 \oplus x_1x_2 \oplus x_2x_3 \oplus x_1x_3$$

$$x_1 \oplus x_2 \oplus x_3 \oplus x_1x_2 \oplus x_2x_3 \oplus x_1x_3 \oplus x_1x_2x_3.$$

Theorem 6.8.3

Every Δ-invariant function $B^n \to B$ is of the form

$$\bigoplus_{\beta \in B^{n-1}} \phi(x_1^{\beta_1} \dots x_{n-1}^{\beta_{n-1}}),$$

where $\phi(f) = \Delta f \oplus f$.

Proof:

Let $\beta = (\beta_1, \dots, \beta_{n-1}, 0)$.

$$x_1^{\beta_1}\cdots x_{n-1}^{\beta_{n-1}} = x_1^{\beta_1}\cdots x_{n-1}^{\beta_{n-1}}x_n^0$$

$$= \begin{cases} 1, & \beta \leq x \\ 0, & \text{otherwise.} \end{cases}$$

$$\Delta(x_1^{\beta_1}\cdots x_{n-1}^{\beta_{n-1}}) = x_1^{\beta_1}\cdots x_{n-1}^{\beta_{n-1}}\overline{x}_n$$

$$= \begin{cases} 1, & \beta = x \\ 0, & \text{otherwise.} \end{cases}$$

Using the notation of Theorem 6.8.1:

$$\phi(x_1^{\beta_1}\cdots x_{n-1}^{\beta_{n-1}}) = \begin{cases} 1, & \beta < x \\ 0, & \text{otherwise.} \end{cases}$$
$$= \phi a_\beta(x),$$

where $\beta_n = 0$. By part (b) of the proof of that theorem, the $\phi a_\beta(x)$ with $\beta_n = 0$ are linearly independent as vectors over B. These 2^{n-1} vectors are Δ-invariant by part (a) of Theorem 6.8.1, and therefore are a basis of the 2^{n-1}-dimensional subspace of Δ-invariant functions $B^n \to B$. ∎

Example 6.8.3.1

There are exactly 256 Δ-invariant functions $B^4 \to B$. Every Δ-invariant function is a linear combination of the following eight functions:

$$\phi(1) = \overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4 \oplus 1$$

$$\phi(x_1) = x_1\overline{x}_2\overline{x}_3\overline{x}_4 \oplus x_1$$

$$\phi(x_2) = \overline{x}_1 x_2\overline{x}_3\overline{x}_4 \oplus x_2$$

$$\phi(x_3) = \overline{x}_1\overline{x}_2 x_3\overline{x}_4 \oplus x_3$$

$$\phi(x_1x_2) = x_1x_2\overline{x}_3\overline{x}_4 \oplus x_1x_2$$

$$\phi(x_1x_3) = x_1\overline{x}_2x_3\overline{x}_4 \oplus x_1x_3$$

$$\phi(x_2x_3) = \overline{x}_1x_2x_3\overline{x}_4 \oplus x_2x_3$$

$$\phi(x_1x_2x_3) = x_1x_2x_3x_4 \oplus x_1x_2x_3.$$

Recall that $\mathcal{P}$ is the set of languages accepted by a Turing machine (TM) in time $P(n)$, where P is a polynomial in the length of the input. $\mathcal{NP}$ is the set of languages accepted by a nondeterministic Turing machine (NTDM) in polynomial time.

The next three definitions are by L. G. Valiant [VL3, VL4].

Definitions 6.9

A **counting Turing machine** is a NTDM with an auxiliary output device that instantly computes and prints the number of accepting computations induced by the input. A counting TM has **time complexity** $f(n)$ if the longest accepting computation induced by the set of all inputs of size n takes $f(n)$ steps. $\#\mathcal{P}$ is the class of functions that can be computed by counting TMs in polynomial time and $\#_k\mathcal{P}$ is the class of functions that can be computed in polynomial time by counting TMs with (mod k) arithmetic.

In [VL4], Valiant shows that the problem of computing the permanent is $\#\mathcal{P}$-complete, and that for all k that are not powers of 2, the problem of computing the permanent (mod k) is $\#_k\mathcal{P}$-complete. Computing the permanent over the integers is believed to be a very hard problem, and no fast procedures are known.

The case $k = 2^r$ behaves a little differently from other values of k. Valiant's argument for the $\#_k\mathcal{P}$-completeness of the permanent (mod k) is inapplicable for these values of k. At the same time, note that for $k = 2$ computing the permanent is identical to computing the determinant, a problem for which efficient algorithms are available. Therefore, it is not immediately clear whether $k = 2$ is a special case because functions in $\#_2\mathcal{P}$ are easy, or because they are hard.

Proposition 6.10

The following problem is $\#_2\mathcal{P}$-complete:

COMPUTING Δf

Input: A circuit computing $f : B^n \rightarrow B$ and an n-tuple $X \in B^n$

Property: $\Delta f(X) = 1$

Proof:

COMPUTING Δf is in $\#_2 P$ since Δf can be computed by nondeterministically selecting each possible Boolean n-tuple, checking if it is less than or equal to X, and if so, simulating the computation of the circuit on that n-tuple. $\Delta f(X) = \bigoplus_{Z \leq X} f(Z)$, so $\Delta f(X) = 1$ if and only if the number of accepting computations (mod 2) is 1.

On the other hand, for $f : B^n \rightarrow B$,

$$\Delta f(1) = \bigoplus_{Z \in B^n} f(Z)$$
$$= \begin{cases} 1, & \text{the number of } Z \text{ with } f(Z) = 1 \text{ equals 1 (mod 2)} \\ 0, & \text{the number of } Z \text{ with } f(Z) = 1 \text{ equals 0 (mod 2)} \end{cases}$$

A polynomial time algorithm can be simulated with a polynomial size circuit. Therefore a fast procedure for computing $\Delta f(X)$ from a circuit for f would give a fast procedure for solving any problem in $\#_2 P$ — simulate the NTDM with a circuit and evaluate $\Delta f(1)$. ∎

Valiant and Vazirani [VAV] have recently shown that a fast random procedure for solving problems in $\#_2 P$ would imply the existence of fast random procedures for any problem in $\mathcal{NP}$.

Various simple functions $B^n \rightarrow B$ like the linear functions, elementary symmetric functions, and threshold functions, which are known to have $O(n)$ circuit complexity, also have Δ-transforms with $O(n)$ circuit complexity. Given these examples, and the notion that a polynomial and its coefficients ought to have similar complexity under a robust complexity measure, it would be interesting to know whether the existence of a small circuit for f always implies the existence of a small circuit for Δf.

— Chapter Seven —

SET COMPLEXITY

This chapter introduces the fundamental notion of *set complexity,* and relates it to previously-defined measures of combinatorial complexity. A number of new results are presented which reduce open questions about circuit complexity to questions about set complexity which, on the surface at least, appear to be more amenable to attack.

Before defining set complexity, let us review the measures of complexity discussed in earlier chapters.

The circuit complexity of a set of functions from $B^n \to B$, or of a function from $B^n \to B^m$, is the size of the smallest circuit which computes the coordinate functions $f_1, f_2, \ldots, f_m$ from the elementary functions $x_1, x_2, \ldots, x_n$, where a node of the circuit is one of the 16 two-input, one-output Boolean operators. By extension, the circuit complexity of an $m \times n$ Boolean matrix $[a_{ij}]$ is the circuit complexity of the m linear functions

$$f_i(x_1, \ldots, x_n) = \bigoplus_{j=1}^{n} a_{ij}x_j, \qquad 1 \le i \le m.$$

The *linear complexity* of an $m \times n$ Boolean matrix is the size of the smallest circuit computing the m linear functions

$$f_i(x_1, \ldots, x_n) = \bigoplus_{j=1}^{n} a_{ij}x_j, \qquad 1 \le i \le m,$$

where a gate takes h_k and h_l to $h_k \oplus h_l$. Since each function produced during the computation is linear, the circuit can be thought of as acting on n-tuples, where

$(a_1, a_2, \ldots, a_n)$ is shorthand for $a_1x_1 \oplus a_2x_2 \oplus \cdots \oplus a_nx_n$. Since a_i has either the value 0 or the value 1, one can also think of the circuit as acting on subsets of an n-element set $S = \{s_1, \ldots, s_n\}$, where the n-tuple $(a_1, \ldots, a_n)$ corresponds to the subset $A \subseteq S$, in which $s_i \in A$ if and only if $a_i = 1$. Thus, the basic operation of the circuit takes the sets A and A' to the symmetric difference of A and A'. Therefore, the problem of computing the linear functions $f_1, \ldots, f_m$ from the elementary functions $x_1, \ldots, x_n$ becomes that of synthesizing the characteristic sets corresponding to the f_i from the atomic sets $\{s_1\}, \{s_2\}, \ldots, \{s_n\}$ using only the symmetric difference operator.

Definition 7.1

The set **complexity** of an $m \times n$ Boolean matrix $[a_{ij}]$ associated with the linear functions

$$f_i(x_1, \ldots, x_n) = \bigoplus_{j=1}^{n} a_{ij}x_j, \qquad 1 \leq i \leq m,$$

is the size of the smallest circuit computing the characteristic sets of the f_i from the atomic sets $\{s_1\}, \{s_2\}, \ldots, \{s_n\}$, over the basis of the 16 two-input, one-output set operations.

By restricting the class of operators, one can define $\cup$-set **complexity** $(C_{\{\cup-set\}})$, in which only union of sets is permitted, or **monotone set complexity** $(C_{monoset})$, in which only union and intersection are permitted. ∇-**set complexity** and $\cap$-**set complexity** can be defined analogously. However, the former measure is exactly identical to the linear complexity $C_{\oplus}$ of Chapter 2, while $\cap$-set complexity is relatively uninteresting, since it is dual to $\cup$-set complexity and since no non-trivial sets can be produced from the disjoint atoms $\{s_1\}, \ldots, \{s_n\}$ by intersection.

The $\cup$-set complexity of a matrix $[a_{ij}]$ is identical to the circuit complexity over the basis $\{\vee\}$ of the functions $\{f_i\} : B^n \rightarrow B$ defined by

$$f_i(x_1, \ldots, x_n) = \bigvee_{j=1}^{n} a_{ij}x_j,$$

and to the complexity over the basis $\{\wedge\}$ of the functions $\{g_i\} : B^n \rightarrow B$ defined by

$$g_i(x_1, \ldots, x_n) = \bigwedge_{j=1}^{n} a_{ij} x_j.$$

Example 7.1.1:

The following row vectors are associated with the nodes of a circuit computing a function whose complexity is $C(F) = 4$:

$$x_1$$

$$x_2$$

$$x_3$$

$$x_4$$

$$x_1 \wedge x_2$$

$$(x_1 \wedge x_2) \bigoplus x_3$$

$$x_3 \vee x_4$$

$$(x_1 \wedge x_2) \bigoplus (x_3 \vee x_4).$$

Example 7.1.2:

The following functions are associated with the nodes of a circuit computing a function whose complexity is $C_{\oplus}(F) = 4$:

$$x_1$$

$$x_2$$

$$x_3$$

$$x_4$$

$$x_1 \bigoplus x_2$$

$$x_1 \bigoplus x_3$$

$$x_1 \bigoplus x_3 \bigoplus x_4$$

$$x_2 \bigoplus x_3 \bigoplus x_4.$$

Example 7.1.3:

The following row vectors are associated with the nodes of a circuit computing a matrix with complexity $C_{set}([a_{ij}]) = 7$:

row vector	*characteristic set*
$(1,0,0,0,0)$	$\{s_1\}$
$(0,1,0,0,0)$	$\{s_2\}$
$(0,0,1,0,0)$	$\{s_3\}$
$(0,0,0,1,0)$	$\{s_4\}$
$(0,0,0,0,1)$	$\{s_5\}$
$(1,0,0,0,1)$	$\{s_1,s_5\} = \{s_1\} \bigcup \{s_5\}$
$(1,0,0,1,1)$	$\{s_1,s_4,s_5\} = \{s_1,s_5\} \bigcup \{s_4\}$
$(1,1,0,0,0)$	$\{s_1,s_2\} = \{s_1\} \bigoplus \{s_2\}$
$(1,1,1,0,0)$	$\{s_1,s_2,s_3\} = \{s_1,s_2\} \bigcup \{s_3\}$
$(1,1,1,1,1)$	$\{s_1,s_2,s_3,s_4,s_5\} = \{s_1,s_4,s_5\} \bigcup \{s_1,s_2,s_3\}$
$(0,0,1,1,1)$	$\{s_3,s_4,s_5\} = \{s_1,s_2,s_3,s_4,s_5\} \bigoplus \{s_1,s_2\}$
$(0,0,0,1,1)$	$\{s_4,s_5\} = \{s_1,s_4,s_5\} \bigcap \{s_3,s_4,s_5\}$.

For a given $m \times n$ matrix $[a_{ij}]$, the following relations follow immediately from the definitions:

$$C_{set}([a_{ij}]) \leq C_{monoset}([a_{ij}]) \leq C_{\bigcup -set}([a_{ij}])$$

$$C_{set}([a_{ij}]) \leq C_{\nabla -set}([a_{ij}]) = C_{\bigoplus}\left(\{ \bigoplus_{j=1}^{n} a_{ij}x_j \}_{1 \leq i \leq m} \right)$$

$$C\left(\{ \bigoplus_{j=1}^{n} a_{ij}x_j \}_{1 \leq i \leq m} \right) \leq C_{\bigoplus -set}([a_{ij}]).$$

Each one of the 2^{n^2} $n \times n$ Boolean matrices can be generated from the identity function under each of these models of computation. By using the constructions of Theorem 3.12 or Lemma 3.11, it is possible to construct any $n \times n$ matrix in

$\sim n^2/2 \log n$ steps, or any $n \times 2^n$ matrix in $2^{n+1} - 2n - 2$ steps under either the linear model or the $\cup$-set model, and hence under any of the models.

On the other hand, by counting the number of n-output circuits that can be constructed with a given number of gates, as in Theorem 2.5.4, one can show that for each of these models some $n \times n$ matrix has complexity $\sim n^2/2 \log n$. Thus, the hardest functions in each model have the same asymptotic complexity.

Set complexity is equivalent to the circuit complexity of a Boolean function on a finite, non-Boolean domain.

Proposition 7.1.4

Suppose $[a_{ij}]$ is an $m \times n$ Boolean matrix and $[b_{ij}]$ is an $k \times n$ Boolean matrix. Then

$$C_{\cap-set}([a_{ij}] \mid [b_{ij}]) = C_{\cap}(F \mid G),$$

where $F : \{1, 2, \ldots, n\} \to B^m$ has coordinate functions

$$f_i(j) = a_{ij}, \qquad 1 \le i \le m,\ 1 \le j \le n,$$

and where $G : \{1, 2, \ldots, n\} \to B^k$ has coordinate functions

$$g_i(j) = b_{ij}, \qquad 1 \le i \le k,\ 1 \le j \le n.$$

Proof:

The function $f_i(j)$ is the characteristic function of the set of entries with value 1 in the i^{th} row of $[a_{ij}]$. The function $g_l(j)$ is the characteristic function of the set of entries with value 1 in the l^{th} row of $[b_{lj}]$. If $\omega \in \{B^2 \to B\}$, then $\omega(h_i, h_l)(j) = 1$ if and only if $\omega(h_{ij}, h_{lj}) = 1$. Thus, any circuit for $[a_{ij}] \mid [b_{ij}]$ in the set complexity model gives a circuit for $F \mid G$ in the circuit complexity model, and vice versa. ∎

In particular, $C_{\cap-set}([a_{ij}]) = C_{\cap}(F \mid G)$, where $G : \{1, 2, \ldots, n\} \to B^k$ has coordinate functions

$$g_i(j) = \begin{cases} 1, & j = i \\ 0, & \text{otherwise.} \end{cases}$$

Since set complexity measures the number of operations necessary to compute one set of subsets of a finite set from another, this measure does not depend on how either the elements of the finite set or the target subsets are ordered. Thus, permuting the rows and columns of a matrix does not change its set complexity.

Proposition 7.1.4.2

If $[A]$ is an arbitrary Boolean matrix, and $[P]$ and $[Q]$ are permutation matrices, then $C_{set}([PAQ]) = C_{set}([A])$.

Proof:

The inputs or outputs of a circuit can be permuted without changing the circuit's complexity, so $C_{set}([P_1 A] \mid [P_2 B]) = C_{set}([A] \mid [B])$ for any permutations P_1 and P_2. Therefore $C_{set}([PAQ]) = C_{set}([PAQ] \mid [I]) = C_{set}([AQ] \mid [I])$. By Corollary 2.2.1.3, $C_{set}([AQ] \mid [I]) = C_{set}([A] \mid [Q^{-1}])$. Since Q is a permutation, this last quantity equals $C_{set}([A] \mid [I]) = C_{set}([A])$. ∎

The formulas for computing $C_\Omega(F \mid G)$ that were given in Chapter 2 remain valid for their set complexity analogs, since the proofs of these theorems were independent of the domain of the functions F and G. According to the following result, it is not difficult to check whether $[G]$ generates $[F]$.

Theorem 7.1.5

For $\Omega = \{\omega : B^2 \rightarrow B\}$, $\{\oplus\}$, $\{\wedge, \vee\}$, or $\{\vee\}$, the following problem is $\in \mathcal{P}$:

BOOLEAN GENERATORS
Input: An $m \times n$ Boolean matrix $[F]$ and an $k \times n$ Boolean matrix $[G]$.
Property: The rows of G generate the rows of F over Ω.

Proof:

In the case of any of these four bases, checking whether $[G]$ generates $[F]$ can be performed in a number of steps that is polynomial in n, m, and k.

Suppose $\Omega = \{E : B^2 \to B\}$. By Theorem 1.12, $[G]$ generates $[F]$ if and only if whenever two columns of $[F]$ are distinct, the corresponding columns of $[G]$ are distinct. Thus, the required checking can be performed by sorting the columns of $[F]$ and comparing them with the corresponding columns of $[G]$.

Suppose $\Omega = \{\bigoplus\}$. Whether $[G]$ generates $[F]$ can be checked by forming the $(m+k) \times n$ matrix $[H]$ whose first m rows are identical to those of $[F]$ and whose last k rows are identical to those of $[G]$, and then putting $[G]$ and $[H]$ into Hermite normal form via Gaussian elimination. $[G]$ generates $[F]$ if and only if $[G]$ and $[H]$ have the same rank.

Suppose that $\Omega = \{\wedge, \vee\}$ and that $[f_i]$ is a row in $[F]$. One can check if $[G]$ generates $[f_i]$ by checking if

$$[f_i] = \bigvee_{j|f_{ij}=1} \left(\bigwedge_{l|g_{lj}=1} [g_l] \right).$$

Suppose $\Omega = \{\vee\}$ and $[f_i]$ is a row in $[F]$. $[G]$ generates $[f_i]$ if and only if

$$[f_i] = \bigvee_{g_l \leq f_i} [g_l].$$

For fixed i and j, only n comparisons are required to determine if $g_l \leq f_i$. ▮

Next, a result analogous to Proposition 2.7 is presented. As in the case of circuit complexity and linear complexity, savings in operations can be had in the set complexity model when computing the same function on disjoint sets of variables.

Theorem 7.1.6

Let $[a_{ij}]$ be an $m \times n$ Boolean matrix, and let $C_{set}([a_{ij}]^{(k)})$ denote the number of set operations necessary to compute k disjoint copies of $[a_{ij}]$. Then

$$C_{set}([a_{ij}]^{(k)}) \leq C_{set}([a_{ij}]) + 2nk + mk - n - k.$$

Proof:

The matrix $[a_{ij}]^{(k)}$ corresponding to k disjoint copies of $[a_{ij}]$ has dimensions $km \times kn$. The $(pk+i,\ qk+j)^{\text{th}}$ entry of $[a_{ij}]^{(k)}$ is a_{ij} if $p = q$ and is 0 if $p \neq q$.

$$[a_{ij}]^{(k)} = \begin{bmatrix} [a_{ij}] & [0] & \cdots & [0] \\ [0] & [a_{ij}] & \cdots & [0] \\ \vdots & \vdots & \vdots & \vdots \\ [0] & [0] & \cdots & [a_{ij}] \end{bmatrix}.$$

The following procedure synthesizes $[a_{ij}]^{(k)}$. First, form the $n \times kn$ matrix consisting of k copies of the $n \times n$ identity matrix:

$$[I_{k,n}] = [[I] \quad [I] \quad \cdots \quad [I]].$$

$[I_{k,n}]$ has set complexity $n(k-1)$. Next, make an $m \times kn$ matrix consisting of k copies of $[a_{ij}]$:

$$[A_k] = [[a_{ij}] \quad [a_{ij}] \quad \cdots \quad [a_{ij}]].$$

The set complexity of $[A_k]$ given $[I_{k,n}]$ is the same as that of $[a_{ij}]$. Next, form the $k \times kn$ matrix

$$[U_{k,n}] = \begin{bmatrix} [1] & [0] & \cdots & [0] \\ [0] & [1] & \cdots & [0] \\ \vdots & \vdots & \vdots & \vdots \\ [0] & [0] & \cdots & [1] \end{bmatrix},$$

where the submatrices [1] and [0] have dimension $1 \times n$ and are filled with 1's and 0's respectively. The set complexity of $[U_{k,n}]$ is $k(n-1)$. Finally, compute the meet of each row of $[A_k]$ and each row of $[U_{k,n}]$. The resulting matrix is $[a_{ij}]^{(k)}$. Thus,

$$C_{set}([A^{(k)}]) = C_{set}([a_{ij}]) + n(k-1) + k(n-1) + km. \quad \blacksquare$$

Therefore if $[a_{ij}]$ is a fixed $n \times n$ matrix, $\lim_{k\to\infty}(1/nk)C_{set}([a_{ij}]^{(k)}) \leq 3 + o(kn)$. That is, in the limit the number of operations per input to necessary to compute $[A^{(k)}]$ does not exceed 3.

Set complexity can be related to the circuit complexity of Boolean functions in the neighborhood of 0.

Definition 7.2

Let $F : B^n \to B^m$, have coordinate functions $f_1, \dots, f_m : B^n \to B$. The impulse matrix of F at $(0, \dots, 0)$, $[I_F]$, is the $m \times n$ Boolean matrix whose ij^{th} entry is

$$f_i(0, \dots, 0) \oplus \frac{\partial f_i}{\partial f_j}(0, \dots, 0) = f_i(x_1, \dots, x_n)|_{\substack{x_j=1 \ \ if \ \ k=j \\ x_k=0 \ \ otherwise}} .$$

$[I_F]$ is equal to $[T_F(0) - F(0)]$, where T_F is the tangent to F.

Proposition 7.3

Let $F : B^n \to B^m$ have impulse matrix $[I_F]$. Then

$$C_{\cap-set}([I_F]) \leq C_\cap(F).$$

Proof:

If the circuit complexity of F is $C(F)$ it means that there exists a circuit with $C(F)$ gates which computes F from $\{x_1, \dots, x_n\}$. If so, consider the restriction of F to elements in B^n of weight 1. Mapping 1 to x_1, 2 to x_2, and so on, this restriction can be pulled back to a function from $\{1, 2, \dots, n\}$ to B^m. In the same way, the functions $x_1, x_2, \dots, x_n$ from $B^n \to B$ correspond to the functions

$$g_i(j) = \begin{cases} 1, & j = i \\ 0, & \text{otherwise} \end{cases}$$

from $\{1, 2, \dots, n\}$ to B. By Corollary 7.1.4.1, there must be a circuit with $C(F)$ gates computing $[I_F]$ over the set complexity model. ∎

Example 7.3.1

SL program from circuit	*polynomial representation*	*row of impulse matrix*	*set*
x_1	x_1	1000	$\{s_1\}$
x_2	x_2	0100	$\{s_2\}$
x_3	x_3	0010	$\{s_3\}$
x_4	x_4	0001	$\{s_4\}$
$x_1 \wedge x_2$	x_1x_2	0000	$\emptyset$
$(x_1 \wedge x_2) \vee x_3$	$x_1x_2x_3 \oplus x_1x_2 \oplus x_3$	0010	$\{s_3\}$
$\overline{(x_1 \wedge x_2) \vee x_3}$	$x_1x_2x_3 \oplus x_1x_2 \oplus x_3 \oplus 1$	1101	$\{s_1, s_2, s_4\}$
$x_3 \vee x_4$	$x_3x_4 \oplus x_3 \oplus x_4$	0011	$\{s_3, s_4\}$
$\overline{((x_1 \wedge x_2) \vee x_3)} \oplus (x_3 \vee x_4)$	$x_1x_2x_3 \oplus x_1x_2 \oplus x_3x_4 \oplus x_4 \oplus 1$	1110	$\{s_1, s_2, s_3\}$

Thus, if there is a circuit computing F over the basis Ω, there is a circuit of the same size which computes $[I_F]$ in the Ω-set model.

The next result equates the set complexity of a matrix to the circuit complexity of a partially-specified Boolean function, or equivalently, to the complexity of the easiest member of a large set of Boolean functions.

Theorem 7.4

Let $[a_{ij}]$ be an $m \times n$ Boolean matrix. Then

$$C_{\Omega-set}([a_{ij}]) = \min_{F:B^n \to B^m} \left\{ C_\Omega(F) \mid [I_F(0)] = [a_{ij}] \right\}.$$

In other words, the set complexity of an $m \times n$ matrix is equal to the circuit complexity of the easiest function $B^n \to B^m$ with that impulse matrix.

Proof:

By the proof of Proposition 7.3, if there exists a circuit for any F with $[I_F] = [a_{ij}]$, there must be a circuit over the set complexity model for $[a_{ij}]$ with the same number of gates. On the other hand, if there exists a circuit for $[a_{ij}]$ over the set model, a circuit with the same Boolean gates computes a function $B^n \rightarrow B^n$ whose impulse matrix is $[a_{ij}]$. Alternatively, the result follows directly from Proposition 7.1.4 and Corollary 2.2.1.1. ∎

Corollary 7.4.1

If F is a linear function with matrix $[a_{ij}]$, then

$$C_{set}([a_{ij}]) \leq C(F) \leq C_{\oplus}(F).$$

Corollary 7.4.2

$$\begin{aligned} C_{set}([a_{ij}]) &\leq C\left(\{\bigvee_{j=1}^{m} a_{ij}x_j\}_{1\leq i\leq m}\right) \\ &\leq C_{\{\vee\}}\left(\{\bigvee_{j=1}^{m} a_{ij}x_j\}_{1\leq i\leq m}\right). \end{aligned}$$

Corollary 7.4.3

Let $F : B^n \rightarrow B^n$, and let $J_F(x_1, \ldots, x_n)$ be the Jacobian of F evaluated at the point $(x_1, \ldots, x_n) \in B^n$. Then

$$C_{set}(J_F(x_1, \ldots, x_n)) \leq C(F) + n.$$

Proof:

Consider the restriction of F to elements in B^n differing from $(x_1, \ldots, x_n)$ in a single component. A circuit for this restriction corresponds to a circuit in the set complexity model which computes $J_F(x_1, \ldots, x_n)$ from the $n \times n$ matrix whose ij^{th} entry is $x_i \oplus 1$ when $i = j$ and is x_i otherwise. Since this last matrix is simply an identity matrix whose i^{th} row is complemented if and only if $x_i = 1$, its set complexity does not exceed n. The triangle inequality gives the stated result. ∎

The next result shows that it is possible to determine $C_{set}([a_{ij}])$ exactly for the hardest matrices with m rows and more than $2^m - m - 2$ columns.

Theorem 7.4.4

$$C_{set}([\mathcal{F}ull_m]) = 2^{m+1} - 3m - 2.$$

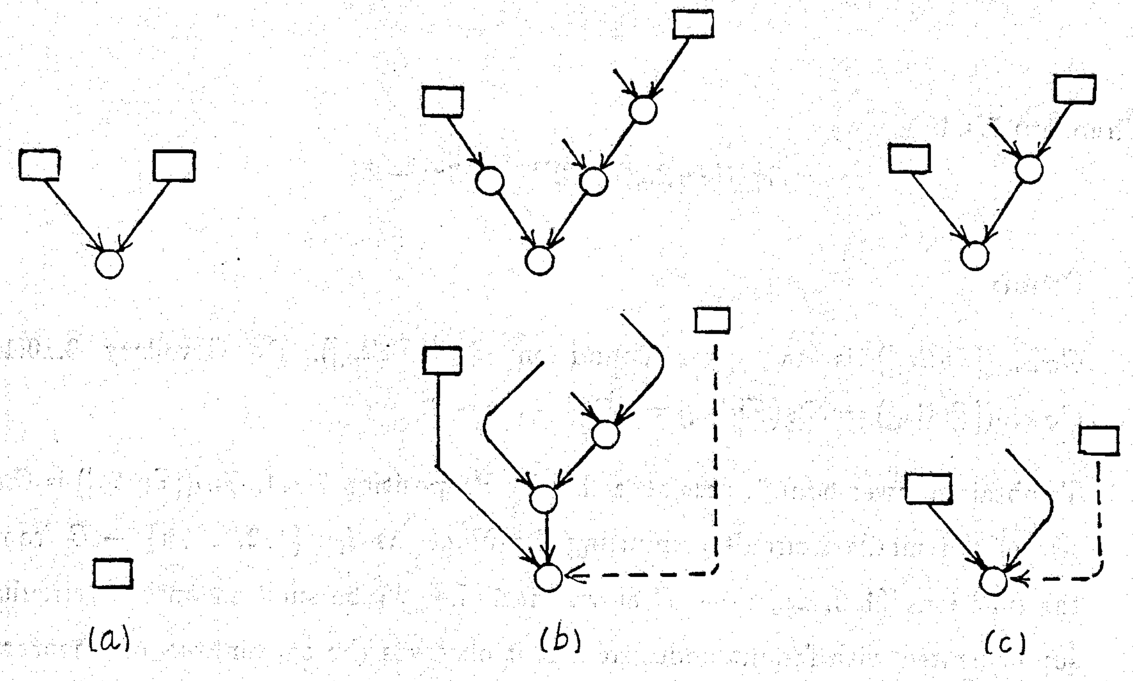
Proof:

$C_{\nabla-set}([\mathcal{F}ull_m])$ is an upper bound on $C_{set}([\mathcal{F}ull_m])$. By Corollary 3.10.1, $C_{\nabla-set}([\mathcal{F}ull_m]) = C_{\oplus}(\mathcal{F}ull_m) = 2^{m+1} - 3m - 2$.

To obtain a lower bound, first note that by Proposition 7.1.4, $C_{set}([\mathcal{F}ull_m])$ is the size of the smallest circuit computing the functions $f_i : \{1, 2, \ldots, n\} \to B$ from the functions $\{1, 2, \ldots, n\} \to B$ of weight 1. Let $\mathcal{N}_1$ be such an optimal circuit, supplemented with fanout nodes so that it observes the conventions of Theorem 3.2. $\mathcal{N}_1$ has $2^m - m - 1$ inputs, m outputs, and $C_{set}([\mathcal{F}ull_m])$ computational nodes. Since each computational node has two inputs and one output and each fanout node has one input and two outputs, for the number of sources and sinks in the network to be equal, the number of fanout nodes in $\mathcal{N}_1$ must satisfy $f + n + 2\,C_{set}([\mathcal{F}ull_m]) = 2^m - m - 1 + 2f + C_{set}([\mathcal{F}ull_m])$. Thus, $C_{set}([\mathcal{F}ull_m]) = f + 2^m - 2m - 1$.

Consider the circuit $\mathcal{N}_2$ obtained by deleting from $\mathcal{N}_1$ every wire leaving a fanout node. Since $\mathcal{N}_2$ has no fanouts it consists of a collection of disconnected components, each of which is a binary tree. If $\mathcal{N}_1$ is optimal, each input vertex has at least one output vertex as a descendant, so the root of each of the trees in $\mathcal{N}_2$ which contains an input vertex must be either an output or a fanout node.

Suppose that some tree in $\mathcal{N}_2$ contains two input vertices, say ones corresponding to the functions $(g_i(x) = 1 \Leftrightarrow x = 1)$ and $(g_j(x) = 1 \Leftrightarrow x = j)$. By hypothesis, these two inputs share a common descendant in $\mathcal{N}_1$ to which they are each connected without passing through any fanout node. Consider the function computed by this common node. If $h(i) = h(j)$, then every node in $\mathcal{N}_1$ which is a descendent of the common node, including the outputs of the circuit, must compute functions which have identical values on the arguments i and j. But this presents a contradiction since the i^{th} and j^{th} columns of $[\mathcal{F}ull_m]$ are distinct. Suppose then that $h(i) \neq h(j)$. But this is also a contradiction, since $\mathcal{N}_1$ is not optimal. Consider the following cases:



Two Inputs in the Same Tree of $\mathcal{N}_2$

(a) Both input vertices are connected directly to the common computational node. If $h(i) = 1$ and $h(j) = 0$, the common node can be replaced by g_i. If $h(i) = 0$ and $h(j) = 1$, the common node can be replaced by g_j.

(b) Both input vertices are separated from the common node by at least one other node. In this case both input vertices can be disconnected from the circuit, and the first computational nodes beneath the inputs eliminated. Since g_i and g_j are 0 on all arguments other than i and j, the two deleted computational nodes must act identically on all arguments different from i and j. That is, the two deleted nodes each compute the identity function, the complement function, or a constant function on all inputs not equal to i or j. This computation can be absorbed into the next lower node(s) of $\mathcal{N}_1$. When g_i and g_j are disconnected, the common node computes a function that is identical on i and j; this function can be restored to its correct value on these arguments by adding to it either g_i or g_j. This costs one gate, but two gates were saved by disconnecting the inputs g_i and g_j, so the resulting circuit computes the same functions as $\mathcal{N}$ and has one fewer gate.

(c) One input vertex is separated from the common node by at least one other node but the other vertex is connected to it directly. As in case (b), each input can be disconnected and the first computational node beneath the nodes eliminated. The common node, which is deleted in this process, is replaced by a node fed by one of the two input vertices to ensure that the correct value of the function is computed on inputs i and j. The action of the topmost deleted node on inputs different from i and j can be absorbed into the first computational node beneath it, and that of the common node into the new node which replaced it.

Therefore each tree in $\mathcal{N}_2$ contains at most one input vertex. However every row of $[\mathcal{F}ull_m]$ has weight 2^{m-1}, so any tree whose root is an output vertex cannot contain input vertices. Therefore each input vertex is contained in a distinct tree whose root is a fanout node, and since there are $2^m - m - 1$ input vertices, $\mathcal{N}_2$ must contain at least $2^m - m - 1$ distinct fanout nodes. $\mathcal{N}_1$ also contains this many fanouts, whence $C_{set}([\mathcal{F}ull_m]) \geq 2^{m+1} - 3n - 2$. ∎

Corollary 7.4.4.1

Suppose $[a_{ij}]$ is a $\log n \times n$ matrix. Then $C_{set}([a_{ij}]) \leq 2^{m+1} - 2m - 1$. This maximum is attained if and only if $[a_{ij}]$ contains $[\mathcal{F}ull_{\log n}]$ as a submatrix and $[a_{ij}]$ contains no columns consisting entirely of zeros.

Proof:

If $[a_{ij}]$ has k columns which are identical, then $C_{set}([a_{ij}]) = k - 1 + C_{set}([b_{ij}])$, where $[b_{ij}]$ is $[a_{ij}]$ with all but one of the duplicate columns deleted. If $[a_{ij}]$ has a column of weight 1 then $C_{set}([a_{ij}]) = 1 + C_{set}([b_{ij}])$, where $[b_{ij}]$ is $[a_{ij}]$ with the singleton column removed. If $[a_{ij}]$ has a column consisting entirely of 0's then $C_{set}([a_{ij}]) = C_{set}([b_{ij}])$, where $[b_{ij}]$ is $[a_{ij}]$ with the column of 0's removed. ∎

The next result shows that the set complexity and monotone set complexity of any matrix are identical, up to a constant factor. This is very different from the model of circuit complexity, where complexity over the monotone basis and over the complete basis can be quite different. For example, Boolean matrix multiplication requires $O(n^3)$ gates over the monotone basis [MEL, PAT] but can be performed with $O(n^{2.81} \log^2 n)$ gates over the basis $\{\omega : B^2 \to B\}$ [FME]. Paul [PAU] has even shown the existence

If gate g_i is connected to gate g_j in a circuit over the $\bigcup$-set model, and f_i and f_j are the n-tuples computed by g_i and g_j respectively, then $f_j \geq f_i$. Therefore, each row of $[a_{ij}]$ must be synthesized independently. Each row contains k 1's, so its synthesis requires $k-1$ $\bigcup$ operations; therefore $C_{\bigcup-set}([a_{ij}]) \sim k^3 = k^{1.5}$.

If $[a_{ij}]_n$ is the matrix described above when $n = k^2$ and the $n \times n$ zero matrix for other values of n, then the characteristic function of $[a_{ij}]$ can be computed by a Turing machine in a number of steps polynomial in n. Turing machines can be efficiently simulated by Boolean circuits [PFI], so this means that the characteristic function of $[a_{ij}]_n$ has a circuit of polynomial size. But this is impossible if C_{set} is bounded below by $KC_{\bigcup-set}$ for any constant K, for then $char[a_{ij}]$ would require exponential-size circuits for some values of n. ∎

Thus, while $C_{monoset}([a_{ij}]) = C_{set}([a_{ij}])$ up to a constant factor, the same cannot be said of $C_{\bigcup-set}([a_{ij}])$.

It should be noted that the set of sequences of matrices with linear $\bigcup$-set complexity is distinct, not only from the set with linear set complexity, but also from the set with linear complexity over $\{\bigoplus\}$. By Lemma 2.2.2, there exists a $\sqrt{n} \times \sqrt{n}$ matrix $[M]$ with $C_{\bigcup-set}([M]) > (1-\epsilon)n^2/\log n$. An $n \times n$ matrix $[a_{ij}]$ consisting of $\sqrt{n}$ copies of $[M]$ along the main diagonal and 0's elsewhere has $C_{\bigcup-set}([a_{ij}]) > (1-\epsilon)n^{1.5}/\log n$.

$$[a_{ij}] = \begin{bmatrix} [M] & 0 & \cdots & 0 \\ 0 & [M] & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & [M] \end{bmatrix}.$$

On the other hand, the Strassen matrix multiplication algorithm [STR; Appendix C] provides a scheme for computing $[a_{ij}]$ with $C_{\nabla-set}([M]) = O(n^{1.404})$ steps. Theorem 7.1.6 provides a scheme for which $C_{set}([a_{ij}]) \leq O(n)$.

The relationship between set complexity and circuit complexity discussed above can be sharpened in several ways. First, note that if $[a_{ij}]$ is exponentially-lopsided, i.e, if $m = K \log n$ for some constant K, then a lower bound of $2n + K \log^2 n \log\log n -$

$2\log n - 2$ on $C_{set}([a_{ij}])$ suffices to give a nonlinear lower bound on the complexity of $char[a_{ij}]$.

This is numerically quite close to the set complexity of known matrices. For example, Corollary 7.4.4.1 presents a matrix with dimensions $\log n \times n$ with $C_{set}([a_{ij}]) = 2n - 2\log n - 1$ and Proposition 3.15 demonstrates a matrix with dimensions $(\log n + \log\log n) \times n$ for which $C_{\cup-set}([a_{ij}]) \geq 2n + \sqrt{2n} + O(\log n)$.

The problem of displaying a concrete function $B^n \to B^n$ with nonlinear circuit complexity is open; this problem can also be reduced to the problem of finding a lower bound on the set complexity of a matrix. Again, consider an $m \times n$ matrix $[a_{ij}]$. This time, instead of looking at the characteristic function of $[a_{ij}]$, consider $[a_{ij}]$ to be sliced into $m/\lceil\log n\rceil$ horizontal bands with $\lceil\log n\rceil$ rows each. Consider the Boolean function whose truth table consists of the $m/\lceil\log n\rceil$ bands of $[a_{ij}]$ placed end-to-end, with enough padding so that both the length of a band and the number of bands are powers of 2. Denote this function, which has $\lceil\log n\rceil$ outputs and no more than $\lceil\log n\rceil + \lceil\log m\rceil$ inputs, $band[a_{ij}]$. $m/\lceil\log n\rceil$ copies of a circuit for $band[a_{ij}]$ suffice, when their "index" variables are set to combinations of 1's and 0's, to compute the m functions whose truth tables are the rows of $[a_{ij}]$. By Theorem 7.9, $C_{set}([a_{ij}]) \leq (m/\lceil\log n\rceil)C(band[a_{ij}]) + 2n - 2\log n - 2$ or

$$C(band[a_{ij}]) \geq \frac{C_{set}([a_{ij}]) - 2n + 2\log n + 2}{m/\lceil\log n\rceil}.$$

This says that if $[a_{ij}]$ is square a lower bound of $n\log^{\epsilon}$, or even $n\log\log n$, on the set complexity of $[a_{ij}]$ suffices to establish a nonlinear lower bound on the complexity of $C(band[a_{ij}])$. If $[a_{ij}]$ is exponentially lopsided, a lower bound as low as $2n + K\log n \log\log\log n - 2\log n - 2$ suffices to establish a nonlinear bound on the complexity of $band[a_{ij}]$.

Thus, in numerical terms, the gap between the largest set complexity now known and the set complexity required to display nonlinear Boolean circuit complexity is very small.

— Chapter Eight —

BOOLEAN GRADIENTS

Loosely, one can say that a measure of computational complexity is **robust**, if minor changes in the structure whose complexity is being measured result in only small changes in complexity under that measure. Intuitively, a good "universal" measure of computational or combinatorial complexity — one which claims to capture the essence of the inherent complexness of a structure, rather than merely the difficulty of synthesizing it with a particular machine — should be robust.

Little seems to be known concerning the question of whether the circuit complexity of a finite Boolean function is robust with respect to geometrical transformations of the matrix representing the function. The purpose of this chapter is to discuss the relationship between a recently-announced result concerning the algebraic complexity of sets of polynomials, and geometrical transformation of Boolean matrices.

Boolean functions can be represented by Boolean matrices in any of several ways. Linear Boolean functions are linear maps between vector spaces, so they can can be represented by matrices in the usual way; i.e., so that the column vectors show the images of canonical basis vectors from the function's domain. A linear function $B^n \to B^m$ is represented by an $m \times n$ matrix.

A second useful representation is the truth table. There is a $1-1$ correspondence between Boolean functions $B^n \to B^m$ and truth tables of dimension $m \times 2^n$; a function $B^n \to B$ is a matrix with a single row. As one traces the progress of a Boolean computation downward in a combinatorial circuit over $\Omega = \{B^2 \to B\}$ and tabulates the row vector which belongs to the function computed at each gate, the truth table representing the aggregate outputs of the gates grows according to the rule that each new row is the component-wise product of two previous rows under some operation from Ω.

Theorem 7.9 and Theorems 7.3/7.4 reflect two different mappings between the set complexity of a matrix and the number of operations necessary to synthesize a given truth table. Theorem 7.9 is based on the observation that the process of computing in the set complexity model and the process of building up a truth table are identical, except for the choice of the starting vectors. The truth table with dimensions $m \times 2^n$ is built up starting from n row vectors of weight 2^{n-1}, while in the set complexity model a matrix with dimensions $m \times 2^n$ is built starting from 2^n row vectors of weight 1. The upper and lower bounds in Theorem 7.9 reflect the cost of synthesizing one set of starting vectors from the other. Theorems 7.3/7.4 are based on the fact that the $m \times 2^n$ truth table contains an $m \times n$ submatrix in which the computation process looks just like computation in the set complexity model.

A function $f : B^n \to B$ has a truth table with a only single row. An alternative to forming a conventional table for f is to break this one very long row vector into pieces of equal length and to stack them up to form a tableau. That is, if $\log m$ of the input variables are distinguished from the remaining variables, then a one-to-one correspondence between functions $B^n \to B$ and $m \times 2^n/m$ tableaus can be established by letting $[a_{ij}] = f(x_1, \ldots, x_{\log m}, x_{\log n+1}, \ldots, x_n)$, where i is the unary representation of the $(\log m)$-tuple $x_1, \ldots, x_{\log m}$ and j is the unary representation of the $n - \log m$ variables $x_{\log m+1}, \ldots, x_n$. In the 1960's and early 70's, O. Lupanov, E. Nechiporuk, V. Orlov, L. Sholomov, and others used this representation of the truth table, together with the "local coding" technique, to establish asymptotically tight upper bounds on the number of gates or contacts required for various combinatorial problems. N. Pippenger [PI3] used the same technique in 1978 to obtain tight upper bounds on the circuit complexity of the hardest monotone Boolean functions.

A third approach to representing Boolean functions by matrices would be to use the representations discussed in Chapter 5, i.e., the gradient, Jacobian, or Hessian matrices.

By Theorem 7.9, the set complexity of a large $n \times n$ matrix and the circuit complexity of the function represented by the $n \times n$ truth table differ by an amount that is of order n. The characteristic function of the matrix is identical to the index function of the function that is represented by the truth table.

The complexity of the characteristic function is less than that of the matrix, plus conversion factors of order n. The matrix, on the other hand, is guaranteed only to be as easy as the characteristic function *times* n. To obtain more powerful results concerning geometrical robustness, it would be desirable to show that the structure of the characteristic function, which contains all of the information in the matrix, can be used to speed up its synthesis.

The following surprising theorem, based on a clever but elementary construction, was announced in 1983 by W. Baur and V. Strassen [BAS].

Theorem 8.1

Let f be a polynomial in $x_1, x_2, \ldots, x_n$ over Z, and let $L(f)$ denote the number of additions and multiplications necessary to compute f from $\{x_1, \ldots, x_n, 1\}$. Then,

$$L\left(f, \frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right) \leq 4L(f).$$

$L(f)$ is the *algebraic complexity* of the function f. There is a close parallel between the algebraic complexity of a set of polynomials over the reals and the circuit complexity of a set of Boolean polynomials. The $\wedge$ and $\oplus$ operations, which correspond to multiplication and addition over the reals, together with the complement operation, which corresponds to adding 1, are a complete set of Boolean operators. That is, combinations of these three operations suffice to generate the other 13 operations $B^2 \rightarrow B$. Therefore, if each gate in a combinatorial circuit is replaced by a combination of gates from $\{\wedge, \oplus, -\}$ that performs the same operation, and if real operations are substituted for the corresponding Boolean ones, then the new circuit computes a set of real polynomials whose algebraic complexity is identical to the circuit complexity of the original function, up to a constant factor.

An elegant way to map between algebraic complexity and Boolean circuit complexity is by using the following technique for representing Boolean functions by real polynomials. This technique is used by Thayse, Davio, and Deschamps in their book on switching theory [DAV], but it is not clear with whom it originated.

Lemma 8.2

Suppose $F : B^n \to B^n$ has coordinate functions f_i, and is computed by a circuit over $\{B^2 \to B\}$. The circuit obtained by replacing each

$x \oplus y$	gate with gates computing	$x + y - 2xy,$
xy	gate with gates computing	$xy,$
$x \vee y$	gate with gates computing	$x + y - xy,$
$\overline{x}$	gate with gates computing	$1 - x,$
0	gate with gates computing	$0,$
1	gate with gates computing	$1,$
$\overline{x} \vee y$	gate with gates computing	$1 - x - xy,$
$\overline{x} \oplus y$	gate with gates computing	$1 - x - y + 2xy,$
$\overline{x}y$	gate with gates computing	$y - xy,$

computes a function $G : \mathcal{R}^n \to \mathcal{R}^n$ with the property that if $f_i(x_1, \ldots, x_n) = 1$ for $\{x_i\} \in \{0, 1\}$ then $g_i(x_1, \ldots, x_n) = 1$, and if $f_i(x_1, \ldots, x_n) = 0$ for $\{x_i\} \in \{0, 1\}$ then $g_i(x_1, \ldots, x_n) = 0$. Moreover, if $L(G)$ is the number of multiplications and additions required to compute G, then

$$C(F) \leq L(G) \leq 4\,C(F).$$

Proof:

Induction on the size of the circuit shows that an output of the circuit for G takes the real value 1 whenever the corresponding gate in the circuit for F takes the Boolean value 1, and takes the real value 0 whenever the corresponding gate in the circuit for F takes the Boolean value 0. Each gate in an optimal circuit for F can be simulated with no more than 4 additions or nonscalar multiplications. Each addition or multiplication in the computation for G can be simulated by $\oplus$ or $\wedge$ respectively to build a circuit for F. ∎

To be able to apply Baur and Strassen's result to Boolean functions, one would like the mapping between real and Boolean operations to survive the differentiation process. Unfortunately this is not quite the case, for the derivatives of the real functions x, x^2 and x^3 are different (mod 2), while the three expressions x, xx, and xxx are identical as Boolean functions.

A more limited result does hold, however:

Lemma 8.2.1

If f^r is a polynomial $\mathcal{R}^n \to \mathcal{R}$ which corresponds to $f : B^n \to B$ on $\{0,1\}$, and if the degree of x_i in f^r is less than 2 then

$$\left(\frac{\partial f}{\partial x_i}\right)^r = \left(\frac{\partial^r f^r}{\partial x_i}\right)^2$$

on $\{0,1\}$, where ∂^r represents ordinary partial differentiation of a real function with respect to a real variable, and ∂ represents differentiation of a Boolean function with respect to a Boolean variable.

Proof: [DAV]

$f^r = a + bx_i$, where a and b are independent of x_i.

$$\frac{\partial^r f^r}{\partial x_i} = b = f(x_i = 1) - f(x_i = 0).$$

Since $\partial f / \partial x_i = f(x_i = 1) \oplus f(x_i = 0)$,

$$\begin{aligned}\left(\frac{\partial f}{\partial x_i}\right)^r &= f(x_i = 1) + f(x_i = 0) - 2f(x_i = 1)f(x_i = 0) \\ &= [f(x_i = 1) - f(x_i = 0)]^2. \ \blacksquare\end{aligned}$$

Example 8.2.2

$$f = x_1x_2x_3 \oplus x_1x_2 \oplus x_3 \oplus 1$$

$$f^r = 1 - x_3 - x_1x_2 + 3x_1x_2x_3 - 2x_1x_2x_3^2$$

$$\frac{\partial f}{\partial x_2} = x_1x_3 \oplus x_1$$

$$\frac{\partial^r f^r}{\partial x_2} = 3x_1x_3 - x_1 - 2x_1x_3^2.$$

Thus, Baur and Strassen's result extrapolates directly only to Boolean polynomials synthesized by circuits whose computations, if modeled over the reals, would never develop any polynomials of degree greater than one. One way to ensure that this happens is to restrict one's attention to *fanout-free* circuits. In this limited context, the gradient of a Boolean function has circuit complexity that is bounded by a constant factor times the complexity of the function itself.

J. Reif [REI] has used this fact to efficiently calculate Boolean derivatives for the purpose of determining all possible "stuck-at" faults in a digital circuit that can be detected by a given Boolean test vector. For this application, the restriction that the portion of the circuit that is to be analyzed be fanout-free is not a critical one.

It does not seem possible to directly extend Baur and Strassen's proof to produce small circuits for the Boolean gradient when the real version of f's circuit generates polynomials of large degree.

Let us suppose for a moment that the Baur and Strassen result *does* apply, at least asymptotically, to Boolean functions. That is, suppose that circuit complexity has the following property.

[Boolean Gradient Property]

There exists a constant K such that for all n sufficiently large, and for all $f : B^n \to B$,

$$C\left(f, \frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right) \leq K\, C(f).$$

The Boolean gradient property says that one can compute the value of f at the n points $(x_1 \oplus 1, x_2, x_3, \ldots, x_n), (x_1, x_2 \oplus 1, x_3, \ldots, x_n), \ldots, (x_1, x_2, x_3, \ldots, x_n \oplus 1)$ at a cost that is at most a constant times the cost of evaluating f at the single point $(x_1, \ldots, x_n)$. Now, let us look at two of the consequences of this powerful property.

First, given the Boolean gradient property, the circuit complexity of $f(x_1, \ldots, x_n, y_1, \ldots, y_k)$ and of $\{f(x_1, \ldots x_n)\}_{y_1,\ldots,y_k}$ are always approximately equal.

Definition 8.3

Suppose $f_i : B^n \to B$, for $1 \leq i \leq m$, are the coordinate functions of $F : B^n \to B^m$. Let $Index_F : B^{n+\lceil \log m \rceil} \to B$ be the function which *selects* the functions f_i, i.e., $Index_F : (x_1, \ldots, x_{n+\lceil \log m \rceil}) = f_i(x_1, \ldots, x_n)$, where $x_{n+1}, \ldots, x_{n+\lceil \log m \rceil}$ is the binary representation of the integer i. $Index_F$ is the **index function** of F.

Proposition 8.3.1

Let $F : B^n \to B^m$. Then

$$C(Index_F) \leq C(F) + O(m).$$

Proof:

If $\{f_i\}$ are the coordinate functions of F, then

$$Index_F(x_1, \ldots, x_n, x_{n+1}, \ldots, x_{n+\lceil \log m \rceil}) = y_1 f_1 \oplus y_2 f_2 \oplus \cdots \oplus y_m f_m,$$

where $y_i = 1 \Leftrightarrow i = \sum_{j=1}^{\lceil \log m \rceil} x_{n+j} 2^{j-1}$. Binary to unary conversion can be accomplished with $2^{\lceil \log m \rceil} + 10\sqrt{2^{\lceil \log m \rceil}} - 6$ gates (Lemma 7.8), and selection of the appropriate coordinate function can be accomplished with $2m - 1$ gates. ∎

On the other hand, if circuit complexity has the Boolean gradient property, it also has the following property.

[Index Property]

There exists some $K \geq 0$ such that for all $n + m$ sufficiently large, and all $F : B^n \to B^m$,

$$C(F) \leq K\, C(Index_F) + O(m).$$

Proof:

Consider the function $g : B^{n+2^{\lceil \log m \rceil}} \to B$ given by $g(x_1, \ldots, x_n, x_{n+1}, \ldots, x_{n+2^{\lceil \log m \rceil}}) = Index_F(x_1, \ldots, x_n, y_1, \ldots, y_{\lceil \log m \rceil})$, where $y_i = \bigvee x_{n+j}$, the disjunction being taken over all j with 1 in the i^{th} digit of its binary representation. $C(g) \leq C(Index_F) +$

$O(m)$, by Lemma 7.7. By the Boolean gradient property there exists a circuit with complexity not exceeding $K\,C(g)$ which computes

$$\left\{g, \frac{\partial g}{\partial x_{n+1}}, \ldots, \frac{\partial g}{\partial x_{n+2^{\lceil \log m \rceil}}}\right\}.$$

But $\frac{\partial g(x_1,\ldots,x_{n+m})}{\partial x_{n+j}} = g(x_1,\ldots,x_n,x_{n+1},\ldots,x_{n+j}\oplus 1,\ldots,x_{n+m}) \oplus f(x_1,\ldots,x_n, x_{n+1},\ldots,x_{n+j},\ldots,x_{n+m})$. If we set $x_{n+1},\ldots,x_{n+m}$ to 0 in this last circuit, and add output g to each of its m other outputs, we have a circuit whose i^{th} output computes $f(x_1,\ldots,x_n,0,\ldots,1,\ldots,0)$ for each possible $(\log m)$-tuple. That is, this circuit computes

$$\{Index_F(x_1,\ldots,x_n,c_1,\ldots,c_{\lceil \log m \rceil})\}_{c\in B^{\lceil \log m \rceil}} = F(x_1,\ldots,x_n). \qquad ∎$$

The index property says that the complexity of a single-output function which can, depending on the values of its index variables, take the value of any of the coordinate functions of a multiple-output Boolean function F, is bounded by a constant times the complexity of F itself, plus a term linear in the number of outputs. Geometrically, this means that rows of a truth table can be broken off and stacked on top of one another to form a new table with relatively little change in the complexity of the represented function.

The index property is not only a consequence of the Boolean gradient property, it is equivalent to it. For suppose $f : B^n \to B$ has complexity $C(f)$. Consider the function $g(x_1,\ldots,x_n,y_1,\ldots,y_{\lceil \log n \rceil}) = f(x_1,x_2,\ldots,x_\alpha \oplus 1,\ldots,x_n)$, where $\alpha = \sum_{k=1}^{\lceil \log n \rceil} y_k 2^{k-1}$. By Lemma 7.8, binary to unary conversion can be accomplished efficiently, so $C(g) \leq C(f) + 5n$. By the index property, there exists a circuit with complexity $K(C(g)+O(n))$ for the n functions

$$f(x_1 \oplus 1, x_2, x_3, \ldots, x_n)$$

$$f(x_1, x_2 \oplus 1, x_3, \ldots, x_n)$$

$$f(x_1, x_2, x_3 \oplus 1, \ldots, x_n)$$

$$\vdots$$

$$f(x_1, x_2, x_3, \ldots, x_n \oplus 1).$$

To complete the synthesis of the n partial derivatives of f, add $f(x_1,\ldots,x_n)$ to each of these last outputs. The cost of the entire construction does not exceed $(K+1)C(f)+(3K+1)n-3K$.

A second consequence of the Boolean gradient property is that an analog of Theorem 3.1 holds true for set complexity, i.e., for all n sufficiently large, and all $n \times m$ matrices $[a_{ij}]$, there exists a constant $K > 0$ such that

$$C_{set}([a_{ij}]^T) \leq K C_{set}([a_{ij}]).$$

By Theorem 7.4, there exists a function $F : B^n \to B^m$ with $[I_F] = [a_{ij}]$ and $C(F) = C_{set}([a_{ij}])$. One may as well assume that $C(F) \geq m$, since rows or columns of weight ≤ 1 may be added to or deleted from any matrix without increasing its set complexity. Expressed as a Boolean polynomial, a typical coordinate function of F has the form

$$f_i(x_1,\ldots,x_n) = a_{i0} \oplus \bigoplus_{j=1}^{n} a_{ij}x_j \oplus r_i(x_1,\ldots,x_n),$$

where $r_i(x_1,\ldots,x_n)$ is a polynomial all of whose terms have degree ≥ 2. Given $f_1,\ldots,f_m$, with $2m-1$ gates one can form the sum

$$\begin{aligned} h(x_1,\ldots,x_n,y_1,\ldots,y_m) &= \bigoplus_{i=1}^{m} y_i f_i \\ &= \bigoplus_{i=1}^{m}\bigoplus_{j=1}^{n} a_{ij}x_jy_i + p(x_1,\ldots,x_n,y_1,\ldots,y_m), \end{aligned}$$

where p is a polynomial all of whose terms have degree ≥ 2 in $x_1,\ldots,x_n$. By the Boolean gradient property, there exists a constant K such that

$$C\left(\frac{\partial h}{\partial x_1},\ldots,\frac{\partial h}{\partial x_n}\right) \leq K\, C(h).$$

A typical derivative of h is

$$\frac{\partial h_i}{x_j} = \bigoplus_{j=1}^{m} a_{ij}y_j + \frac{\partial p}{\partial x_i},$$

where each term in $\partial p/\partial x_i$ has degree ≥ 1 in $x_1, \ldots, x_n$. Therefore, if $x_1, \ldots, x_n$ are set to 0 in the circuit whose existence is guaranteed by the Boolean gradient property, then what remains is a circuit which computes a linear function G whose n coordinate functions are $\bigoplus_{j=1}^{m} a_{ij}y_j$. But $[G] = [I_G] = [a_{ij}]^T$, so by Proposition 7.3, $C_{set}([a_{ij}]^T) \leq C(G)$.

Thus, given that the Boolean gradient property holds for circuit complexity, the set complexity of a matrix is of the same order as the set complexity of its transpose.

— Chapter Nine —

ONE-WAY FUNCTIONS

In this chapter, results given earlier are applied to examine the relationship between the circuit complexity of a invertible function and that of its inverse.

Definition 9.1

A sequence of functions $\{F_n : B^n \rightarrow B^n\}_{n=1,\ldots,\infty}$ is one-way if each F_n is invertible, and

$$\lim_{n\rightarrow\infty} \frac{C_{\cap}(F_n^{-1})}{C_{\cap}(F_n)} = \infty.$$

Whether one-way functions, in the sense of Definition 9.1, exist at all is an open question. No proof of one-wayness for a sequence of finite functions has ever been exhibited.

By analogy with the transpose theorem and with Baur and Strassen's result on computing gradients, if circuit complexity is truly robust one might speculate that there exists $k > 0$ such that for all n and all invertible $F : B^n \rightarrow B^n$, $C_{\cap}(F^{-1}) \leq kC_{\cap}(F)$. An equivalent formulation is the proposition that for all invertible F and G, $C_{\cap}(F \mid G) \leq kC_{\cap}(G \mid F)$, since by Corollary 2.2.1.3 $C_{\cap}(F \mid G) = C_{\cap}(FG^{-1}) = C_{\cap}((GF^{-1})^{-1})$ and $C_{\cap}(G \mid F) = C_{\cap}(GF^{-1})$, and since $C_{\cap}(F \mid G)$ reduces to $C_{\cap}(F)$ and $C_{\cap}(G \mid F)$ to $C_{\cap}(F^{-1})$ when $G = I$. No proof of this proposition is evident, however.

Most invertible functions $B^n \rightarrow B^n$ have near-maximal complexity. Because of this, a sequence of functions $\{F_n : B^n \rightarrow B^n\}$ chosen at random is one-way with probability zero. In fact, a stronger statement is possible:

Proposition 9.2

If $\{F_n : B^n \to B^n\}$ is a sequence of invertible functions, then $C(F_n^{-1}) \sim C(F_n)$ with probability one.

Proof:

For any $\epsilon > 0$, if

$$\left| \frac{C(F)}{C(F^{-1})} - 1 \right| > \epsilon$$

then either $(1-\epsilon)C(F^{-1}) > C(F)$ or $(\frac{1}{1+\epsilon})C(F) > C(F^{-1})$. By Corollary 2.4.3, the complexity of any function $B^n \to B^n$ is less than $2^n + o(2^n)$, so F can have this property only if either $C(F)$ or $C(F^{-1})$ is less than $(1-\epsilon)2^n + o(2^n)$. By Lemma 2.2.2, the number of functions $F : B^n \to B^n$ with $C(F) \leq k2^n$ does not exceed $(12|\Omega|)^{k2^n}(k2^n)^{k2^n+n}$. Therefore the number of functions $B^n \to B^n$ with either $C(F) \leq k2^n$ or $C(F^{-1}) \leq k2^n$ does not exceed $N_k(n) = 2(12|\Omega|)^{k2^n}(k2^n)^{k2^n+n}$. Let $N(n) = (2^n)!$ be the number of invertible functions $B^n \to B^n$. $N(n) \geq 2^{\frac{n}{2}} 2^{-(\log e)2^n} 2^{n2^n}$. Therefore for any $0 < k < 1$,

$$\lim_{s\to\infty} \prod_{n=s}^{\infty} \left(\frac{N(n) - N_k(n)}{N(n)} \right) \geq \lim_{s\to\infty} \prod_{n=s}^{\infty} \left(1 - \frac{1}{2^d}\right),$$

where $d = (1-k)n2^n - (k\log k + 7.59k - 1.44)2^n - n^2 + (\frac{1}{2} - \log k)n - 1$. But the last limit is 1, since the product $\prod_{n=1}^{\infty}\left(1 - \frac{1}{2^d}\right)$ is bounded below by $\prod_{n=1}^{\infty}\left(1 - \frac{1}{2^k}\right)$, and hence converges. ∎

Most invertible linear Boolean functions also have near-maximal complexity: by Lemma 2.2.2 and Proposition 4.5 all but a vanishing portion of the invertible linear functions $F : B^n \to B^n$ have

$$C_{\oplus}(F) \geq C(F) \geq (1-\epsilon)\frac{n^2}{2\log n}.$$

As a consequence, the same counting argument that was used in Proposition 9.2 can be applied to linear invertible functions.

Proposition 9.3.1

If $\{F_n : B^n \to B^n\}$ is a sequence of invertible linear functions, then $C_{\oplus}(F_n^{-1}) \sim C_{\oplus}(F_n)$ and $C(F_n^{-1}) \sim C(F_n)$ with probability one.

Proof:

By Theorem 3.12, $C_{\oplus}(F)$ for any linear $F : B^n \to B^n$ does not exceed

$$\frac{n^2}{2\log n} + o\left(\frac{n^2}{2\log n}\right).$$

By Lemma 2.2.2, the number of functions $B^n \to B^n$ with either $C(F) \le kn^2/2\log n$ or $C(F^{-1}) \le kn^2/2\log n$ satisfies

$$\begin{aligned} N_k(n) &\le 2(12|\Omega|)^{\left(\frac{kn^2}{2\log n}\right)}\left(\frac{kn^2}{2\log n}\right)^{\left(\frac{kn^2}{2\log n}+n\right)} \\ &\le 2^{\left(kn^2+O(n^2/\log n)\right)}. \end{aligned}$$

By Proposition 4.5 the number of invertible linear functions $B^n \to B^n$ satisfies

$$N(n) \ge (2.8)2^{n^2} > 2^{n^2 \cdot 2}.$$

Therefore for any $0 < k < 1$,

$$\lim_{s\to\infty} \prod_{n=s}^{\infty} \left(\frac{N(n) - N_k(n)}{N(n)}\right) \ge \lim_{s\to\infty} \prod_{n=s}^{\infty} \left(1 - \frac{1}{2^{(1-k)n^2 - O(n^2/\log n)}}\right) = 1. \quad \blacksquare$$

Propositions 9.2 and 9.3.1 are based on the fact that nearly all functions from $B^n \to B^n$ have near-maximal complexity, which in turns reflects the efficiency of combinatorial circuits in generating a large number of functions with a small number of gates. That is, since there are very few easy functions a randomly chosen invertible linear function nearly always has complexity close to $\frac{n^2}{2\log n}$ and, for the same reason, its inverse nearly always has complexity close to $\frac{n^2}{2\log n}$. A deeper question is whether a function chosen at random among *easy* functions has an easy inverse. The upper bound on the complexity of memoryless functions that was established in Chapter 4 establishes a limited result along these lines:

Proposition 9.3.2

Let N_C be the number of invertible linear functions $F : B^n \to B^n$ with circuit complexity not exceeding C. Then for all n sufficiently large and all $C \geq 2n$, at least $\sqrt{N_C} - o(\sqrt{N_C})$ such functions have

$$.5 \leq \frac{C(F^{-1})}{C(F)} \leq 2.$$

In addition, at least $\sqrt{N_C} - o(\sqrt{N_C})$ such functions have

$$.5 \leq \frac{C_{\oplus}(F^{-1})}{C_{\oplus}(F)} \leq 2.$$

Proof:

By Lemma 2.2.2, $N_C \leq K^C C^{C+n}$, for a constant K depending on the basis. Let M^C be the number of functions $F : B^n \to B^n$ for which $\mathcal{M}_{\oplus}(F) \leq C$. By Theorem 4.4, every invertible linear function $B^n \to B^n$ can be computed by a memoryless circuit of size $\frac{n^2}{\log n} + o\left(\frac{n^2}{\log n}\right)$. But a memoryless circuit of this size can be considered as a cascade of $\frac{n^2}{C \log n} + o\left(\frac{n^2}{C \log n}\right)$ circuits of size C. By Proposition 4.5, there are more than $(2.8)2^{n^2}$ invertible linear functions, so M_C must satisfy

$$M_C^{\frac{n^2}{C \log n} + o\left(\frac{n^2}{C \log n}\right)} \geq (2.8)2^{n^2}.$$

That is, $\log M_C \geq C \log n - o(\log n)$. For all n sufficiently large, $M_C \geq n^{c-1}$.

All of the functions with $\mathcal{M}_{\oplus} \leq C$ have $\mathcal{M}_{\oplus}(F^{-1}) \leq C$, since reversing the order of the operations in a memoryless circuit for F produces a memoryless circuit for F^{-1}.

By Lemma 2.2.2 at most $K^{\left(\frac{C}{2}\right)}\left(\frac{C}{2}\right)^{\left(\frac{C}{2}+n\right)}$ functions $B^n \to B^N$ can have $C(F) \leq C/2$. By the same counting argument, at most $K^{\left(\frac{C}{2}\right)}\left(\frac{C}{2}\right)^{\left(\frac{C}{2}+n\right)}$ functions can have $C(F^{-1}) \leq C/2$. Therefore there are not less than

$$n^{C-1} - 2K^{\left(\frac{C}{2}\right)}\left(\frac{C}{2}\right)^{\left(\frac{C}{2}+n\right)}$$

distinct functions $F : B^n \to B^n$ for which $C(F) \leq C$, $C(F^{-1}) \leq C$, $C(F) \geq C/2$, and $C(F^{-1}) \geq C/2$.

A direct calculation shows that for all n sufficiently large and $3n \leq C \leq n^{2-\epsilon}$,

$$n^{C-1} - 2K^{(\frac{C}{2})}\left(\frac{C}{2}\right)^{(\frac{C}{2}+n)} > 0,$$

and that the square of this quantity is less than $K^C C^{C+n}$. ∎

If C is less than $\frac{(1-\epsilon)n^2}{2\log n}$, somewhat stronger results can be obtained by this same argument. For example, given any $\epsilon > 0$, for $C = n\log n$ and all n sufficiently large, the number of linear invertible functions with

$$.5 \leq \frac{C(F^{-1})}{C(F)} \leq 2$$

is not less than $N_C^{(1-\epsilon)}$.

According to Theorem 3.1, the complexity over $\{\bigoplus\}$ of every square matrix is *exactly* equal to that of its transpose. Is it possible that a similar relation holds for the circuit complexity of a linear function and that of its inverse? What about the circuit complexity of an arbitrary invertible function $B^n \to B^n$ and that of its inverse?

Each of the 24 invertible functions from B^2 to B^2 does require exactly as many gates as does its inverse (either one or two, depending on the function.) What about the case $n = 3$? There are $2^3! = 8! = 40320$ invertible functions with three inputs and three outputs. It is not necessary to examine each individual function to verify the hypothesis that $C(F) = C(F^{-1})$, however. If $F_1 = P_1F_2P_2$, where P_1 and P_2 are permutations, F_1 and F_2 have the same circuit complexity. In addition, if C_1 and C_2 are complementations of inputs and outputs respectively, F_1 and $C_2F_1C_1$ will normally have the same circuit complexity over the 16-gate basis $\{\omega : B^2 \to B\}$, since complementation of inputs and outputs can be absorbed into the top and bottom gates in the circuit without changing the total gate count. The only exception occurs when an input wire is connected directly to an output wire without any intervening gates.

However, when this happens, complementation increases the complexity of a function and its inverse by the same number of gates. Thus, it suffices to consider one member of each equivalence class under permutation and complementation. These classes have been enumerated by C. S. Lorens [LOR] by using a variation of DeBruijn's Theorem: he finds that there are 52 classes, and lists representatives of each. 24 of the functions are equivalent to their own inverses. The remaining 28 functions are listed in Appendix D. Heuristically determined minimal networks for these functions are shown in Appendix E. In each case, the circuit complexity of the function is same as that of its inverse.

It turns out *not* to be true, however, that $C(F)$ is exactly equal to $C(F^{-1})$ for all n, or that $C_\oplus(F)$ equals $C_\oplus(F^{-1})$ for all n. A sequence of linear functions $\{F_n\}$ for which

$$\frac{C(F_n^{-1})}{C(F_n)} = \frac{C_\oplus(F_n^{-1})}{C_\oplus(F_n)} > 1$$

is presented below.

Lemma 9.4

Let S be a set of functions $\{f_1, f_2, \ldots, f_{n-r}, g_1, g_2, \ldots, g_r\}$ in the variables $\{x_1, x_2, \ldots, x_{n-r}\}$, and assume that none of $g_1, \ldots, g_r$ are identically zero. Let S_r be the set of functions

$$\{f_1, f_2, \ldots, f_{n-r}, g_1 \oplus x_{n-r+1}, g_2 \oplus x_{n-r+2}, \ldots, g_r \oplus x_n\}$$

in the variables $\{x_1, x_2, \ldots, x_n\}$. Then

$$C_\oplus(S_r) = C_\oplus(S) + r$$

and

$$C(S_r) = C(S) + r.$$

Proof:

The transpose theorem (Theorem 3.1) gives the first result. By definition, the $n \times n$ matrix

$$\begin{bmatrix} F & 0 \\ G & I \end{bmatrix},$$

which contains the $r \times r$ identity matrix and the $(n-r) \times (n-r)$ matrix $[F]$ as submatrices, has complexity $C_{\oplus}(S_r)$. By Theorem 3.1,

$$\begin{bmatrix} F^T & G^T \\ 0 & I \end{bmatrix}$$

has complexity $C_{\oplus}(S_r)$. Since the last r rows simply represent an identity function on the last r inputs, their synthesis takes no gates at all. Therefore, the complexity of

$$\begin{bmatrix} F^T & G^T \end{bmatrix}$$

is also $C_{\oplus}(S_r)$. Using the transpose theorem again, the complexity of

$$\begin{bmatrix} F \\ G \end{bmatrix}$$

is $C(S_r) - r$. But this last matrix represents S.

The second result may be verified by the following argument. Suppose that we have a circuit for S_r with $C(S_r)$ gates. Successively set $x_{n-r+1}, x_{n-r+2}, \ldots, x_n$ to zero. Since in each case an output depends on one of the x_{n-r+i}, together with at least one other variable, zeroing x_{n-r+i} allows us to eliminate one gate from the circuit. When all the x_{n-r+i} are zero, the circuit computes S, so $C(S_r) - r \geq C(S)$. On the other hand, $C(S_r)$ can be synthesized by first building S, then adding $x_{n-r+1}, \ldots, x_n$ to the outputs $g_1, \ldots, g_r$. Thus, $C(S_r) - r = C(S)$. ∎

Lemma 9.5

For any integer $d > 0$, and $n = 4d$, there exists an n-input invertible linear Boolean function F on the variables $x_1, x_2, \ldots, x_n$, and a circuit computing F, with the following properties. Denoting the set of intermediate functions computed by the circuit as $INTER$, and the set of coordinate output functions by $FINISH$, we have:

$$C(INTER) = 4d - 1$$

$$C(FINISH \mid INTER \cup \{x_1, \ldots, x_n\}) = 2d$$

$$C(INTER \cup \{x_1, \ldots, x_n\} \mid FINISH) = 7d - 2.$$

Moreover, these relationships hold if C is replaced by $C_{\oplus}$ in each equation.

Proof:

The coordinate functions of F are

$$\begin{aligned}
f_1 &= x_1 \\
f_2 &= x_2 \\
&\vdots \\
f_{2d} &= x_{2d} \\
f_{2d+1} &= x_1 \oplus \ldots \oplus x_{2d} \oplus x_{2d+2} \oplus \ldots \oplus x_{4d} \\
f_{2d+2} &= x_1 \oplus \ldots \oplus x_{2d+1} \oplus x_{2d+3} \oplus \ldots \oplus x_{4d} \\
&\vdots \\
f_{4d} &= x_1 \oplus \ldots \oplus x_{4d-1}.
\end{aligned}$$

If the $2d \times 2d$ matrix which consists of all ones is denoted by [1] and the $2d \times 2d$ identity matrix by $[I]$, then $[F]$ has the form

$$\begin{bmatrix} I & 0 \\ 1 & 1 \oplus I \end{bmatrix}.$$

$[1]^2 = [0]$, $[1] + ([1] + [I])[1] = [0]$, and $([1] + [I])([1] + [I]) = [I]$. Therefore $[F]^2$ is the $4d \times 4d$ identity matrix, and F is non-singular. The circuit computing F calculates successively:

Phase I (d gates)

$$x_1 \oplus x_{2d+1}$$

$$x_1 \oplus x_2 \oplus x_{2d+1}$$

$$x_1 \oplus x_2 \oplus x_3 \oplus x_{2d+1}$$

$$\vdots$$

$$x_1 \oplus \ldots \oplus x_d \oplus x_{2d+1}$$

Phase II ($2d-1$ gates)

$$x_1 \oplus \ldots \oplus x_d \oplus x_{2d+1} \oplus x_{2d+2}$$

$$x_1 \oplus \ldots \oplus x_d \oplus x_{2d+1} \oplus x_{2d+2} \oplus x_{2d+3}$$

$$\vdots$$

$$x_1 \oplus \ldots \oplus x_d \oplus x_{2d+1} \oplus \ldots \oplus x_{4d}$$

Phase III (d gates)

$$x_1 \oplus \ldots \oplus x_d \oplus x_{d+1} \oplus x_{2d+1} \oplus \ldots \oplus x_{4d}$$

$$x_1 \oplus \ldots \oplus x_d \oplus x_{d+1} \oplus x_{d+2} \oplus x_{2d+1} \ldots x_{4d}$$

$$\vdots$$

$$x_1 \oplus \ldots \oplus x_{4d}.$$

The results of phases I, II, and III comprise $INTER$; $4d-1$ gates are necessary and sufficient. $FINISH$ consists of the functions $x_1, x_2, \ldots, x_{2d}$ and

$$x_{2d+1} \oplus (x_1 \oplus \ldots \oplus x_{4d})$$

$$x_{2d+2} \oplus (x_1 \oplus \ldots \oplus x_{4d})$$

$$\vdots$$

$$x_{4d} \oplus (x_1 \oplus \ldots \oplus x_{4d}).$$

Given $\{x_1, \ldots, x_n\}$ and the last output of phase III, $2d$ gates are necessary and sufficient to compute $FINISH$.

Given $FINISH$, how many gates are required to calculate $INTER$ and $\{x_1, \ldots, x_n\}$? $FINISH$ and $\{x_1, \ldots, x_n\}$ have $2d$ functions in common, namely $x_1, x_2, \ldots, x_{2d}$, so $INTER \cup \{x_1, \ldots, x_n\}$ contains $6d-1$ functions not in $FINISH$. Therefore, any circuit which computes $INTER \cup \{x_1, \ldots, x_n\}$ must have at least

$6d-1$ output gates. Now note that, since F is non-singular, $\{x_1, \ldots, x_n\}$ and $FINISH$ are each bases of a $4d$-dimensional vector space over B. We will now show that, when expressed in terms of the basis $FINISH$, the weight [number of non-zero components] of each of the $6d-1$ linear functions to be synthesized is at least $d+1$. This finishes the lemma, since it means there must be at least $d-1$ gates in the network which do not synthesize any output function.

Since F is self-inverse, $[F]^T$ is itself the change of basis matrix which converts an input vector in the basis $x_1, \ldots, x_n$ to one in the basis $f_1, \ldots, f_n$. It is easily checked that

$$x_1 = f_1$$
$$x_2 = f_2$$
$$\vdots$$
$$x_{2d} = f_{2d}$$
$$x_{2d+1} = f_{2d+1} \oplus (f_1 \oplus \cdots \oplus f_{4d})$$
$$x_{2d+2} = f_{2d+2} \oplus (f_1 \oplus \cdots \oplus f_{4d})$$
$$\vdots$$
$$x_{4d} = f_{4d} \oplus (f_1 \oplus \cdots \oplus f_{4d}).$$

Therefore, the functions computed in phase I all have weight at least $3d-1$, since each has a non-zero projection on $f_{d+1}, \ldots, f_{2d}$ and $f_{2d+2}, \ldots, f_{4d}$. The functions computed in phase II all have weight at least $d+1$, because each has nonzero projections on either $f_1, \ldots, f_d$ or on $f_{d+1}, \ldots, f_{2d}$, depending on the parity of its rank in the phase, and each has at least one nonzero projection on one of the $f_{2d+1}, \ldots, f_{4d}$. Each function computed in phase III has weight at least $3d$, since each has nonzero projections on $f_1, \ldots, f_d$ and $f_{2d+1}, \ldots, f_{4d}$. Finally, the functions in $FINISH$ which are not in $\{x_1, \ldots, x_n\}$ all have weight $4d-1$, since F is its own inverse. ∎

The following example, with $d = 4$ and $n = 16$, demonstrates the weight distributions discussed above.

```
1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0
1 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0
1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0

1 1 1 1 0 0 0 0 1 1 0 0 0 0 0 0
1 1 1 1 0 0 0 0 1 1 1 0 0 0 0 0
1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0
1 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0
1 1 1 1 0 0 0 0 1 1 1 1 1 1 0 0
1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 0
1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 1

1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1
1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

In terms of the basis *FINISH*, these vectors are:

```
0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1
0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1
0 0 0 1 1 1 1 1 0 1 1 1 1 1 1 1
0 0 0 0 1 1 1 1 0 1 1 1 1 1 1 1

1 1 1 1 0 0 0 0 1 1 0 0 0 0 0 0
0 0 0 0 1 1 1 1 0 0 0 1 1 1 1 1
1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0
0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 1
1 1 1 1 0 0 0 0 1 1 1 1 1 1 0 0
0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 1
1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 0
```

$$\begin{array}{cccccccccccccccc}
1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{array}$$

Note that each vector has weight at least $d+1=5$.

Theorem 9.6

For any integer $d > 0$, there exists an invertible linear Boolean function $\mathcal{F} : B^{8d-1} \to B^{8d-1}$, with

$$C(\mathcal{F}) = 10d-2$$

and

$$C(\mathcal{F}^{-1}) = 11d-3.$$

Proof:

Take $F : B^{4d} \to B^{4d}$ as in Lemma 9.5. Consider the function defined by the matrix

$$[\mathcal{F}] = \begin{bmatrix} FINISH & 0 \\ INTER & I \end{bmatrix},$$

where $FINISH$ and $INTER$ are the sets of functions defined in Lemma 9.5. By Lemma 9.4,

$$C(\mathcal{F}) = C\left(\begin{bmatrix} FINISH \\ INTER \end{bmatrix}\right) + 4d-1$$

$$= (4d-1) + (2d) + 4d - 1$$

$$= 10d-2.$$

On the other hand,

$$C(\mathcal{F}^{-1}) = C\left(\begin{bmatrix} FINISH^{-1} & 0 \\ (INTER)(FINISH^{-1}) & I \end{bmatrix}\right)$$

$$= C\left(\begin{bmatrix} FINISH^{-1} \\ (INTER)(FINISH)^{-1} \end{bmatrix}\right) + 4d - 1$$

$$= C\left(\begin{bmatrix} I \\ INTER \end{bmatrix} [FINISH]^{-1}\right) + 4d - 1$$

$$= C(\{x_1, \ldots, x_n\} \cup INTER \mid FINISH) + 4d - 1$$

$$= (7d - 2) + (4d - 1)$$

$$= 11d - 3. \quad \blacksquare$$

Corollary 9.7

For any integer $d > 0$, there exists an invertible linear Boolean function $\mathcal{F} : B^{8d-1} \rightarrow B^{8d-1}$, with

$$C_{\oplus}(\mathcal{F}) = 10d - 2$$

and

$$C_{\oplus}(\mathcal{F}^{-1}) = 11d - 3.$$

Example 9.7.1

For $d = 2$ and $n = 15$,

$$[\mathcal{F}] = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}.$$

$C(\mathcal{F}) = C_{\oplus}(\mathcal{F}) = 18$, while $C(\mathcal{F}^{-1}) = C_{\oplus}(\mathcal{F}^{-1}) = 19$.

We finish this chapter with the observation that a fast general procedure for finding the Δ-transform of a function yields a fast procedure for finding the inverse of a function. Suppose it were easy to compute the Δ-transform of $F : B^n \rightarrow B^n$ from a small circuit for F. If F is invertible, there is only a single value of $X \in B^n$ for which $F(X) = Y$. Therefore a fast procedure for ΔF could be used to determine $F^{-1}(Y)$ by checking whether the number of solutions to $F(X) = Y$ with $X \leq X_{test}$ is odd or even for various values of X_{test}.

No general fast procedure is known for computing ΔF from F. It is quite possible, however, that ΔF has always has small circuits when F does, without those circuits being easy to find. If so, then F and F^{-1} must also always have nearly the same circuit complexity.

Theorem 9.8

Suppose that for all $f : B^n \to B$ $C(\Delta f)/C(f) = O(1)$. Then for all $F : B^n \to B^n$ with $C(F) > n$

$$\frac{1}{O(n)} \leq \frac{C(F^{-1})}{C(F)} \leq O(n).$$

Proof:

Consider an optimal circuit $\mathcal{N}_1$ computing $F(x_1, \ldots, x_n)$ from $x_1, \ldots, x_n$. Appending $2n-1$ gates to $\mathcal{N}_1$ which successively compare each of the n outputs of $\mathcal{N}_1$ with the corresponding variable $y_1, \ldots, y_n$ forms a $2n$-input, one-output circuit which evaluates the predicate $(F(x_1, \ldots, x_n) = (y_1, \ldots, y_n))$. Call this new circuit $\mathcal{N}_2$. The function computed by $\mathcal{N}_2$ is

$$\bigwedge_{i=1}^{n} (f_i(x_1, \ldots, x_n) \oplus \overline{y}_i)$$

and the Δ-transform of this function is

$$\bigoplus_{\substack{(\alpha_1, \ldots, \alpha_n) \leq (x_1, \ldots, x_n) \\ (\beta_1, \ldots, \beta_n) \leq (y_1, \ldots, y_n)}} (F(\alpha_1, \ldots, \alpha_n) = (\beta_1, \ldots, \beta_n)).$$

Now consider the circuit $\mathcal{N}_3$, which is identical to $\mathcal{N}_2$, except that there are two input lines for each element in y_i and that the n pairs $y_1, z_1, \ldots, y_n, z_n$ are connected to a $2n-1$ gate "protocol-checking" circuit which computes

$$\bigwedge_{i=1}^{n} (y_i \oplus z_i).$$

The output of the protocol checker, which is multiplied by the output of $\mathcal{N}_2$, is 1 if and only if y_i is the complement of z_i for each $1 \leq i \leq n$. The function computed by $\mathcal{N}_3$ is equal to $(F(x_1, \ldots, x_n) = (y_1, \ldots, y_n))$ if every pair y_i, z_i are complementary, but is 0 otherwise. $\mathcal{N}_3$ has $C(F) + 4n - 1$ gates.

By hypothesis, $C(\Delta f) \leq KC(f)$ for all $f : B^n \to B$ and all n sufficiently large. In particular, there exists a circuit $\mathcal{N}_4$ of size $\leq K(C(F) + 4n - 1)$ for the Δ-transform of the function computed by $\mathcal{N}_3$. $\mathcal{N}_4$ computes

$$\bigoplus_{\substack{(\alpha_1,\ldots,\alpha_n)\leq(x_1,\ldots,x_n)\\(\beta_1,\ldots,\beta_n)<(y_1,\ldots,y_n)\\(\gamma_1,\ldots,\gamma_n)\leq(z_1,\ldots,z_n)}} ((F(\alpha_1,\ldots,\alpha_n)=(\beta_1,\ldots,\beta_n))\wedge(\mathit{protocol}\quad \mathit{satisfied})).$$

If z_i is set equal to $\overline{y}_i$ for each i, then the last sum is equal to

$$\bigoplus_{\substack{(\alpha_1,\ldots,\alpha_n)\leq(x_1,\ldots,x_n)\\(\beta_1,\ldots,\beta_n)\leq(y_1,\ldots,y_n)\\(\gamma_1,\ldots,\gamma_n)\leq(\overline{y}_1,\ldots,\overline{y}_n)}} ((F(\alpha_1,\ldots,\alpha_n)=(\beta_1,\ldots,\beta_n))\wedge((\gamma_1,\ldots,\gamma_n)=(\overline{\beta}_1,\ldots,\overline{\beta}_n)))$$

$$= \bigoplus_{(\alpha_1,\ldots,\alpha_n)\leq(x_1,\ldots,x_n)} (F(\alpha_1,\ldots,\alpha_n)=(y_1,\ldots,y_n))$$

$$= \bigoplus_{(\alpha_1,\ldots,\alpha_n)\leq(x_1,\ldots,x_n)} ((\alpha_1,\ldots,\alpha_n)=F^{-1}(y_1,\ldots,y_n))$$

$$= \begin{cases} 1, & \text{if } F^{-1}(y_1,\ldots,y_n)\leq(x_1,\ldots,x_n)\\ 0, & \text{otherwise,}\end{cases}$$

since $\beta_i \leq y_i$ and $\overline{\beta}_i \leq \overline{y}_i$ together imply that $\beta_i = y_i$.

Let $\mathcal{N}_5$ be an n-input m-output circuit consisting of n copies of $\mathcal{N}_4$, where the ith copy has inputs $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$ set to 1 and input x_i set to 0, and where each of the copies has inputs $z_1, \ldots, z_n$ set to $\overline{y}_1, \ldots, \overline{y}_n$. The output of each copy of $\mathcal{N}_4$ is complemented to produce the final outputs for $\mathcal{N}_5$. The ith output of $\mathcal{N}_5$ is 1 if and only if the ith component of $F^{-1}(y_1,\ldots,y_n)$ is not ≤ 0. That is, $\mathcal{N}_5$ computes $F^{-1}(y_1,\ldots,y_n)$. $\mathcal{N}_5$ contains no more than $KnC(F)+4Kn^2+n^2-Kn+2$ gates. ∎

This result can be improved to $C(F^{-1}) = O(C(F))$ if it should be true that circuit complexity possesses the Boolean gradient property.

— Chapter Ten —

SUMMARY AND CONCLUSIONS

In addition to definitions, technical lemmas, and revised proofs of previously-known results, this paper has presented the following new combinatorial or complexity-theoretic results:

(a) CIRCUIT COMPLEXITY OVER $\{\oplus\}$ and CIRCUIT COMPLEXITY OVER $\{\vee\}$ are $\mathcal{NP}$-complete.

(b) Given the matrix associated with a function that is generated by a commutative, associative operation, the complexity of the function and the complexity of the function associated with the transpose matrix are the same. A simple procedure generates one circuit from the other.

(c) J. Olivos's bound on the complexity of a monomial with respect to an addition chain can be extended to the simultaneous computation of sets of monomials, and with a minor modification, to computation by addition-subtraction chains.

(d) $(.25)n^2 \log^{-1} n + o(n^2 \log^{-1} n)$ is an upper bound on the circuit complexity of the hardest quadratic monotone Boolean function. This matches the previously known lower bound, which was based on a counting argument.

(e) The complexity of the hardest linear function with respect to circuits of width n is $n^2 \log^{-1} n + o(n^2 \log^{-1} n)$. This is twice the complexity of the hardest linear function with respect to circuits of arbitrary width.

(f) Line integrals on Boolean functions, defined analogously to their real or complex counterparts, satisfy the same conditions for path independence.

(g) There are exactly 2^{2^n-1} Boolean polynomials whose functional values are identical to their coefficients.

(h) There exist Boolean functions whose circuit complexity differs from that of their inverses.

(i) If Δf has a small circuit whenever f does, then the circuit complexity of a function and its inverse cannot differ by more than a factor of n.

(j) Nearly all n-input n-output functions with the property that all of the outputs take the value 1 whenever two or more of their inputs are 1 have circuit complexity over the monotone basis that is greater than $\frac{(1-\epsilon)n^2}{2\log n}$. However, any sequence of such functions with monotone circuit complexity greater than $n^{1+\delta}$ is (1) hard to describe, and (2) can be used to define a sequence of functions $B^n \to B$ whose circuit complexity grows exponentially.

(k) Displaying a set of subsets of a finite set whose complexity with respect to a complete basis exceeds $2n + O(\log n)$ would be tantamount to displaying a Boolean function with nonlinear complexity. A concrete set is given whose set complexity is $2n - O(\log n)$ with respect to the complete basis, as well as a set whose complexity is at least $2n + O\sqrt{n}$ with respect to union operations.

(l) If circuit complexity satisfies a property analogous to one possessed by algebraic complexity, then the circuit complexity of a function with n inputs and 2^n outputs is within a constant factor of the complexity of the function with transposed truth table.

It has been known since at least 1950 that nearly all finite Boolean functions have exponential circuit complexity. However, as yet, no one has demonstrated a concrete example of a function with nonlinear circuit complexity, except via diagonalization-type arguments [e.g., STO]. This fact has hampered progress in other areas of complexity theory. Perhaps the most outstanding single open problem in the theory of the complexity of finite functions is to display a well-defined Boolean function with at least polynomial circuit complexity.

One possibility that might explain this difficulty is that, up to linear factors, the circuit complexity of a Boolean function may be unchanged under a wide variety of combinatorially or geometrically simple alterations of its truth table.

This amounts to the statement that $C(F)$, the circuit complexity of $F : B^n \rightarrow B^m$, is a *robust* measure of complexity. If so, it would be easy to see why finding examples of Boolean functions with nonlinear circuit complexity is so difficult: a candidate function would have to be very different from the simple functions of ordinary discourse, since a simple transform of an easy function could not have large complexity.

Whether circuit complexity actually is robust in this sense remains an open question. What this paper has shown, at most, is that several different notions of robustness are closely related. The hypothesis that circuit complexity *is* robust has not been ruled out by any of the counting or combinatorial arguments currently available in the literature, however.

If it should turn out that circuit complexity is *not* robust, it would be interesting to know if there is another combinatorial measure of Boolean function complexity that is robust, while still being useful. Paradoxically, the more robust a measure is, the harder it seems to be to prove lower bounds for the complexity of particular functions under that measure, except by self-reference. For example, if one measures the complexity of a sequence of finite functions by the size of the programs necessary to compute them, one has a measure that is robust, but for which it is difficult to display concrete functions with nonlinear lower bounds. The most robust measure of all would be one for which any easily-described transformation of a function's description would not change the function's complexity. Difficult functions under such a measure would probably be very difficult to describe, indeed.

Among the key open problems related to the topics discussed in this paper are the following:

- Does $C_{\oplus}(F) = C(F)$ for all linear functions $F : B^n \rightarrow B^m$? Are the two complexities related by a constant factor?

- Are $C_{set}([a_{ij}]) = C_{set}([a_{ij}]^T)$ related by a constant factor?

- If the linear function $F : B^n \rightarrow B^n$ has matrix $[a_{ij}]$, are $C_{set}([a_{ij}])$ and $C(F)$ linearly related? What about $C_{set}([a_{ij}])$ and $C_{\oplus}$?

- Does Δf have a small circuit if f does?

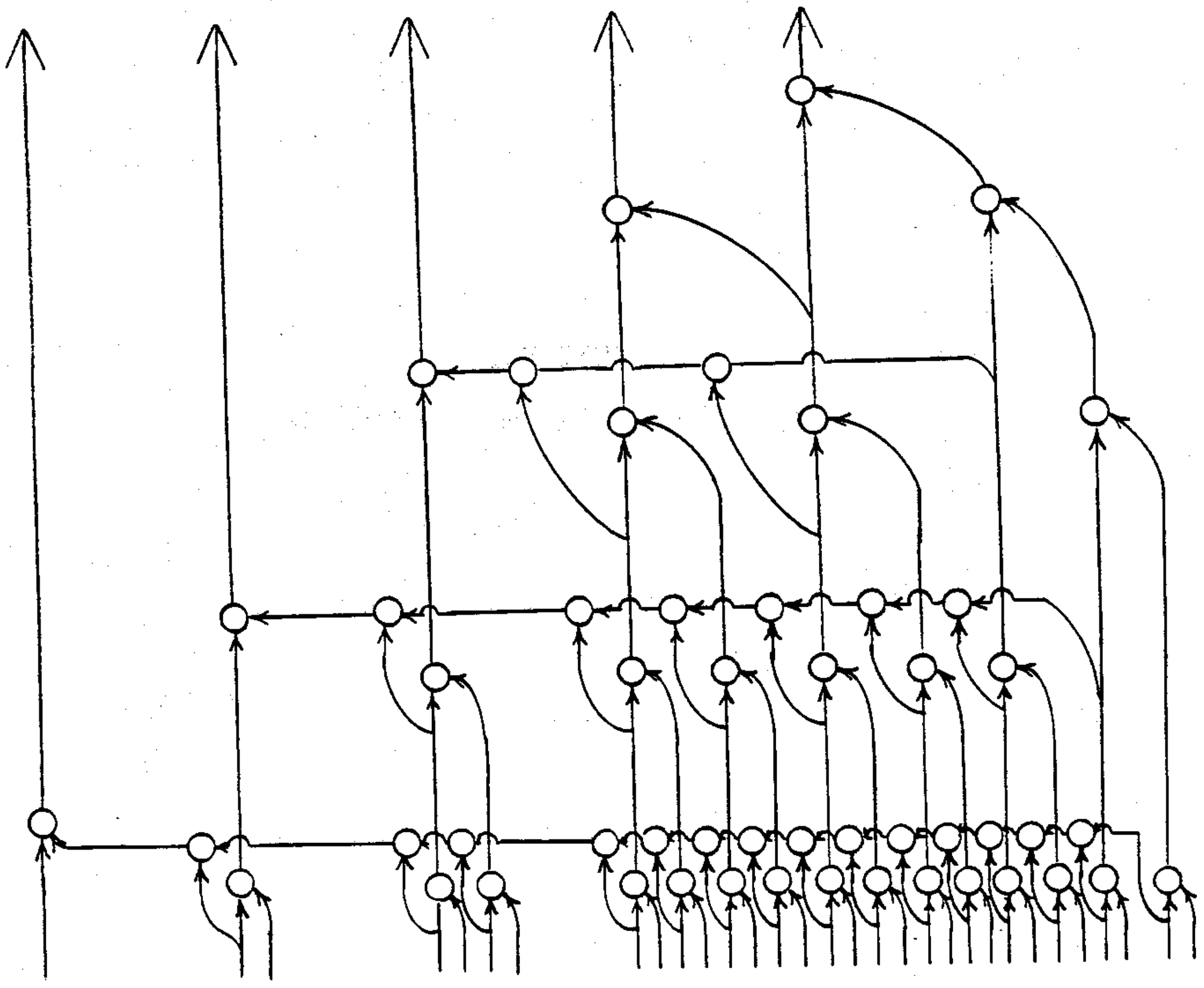
- Does the gradient of f have a small circuit if f does?

- Does F^{-1} have a small circuit if F does?

Answers to any of these questions, either positive or negative, would go a long way to elucidating the nature of combinatorial complexity.

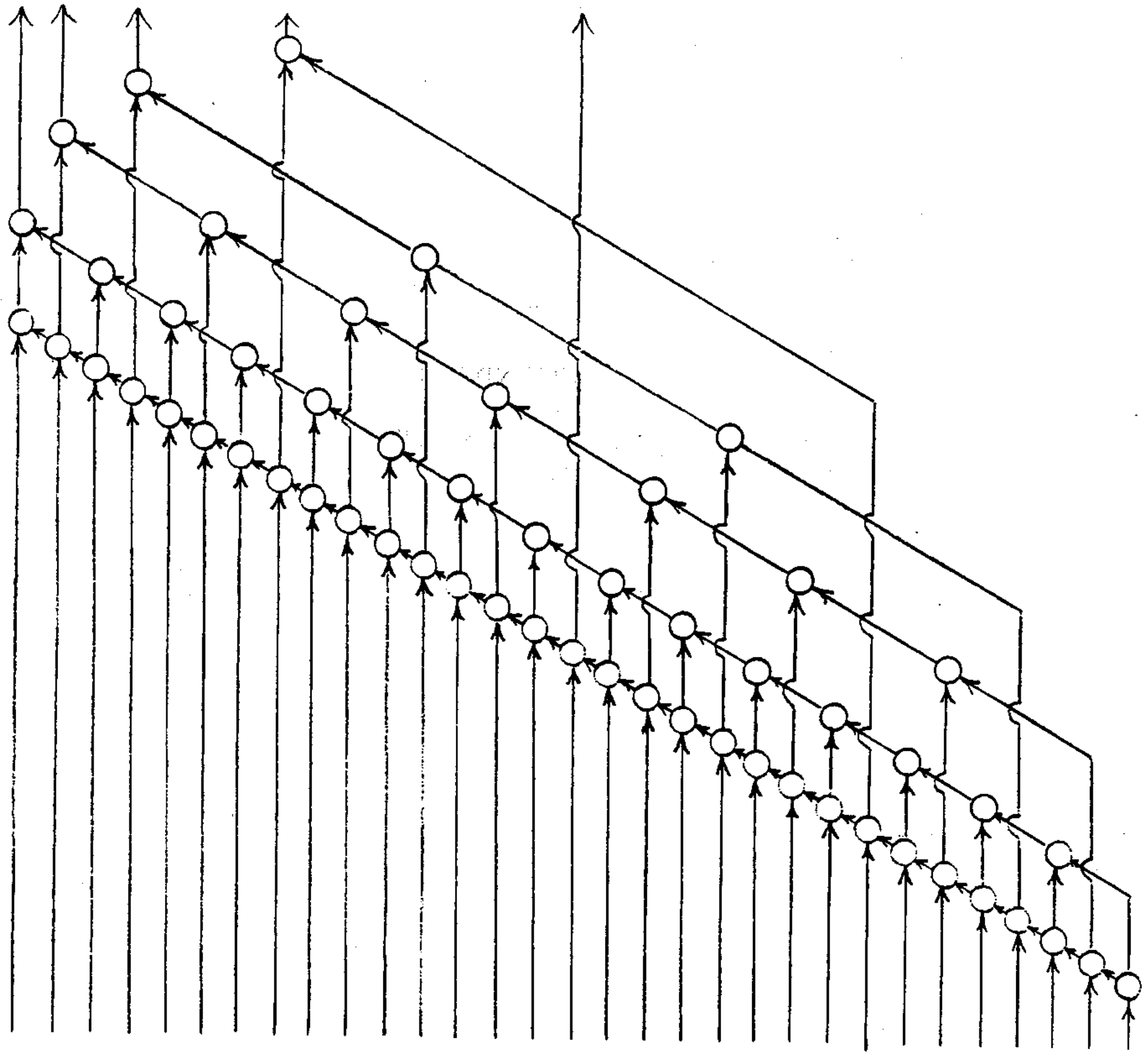
APPENDIX A

Examples of Circuits Based on Band Synthesis



Circuit Based on Synthesizing Vectors in Order of Weight

52 gates suffice to compute any 31-input, 5-output, function over a single commutative, associative operation whose *index* + *period* = 2. ($n + 21$ gates suffice for any n-input, 5-output function.)



Circuit Based on Reflected Gray Code

53 gates suffice to compute any 32-input, 5-output linear Boolean function.

APPENDIX B

Table of $\Delta(F)$ for $F: B^3 \rightarrow B^3$

$f(x_1, x_2, x_3)$	$\Delta f(x_1, x_2, x_3)$
0 0 0 0 1 1 1 1	0 0 0 0 1 1 1 1
0 0 1 1 0 0 1 1	0 0 1 1 0 0 1 1
0 1 0 1 0 1 0 1	0 1 0 1 0 1 0 1
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 1
0 0 0 0 0 0 1 0	0 0 0 0 0 0 1 1
0 0 0 0 0 0 1 1	0 0 0 0 0 0 1 0
0 0 0 0 0 1 0 0	0 0 0 0 0 1 0 1
0 0 0 0 0 1 0 1	0 0 0 0 0 1 0 0
0 0 0 0 0 1 1 0	0 0 0 0 0 1 1 0
0 0 0 0 0 1 1 1	0 0 0 0 0 1 1 1
0 0 0 0 1 0 0 0	0 0 0 0 1 1 1 1
0 0 0 0 1 0 0 1	0 0 0 0 1 1 1 0
0 0 0 0 1 0 1 0	0 0 0 0 1 1 0 0
0 0 0 0 1 0 1 1	0 0 0 0 1 1 0 1
0 0 0 0 1 1 0 0	0 0 0 0 1 0 1 0
0 0 0 0 1 1 0 1	0 0 0 0 1 0 1 1
0 0 0 0 1 1 1 0	0 0 0 0 1 0 0 1
0 0 0 0 1 1 1 1	0 0 0 0 1 0 0 0
0 0 0 1 0 0 0 0	0 0 0 1 0 0 0 1
0 0 0 1 0 0 0 1	0 0 0 1 0 0 0 0
0 0 0 1 0 0 1 0	0 0 0 1 0 0 1 0
0 0 0 1 0 0 1 1	0 0 0 1 0 0 1 1
0 0 0 1 0 1 0 0	0 0 0 1 0 1 0 0
0 0 0 1 0 1 0 1	0 0 0 1 0 1 0 1
0 0 0 1 0 1 1 0	0 0 0 1 0 1 1 1
0 0 0 1 0 1 1 1	0 0 0 1 0 1 1 0
0 0 0 1 1 0 0 0	0 0 0 1 1 1 1 0
0 0 0 1 1 0 0 1	0 0 0 1 1 1 1 1
0 0 0 1 1 0 1 0	0 0 0 1 1 1 0 1
0 0 0 1 1 0 1 1	0 0 0 1 1 1 0 0
0 0 0 1 1 1 0 0	0 0 0 1 1 0 1 1
0 0 0 1 1 1 0 1	0 0 0 1 1 0 1 0
0 0 0 1 1 1 1 0	0 0 0 1 1 0 0 0
0 0 0 1 1 1 1 1	0 0 0 1 1 0 0 1

$f(x_1, x_2, x_3)$	$\Delta f(x_1, x_2, x_3)$
0 0 0 0 1 1 1 1	0 0 0 0 1 1 1 1
0 0 1 1 0 0 1 1	0 0 1 1 0 0 1 1
0 1 0 1 0 1 0 1	0 1 0 1 0 1 0 1
0 0 1 0 0 0 0 0	0 0 1 1 0 0 1 1
0 0 1 0 0 0 0 1	0 0 1 1 0 0 1 0
0 0 1 0 0 0 1 0	0 0 1 1 0 0 0 0
0 0 1 0 0 0 1 1	0 0 1 1 0 0 0 1
0 0 1 0 0 1 0 0	0 0 1 1 0 1 1 0
0 0 1 0 0 1 0 1	0 0 1 1 0 1 1 1
0 0 1 0 0 1 1 0	0 0 1 1 0 1 0 1
0 0 1 0 0 1 1 1	0 0 1 1 0 1 0 0
0 0 1 0 1 0 0 0	0 0 1 1 1 1 0 0
0 0 1 0 1 0 0 1	0 0 1 1 1 1 0 1
0 0 1 0 1 0 1 0	0 0 1 1 1 1 1 1
0 0 1 0 1 0 1 1	0 0 1 1 1 1 1 0
0 0 1 0 1 1 0 0	0 0 1 1 1 0 0 1
0 0 1 0 1 1 0 1	0 0 1 1 1 0 0 0
0 0 1 0 1 1 1 0	0 0 1 1 1 0 1 0
0 0 1 0 1 1 1 1	0 0 1 1 1 0 1 1
0 0 1 1 0 0 0 0	0 0 1 0 0 0 1 0
0 0 1 1 0 0 0 1	0 0 1 0 0 0 1 1
0 0 1 1 0 0 1 0	0 0 1 0 0 0 0 1
0 0 1 1 0 0 1 1	0 0 1 0 0 0 0 0
0 0 1 1 0 1 0 0	0 0 1 0 0 1 1 1
0 0 1 1 0 1 0 1	0 0 1 0 0 1 1 0
0 0 1 1 0 1 1 0	0 0 1 0 0 1 0 0
0 0 1 1 0 1 1 1	0 0 1 0 0 1 0 1
0 0 1 1 1 0 0 0	0 0 1 0 1 1 0 1
0 0 1 1 1 0 0 1	0 0 1 0 1 1 0 0
0 0 1 1 1 0 1 0	0 0 1 0 1 1 1 0
0 0 1 1 1 0 1 1	0 0 1 0 1 1 1 1
0 0 1 1 1 1 0 0	0 0 1 0 1 0 0 0
0 0 1 1 1 1 0 1	0 0 1 0 1 0 0 1
0 0 1 1 1 1 1 0	0 0 1 0 1 0 1 1
0 0 1 1 1 1 1 1	0 0 1 0 1 0 1 0

$f(x_1, x_2, x_3)$	$\Delta f(x_1, x_2, x_3)$
0 0 0 0 1 1 1 1	0 0 0 0 1 1 1 1
0 0 1 1 0 0 1 1	0 0 1 1 0 0 1 1
0 1 0 1 0 1 0 1	0 1 0 1 0 1 0 1
0 1 0 0 0 0 0 0	0 1 0 1 0 1 0 1
0 1 0 0 0 0 0 1	0 1 0 1 0 1 0 0
0 1 0 0 0 0 1 0	0 1 0 1 0 1 1 0
0 1 0 0 0 0 1 1	0 1 0 1 0 1 1 1
0 1 0 0 0 1 0 0	0 1 0 1 0 0 0 0
0 1 0 0 0 1 0 1	0 1 0 1 0 0 0 1
0 1 0 0 0 1 1 0	0 1 0 1 0 0 1 1
0 1 0 0 0 1 1 1	0 1 0 1 0 0 1 0
0 1 0 0 1 0 0 0	0 1 0 1 1 0 1 0
0 1 0 0 1 0 0 1	0 1 0 1 1 0 1 1
0 1 0 0 1 0 1 0	0 1 0 1 1 0 0 1
0 1 0 0 1 0 1 1	0 1 0 1 1 0 0 0
0 1 0 0 1 1 0 0	0 1 0 1 1 1 1 1
0 1 0 0 1 1 0 1	0 1 0 1 1 1 1 0
0 1 0 0 1 1 1 0	0 1 0 1 1 1 0 0
0 1 0 0 1 1 1 1	0 1 0 1 1 1 0 1
0 1 0 1 0 0 0 0	0 1 0 0 0 1 0 0
0 1 0 1 0 0 0 1	0 1 0 0 0 1 0 1
0 1 0 1 0 0 1 0	0 1 0 0 0 1 1 1
0 1 0 1 0 0 1 1	0 1 0 0 0 1 1 0
0 1 0 1 0 1 0 0	0 1 0 0 0 0 0 1
0 1 0 1 0 1 0 1	0 1 0 0 0 0 0 0
0 1 0 1 0 1 1 0	0 1 0 0 0 0 1 0
0 1 0 1 0 1 1 1	0 1 0 0 0 0 1 1
0 1 0 1 1 0 0 0	0 1 0 0 1 0 1 1
0 1 0 1 1 0 0 1	0 1 0 0 1 0 1 0
0 1 0 1 1 0 1 0	0 1 0 0 1 0 0 0
0 1 0 1 1 0 1 1	0 1 0 0 1 0 0 1
0 1 0 1 1 1 0 0	0 1 0 0 1 1 1 0
0 1 0 1 1 1 0 1	0 1 0 0 1 1 1 1
0 1 0 1 1 1 1 0	0 1 0 0 1 1 0 1
0 1 0 1 1 1 1 1	0 1 0 0 1 1 0 0

$f(x_1, x_2, x_3)$	$\Delta f(x_1, x_2, x_3)$
0 0 0 0 1 1 1 1	0 0 0 0 1 1 1 1
0 0 1 1 0 0 1 1	0 0 1 1 0 0 1 1
0 1 0 1 0 1 0 1	0 1 0 1 0 1 0 1
0 1 1 0 0 0 0 0	0 1 1 0 0 1 1 0
0 1 1 0 0 0 0 1	0 1 1 0 0 1 1 1
0 1 1 0 0 0 1 0	0 1 1 0 0 1 0 1
0 1 1 0 0 0 1 1	0 1 1 0 0 1 0 0
0 1 1 0 0 1 0 0	0 1 1 0 0 0 1 1
0 1 1 0 0 1 0 1	0 1 1 0 0 0 1 0
0 1 1 0 0 1 1 0	0 1 1 0 0 0 0 0
0 1 1 0 0 1 1 1	0 1 1 0 0 0 0 1
0 1 1 0 1 0 0 0	0 1 1 0 1 0 0 1
0 1 1 0 1 0 0 1	0 1 1 0 1 0 0 0
0 1 1 0 1 0 1 0	0 1 1 0 1 0 1 0
0 1 1 0 1 0 1 1	0 1 1 0 1 0 1 1
0 1 1 0 1 1 0 0	0 1 1 0 1 1 0 0
0 1 1 0 1 1 0 1	0 1 1 0 1 1 0 1
0 1 1 0 1 1 1 0	0 1 1 0 1 1 1 1
0 1 1 0 1 1 1 1	0 1 1 0 1 1 1 0
0 1 1 1 0 0 0 0	0 1 1 1 0 1 1 1
0 1 1 1 0 0 0 1	0 1 1 1 0 1 1 0
0 1 1 1 0 0 1 0	0 1 1 1 0 1 0 0
0 1 1 1 0 0 1 1	0 1 1 1 0 1 0 1
0 1 1 1 0 1 0 0	0 1 1 1 0 0 1 0
0 1 1 1 0 1 0 1	0 1 1 1 0 0 1 1
0 1 1 1 0 1 1 0	0 1 1 1 0 0 0 1
0 1 1 1 0 1 1 1	0 1 1 1 0 0 0 0
0 1 1 1 1 0 0 0	0 1 1 1 1 0 0 0
0 1 1 1 1 0 0 1	0 1 1 1 1 0 0 1
0 1 1 1 1 0 1 0	0 1 1 1 1 0 1 1
0 1 1 1 1 0 1 1	0 1 1 1 1 0 1 0
0 1 1 1 1 1 0 0	0 1 1 1 1 1 0 1
0 1 1 1 1 1 0 1	0 1 1 1 1 1 0 0
0 1 1 1 1 1 1 0	0 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1	0 1 1 1 1 1 1 1

$f(x_1, x_2, x_3)$	$\Delta f(x_1, x_2, x_3)$
0 0 0 0 1 1 1 1	0 0 0 0 1 1 1 1
0 0 1 1 0 0 1 1	0 0 1 1 0 0 1 1
0 1 0 1 0 1 0 1	0 1 0 1 0 1 0 1
1 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1
1 0 0 0 0 0 0 1	1 1 1 1 1 1 1 0
1 0 0 0 0 0 1 0	1 1 1 1 1 1 0 0
1 0 0 0 0 0 1 1	1 1 1 1 1 1 0 1
1 0 0 0 0 1 0 0	1 1 1 1 1 0 1 0
1 0 0 0 0 1 0 1	1 1 1 1 1 0 1 1
1 0 0 0 0 1 1 0	1 1 1 1 1 0 0 1
1 0 0 0 0 1 1 1	1 1 1 1 1 0 0 0
1 0 0 0 1 0 0 0	1 1 1 1 0 0 0 0
1 0 0 0 1 0 0 1	1 1 1 1 0 0 0 1
1 0 0 0 1 0 1 0	1 1 1 1 0 0 1 1
1 0 0 0 1 0 1 1	1 1 1 1 0 0 1 0
1 0 0 0 1 1 0 0	1 1 1 1 0 1 0 1
1 0 0 0 1 1 0 1	1 1 1 1 0 1 0 0
1 0 0 0 1 1 1 0	1 1 1 1 0 1 1 0
1 0 0 0 1 1 1 1	1 1 1 1 0 1 1 1
1 0 0 1 0 0 0 0	1 1 1 0 1 1 1 0
1 0 0 1 0 0 0 1	1 1 1 0 1 1 1 1
1 0 0 1 0 0 1 0	1 1 1 0 1 1 0 1
1 0 0 1 0 0 1 1	1 1 1 0 1 1 0 0
1 0 0 1 0 1 0 0	1 1 1 0 1 0 1 1
1 0 0 1 0 1 0 1	1 1 1 0 1 0 1 0
1 0 0 1 0 1 1 0	1 1 1 0 1 0 0 0
1 0 0 1 0 1 1 1	1 1 1 0 1 0 0 1
1 0 0 1 1 0 0 0	1 1 1 0 0 0 0 1
1 0 0 1 1 0 0 1	1 1 1 0 0 0 0 0
1 0 0 1 1 0 1 0	1 1 1 0 0 0 1 0
1 0 0 1 1 0 1 1	1 1 1 0 0 0 1 1
1 0 0 1 1 1 0 0	1 1 1 0 0 1 0 0
1 0 0 1 1 1 0 1	1 1 1 0 0 1 0 1
1 0 0 1 1 1 1 0	1 1 1 0 0 1 1 1
1 0 0 1 1 1 1 1	1 1 1 0 0 1 1 0

$f(x_1, x_2, x_3)$	$\Delta f(x_1, x_2, x_3)$
0 0 0 0 1 1 1 1	0 0 0 0 1 1 1 1
0 0 1 1 0 0 1 1	0 0 1 1 0 0 1 1
0 1 0 1 0 1 0 1	0 1 0 1 0 1 0 1
1 0 1 0 0 0 0 0	1 1 0 0 1 1 0 0
1 0 1 0 0 0 0 1	1 1 0 0 1 1 0 1
1 0 1 0 0 0 1 0	1 1 0 0 1 1 1 1
1 0 1 0 0 0 1 1	1 1 0 0 1 1 1 0
1 0 1 0 0 1 0 0	1 1 0 0 1 0 0 1
1 0 1 0 0 1 0 1	1 1 0 0 1 0 0 0
1 0 1 0 0 1 1 0	1 1 0 0 1 0 1 0
1 0 1 0 0 1 1 1	1 1 0 0 1 0 1 1
1 0 1 0 1 0 0 0	1 1 0 0 0 0 1 1
1 0 1 0 1 0 0 1	1 1 0 0 0 0 1 0
1 0 1 0 1 0 1 0	1 1 0 0 0 0 0 0
1 0 1 0 1 0 1 1	1 1 0 0 0 0 0 1
1 0 1 0 1 1 0 0	1 1 0 0 0 1 1 0
1 0 1 0 1 1 0 1	1 1 0 0 0 1 1 1
1 0 1 0 1 1 1 0	1 1 0 0 0 1 0 1
1 0 1 0 1 1 1 1	1 1 0 0 0 1 0 0
1 0 1 1 0 0 0 0	1 1 0 1 1 1 0 1
1 0 1 1 0 0 0 1	1 1 0 1 1 1 0 0
1 0 1 1 0 0 1 0	1 1 0 1 1 1 1 0
1 0 1 1 0 0 1 1	1 1 0 1 1 1 1 1
1 0 1 1 0 1 0 0	1 1 0 1 1 0 0 0
1 0 1 1 0 1 0 1	1 1 0 1 1 0 0 1
1 0 1 1 0 1 1 0	1 1 0 1 1 0 1 1
1 0 1 1 0 1 1 1	1 1 0 1 1 0 1 0
1 0 1 1 1 0 0 0	1 1 0 1 0 0 1 0
1 0 1 1 1 0 0 1	1 1 0 1 0 0 1 1
1 0 1 1 1 0 1 0	1 1 0 1 0 0 0 1
1 0 1 1 1 0 1 1	1 1 0 1 0 0 0 0
1 0 1 1 1 1 0 0	1 1 0 1 0 1 1 1
1 0 1 1 1 1 0 1	1 1 0 1 0 1 1 0
1 0 1 1 1 1 1 0	1 1 0 1 0 1 0 0
1 0 1 1 1 1 1 1	1 1 0 1 0 1 0 1

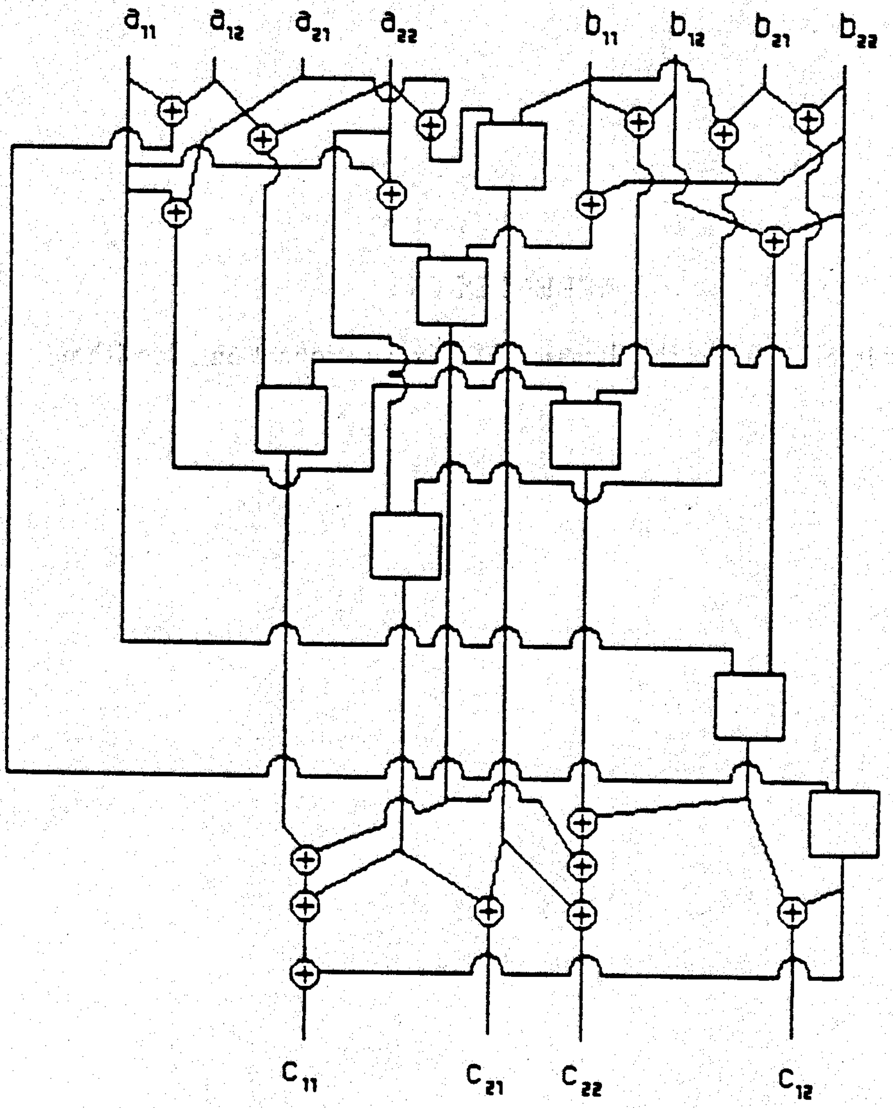
$f(x_1, x_2, x_3)$	$\Delta f(x_1, x_2, x_3)$
0 0 0 0 1 1 1 1	0 0 0 0 1 1 1 1
0 0 1 1 0 0 1 1	0 0 1 1 0 0 1 1
0 1 0 1 0 1 0 1	0 1 0 1 0 1 0 1
1 1 0 0 0 0 0 0	1 0 1 0 1 0 1 0
1 1 0 0 0 0 0 1	1 0 1 0 1 0 1 1
1 1 0 0 0 0 1 0	1 0 1 0 1 0 0 1
1 1 0 0 0 0 1 1	1 0 1 0 1 0 0 0
1 1 0 0 0 1 0 0	1 0 1 0 1 1 1 1
1 1 0 0 0 1 0 1	1 0 1 0 1 1 1 0
1 1 0 0 0 1 1 0	1 0 1 0 1 1 0 0
1 1 0 0 0 1 1 1	1 0 1 0 1 1 0 1
1 1 0 0 1 0 0 0	1 0 1 0 0 1 0 1
1 1 0 0 1 0 0 1	1 0 1 0 0 1 0 0
1 1 0 0 1 0 1 0	1 0 1 0 0 1 1 0
1 1 0 0 1 0 1 1	1 0 1 0 0 1 1 1
1 1 0 0 1 1 0 0	1 0 1 0 0 0 0 0
1 1 0 0 1 1 0 1	1 0 1 0 0 0 0 1
1 1 0 0 1 1 1 0	1 0 1 0 0 0 1 1
1 1 0 0 1 1 1 1	1 0 1 0 0 0 1 0
1 1 0 1 0 0 0 0	1 0 1 1 1 0 1 1
1 1 0 1 0 0 0 1	1 0 1 1 1 0 1 0
1 1 0 1 0 0 1 0	1 0 1 1 1 0 0 0
1 1 0 1 0 0 1 1	1 0 1 1 1 0 0 1
1 1 0 1 0 1 0 0	1 0 1 1 1 1 1 0
1 1 0 1 0 1 0 1	1 0 1 1 1 1 1 1
1 1 0 1 0 1 1 0	1 0 1 1 1 1 0 1
1 1 0 1 0 1 1 1	1 0 1 1 1 1 0 0
1 1 0 1 1 0 0 0	1 0 1 1 0 1 0 0
1 1 0 1 1 0 0 1	1 0 1 1 0 1 0 1
1 1 0 1 1 0 1 0	1 0 1 1 0 1 1 1
1 1 0 1 1 0 1 1	1 0 1 1 0 1 1 0
1 1 0 1 1 1 0 0	1 0 1 1 0 0 0 1
1 1 0 1 1 1 0 1	1 0 1 1 0 0 0 0
1 1 0 1 1 1 1 0	1 0 1 1 0 0 1 0
1 1 0 1 1 1 1 1	1 0 1 1 0 0 1 1

$f(x_1, x_2, x_3)$	$\Delta f(x_1, x_2, x_3)$
0 0 0 0 1 1 1 1	0 0 0 0 1 1 1 1
0 0 1 1 0 0 1 1	0 0 1 1 0 0 1 1
0 1 0 1 0 1 0 1	0 1 0 1 0 1 0 1
1 1 1 0 0 0 0 0	1 0 0 1 1 0 0 1
1 1 1 0 0 0 0 1	1 0 0 1 1 0 0 0
1 1 1 0 0 0 1 0	1 0 0 1 1 0 1 0
1 1 1 0 0 0 1 1	1 0 0 1 1 0 1 1
1 1 1 0 0 1 0 0	1 0 0 1 1 1 0 0
1 1 1 0 0 1 0 1	1 0 0 1 1 1 0 1
1 1 1 0 0 1 1 0	1 0 0 1 1 1 1 1
1 1 1 0 0 1 1 1	1 0 0 1 1 1 1 0
1 1 1 0 1 0 0 0	1 0 0 1 0 1 1 0
1 1 1 0 1 0 0 1	1 0 0 1 0 1 1 1
1 1 1 0 1 0 1 0	1 0 0 1 0 1 0 1
1 1 1 0 1 0 1 1	1 0 0 1 0 1 0 0
1 1 1 0 1 1 0 0	1 0 0 1 0 0 1 1
1 1 1 0 1 1 0 1	1 0 0 1 0 0 1 0
1 1 1 0 1 1 1 0	1 0 0 1 0 0 0 0
1 1 1 0 1 1 1 1	1 0 0 1 0 0 0 1
1 1 1 1 0 0 0 0	1 0 0 0 1 0 0 0
1 1 1 1 0 0 0 1	1 0 0 0 1 0 0 1
1 1 1 1 0 0 1 0	1 0 0 0 1 0 1 1
1 1 1 1 0 0 1 1	1 0 0 0 1 0 1 0
1 1 1 1 0 1 0 0	1 0 0 0 1 1 0 1
1 1 1 1 0 1 0 1	1 0 0 0 1 1 0 0
1 1 1 1 0 1 1 0	1 0 0 0 1 1 1 0
1 1 1 1 0 1 1 1	1 0 0 0 1 1 1 1
1 1 1 1 1 0 0 0	1 0 0 0 0 1 1 1
1 1 1 1 1 0 0 1	1 0 0 0 0 1 1 0
1 1 1 1 1 0 1 0	1 0 0 0 0 1 0 0
1 1 1 1 1 0 1 1	1 0 0 0 0 1 0 1
1 1 1 1 1 1 0 0	1 0 0 0 0 0 1 0
1 1 1 1 1 1 0 1	1 0 0 0 0 0 1 1
1 1 1 1 1 1 1 0	1 0 0 0 0 0 0 1
1 1 1 1 1 1 1 1	1 0 0 0 0 0 0 0

APPENDIX C

Circuit Implementing the Strassen Matrix Multiplication Algorithm

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$



Strassen Multiplication Circuit

APPENDIX D

Equivalence Classes of Invertible Functions $B^3 \rightarrow B^3$

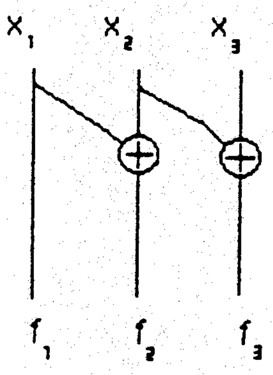
function	f_1	f_2	f_3	*inverse*
F_1	x_1	$x_1 \oplus x_2$	$x_2 \oplus x_3$	F_{15}
F_2	x_1	$x_2 \oplus x_3$	$x_1x_2 \oplus \bar{x}_1x_3$	F_{16}
F_3	x_1	$x_1 \oplus x_2 \oplus x_3$	$x_1x_2 \oplus \bar{x}_1x_3$	F_{17}
F_4	$x_1 \oplus x_2$	$x_1 \oplus x_3$	$x_1x_2 \oplus x_2x_3 \oplus x_3x_1$	F_{18}
F_5	$x_2 \oplus x_3$	$x_1 \oplus x_2 \oplus x_3$	$x_1x_2 \oplus \bar{x}_1x_3$	F_{19}
F_6	$x_2 \oplus x_3$	$x_1x_2 \oplus x_2x_3 \oplus x_3x_1$	$x_1x_2 \oplus x_2\bar{x}_3 \oplus \bar{x}_3x_1$	F_{20}
F_7	$x_2 \oplus x_3$	$x_1x_2 \oplus x_2x_3 \oplus x_3x_1$	$x_1 \oplus x_2\bar{x}_3$	F_{21}
F_8	$x_2 \oplus x_3$	$x_1x_2 \oplus x_2x_3 \oplus x_3x_1$	$x_1x_2 \oplus \bar{x}_1\bar{x}_3$	F_{22}
F_9	$x_2 \oplus x_3$	$x_1 \oplus x_2x_3$	$x_1x_2 \oplus \bar{x}_1x_3$	F_{23}
F_{10}	$x_1 \oplus x_2 \oplus x_3$	$x_1x_2 \oplus \bar{x}_1x_3$	$x_1x_2 \oplus \bar{x}_2\bar{x}_3$	F_{24}
F_{11}	$x_2 \oplus x_3$	$x_1x_2 \oplus \bar{x}_1x_3$	$x_1\bar{x}_3 \oplus \bar{x}_1x_2$	F_{25}
F_{12}	$x_1x_2 \oplus x_2x_3 \oplus x_3x_1$	$\bar{x}_1x_2 \oplus x_2x_3 \oplus x_3\bar{x}_1$	$x_1 \oplus x_2\bar{x}_3$	F_{26}
F_{13}	$x_1x_2 \oplus x_2x_3 \oplus x_3x_1$	$x_1 \oplus x_2\bar{x}_3$	$x_3 \oplus x_2\bar{x}_1$	F_{27}
F_{14}	$x_1x_2 \oplus x_2x_3 \oplus x_3x_1$	$x_1 \oplus x_2\bar{x}_3$	$x_1x_2 \oplus \bar{x}_3\bar{x}_1$	F_{28}
F_{15}	x_1	$x_1 \oplus x_2$	$x_1 \oplus x_2 \oplus x_3$	F_1
F_{16}	x_1	$x_3 \oplus \bar{x}_1x_2$	$x_3 \oplus x_1x_2$	F_2
F_{17}	x_1	$x_3 \oplus \bar{x}_1x_2$	$x_3 \oplus x_1\bar{x}_2$	F_3
F_{18}	$x_3 \oplus x_1x_2$	$x_3 \oplus x_1\bar{x}_2$	$x_3 \oplus \bar{x}_1x_3$	F_4
F_{19}	$x_1 \oplus x_2$	$x_3 \oplus x_1x_2$	$x_3 \oplus x_1\bar{x}_2$	F_5
F_{20}	$x_1x_2 \oplus \bar{x}_1x_3$	$x_1x_3 \oplus \bar{x}_1x_2$	$\bar{x}_1x_2 \oplus x_1\bar{x}_3$	F_6
F_{21}	$x_1x_2 \oplus \bar{x}_1\bar{x}_3$	$x_2 \oplus x_1x_3$	$x_2 \oplus x_1\bar{x}_3$	F_7
F_{22}	$x_2 \oplus \bar{x}_1\bar{x}_3$	$x_1x_3 \oplus \bar{x}_1x_2$	$x_1\bar{x}_3 \oplus \bar{x}_1x_2$	F_8
F_{23}	$x_2 \oplus \bar{x}_1x_3$	$x_3 \oplus x_1\bar{x}_2$	$x_3 \oplus x_1x_2$	F_9
F_{24}	$x_3 \oplus \bar{x}_1\bar{x}_2$	$x_2 \oplus x_1\bar{x}_3$	$x_1x_2 \oplus \bar{x}_1\bar{x}_3$	F_{10}
F_{25}	$x_1 \oplus x_2 \oplus x_3$	$x_1x_3 \oplus \bar{x}_1x_2$	$x_1\bar{x}_2 \oplus \bar{x}_1x_2$	F_{11}
F_{26}	$x_1\bar{x}_2 \oplus \bar{x}_2x_3 \oplus x_3x_1$	$x_1\bar{x}_3 \oplus x_2x_3$	$x_1x_3 \oplus x_2\bar{x}_3$	F_{12}
F_{27}	$x_1\bar{x}_3 \oplus x_2x_3$	$x_1 \oplus x_2\bar{x}_3$	$x_1x_2 \oplus \bar{x}_2x_3$	F_{13}
F_{28}	$x_1x_3 \oplus x_2\bar{x}_3$	$x_1\bar{x}_2 \oplus x_2x_3$	$x_1x_2 \oplus \bar{x}_2\bar{x}_3$	F_{14}

Reference: [LOR].

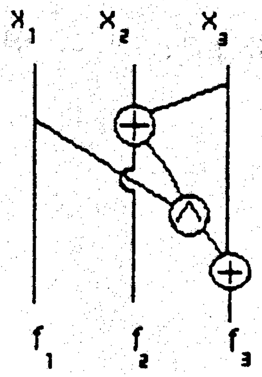
APPENDIX E

Optimal Circuits for Invertible $F: B^3 \rightarrow B^3$

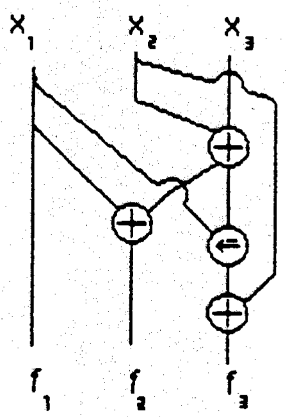
1


x_1
x_2
x_3
f_1
f_2
f_3

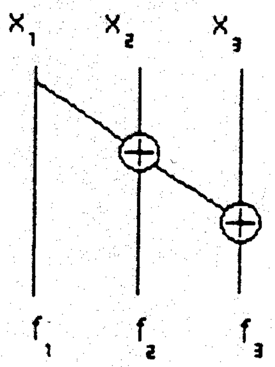
2


x_1
x_2
x_3
f_1
f_2
f_3

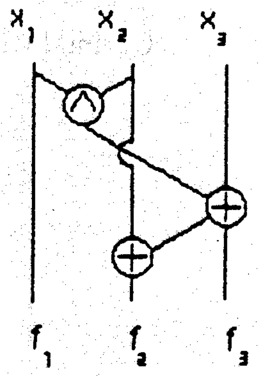
3


x_1
x_2
x_3
f_1
f_2
f_3

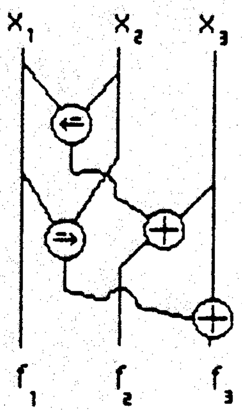
15


x_1
x_2
x_3
f_1
f_2
f_3

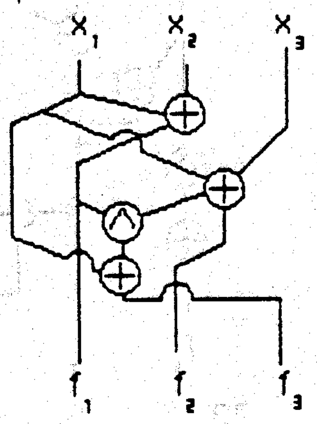
16


x_1
x_2
x_3
f_1
f_2
f_3

17

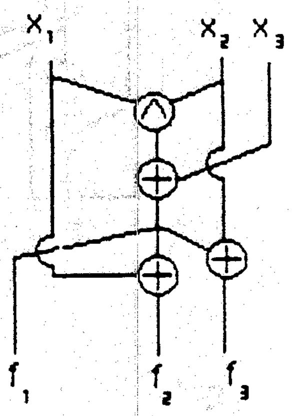

x_1
x_2
x_3
f_1
f_2
f_3

4



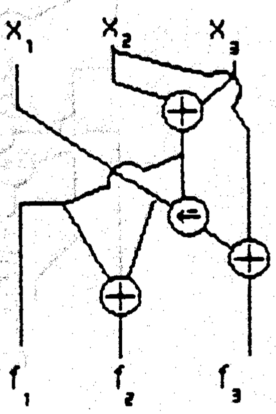
x_1
x_2
x_3
f_1
f_2
f_3

18



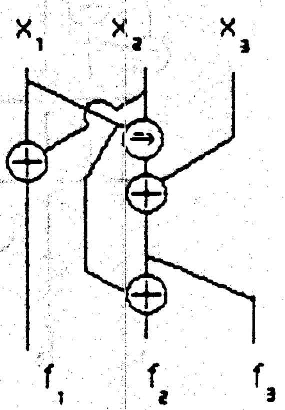
x_1
x_2
x_3
f_1
f_2
f_3

5



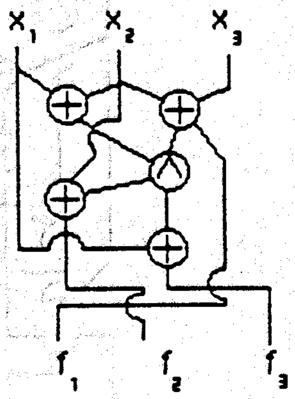
x_1
x_2
x_3
f_1
f_2
f_3

19



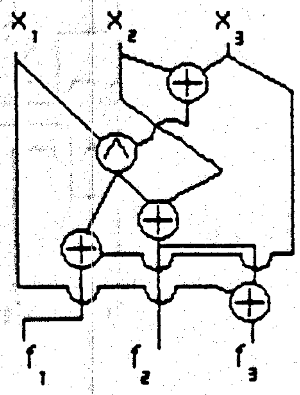
x_1
x_2
x_3
f_1
f_2
f_3

6



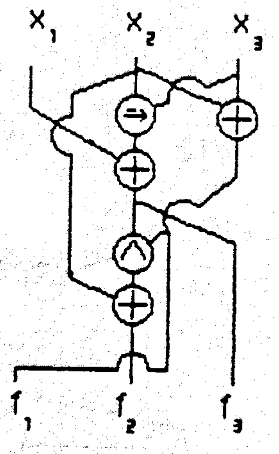
x_1
x_2
x_3
f_1
f_2
f_3

20



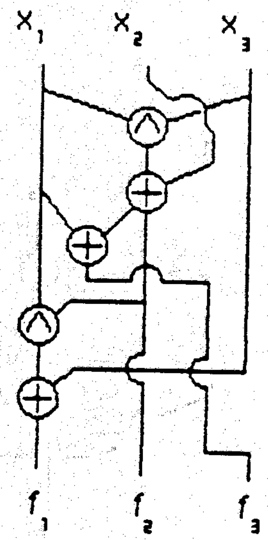
x_1
x_2
x_3
f_1
f_2
f_3

7



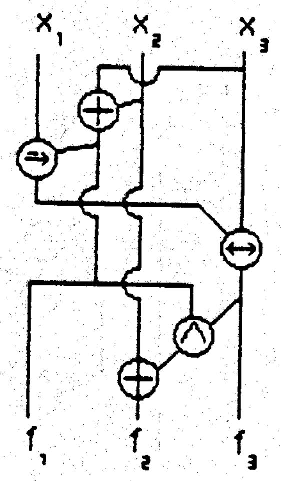
x_1 x_2 x_3
f_1 f_2 f_3

21



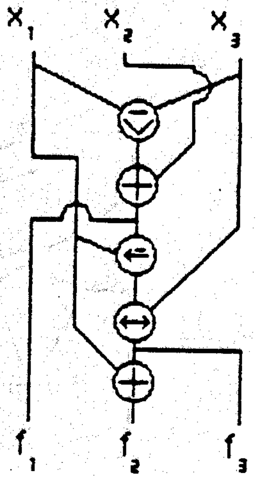
x_1 x_2 x_3
f_1 f_2 f_3

8



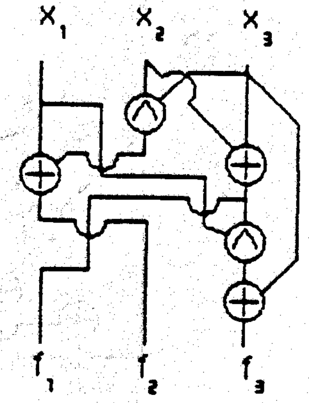
x_1 x_2 x_3
f_1 f_2 f_3

22



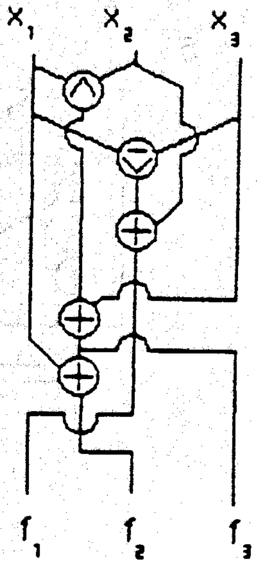
x_1 x_2 x_3
f_1 f_2 f_3

9



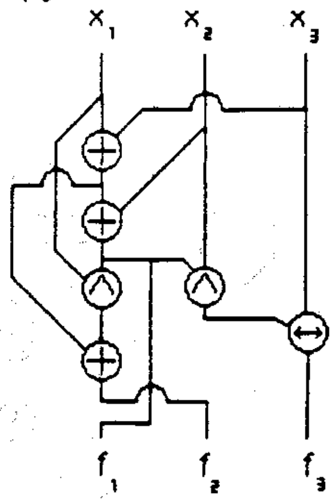
x_1 x_2 x_3
f_1 f_2 f_3

23

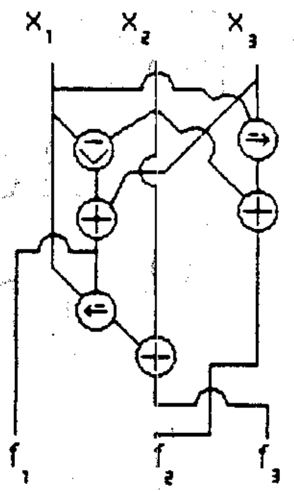


x_1 x_2 x_3
f_1 f_2 f_3

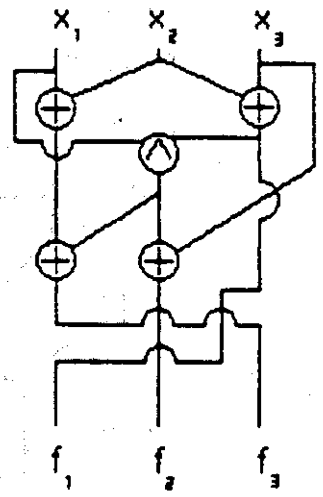
10



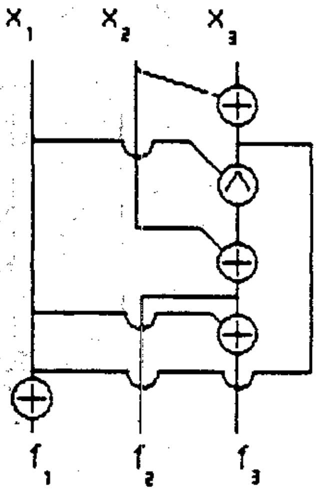
24



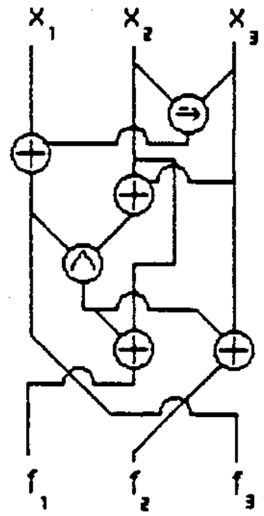
11



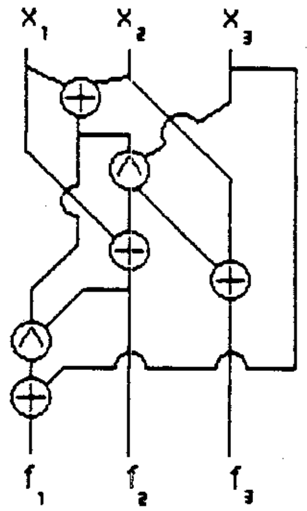
25



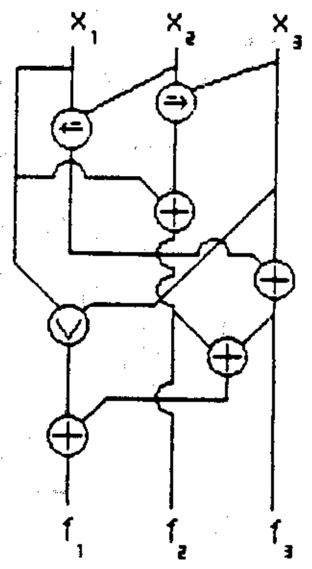
12



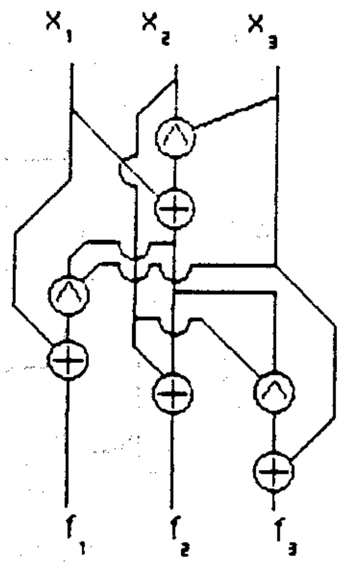
26



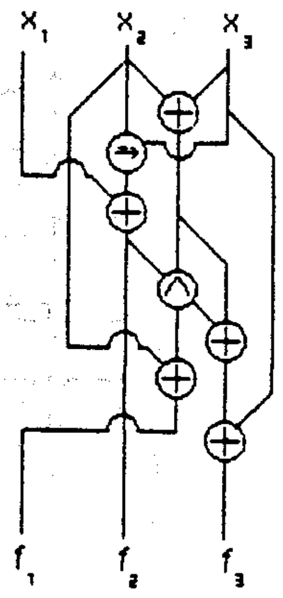
13



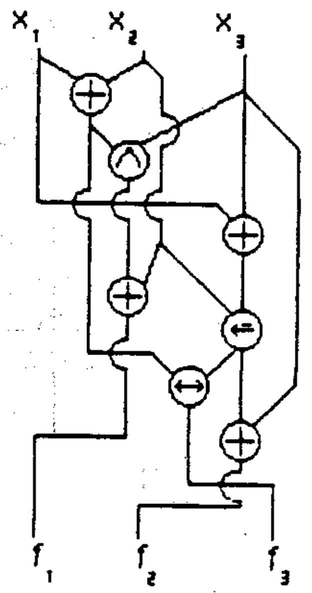
27



14



28



APPENDIX F

Rate of Growth Notation

Suppose that $f(n)$ and $g(n)$ are nonnegative real-valued functions defined on the natural numbers. The following notations are used to describe the relative rates of growth of f and g with respect to changes in n.

$f(n) = o(g(n))$ if for any $\epsilon > 0$, $f(n)/g(n) < \epsilon$ for all n sufficiently large. That is, $f = o(g)$ if $\lim_{n\to\infty} f(n)/g(n) = 0$.

$f(n) = O(g(n))$ if there exists a constant $c > 0$ such that $f(n) \leq cg(n)$ for all n sufficiently large.

$f(n) \sim g(n)$ if for any $\epsilon > 0$, $(f(n) - g(n))/g(n) < \epsilon$ for all n sufficiently large. That is, $f(n) \sim g(n)$ if $\lim_{n\to\infty} f(n)/g(n) = 1$.

$f(n) = U(g(n))$ if $\log f(n) = O(g(n))$.

APPENDIX G

Notation Index

NOTATION	*REFERENCE*
A_n	Exp. 1.13.1
B	Prop. 1.2
B^n	Prop. 1.3
C	Def. 2.1
C_Ω	Def. 2.1
$C_\oplus$	Thm. 2.5.1
$C_{\{\vee\}}$	Def. 2.1
$C_{monosct}$	Def. 7.1
C_{sct}	Def. 7.1
Δf	Def. 6.1
∂f	Def. 5.1
∇f	Def. 5.2
$[I_F]$	Def. 7.2
$[J_F]$	Def. 5.15
$F^{(k)}$	Thm. 2.7.1
$\mathcal{F}ull_m$	Def. 3.9
l	Cor. 3.5
$\mathcal{M}_\oplus$	Def. 4.2.1
$\mathcal{M}'_\oplus$	Prop. 4.3
$O(\cdot)$	App. F
$o(\cdot)$	App. F
$\Omega(\cdot)$	App. F
$Q_{[a]}$	Thm. 7.10
T_F	Def. 5.3
τ_k^n	Thm. 7.10
G^T	Thm. 3.1
$\sim$	App. F
$\int_P \cdot dx$	Def. 5.18
$*$	Def. 6.4
$\bar{x}$	Prop. 1.1
x'	Lemma 7.8
f'	Def. 5.4
G'	Prop. 2.2.1
$x_i^{\alpha_i}$	Prop. 1.10
$\#_k\mathcal{P}$	Def. 7.2

APPENDIX H

BIBLIOGRAPHY

[ADK] V. Arlazarov, E. Dinic, M. Kronrod, and I. Faradzev, "On the Economical Construction of the Transitive Closure of a Directed Graph," Dokl. Acad. Nauk SSSR, **194**, 1970, pp. 487–488. Translated in Soviet Math. Dokl., **11**, No. 5, pp. 1209-1210.

[AHU] A. Aho and J. Hopcroft, J. Ullman,**The Design and Analysis of Computer Algorithms**, Addison-Wesley, Reading, MA, 1974.

[AKE] S. Akers, Jr., "On a Theory of Boolean Functions," SIAM Journal, **7**, 1959, pp. 487–498.

[BAS] W. Baur and V. Strassen, "The Complexity of Partial Derivatives," Theoretical Comp. Sci., **22**, 1983, pp. 317–330.

[BEG] A. Ben-Israel and T. Greville, **Generalized Inverses: Theory and Applications**, John Wiley and Sons, New York, 1974, pp. 7–18.

[BEM] A. Bertoni and G. Mauri, "On Efficient Computation of the Coefficients of Some Polynomials with Applications to Some Enumeration Problems," Information Processing Letters, **12**, No. 3, 13 June 1981, pp. 142–145.

[BER] E. Berlekamp, R. McEliece, H. Van Tilborg, "On the Inherent Intractibility of Certain Coding Problems," IEEE Trans. Information Theory, **IT–24**, No. 3, May, 1978, pp. 384–386.

[BHR] P. Bloniarz, H. Hunt, III and D. Rosenkrantz, "Algebraic Structures with Hard Equivalence and Minimization Problems," Journal Assoc. Comp. Machinery, **31**, No. 4, 1984, pp. 879–904.

[BLO] P. Bloniarz, "The Complexity of Monotone Boolean Functions and An Algorithm for Finding Shortest Paths in a Graph," Ph. D. Thesis, Massachusetts Institute of Technology. Republished as MIT/LCS/TR–28, Jan. 1979.

[BOM] A. Borodin and I. Munro, **The Computational Complexity of Algebraic and Numeric Problems**, American Elsevier, New York, 1975.

[CHA] G. Chaitin, "On the Length of Programs for Computing Finite Binary Sequences," Journal Assoc. for Computing Mach., **15**, No. 4, 1966, pp. 547–569.

[DAV] M. Davio, J. Deschamps, and A. Thayse, **Discrete and Switching Functions**, Georgi Pub. Co., St. Saphorin, Switzerland, 1978, pp. 395–412.

[FME] M. Fischer and A. Meyer, "Boolean Matrix Multiplication and Transitive Closure," 12th Annual IEEE Symposium on Switching and Automata Theory, pp. 129–131.

[FIS] M. Fischer, "The Complexity of Negation-Limited Networks," Report MAC TM 65, Massachusetts Institute of Technology, 1975.

[GAR] M. Garey and D. Johnson, **Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness**, W. H. Freeman and Co., San Francisco, 1978.

[HAL] P. Halmos, **Lectures on Boolean Algebras**, Springer-Verlag, New York, 1974.

[HAP] F. Harary and E. Palmer, **Graphical Enumeration**, Academic Press, New York, 1973, pp. 209–214.

[HOM] J. Hopcroft and J. Musinski, "Duality Applied to the Complexity of Matrix Multiplication and Other Bilinear Forms," SIAM J. Computing, **2**, No. 3, Sept 1973, pp. 159–173.

[HR1] M. Harrison, "The Number of Classes of Invertible Boolean Functions," J. Assoc. for Computing Mach., **10**, Jan 1963, pp. 25–28.

[HR2] M. Harrison, "On the Number of Classes of (n, k) Switching Networks," Journal of the Franklin Institute, **276**, Oct 1963, pp. 313–327 (with correction p. 588).

[HR3] M. Harrison, **Introduction to Switching and Automata Theory**, McGraw-Hill, New York, 1965, Chapters 4, 5, 6, 7.

[HUP] D. Hughes and F. Piper, **Projective Planes**, Springer-Verlag, New York, 1973, pp. 76–80.

[KAR] R. Karp, "On the Computational Complexity of Combinatorial Problems," Networks, **5**, 1975, pp. 45–68.

[KOL] A. Kolmogorov, "Three Approaches to the Qualitative Definition of Information," Problemy Peredachi Informatsii, **1**, No. 1, 1965, pp. 3–11.

[KNU] D. Knuth, **The Art of Computer Programming, Vol. 2, Semi-numerical Algorithms**, Addison-Wesley, Reading, Massachusetts, 1969, pp. 402–422 & Exercise 4.6.3.32.

[LOR] C. Lorens, "Invertible Boolean Functions," IEEE Trans. Elec. Comp, **EC-13**, No. 5, 1964, pp. 529–541.

[LSZ] A. Lempel, G. Serouss and J. Ziv, "On the Power of Straight-Line Computations in Finite Fields," IEEE Trans. Info. Theory, **IT–28**, No. 6, Nov. 82, pp. 875–880.

[LUP] O. Lupanov, "A Method of Circuit Synthesis," Izvestia V. U. Z. Radiofizika, **1**, No. 1, 1958, pp. 120–140.

[LU2] O. Lupanov, "On Diode and Diode Networks," Doklady Acad. Nauk SSSR, **111**, No. 6, 1956, pp. 1171–1174.

[LU3] O. Lupanov, "On the Synthesis of Contact Networks," Dokl. Acad. Nauk SSSR, **119**, 1958, pp. 23–26. Translated in Automat. Express, **1**, pp. 7–8.

[LU4] O. Lupanov, "On an Approach to the Synthesis of Logical Systems — The Principle of Local Coding," Problems of Cybernetics, **14**, 1965, pp. 31–110.

[MEL] K. Melhorn, "On the Complexity of Monotone Realizations of Matrix Multiplication," Technical Report A74-11, Fachbereich Angewandte Mathematik und Informatik, Universität des Saarlandes, Sep. 74.

[MOR] J. Morgenstern, "Linear Tangent Algorithms and Complexity," Comptes Rendues Acad. Sc. Paris, **277**, No. 9, Série A, 3 Sept 1973, 367–369.

[MO2] J. Morgenstern, "The Linear Complexity of Computation," J. Assoc. Comp. Mach., **22**, No. 2, April 1975, pp. 184–194.

[MUK] A. Mukhopadhyay, **Recent Developments in Switching Theory**, Academic Press, New York, 1971, pp. 57–83.

[MUL] D. Muller, "Application of Boolean Algebra to Switching Circuit Design and to Error Detection," Trans. IRE., **PGEC–3**, 1954, pp. 6–12.

[NEC] E. Nechiporuk, "Rectifier Networks," Doklady Akademii Nauk SSSR, **148**, No. 1, Jan. 1963, pp. 50–53. Translated in Soviet Physics – Doklady, 8, No. 1, Jul. 1963, pp. 5–7.

[NE2] E. Nechiporuk, "On the Topological Principles of Self-Correction," Problemy Kibernetiki, **21**, 1971. Translated in Systems Theory Research, **21**, 1971, pp. 1–99.

[NE3] E. Nechiporuk, "On a Boolean Matrix," Problemy Kibernetiki, **21**, 1971, pp. 237–240. Translated in Systems Theory Research, **21**, 1971, pp. 236–239.

[NIV] I. Niven, "Formal Power Series," American Mathematical Monthly, **76**, 1969, pp. 871–889.

[OLI] J. Olivos, "On Vectorial Addition Chains", Journal of Algorithms, **2**, 1981, pp. 13–21.

[ORL] V. Orlov, "Realization of Narrow Matrices by Means of Gate Networks," Problemy Kibernetiki, **22**, 1970, pp. 45–52. Translated in Systems Theory Research, Vol. **22**, 1970, pp. 42–50.

[OTT] R. Otter, "The Number of Trees," Ann. of Math., **49**, 1948, pp. 583–599.

[PAL] W. Paul, "A 2.5n Lower Bound for the Computational Complexity of Boolean Functions," Proc. of the 7th Annual ACM Symposium on the Theory of Computing, 1975, pp. 27–36.

[PAT] M. Paterson, "Complexity of Monotone Networks for Boolean Matrix Product," Theoretical Comp. Sci., **1**, pp. 13–20.

[PAU] W. Paul, "Realizing Boolean Functions on Disjoint Sets of Variables," Theoretical Comp. Sci., **2**, 1976, pp. 383–396.

[PAZ] C. Papadimitriou and S. Zachos, "Two Remarks on the Power of Counting," Report MIT/LCS/TM-228, Massachusetts Institute of Technology, 1982

[PFI] N. Pippenger and M. Fischer, "Relationships Among Complexity Measures," Report RC-6569, IBM T. J. Watson Research Center, Yorktown Heights, New York, 1977.

[PIP] N. Pippenger, "On Another Boolean Matrix," Theoretical Comp. Sci., **11**, 1980, pp. 49–56.

[PI2] N. Pippenger, "On the Evaluation of Powers and Related Problems," Proceedings 17th Annual IEEE Conf. on the Foundations of Computer Science, Oct. 1976, pp. 258–263.

[PI3] N. Pippenger, "The Complexity of Monotone Boolean Functions," Math. Systems Theory, **11**, 1978, pp. 289–316.

[PI4] N. Pippenger, "The Minimum Number of Edges in Graphs with Prescribed Paths," Math. Systems Theory, **12**, 1979, pp. 325–346.

[RAO] C. Rao, **Linear Statistical Inference and Its Applications,** Wiley & Sons, New York, 1965, pp. 24–28.

[RED] I. Reed, "A Class of Multiple-Error-Correcting Codes and the Decoding Scheme," Trans. IRE, **PGIT–4**, 1954, pp. 38–49.

[REI] J. Reif, "Efficient VSLI Fault Simulation," Report TR–03–85, Aiken Computation Lab., Harvard University, 1985.

[RUD] S. Rudenau, **Boolean Functions and Equations**, North-Holland Pub. Co., Amsterdam, 1974.

[SAN] N. Santoro, "Four $O(n^2)$ Multiplication Methods for Sparse and Dense Boolean Matrices," Report CS–80–25, 1980, University of Waterloo, Canada.

[SAV] J. Savage, **The Complexity of Computing**, Wiley & Sons, New York, 1976.

[SCH] C. Schnorr, "A $3n$-Lower Bound on the Network Complexity of Boolean Functions," Theoretical Comp. Sci., **10**, 1980, pp. 83–92.

[SHA] C. Shannon, "The Synthesis of Two-Terminal Switching Circuits," Bell System Technical Journal, **28**, 1949, pp. 59–98.

[SHO] L. Sholomov, "On the Realization of Incompletely-Defined Boolean Functions by Circuits of Functional Elements," Problemy Kibernetiki, Vol. **21**, 1969, pp. 215–226. Translated in Systems Theory Research, Vol. **21**, 1972, pp. 211–233.

[STA] V. Strassen, "Gaussian Elimination is Not Optimal," Numerische Mathematik, **13**, 1969, pp. 354–356.

[STO] L. Stockmeyer, "The Complexity of Decision Problems in Automata Theory and Logic," Ph. D. Thesis, Massachusetts Institute of Technology, July 1974. Republished as MIT/LCS/TR–133, July 1974.

[STR] M. Shaw and J. Traub, "On the Number of Multiplications for the Evaluation of a Polynomial and All Its Derivatives," Conference Record, 13th Annual Symposium on Switching & Automata Theory, 1972, pp. 105–107.

[THA] A. Thayse, "Boolean Differential Calculus," Philips Res. Rep., **26**, 1971, pp. 229–246.

[ULI] D. Ulig, "On the Synthesis of Self-Correcting Schemes from Functional Elements with a Small Number of Reliable Elements," Matematicheskie Zametki, **15**, 1974, pp. 937–944. Translated in Math. Notes Acad. Sci. USSR, **15**, 1974, pp. 558–562.

[VAV] L. Valiant and V. Vazirani, "$\mathcal{NP}$ Is As Easy As Detecting Unique Solutions," to appear 17th Annual Symposium on the Theory of Computing, May 85.

[VL1] L. Valiant, "Negation Can Be Exponentially Powerful," Theoretical Comp. Sci., **12**, 1980, pp. 303–314.

[VL2] L. Valiant, "The Relative Complexity of Checking and Evaluating," Information Processing Letters, **5**, No. 1, May 1976, pp. 20–23.

[VL3] L. Valiant, "The Complexity of Enumeration and Reliability Problems," SIAM J. Comp., **8**, No. 3, Aug 1979, pp. 410–421.

[VL4] L. Valiant, "The Complexity of Computing the Permanent," Theoretical Comp. Sci., 8, 1979, pp. 189–201.

[WEG] I. Wegner, "Switching Functions Whose Monotone Complexity is Nearly Quadratic," Theoretical Comp. Sci., **9**, 1979, pp. 83–97.

[WIL] C. Wilson, "Relativized Circuit Complexity," Proceedings 24th Annual IEEE Symposium on the Foundations of Computer Science, Nov. 1984, pp. 329–334.

[YAB] S. Yablonksi, "The Impossibility of Eliminating the Summing of All Functions of P_2 in the Solution of Some Problems of Circuit Theory," Dokl. Acad. Nauk SSSR, **124**, No. 1, 1959, pp. 44–47.

[YA2] S. Yablonksi, "The Algorithmic Difficulties of Synthesizing Minimal Switching Circuits," Problemy Kibernetiki, **2**. Translated in Problems of Cybernetics, **2**, 1961, pp. 401–457.