COMPLEXITY OF INFORMATION EXTRACTION

Thesis by

Yaser Said Abu-Mostafa

In Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

California Institute of Technology
Pasadena, California

1983

(Submitted May 23, 1983)

Acknowledgement

I should like to acknowledge Professor D. Psaltis, my dissertation advisor, for his substantial impact on the initiation, development, and refinement of my doctoral thesis. He had the insight, knowledge, and dedication to supervise and guide a diversified line of research that culminated in this work. His friendly and objective attitude made my study at Caltech enjoyable and fruitful.

Acknowledgement is also due the members of my doctoral committee, Professors A. Kechris, R. McEliece, C. Mead, E. Posner, H. Ryser, and R. Wilson, for their thorough study and direction of my thesis as well as their continuous encouragement. They have been very generous with their time, attention, and advice. I also like to thank Professors A. Martin and J. van Lint for their cooperation.

YSA
5.23.83

Abstract

This thesis describes a mathematical theory that interrelates the basic concepts of complexity, cost, information and reliability. The accessibility of information, as opposed to its availability, is characterized. Universal bounds for complexity distribution, implementation cost and decision reliability are estimated. These bounds give rise to a methodology for any consistent definition of a complexity measure. The basic notions of pattern recognition and information theory are directly related to computational complexity.

Contents

To my parents

who taught me the theorems I could not prove

CHAPTER I

Introduction

The accessibility of available information is a central issue in pattern recognition as well as in coding theory and computability theory. Several problems of different outlook can be reduced to a yes/no decision based on substantial amount of information and the bottleneck in making use of this information lies in the high complexity encountered in extracting the right "bit" from the raw data. This thesis provides a mathematical theory that interrelates the basic concepts of complexity, the practical implementation of decision-making systems, and the established notions of information.

< 1.1> Overview:-

The specific problem addressed here is that of characterizing the inherent complexity of binary decisions based on a number of binary observations and relating this complexity to different aspects of decision-making such as the reliability of the decision, the implementation cost, and the information-theoretic properties. The purpose is to introduce a quantitative measure of inherent complexity in a well-defined sense and relate it to the inevitable cost of implementation and to the structure of observations or data. The theory is developed from first principles in a mathematical way.

The word "complexity" is used in the literature in a diversity of contexts and meanings. In many reports on the complexity of Boolean functions, it is shown that almost all functions are "very complex" [2],[21]. Several approaches deal with the complexity in terms of implementation cost using standard gates [12]. Another methodology takes the size of programs as a basis for the definition [8]. Some reports provide an anatomy of the computation process [1]. The functional decomposition is also a model for defining complexity [13]. In pattern recognition [9] and coding theory [15], complexity is a central consideration that is usually tackled in an algorithmic way. In this thesis, we try to use these different concepts to build a uniform relationship between theoretical and practical aspects of complexity and information theory.

In chapter II, the model on which the complexity definition is based is defined and analyzed. This model is meant to be both ideal and realizable. The definition of complexity relies on abstract concepts such as reducibility and comparativeness applied to this model. This definition is given in chapter III together with elaboration, examples and justification. The concept of implementation cost is formalized and the complexity measure is related to cost in a direct way. Although the complexity definition is based on a specific model, a fundamental theorem about its universal significance is proved. This theorem also indicates the complexity distribution that must be met by any definition of complexity that is consistent with implementation considerations. It is shown that an important class of functions has low complexity, an essential fact for practical significance.

In chapter IV, the probabilistic considerations are fit into the complexity model in a way that relates the information-theoretic quantities to the complexity measure. This relationship enables us to describe certain phenomena in pattern recognition and coding theory in a specific way. The tradeoff between decision complexity and reliability is expressed in terms of the quantities involved. The application of the theory to actual engineering problems is surveyed and a conclusive discussion about the significance, scope and methodology is included.

< 1.2> Original Results:-

In this section, an account of the original ideas, definitions, methods, theorems, proofs, observations and conclusions is provided. The following is a summary of the original results:-

1. Defining a quantitative measure of complexity that applies to all Boolean functions of any number of variables in a way that is compatible with information theory.
2. Use of reducibility, comparativeness, and functional decomposition to formalize inherent complexity.
3. Defining implementation cost and computing facility and deriving the complexity distribution and the cost-complexity bounds.
4. Introduction of the probabilistic model and use of the source coding theorem and the entropy-complexity relation to describe the complexity-reliability tradeoff.
5. Relating the complexity to different aspects of coding theory and pattern recognition such as the code rate, channel noise and pattern normalization.

The idea, formulation and proof of the following lemmas and theorems are entirely original: lemma 1, lemma 3, lemma 4 (implementation limitation), lemma 5, theorem 1b, theorem 2b, theorem 4a,b (normal form cost), theorem 5a,b (complexity distribution), theorem 6a,b (universality), theorem 7b, theorem 8 (entropy-complexity), and theorem 9a,b (complexity-reliability). Lemma 2 was found independently by the author. All the examples given are original. The rest of the theorems and theorem parts are influenced in different aspects and to different degrees by similar methods in the literature.

< 1.3> Preliminaries:-

In this section, we introduce some notation, notions and lemmas that will be used to prove the main results in the following chapters. The lemmas of this section are formulated to suit this purpose and are by no means the strongest statements that can be made in their context. All logarithms (log) and exponentials (exp) are to the base 2.

The uncertainty (entropy) function [20] is defined by $H(x) = x \log \frac{1}{x} + (1-x)\log \frac{1}{1-x}$ for $0 < x < 1$ and $H(0) = H(1) = 0$. The following lemma estimates $H(x)$.

Lemma 1. For $0 < x < 1$, we have $x \log \frac{1}{x} < H(x) < x\,(2 + \log \frac{1}{x})$.

Proof. The LHS follows from the positivity of $(1-x)\log \frac{1}{1-x}$. For the RHS, we estimate this term. $(1-x)\log \frac{1}{1-x} = \frac{1-x}{\ln 2} \ln \frac{1}{1-x}$ where "ln" denotes the natural logarithm. Since $\ln 2$ is greater than 0.5, we have the over-estimate $2(1-x)\ln \frac{1}{1-x} = 2(1-x)(x + \frac{x^2}{2} + \frac{x^3}{3} + \cdots)$. This is less than $2(1-x)(x + x^2 + x^3 + \cdots) = 2(1-x)x\,(1 + x + x^2 + \cdots)$. The expansion reduces to $(1-x)^{-1}$ and therefore we get the final estimate $2x$ and the RHS follows. |

Stirling's formula [14] estimates $n!$ for $n > 0$ by $\alpha\sqrt{n}\left[\frac{n}{e}\right]^n$ where α is a number between 2 and e. The following lemma uses this formula to estimate $\binom{n}{r}$. Similar estimates are found in appendix A in [18].

Lemma 2. Let $\binom{n}{r} = \frac{n!}{r!(n-r)!}$.

a. If $0 < r < n$ then $\frac{1}{4\sqrt{r}} 2^{n H(r/n)} \le \binom{n}{r} \le 2^{n H(r/n)}$.

b. If $0 < t < n/2$ then $\frac{1}{4\sqrt{t}} 2^{n H(t/n)} \le \sum_{r=0}^{t} \binom{n}{r} \le t\, 2^{n H(t/n)}$.

Proof. We manipulate $\binom{n}{r}$ using Stirling's formula.

a. $\binom{n}{r} = \frac{n!}{r!(n-r)!} = \frac{\alpha_1}{\alpha_2\alpha_3}\left[\frac{n}{r(n-r)}\right]^{1/2}\left[\frac{(n/e)^n}{(r/e)^r\,(\,(n-r)/e\,)^{n-r}}\right]$ where $\alpha_1, \alpha_2, \alpha_3$ are between 2 and e. It is simple to show that $\frac{\alpha_1}{\alpha_2\alpha_3}$ is between $1/4$ and $1/\sqrt{2}$. Also, $\left[\frac{n}{r(n-r)}\right]^{1/2}$ is between $\frac{1}{\sqrt{r}}$ and $\sqrt{2}$. Therefore, their product is between $\frac{1}{4\sqrt{r}}$ and 1. We now evaluate $\left[\frac{(n/e)^n}{(r/e)^r\,(\,(n-r)/e\,)^{n-r}}\right]$. This can be rearranged as

$\left[(r/n)^r \left((n-r)/n \right)^{n-r}\right]^{-1}$. Taking the logarithm, we get $n\ H(r/n)$ and part (**a**) follows by exponentiation.

b. The LHS considers only the last term in the summation and applies the LHS in (**a**). The RHS over-estimates each term by the last term (since $0 < t < n/2$) and then applies the RHS in (**b**). This completes the proof. |

A *multiset* is a collection of objects where repetition is allowed and the order doesn't matter. The objects are called the *components* of the multiset and the number of times a certain object appears in the multiset is called its *multiplicity*. The multiset is denoted by listing its components enclosed in $< \ >$. If A and B are multisets, then the union $A \cup B$ is the multiset formed by all the objects of A and B, with the overall multiplicities. The following lemma estimates the number of different multisets of a special class.

Lemma 3. Let ρ_M be the number of different multisets $X_M = <n_1, \cdots, n_Q>$ where the n_i's are positive integers satisfying $\sum_{i=1}^{Q} 2^{n_i} = 2^M$ for the positive integer M. Then $\rho_M \le 2^{2^M}$.

Proof. We claim that any X_{M+1} can be written as the union of two X_M's except $X_{M+1} = < M+1 >$. To see this, we take the X_{M+1} and replace each two 1's in it by a single 2 (conceivably leaving a single 1 at the end). We next replace each two 2's by a single 3 and continue in this fashion until we replace the $M-1$'s by M's. This procedure must yield exactly $< M, M >$ because there is at most a single 1, a single 2, ..., a single $M-1$ left and these cannot contribute to $\sum_{i=1}^{Q} 2^{n_i}$ more than $2^M - 2$. Therefore, we go back and decompose the two M's getting two X_M's which proves the claim. Hence ρ_{M+1} is at most $1 + \frac{\rho_M(\rho_M+1)}{2}$. For $\rho_M \ge 2$, which holds for $M \ge 2$, this is at most ρ_M^2. The proof now follows by induction after over-estimating ρ_1 and ρ_2 by 2^{2^1} and 2^{2^2} respectively. |

In [11], an asymptotic estimate for ρ_M is given, but the result derived there could not be used to improve on main results.

< 1.4> Literature:-

The bibliography includes the books and technical papers that affected this work directly or indirectly. Among the several approaches for defining complexity, [2], [5], [22] had the most relevant notions. For devices and implementation, [1], [6], [18] were particularly useful. [7], [23] clarified general concepts. The author was affected by [14] and [19] in combinatorial methods, by [9] in the concepts of pattern recognition, by [10] and [15] in information-theoretic analysis, by [4], [12] and [17] in digital logic, and by [3] and [20] in approaching the problem.

CHAPTER II

Configurations

This chapter is devoted to the development of the basic notions and relations used in the definition and application of the complexity measure in the next chapter.

< 2.1> Boolean Functions:-

The purpose of this section is twofold. On the one hand, it sets up the notion of Boolean function in a formal way that excludes the redundancy encountered in the standard definition of a function. For example, if f is a Boolean function of one variable and g is a Boolean function of two variables such that their values are always 1 (constant functions), the two functions are formally different (according to the standard definition) because their domains are different, but they are the same "function" in our definition. On the other hand, our definition makes the distinction between functions in the sense of "operators" which take a point in the domain to a point in the range and functions which represent Boolean variables that are dependent on a set of independent Boolean variables. This framework is necessary for the development of the theory.

Let n be a positive integer. A *Boolean mapping on* n, denoted by $f_n(.)$, is a mapping from $\{0,1\}^n$ (the set of all binary n-tuples) to $\{0,1\}$. The mapping $f_n(.)$ is just an operator which expects an n-tuple of 0's and 1's as an argument and produces 0 or 1 according to some specific rule.

Let **U** be a universal set of indeterminates which can be thought of as independent Boolean variables assuming the values 0 or 1 only. The cardinality of **U** is potentially infinite. We refer to any specific assignment of 0's and 1's to all Boolean variables in **U** as the *state* of the system. Let $S = \{s_1, \cdots, s_N\}$ be any non-empty finite subset of **U** for some positive integer N. A *Boolean mapping on* S (which is a set, not an integer), denoted by f_S (without further arguments), is a mapping from $\{0,1\}^S$ (the set of all binary N-tuples indexed by the elements of S) to $\{0,1\}$. The Boolean mapping f_S defines a dependent Boolean variable whose value is determined by the values of the independent variables which are the elements of S. In contrast to the Boolean mapping $f_n(.)$ on an integer n, the value of f_S is determined by the state of the system.

For a fixed S of cardinality N, there are 2^N possible assignments of 0's and 1's to the Boolean variables in S, and hence there are 2^{2^N} different Boolean mappings f_S. The set of all Boolean mappings f_S on a set S for all choices of S (finite non-empty subsets of **U**) is denoted by $\mathbf{M_U}$. The cardinality of $\mathbf{M_U}$ is potentially infinite. Some elements of $\mathbf{M_U}$ are equivalent in the sense that for all possible assignments of 0's and 1's to their independent Boolean variables, they always assume the same value. This will happen when some f_S is actually independent of some of the Boolean variables in S. We want to merge (identify) these mappings into one entity.

Definition. The relation $\equiv$ is defined on $\mathbf{M_U}$ as follows. $g_{S_1} \equiv h_{S_2}$ if, and only if, for all states of the system, the values of g_{S_1} and h_{S_2} are the same.

It is clear that $\equiv$ is an equivalence relation (reflexive, symmetric and transitive). Therefore, it induces a partition of $\mathbf{M_U}$ into equivalence classes. Each equivalence class is a set of all Boolean mappings like g_{S_1} and h_{S_2} that are mutually equivalent (related by $\equiv$). We now identify each equivalence class as an object and introduce the following definition.

Definition. A *Boolean function* f is an equivalence class of the relation $\equiv$ on $\mathbf{M}_{\mathbf{U}}$. The set of all Boolean functions is denoted by $\mathbf{F}$.

Notice that any Boolean function f "depends" on a finite number of Boolean variables in $\mathbf{U}$. This is because of the restriction in the definition of Boolean mappings on a set S that S must have finite cardinality. The smallest set of variables on which a Boolean function depends is of special interest.

Definition. The *support* of a Boolean function f, denoted by $T(f)$, is the intersection of all sets S for which some Boolean mapping g_S belongs to (the equivalence class) f. The *rank* of f, denoted by $r(f)$, is the cardinality of its support, $r(f) = |T(f)|$.

We shall adopt the usual abuse of notation in the context of equivalence classes and treat f as an actual function rather than a set of equivalent Boolean mappings whenever no confusion as to what is meant can arise. We shall also refer to the value of f simply by f. We start by saying that only the constant functions $f=0$ and $f=1$ have empty support $T(f) = \Phi$ (zero rank, $r(f) = 0$). If S is a subset of $\mathbf{U}$ with cardinality N, the number of Boolean functions whose supports are *subsets* of S is 2^{2^N} whereas the number of Boolean functions of support S is, by the principle of inclusion and exclusion, $\sum_{r=0}^{N} \binom{N}{r} (-1)^{N-r}\ 2^{2^r}$. On the other hand, the number of Boolean functions which depend on N variables, i.e., whose rank is N, is potentially infinite.

< 2.2> Normal Form:-

In this section, we introduce the fundamental notions on the way to defining complexity. We start by discussing what is involved in proposing a valid and useful measure of complexity. From a practical point of view, a function whose complexity measure is large must require a *costly* implementation. However, from a theoretical point of view, a measure of inherent complexity should be essentially independent of any *specific* implementation devices that may be available. A significant definition of complexity has to capture both aspects.

Suppose that we have n-input "universal gates," e.g., programmable devices (section 2.12 in [17]) such as a programmable read-only memory (PROM) with n address lines and 1 data line, or a finite Turing machine, which can simulate any Boolean mapping with n arguments. All Boolean functions which depend on n variables (i.e., whose rank is n) can be simulated by this universal gate. However, some of these functions depend on their n variables in a simple way and do not really require a *universal* gate to simulate them. They can be implemented using less powerful devices of n inputs or several smaller universal gates interconnected together. The implementation of a universal gate with n input lines requires the order of 2^n standard switching devices (e.g., NAND gates) or the same order of memory locations. Therefore, a decrease in the number of inputs per universal gate at the cost of increasing the number of required gates (by less than an exponential order) is a valid implementation strategy. Using finite automata instead of combinational logic to simulate a universal gate introduces a tradeoff between time, space, size, and depth [6].

We shall consider a specific standard way of interconnecting the universal gates. It is conceivable that restricting the interconnection scheme may deprive the complexity measure of its universal significance. Informally, we may decide that some function is very complex only because the enforced standard way of interconnecting the gates is so poorly suited to the function that we require many more gates than we might have needed if we were free to choose the interconnection scheme of these gates. The universality theorem which will be proved in the next chapter shows that this dilemma almost never happens. Other standard forms can also be used to build the theory. Also, the consideration of universal gates (ideal maximum cost) and interconnections (ideal no cost) is essential to the generality of this approach. For example, if we consider special-purpose devices such as linear sequential circuits or non-universal devices such as field-programmable logic arrays (FPLA's) as a basis of defining complexity, the definition will not be general as such.

The standard way we adopt here for interconnecting the smaller universal gates is the *normal form*. There will be a first stage of universal gates which take their inputs directly from the function arguments (input Boolean variables). The outputs of these gates are then input to a final (second stage) universal gate (figure 1). A normal form can be thought of as an interconnection of devices analogous to the sum-of-products or the product-of-sums where the AND's and OR's are now replaced by universal gates. The normal form resembles a standard system of pattern recognition where the classification decision is based on a number of features that are extracted from the inputs. This suggests the following terminology.

Nomenclature. In the normal form, the universal gates of the first stage are called the primary gates or *feature extractors* and the Boolean mappings they simulate are called primary functions or *features*, denoted by F's. The universal gate of the final stage is called the secondary gate or *classifier* and the Boolean mapping it simulates is called the secondary function or *classification decision*, denoted by D.

Hence, the normal form is a simple decomposition of the function in question into a classification decision based on extracted features. Normal form configurations are a variation of support systems which were introduced in the context of local and global computation [1]. We now give the definition that formalizes the normal form structure.

Definition. A *normal form input configuration* (or simply a *configuration*) C is a finite multiset of finite non-empty subsets of **U**, $C = < S_1, \cdots ,S_L >$. The S_i's are called the *components* of the configuration C. The *support* of C, denoted by $T(C)$, is defined as $\bigcup_{i=1}^{L} S_i$ (all Boolean variables which appear in any component of the configuration). The cardinality of the support is called the *rank* of C and denoted by $r(C)$ (dimensionality of the space of Boolean variables in the configuration). The *length* of C, denoted by $l(C) = L$, is the number of the not-necessarily-distinct components of the multiset. The *degree* of C, denoted by $d(C)$, is the maximum cardinality $n_i = | S_i |$ of a component S_i in C (zero for the empty configuration).

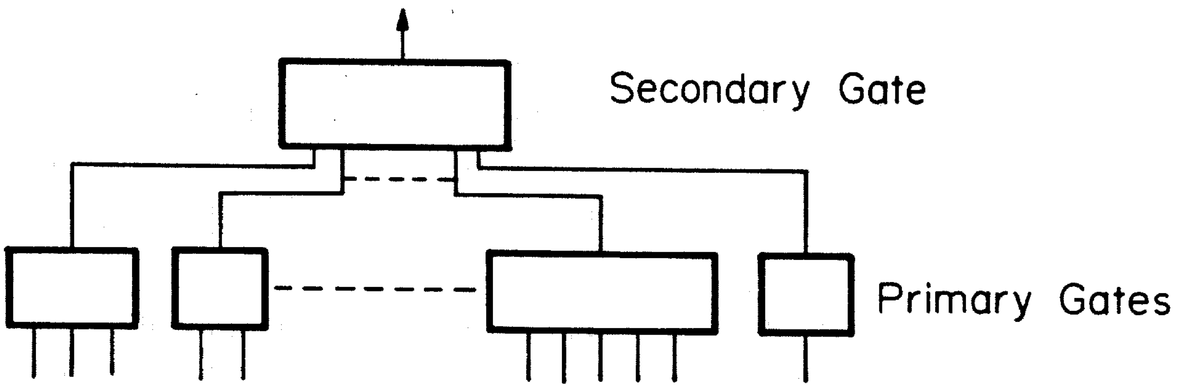


Figure 1: Normal Form Structure

Notice that the configuration (the multiset) can be empty, but if it has components none of these components can be empty. Also, nothing infinite is allowed in a configuration. For each $i = 1, \cdots, L$, the n_i Boolean variables in S_i will be the inputs to one of L primary gates. The outputs of these L gates, i.e., the features, are then input to the secondary gate whose output (the classification decision) becomes the overall implemented function.

Definition. A non-empty configuration $C = < S_1, \cdots, S_L >$ is said to *admit* a Boolean function f if there exist Boolean mappings $F^1_{S_1}, \cdots, F^L_{S_L}$ on the subsets $S_1, \cdots, S_L$ of **U** (the superscript distinguishes between the F's) and a Boolean mapping $D_L(.)$ on the integer L such that, for all states of the system, $f = D_L\left[F^1_{S_1}, \cdots, F^L_{S_L}\right]$. The empty configuration admits the constant functions only.

A function admitted by a configuration is one that can be implemented using a normal form with the inputs specified by the configuration. Since the configuration is defined as a multiset, configurations will be *equal* if, and only if, they have the same components (with the same multiplicities). However, some unequal configurations are functionally the same.

Definition. The set of all Boolean functions f admitted by a configuration C is denoted by $\mathbf{F}(C)$. Two configrations C_1 and C_2 are said to be *equivalent* if they admit the same functions, i.e., if $\mathbf{F}(C_1) = \mathbf{F}(C_2)$. The number of functions admitted by C is denoted by $N(C)$ $(= |\mathbf{F}(C)|)$.

We observe immediately that for a function f that is admitted by a configuration C, $T(f)$, the support of f, must be a subset of $T(C)$, the support of C. The degree of C cannot be bigger than its rank, i.e., $d(C) \leq r(C)$. The set $\mathbf{F}(C)$ contains all Boolean functions that can be implemented in a normal form using the input configuration C. The number of functions $N(C)$ admitted to a configuration C expresses the power of the configuration or its versatility. Notice that each function is counted as "1" regardless of its "complexity." Since the gates of the normal form in question are universal, the configuration which admits an inherently complex function will be powerful enough to admit a large number of simpler functions and $N(C)$ will be indeed large. This would not hold if the building blocks were special-purpose devices. The next two sections develop the relations between the different parameters of the configuration as well as its structural properties.

< 2.3> Parameter relations:-

The parameters of a configuration C are interrelated. The following theorem describes several bounds on $N(C)$, the number of functions admitted by C, in terms of the length, degree, and rank of C. These bounds will prove vital in estimating the complexity of Boolean functions in the next chapter and will provide insight into the nature of configurations.

Theorem (1)

Let $C = < S_1, \cdots, S_L>$ be a configuration which admits $N(C)$ Boolean functions. Let $r(C)$, $l(C)$ $(= L)$, $d(C)$ be the rank, length and degree of C respectively. Then

a. $d(C) \leq \log\log N(C) \leq \max(l(C), d(C)) + \log\left[\max(l(C), d(C)) + 1\right]$.

b. $\sqrt{r(C)} \leq \log\log N(C) \leq r(C)$.

Proof. We first maintain that $\log\log N(C)$ exists (and is non-negative) by observing that $N(C) \geq 2$ since all configurations admit the two constant functions (including the empty configuration by definition). All statements are trivial for the empty configuration and we now assume that C is non-empty.

a. Let S_{i^*} be a component of maximal cardinality in the configuration. By definition, $|S_{i^*}| = d(C)$. Now C can implement at least the $2^{2^{d(C)}}$ different functions whose support is a subset of S_{i^*}. The LHS inequality in (**a**) follows by taking the logarithm twice. To prove the RHS inequality, we observe that the number of different mappings that can be implemented by the primary gate on S_i is $2^{2^{|S_i|}}$ and the number of mappings that can be implemented by the secondary gate in terms of its inputs is $2^{2^{l(C)}}$. Therefore, $N(C)$ is at most $\exp\left[\sum_{i=1}^{l(C)} 2^{|S_i|} + 2^{l(C)}\right]$. We increase this number by substituting for each $|S_i|$ and for $l(C)$ by $M = \max(l(C), d(C))$. This gives $N(C) \leq 2^{(M+1)2^M}$ which yields the RHS inequality in (**a**).

b. The support $T(C)$ contains the support $T(f)$ of any function f admitted by C. Since $T(C)$ has $r(C)$ Boolean variables, $N(C)$ is at most $2^{2^{r(C)}}$. Taking the logarithm twice yields the RHS inequality in (**b**). Now let $C^* = < S_1^*, \cdots, S_{L^*}^* >$ be the configuration defined by $S_1^* = S_1$ and $S_i^* = S_i \setminus \bigcup_{j=1}^{i-1} S_j$ (if non-empty, otherwise skip and relabel) for $i = 2, \cdots, L^*$ ($L^* \leq L$). We shall prove the LHS inequality in (**b**) for C^*. If $d(C^*) \geq \sqrt{r(C^*)}$, we are done (LHS inequality in (**a**)), so we assume that $d(C^*) < \sqrt{r(C^*)}$. But $l(C^*)d(C^*) \geq r(C^*)$ (since the $r(C^*)$ variables are contained in the $l(C^*)$ components and each component has at most $d(C^*)$ variables). Therefore, $l(C^*) > \sqrt{r(C^*)}$. Since the S_i^*s are disjoint and non-empty, C^* can implement at least $\exp 2^{\sqrt{r(C^*)}}$ functions (those implemented by the secondary gate when each primary gate "passes" a distinct variable). Taking the logarithm of $N(C^*)$ twice, we get the LHS inequality in (**b**) for C^*. By construction, each variable in the S_i's is contained in some S_i^* and vice versa, hence $r(C^*) = r(C)$. Also, C^* is a "reduced" form of C which implies that C admits all the functions admitted by C^*. Therefore, $N(C) \geq N(C^*)$ and the LHS inequality in (**b**) follows. This completes the proof. |

Notice that max $(l(C), d(C))$ has the interpretation of being the maximum number of inputs in any gate of the configuration. The observation here is that 2^{2^n} is a tremendously increasing function of n which makes $N(C)$ practically depend only on the size of the largest gate in the configuration and nothing else. Also, if a configuration C has $r(C)$ variables, then no matter how these variables are distributed on different components, the size of one of the gates (possibly the secondary one) must be at least $\sqrt{r(C)}$.

< 2.4> Structure:-

Although the configuration is just a multiset of subsets, it has the functional interpretation of implementing Boolean functions on normal forms. This fact led to the equivalence of configurations which implement the same functions, i.e., which have the same $\mathbf{F}(C)$. This equivalence relation makes the configuration a combinatorial object distinct, for example, from hypergraphs. In this section, we describe the structure of configurations and derive useful results for the complexity analysis in the next chapter.

Definition. A configuration $C = < S_1, \cdots ,S_L >$ is said to be ***redundant*** if one of the S_i's can be omitted without diminishing $\mathbf{F}(C)$.

This definition means that a redundant configuration is one which has an unnecessary gate among its primary gates. For example, it is easy to show that $\{s_1,s_2\}$ can be omitted from $C = < \{s_1,s_2\},\{s_1,s_2,s_3\} >$ without diminishing $\mathbf{F}(C)$. The following theorem deals with redundant and non-redundant configurations.

Theorem (2)

a. If the configuration $C = < S_1, \cdots ,S_L >$ is not redundant, then for all subsets λ of $\{1, \cdots ,L\}$:

(1) $$|\bigcup_{i \in \lambda} S_i| \geq |\lambda|.$$

b. Any configuration $C = < S_1, \cdots ,S_L >$ is equivalent to a configuration $C^* = C_1^* \cup C_2^*$ (figure 2) where $C_1^* = < S_1^*, \cdots ,S_l^* >$ and $C_2^* = < S_{l+1}^*, \cdots ,S_{L^*}^* >$ such that $S_i^* = \{s_i\}$ for $i = 1, \cdots ,l$ with distinct s_i's, the partition $C_1^* \cup C_2^*$ is unique, and for any subset λ of $\{1, \cdots ,L^*\}$ where $\lambda \cap \{l+1, \cdots ,L^*\} \neq \Phi$:

(2) $$|\bigcup_{i \in \lambda} S_i^*| > |\lambda|.$$

Proof. We shall use the Philip Hall Theorem [19] about the existence of a system of distinct representatives (SDR), which is a collection of distinct elements such that each element belongs to (represents) a specific subset within a given collection of subsets.

a. Suppose that condition (1) does not hold. Then there is a subset λ of $\{1, \cdots ,L\}$ for which $|\bigcup_{i \in \lambda} S_i| < |\lambda|$. Let λ be a minimal subset satisfying this condition. Let j be any element of λ (λ is non-empty since $|\lambda| > 0$). Let $\lambda_j = \lambda \setminus \{j\}$. Since λ is minimal, we have $|\bigcup_{i \in \lambda_j} S_i| \geq |\lambda_j|$. But $|\lambda_j| = |\lambda| - 1$ and $|\lambda| > |\bigcup_{i \in \lambda} S_i| \geq |\bigcup_{i \in \lambda_j} S_i|$. This forces both $\bigcup_{i \in \lambda} S_i$ and $\bigcup_{i \in \lambda_j} S_i$ to have cardinality

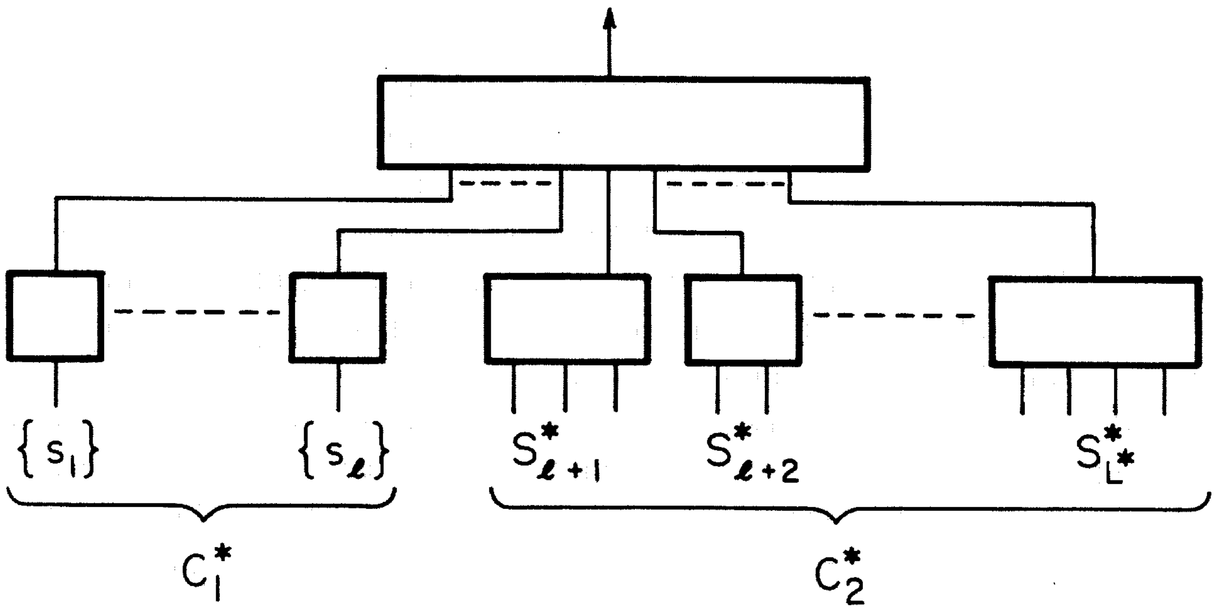


Figure 2: Compact Configuration

$|\lambda| - 1$ (= $|\lambda_j|$) and forces S_j to be contained in $\bigcup_{i\epsilon\lambda_j} S_i$. By minimality of λ again, the sets S_i with i in λ_j satisfy the Hall condition for an SDR. This SDR has $|\lambda_j|$ elements and so does $\bigcup_{i\epsilon\lambda_j} S_i$. Hence the SDR covers all the elements of the S_i's with i in λ_j. Now S_j is a subset of $\bigcup_{i\epsilon\lambda_j} S_i$ and we can omit it without diminishing $\mathbf{F}(C)$ since the SDR can be "passed" by the other primary gates to the secondary gate which can then implement any function of the variables in $\bigcup_{i\epsilon\lambda} S_i$. Hence C is redundant and part (**a**) is proved.

b. We use the argument of part (**a**) inductively and omit the redundant S_j's until condition (1) is established. Now, if any collection of S_i's satisfies condition (1) with equality and are not all one-element sets, we use a similar argument to replace them by the collection of (the same number of) one-element sets which is the SDR (the SDR exists since condition (1) holds for all subcollections) without diminishing (or increasing) $\mathbf{F}(C)$. We then re-establish condition (1). Repeating these two steps for a finite multiset of finite subsets cannot continue forever since each step decreases either the number of components in C or the number of elements in a component and does not increase the other. We show that the resulting configuration C^* satisfies the required conditions. Let C_1^* be the multiset of all one-element components of C^* and let C_2^* be the multiset of all multi-element components of C^*. No two components of C_1^* can be equal since this would violate condition (1) (taking these two components alone). Condition (2) holds since condition (1) holds, each component of C_2^* has more than one element, and our procedure leaves no collection of S_i^*s satifying condition (1) with equality except the one-element components. Uniqueness of the partition follows because if we included any one-element component in C_2^*, this would violate condition (2) (taking this component alone). This completes the proof. |

We can call the reduced form in part (**b**) of theorem 2 a *compact* configuration. It should be noted that even the compact configuration can be redundant, as in $C = < \{s_1\} . \{s_1, s_2, s_3\} >$. We now make use of theorem 2 in proving that a configuration does not have to be very long and conclude that the set of possible values for $\mathbf{F}(C)$, where the support of C is any subset of a given N-set S, has drastically fewer elements than the obvious upper bound of

$2^{2^{2N}}$.

Theorem (3)

a. If C is a configuration with $l(C) > r(C)$, then there is a configuration C^* with $l(C^*) \leq r(C^*)$ which is equivalent to C.
b. Let S be an N-subset of the universal set **U**. There are at most 2^{N^2} possible values for $\mathbf{F}(C)$ for all configurations C whose support $T(C)$ is a subset of S.

Proof. a. By part (**a**) of theorem 2, C must be redundant since $l(C) > r(C)$ means that condition (1) does not hold for C with $\lambda = \{1, \cdots, l(C)\}$. We keep omitting the unnecessary S_j's until we get $l(C^*) \leq r(C^*)$.
b. From part (**a**), we need to consider only those configurations with $l(C) \leq N$. Since $d(C) \leq N$, we can now enumerate the number of different C's. Each component can be assigned 2^N different subsets of S and there are at most N such components. Therefore, the total number of different configurations is at most $\prod_{i=1}^{N} 2^N = 2^{N^2}$. Since the different $\mathbf{F}(C)$'s can be at most that many, the proof follows. |

Theorem 3 shows that we only need to consider a finite, and relatively small, set of configurations for any finite support. Since all N-sets are isomorphic, we conclude that the number of different values for $N(C)$ for all configurations C of rank N or less is at most 2^{N^2}, which is far less (for large N) than the conceivably possible $2^{2^N} + 1$ values. This says that the function $N(C)$ is, in some sense, *quantum*. We are now in a position to define complexity in terms of normal form input configurations.

CHAPTER III

Complexity

In this chapter, a measure of complexity is introduced and discussed. A fundamental result about the universal significance of this measure is proved. Finally, low-entropy functions are defined and their complexity is estimated.

< 3.1> Basic Definition:-

Suppose we have a number of objects which possess a certain property to different degrees. We are required to give a quantitative measure of how much the object X possesses this property. The most obvious way to do so is to introduce some ordering of these objects according to the degree they possess the property, then define the measure for X to be the number of objects which possess the property to a lesser degree than X itself. We call this approach a *comparative* approach.

The choice of such an approach does not readily produce a quantitative measure of complexity for Boolean functions. The property of complexity has yet to be defined in order to make the ordering of Boolean functions possible. The notion of ***reducibility*** [23] is a natural way of comparing the complexity of two procedures. If some procedure A can be carried out by transforming it in a simple way to another procedure B and then carrying out the procedure

B instead, A cannot be more complex than B. In our case, we used configurations as a basis for reducibility, which resembles some established ideas of reducibility like *projection* [22]. A similar approach is introduced in [2]. The point is that if we consider the minimal configuration C that admits a Boolean function f, then the other functions admitted by C are at most as complex as f since they are reducible to f by reprogramming the universal gates in C, the configuration that *just* implements f.

Definition. The *comparative normal form complexity* (or simply the *complexity*) of a Boolean function f, denoted by $K(f)$, is defined by:-

$$K(f) = \log\log\min\{\, N(C) \mid C \text{ admits } f \,\}$$

The units of $K(f)$ are *bits* (binary digits).

We first dispose of the "log log" as being a scale down for $N(C)$ which is typically of the form 2^{2^N}. The definition says that we consider all configurations C that admit the function f, choose the minimal configuration of these with respect to the number of functions it admits, and take this number as a measure for the complexity of f. Since $N(C) \geq 2$ for all configurations, taking the logarithm twice is valid and $K(f) \geq 0$ (with equality if, and only if, f is a constant function). Notice that the normal form served as a "catalyst" in the definition of complexity. Conceivably, another definition along the same lines but based on a general setup (instead of normal form input configurations) would be more general. The following example shows that this generalization cannot be pursued without limit.

Example. Consider a device D to be formally defined as any fixed subset of the set $\mathbf{F}$ of all Boolean functions (the subset being those functions admitted by the device). Suppose we define the complexity $\hat{K}(f)$ of the Boolean function f to be:-

$$\hat{K}(f) = \min\{\, N(D) \mid D \text{ admits } f \,\}$$

Then *every* function has complexity 1 by taking the (special-purpose) device that simulates the function and nothing else.

This example is in some sense similar to Berry's Paradox [7], and essentially says that the measure of complexity cannot take care of every individual function. The objective is to propose a definition that sets a consistent hierarchy of inherent complexity for "most" functions. Here is an example of an inherently simple function.

Example. Let $f = s_1 \oplus s_2 \oplus \cdots \oplus s_N$ where "$\oplus$" denotes the modulo-two sum. It is easy to show that f is of complexity $o(N)$ by constructing a configuration that admits it which has all of its gates with approximately $\sqrt{N}$ inputs where each gate simulates the modulo-two sum. Notice that this simple function requires a maximal sum-of-products or product-of-sums implementation since its Karnaugh map [12] looks like a chessboard and the prime implicants have to be taken as the minterms or maxterms.

Other simple functions like symmetric or linear functions can also be shown to have low complexity in a similar way. This is not immediately obvious since the normal form which implements these functions in our framework has only universal gates and is not particularly oriented towards linear functions for example. Another important class of functions of moderate complexity is the class of low-entropy functions which will be discussed in section < **3.4**> .

In order to have practical significance for the definition of complexity, $K(f)$ should be related to the cost of implementing f. We start by defining the cost.

Definition. The *cost*, denoted by c, of a universal gate of n inputs is defined to be 2^n. The units of the cost are *cells*. The cost of an interconnection is zero cells. The cost of a collection of gates and interconnections is defined as the sum of the costs of the components.

This definition is motivated by the actual number of cells or switching points on an integrated circuit that simulates a universal gate. Notice that, in practice, an interconnection has a non-zero cost. However, this fact can only strengthen the main results to be proved shortly. It is now possible to estimate that cost of normal forms and one might consider defining the complexity of a Boolean function directly in terms of the minimum cost of a normal form that implements the function. Although such a definition is not a priori justified, it turns out to be almost identical, ***numerically***, to $2^{K(f)}$. Inherent complexity is to implementation cost what mass is to weight, an intrinsic property which is different from, but directly related to, a practical impact.

Theorem (4)

Let f be a Boolean function of complexity $K(f) = K$. Define $o_K = \log(1+K)$. Then

a. Any normal form implementation of f costs at least 2^K cells.

b. There is a normal form implementation of f which costs at most 2^{K+o_K} cells.

Proof. The proof relies on theorems (1) and (2).

a. Let $C = <S_1, \cdots, S_L>$ be a configuration that admits f. From the definition of $K(f)$, $2^{2^K} \leq N(C)$. Let $n_1, \cdots, n_L$ be the cardinalities of $S_1, \cdots, S_L$ respectively (number of inputs to each primary gate). Now $N(C) \leq \exp\left[2^L + \sum_{i=1}^{L} 2^{n_i}\right]$ (same argument as in theorem (1)). Taking the logarithm yields part (**a**).

b. Let C be a minimal configuration (in the sense of $N(C)$) that admits f. If C is redundant, omit unnecessary S_i's until condition (1) of theorem (2) is satisfied. We show that $\max(l(C), d(C)) \leq K$. Suppose not: if $d(C)$ is greater than K, then C admits more than 2^{2^K} functions using the primary gate with $d(C)$ inputs; else, if $l(C)$ is greater than K, then by passing an SDR from the primary gates to the secondary gate, C also admits more than 2^{2^K} functions, a contradiction. Hence the cost of C is at most $(K+1)\, 2^K$ cells and (**b**) follows by re-arrangement. |

This result does not restrict f or $K(f)$, but it restricts the implementation to the normal form. A much stronger limit result will be proved shortly, but we first turn to some estimates for the complexity distribution which will be used in the proof.

< 3.2> Distribution:-

One of the "health" properties of any complexity measure is that it should resolve different levels of complexity. We now show that $K(f)$ meets this requirement by estimating the range of $K(f)$ for different functions and the number of functions with different $K(f)$'s.

Theorem (5)

Let $\mathbf{F}_S$ be the set of all Boolean functions f whose support is a subset of a non-empty N-set S. Define N_K to be $|\{f \varepsilon \mathbf{F}_S \mid K(f) \leq K\}|$. We have :-

a. If f depends on M variables then $\sqrt{M} \leq K(f) \leq M$.

b. If $0 \leq K \leq N$ then $K-1 \leq \log\log N_K \leq K + 2\log N$.

Proof. We shall use theorems (1) and (3).

a. Since f depends on M variables, any configuration C admitting f must have rank $r(C) \geq M$. The LHS inequality in (**a**) follows from part (**b**) of theorem (1). Now, $C = < T(f) >$ admits f and has $r(C) = M$. The RHS inequality in (**a**) follows from the minimality in the definition of $K(f)$ and part (**b**) of theorem (1).

b. All functions f depending on at most $\lfloor K \rfloor$ variables have $K(f) \leq K$ by part (**a**). There are at least $2^{2^{K-1}}$ such functions in $\mathbf{F}_S$ and the LHS inequality in (**b**) follows by taking the logarithm twice. From theorem (3), there are at most 2^{N^2} different $\mathbf{F}(C)$'s with $T(C)$ a subset of S. Each function f of complexity $K(f) \leq K$ must belong to an $\mathbf{F}(C)$ whose cardinality is at most 2^{2^K}. Therefore, $N_K \leq 2^{N^2} 2^{2^K}$ and the RHS inequality in (**b**) follows by taking the logarithm twice. This completes the proof. |

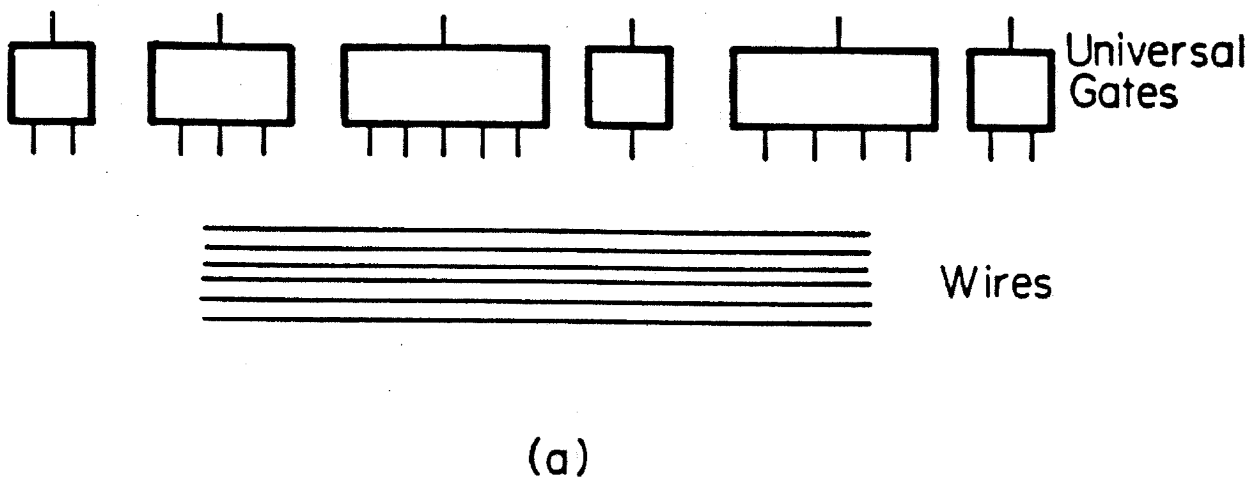
Theorem 5 provides another significance for the quantity $K(f)$. Apart from being a measure of the comparative complexity of f (from the definition) and a measure of the cost of any normal form implementation of f (from theorem 4), it is also a measure of how many functions, depending on variables within a finite set S, have complexity at most $K(f)$.

< 3.3> Universality:-

Although restricting the implementation devices to universal gates was essential to the independence of the theory, restricting the interconnection scheme of these universal gates to the normal form is a simplification that may affect the universal significance. In this section, we prove that such a systematic way is not far from the most general schemes of interconnecting the gates. Suppose that we have a "computing facility", i.e., a number of universal gates of different sizes and several "wires" for interconnection (figure 3a). It is conceivable that with the freedom to choose the interconnection scheme of these gates, one can do much better than in theorem 4. This would mean that $K(f)$ is not really a characteristic measure of cost, but merely a description of the behavior of functions when implemented on normal forms. We want to prove that this is not the case.

Definition. A *computing facility* P is a finite multiset of positive integers, $P = <n_1, \cdots, n_Q>$.

These integers are meant to be the number of inputs to each of the Q available universal gates. It follows that the cost of this facility, $c(P)$, is given by $\sum_{i=1}^{Q} 2^{n_i}$ cells. Given N Boolean variables $x_1, \cdots, x_N$ and calling the outputs of the Q gates $y_1, \cdots, y_Q$, we have (at most) $N+Q$ different variables that can be input to each gate. Therefore, for each computing facility P we have at most $\prod_{i=1}^{Q} \binom{N+Q}{n_i}$ possible interconnection schemes or "circuits" (entering the same variable twice to the same gate or interchanging the inputs within a gate cannot increase the implemented functions because the gates are universal). For each circuit (figure 3b), there are at most $\prod_{i=1}^{Q} 2^{2^{n_i}}$ ways to program the universal gates and, for each of these, we have Q implemented functions out of the Q gates. Therefore, the number of functions that can be implemented on a circuit is at most $Q \prod_{i=1}^{Q} 2^{2^{n_i}}$. In the following lemma, we estimate the number of functions whose support is a subset of given set S and are implementable on some computing facility given the fixed cost of any such



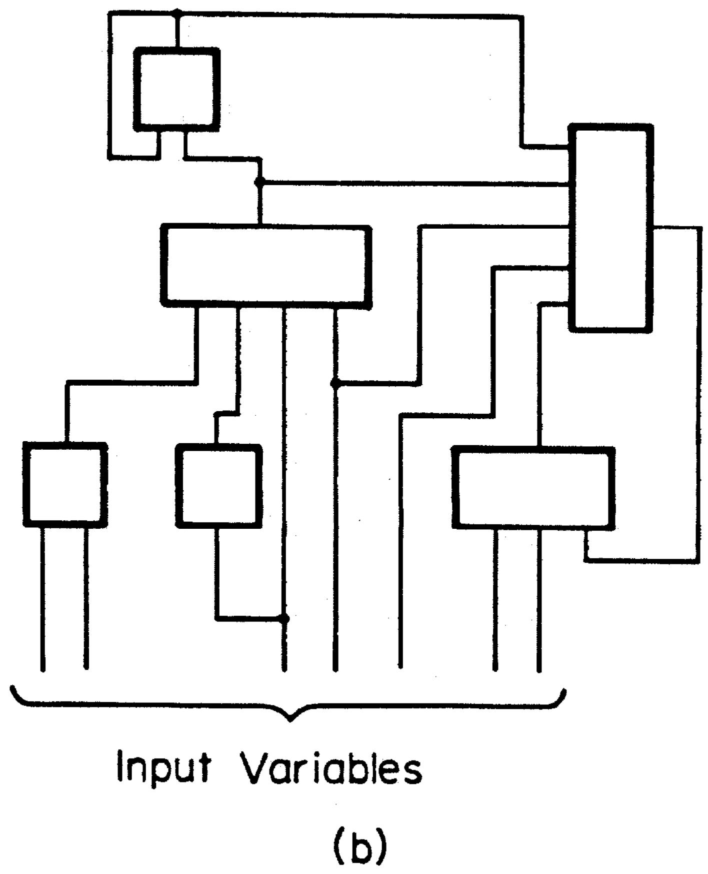


Figure 3: (a) Computing Facility (b) Circuit

facility. The lemma will practically yield the proof of the main theorem.

Lemma 4. Let S be a non-empty N-subset of **U**. The number of functions f, with $T(f)$ a subset of S, that can be implemented on some computing facility that costs 2^M cells for some positive integer M is at most $2^{2^{M+\delta}}$ where $\delta = \log(4 + \frac{1}{2}\log(2^{M-1} + N))$.

Proof. We will estimate the number of different computing facilities for a given cost, the number of circuits that can be formed using a given computing facility, and the number of Boolean functions that can be implemented on a given circuit.

1. The number of computing facilities $P = <n_1, \cdots ,n_Q>$ satisfying $\sum_{i=1}^{Q} 2^{n_i} = 2^M$ (i.e., whose cost is 2^M cells) is at most 2^{2^M} by lemma 3.

2. Given a computing facility $P = <n_1, \cdots ,n_Q>$ which costs 2^M cells and the set $S = \{s_1, \cdots ,s_N\}$ of Boolean variables, the number of different circuits that can be formed using P and S is at most $\prod_{i=1}^{Q} \binom{Q+N}{n_i}$. By lemma 2, substituting for each term in the product, this number is at most $\exp\left[(Q+N)\sum_{i=1}^{Q} H\left(\frac{n_i}{Q+N}\right)\right]$ where $H(x)$ is the uncertainty function defined in section < **1.3**>. Substituting for each H using lemma 1, this number is at most $\exp\left[(Q+N)\sum_{i=1}^{Q}\left(\frac{n_i}{Q+N}\right)\left(2 + \log\frac{Q+N}{n_i}\right)\right]$. Since each n_i is at least 1, this number is at most $\exp\left[(2 + \log(Q+N))\sum_{i=1}^{Q} n_i\right]$. Subject to $\sum_{i=1}^{Q} 2^{n_i} = 2^M$, the maximum value of $\sum_{i=1}^{Q} n_i$ occurs when all the n_i's are 1's and is 2^{M-1}. Hence, we get the following over-estimate: $\exp\left[(1 + \frac{1}{2}\log(2^{M-1} + N))\, 2^M\right]$.

3. Given a circuit formed by the above computing facility, the number of functions that can be implemented on the circuit is at most $Q\prod_{i=1}^{Q} \exp 2^{n_i}$ which is less than $2^{2 \times 2^M}$.

Therefore, from 1,2,3, the number of functions f whose support is a subset of S and can be implemented on some computing facility of cost 2^M is at most the multiplication of the three estimates, namely $\exp\left[(4 + \frac{1}{2}\log(2^{M-1} + N))2^M\right]$ which yields the required formula after re-arrangement. |

This lemma demonstrates the nature of the function 2^{2^n}. Each of the three estimates is of the order 2^{2^M} and their multiplication is of the same order (w.r.t. M). We now state the main theorem.

Theorem (6)

Given $\varepsilon > 0$, there exists a positive integer N_0 such that for any Boolean function f whose support is a subset of a fixed N-set S where $N \geq N_0$, the following holds:-

a. If $K(f)=K$, then there is a computing facility which costs at most $2^{K+\varepsilon N}$ cells that can implement f.

b. The fraction of functions f in the complexity range $K \leq K(f) \leq K + \varepsilon N$ that can be implemented on some computing facility that costs at most 2^K cells is less than ε.

Proof. We shall use theorems 4 and 5, and lemma 4.

a. From theorem 4, there is, in particular, a normal form implementation of f which costs at most 2^{K+o_K}. Taking N large enough, εN will be greater than $o_K = \log(1+K)$ since $K \leq N$. The asserted computing facility can be taken as $P = <|S_1|, \cdots, |S_L|, L>$ where $<S_1, \cdots, S_L>$ is the configuration of the above normal form.

b. From theorem 5, we have the following estimates: $N_K \leq \exp 2^{K+2\log N}$ and $N_{K+\varepsilon N} \geq \exp 2^{K+\varepsilon N-1}$. Therefore, taking N large enough, the number of functions whose complexity is between K and $K + \varepsilon N$ is at least $\exp 2^{K+(\varepsilon/2)N}$. But from lemma 4, at most $\exp 2^{M+\delta}$ functions can be implemented by the facilities which cost 2^M cells for any positive integer M. Since any function implemented on a facility that costs at most 2^M cells can be implemented on a facility that costs exactly 2^M (by adding some one-input gates without using their outputs) and from the definition of δ, N can be taken large enough to make the number of implementable functions using the facilities which cost at most $2^{[K]}$ cells less than $\exp 2^{K+(\varepsilon/4)N}$. Finally, by taking N large enough, the ratio of $\exp 2^{K+(\varepsilon/4)N}$ to $\exp 2^{K+(\varepsilon/2)N}$ can be made less than ε which completes the proof. |

Informally, this theorem says that if you take the functions of complexity $K(f)$ and try to implement them using a circuit whose cost is consistent with $K(f)$, you will always succeed, whereas if you try to "save a little bit", you will fail in almost all cases. Notice that if $K(f)$ itself is of the order of εN, then the theorem is not as significant. This has the nice interpretation that if the function is very simple, it pays to try to study it more closely to come up with a compact unsystematic implementation, whereas in the much harder case of a function of considerable complexity, it doesn't pay to do so. In [16], the structured design approach in VLSI systems is emphasized throughout for practical reasons. Theorem 6 provides theoretical justification for such an approach at a basic level.

< 3.4> Deterministic Entropy:-

The argument in the proof of part (**b**) in theorem 5 may raise the question as to whether or not there is an important class of functions whose complexity is at most K, other than those functions which depend on at most $\lfloor K \rfloor$ variables. The answer to this question is fortunately *yes*. Indeed, if such was not the case, the significance of $K(f)$ would be doubtful.

Definition. Let S be a fixed non-empty N-subset of **U**. A non-constant function f whose support is a subset of S is said to be of (deterministic) *entropy* H if it assumes the value 1 (or the value 0) in 2^H ($H \leq N-1$) states of the N variables in S.

The motivation for this terminology comes from information theory (chapter 3 in [10]). Notice that the definition of H depends on (the fixed) N.

Example. Let $S = \{s_1, \cdots, s_N\}$. The function $f = s_1$ has entropy $H = N-1$ (maximal entropy) since it assumes the value 1 in exactly 2^{N-1} states of the variables in S, those for which $s_1 = 1$.

In spite of this example, most functions of entropy H have their 2^H 1's (or 0's) distributed "at random" in the Karnaugh map. The complexity of a function of entropy H will prove vital in the next chapter. We now develop some prerequisites. Without loss of generality, we shall consider only the functions with 2^H 1's.

Definition. For a given function f of entropy H, the state of the variables in a subset S_i of S is said to be *positive* if there is an assignment of 0's and 1's to the rest of the variables in S that makes $f = 1$.

For a function f of entropy H, there are at most 2^H positive states for any subset S_i. Therefore, as far as f is concerned, the state of the variables in S_i can be encoded using $m = \lceil \log(1+2^H) \rceil$ binary variables (the extra 1 represents "the state is not positive"). Therefore, by taking $|S_i| > m$, this encoding constitutes information compression. We shall use this fact to implement low-entropy functions on normal forms of moderate size.

Example. Suppose that $H << N$. Partition S into $\sqrt{N/m}$ subsets each of cardinality $\sqrt{mN}$ (approximately). Consider the configuration C (figure 4) that has m duplicates of each of these subsets. Each primary gate has $\sqrt{mN}$ inputs and the secondary gate also has $m \times \sqrt{N/m} = \sqrt{mN}$ inputs. Since m is of the order of H, $\log\log N(C)$ will be of the order $\sqrt{HN}$ (by theorem (1)) which is much smaller than N. Furthermore, C admits any function f of entropy H since the m (duplicate) primary gates can be programmed to encode the (relevant) state of their inputs and the secondary gate can decide whether $f=1$ (matched positive states) or $f=0$ (unmatched positive states, or some non-positive state).

This example shows that low-entropy functions have low complexity. Even if H is not much smaller than N, a configuration C can be constructed using m duplicates of a subset of S having approximately $\frac{N+H}{2}$ elements together with the rest of the elements in S appearing as singletons in C. This configuration has $\log\log N(C)$ of the order $\frac{N+H}{2}$ which is still less than N.

Definition. $H{*}N$ is defined to be the minimum $\log\log N(C)$ for a configuration

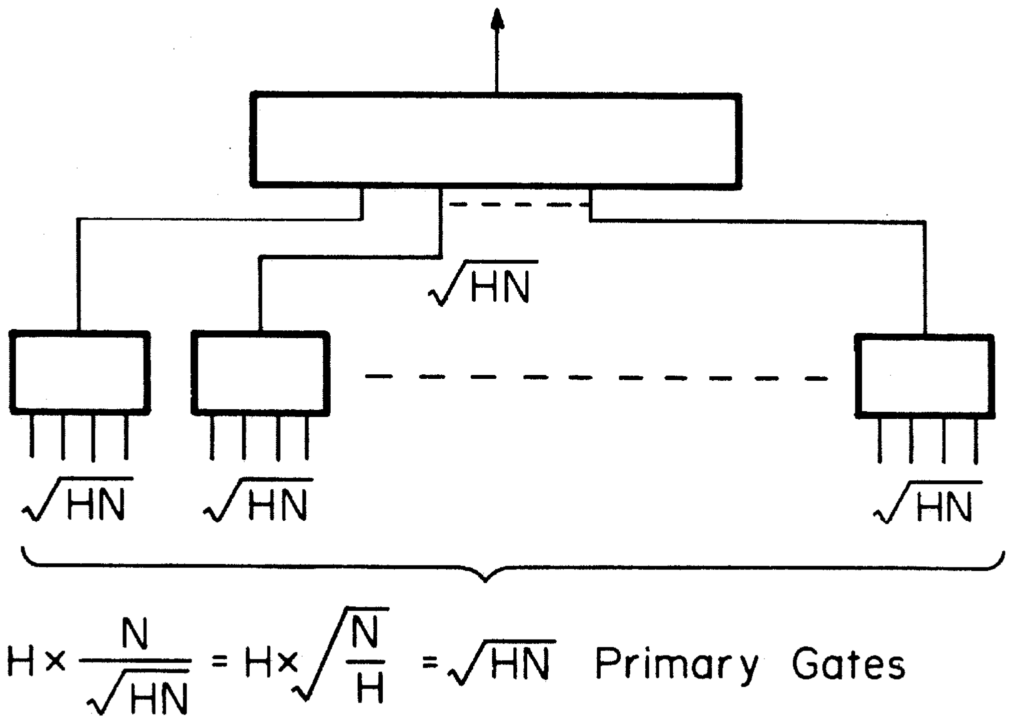

$\sqrt{HN}$
$\sqrt{HN}$
$\sqrt{HN}$
$\sqrt{HN}$
$H \times \frac{N}{\sqrt{HN}} = H \times \sqrt{\frac{N}{H}} = \sqrt{HN}$ Primary Gates

Figure 4: Implementation of low-entropy functions on normal forms

C that admits *all* functions f of entropy H with $T(f)$ a subset of S.

From our discussion, the order of $H{*}N$ is at most $\sqrt{HN}$ for small H and $\frac{N+H}{2}$ for any H. Whether *every* function of entropy H can be implemented using some special configuration C with $\log\log N(C)$ significantly less than $H{*}N$ is still an open question [24]. The lower bound given by the following theorem is asymptotically achievable using a multistage configuration.

Theorem (7)

Let H be the entropy of a Boolean function f whose support is a subset of the N-set S. We have:-

a. $K(f) \leq H{*}N$.

b. At least one of these functions f has $K(f) > H - 3\log N$.

Proof. We shall use lemmas 1 and 2 and theorem 5.

a. The proof is immediate from the definition of $H{*}N$.

b. The statement is trivial for (at least) $H \leq 4$ since $N \geq H+1$ and $K(f) > 0$. Now assume $H > 4$. The number of different f's with 2^H 1's is $\binom{2^N}{2^H}$. From lemma 2, this is at least $(2^{-(2+H/2)}) \times \exp\left[2^N H(2^{H-N})\right]$. By lemma 1, this is at least $(2^{-(2+H/2)}) \times \exp\left[(N-H)2^H\right]$. Since $H > 4$ and $N > H$, this is greater than $\exp\left[(N - H - \frac{1}{2})2^H\right] \geq \exp 2^{H-1} > \exp 2^{H-\log N}$. From theorem 5 (or the positivity of $K(f)$), the number of functions f with $K(f) \leq H - 3\log N$ is at most $\exp 2^{H-\log N}$. Therefore, it is impossible that all the functions f of entropy H have $K(f) \leq H - 3\log N$ which completes the proof. |

Notice that the upper bound on $K(f)$ of theorem 7 is significantly less than the a priori upper bound without regard to H (which is N) since the cost of the implementation of f is exponential in $K(f)$. The value of $H{*}N$ is better than halfway between N and H.

CHAPTER IV

Information

In this section, a probability measure is introduced and the complexity results of the previous chapters are discussed in a probabilistic context. The application of the complexity analysis to engineering problems is demonstrated by examples in pattern recognition and noisy decoding. Finally, a conclusive discussion about the methodology of the theory draws on the results in complexity, implementation and information.

< 4.1> Probabilistic Considerations:-

In this section, we consider the case where the Boolean variables $s_1, \cdots, s_N$ become random variables under some probability measure, and investigate the consequences of having this extra factor. Let S be a fixed (finite) non-empty N-subset of the universal set **U**. Define **S** to be the set $\{0,1\}^S$ of all binary N-tuples indexed by the Boolean variables $s_1, \cdots, s_N$ of S. Let p be a (fixed) probability measure on **S** (formally, let $(\mathbf{S}, P(\mathbf{S}), p)$ be a probability space where $P(\mathbf{S})$ denotes the power set of **S**). This measure induces a measure on all Boolean functions f whose support $T(f)$ is a subset of S, i.e., these functions become (dependent) random variables under this measure. We shall call **S** or the pair $(\mathbf{S}, p)$ the ***ensemble***.

Suppose that we can afford a non-zero probability of error ε in implementing the function f. Using this freedom, it is conceivable that we can reduce the complexity of f by "approximating" f by another function g which is less complex than, but doesn't *often* differ from, f.

Definition. The *ε-complexity* of a Boolean function f, denoted by $K_\varepsilon(f)$, is defined for $0 \leq \varepsilon \leq 1$ by:-

$$K_\varepsilon(f) = \min \{ K(g) \mid \Pr(f \neq g) \leq \varepsilon \}$$

Since the minimization domain includes f itself, it follows that $K_\varepsilon(f) \leq K(f)$. Also, it is obvious that for $\varepsilon \geq \frac{1}{2}$, $K_\varepsilon(f) = 0$ for any function f since one of the two constant functions $g = 0$ or $g = 1$ will be in the minimization domain. We now investigate the (possibility and) conditions for having $K_\varepsilon(f)$ significantly less than $K(f)$ for small ε.

Definition. The *source coding function* for the ensemble $(\mathbf{S}, p)$, denoted by $\hat{H}(\varepsilon)$, is the (real) function from $[0,1]$ to $[0,\infty)$ defined as follows. $\hat{H}(\varepsilon) = \hat{H}$ means that $\hat{H}$ is the minimum number such that $\mathbf{S}$ can be partitioned into $\mathbf{S}_T \cup \mathbf{S}_A$ (T for typical and A for atypical) where $|\mathbf{S}_T| \leq 2^{\hat{H}}$ and $\Pr(\mathbf{S}_A) \leq \varepsilon$.

The terminology is motivated by chapter 3 in [10]. $\hat{H}(\varepsilon)$ becomes very significant when the partition is such that $\mathbf{S}_T$ has most of the probability (small ε) while $\mathbf{S}_A$ has most of the points (small $\hat{H}$). For many probability measures of interest [25], such as the independent identically distributed s_i's, a fundamental theorem in information theory [20] says that, for small ε and large N, $\hat{H}(\varepsilon)$ is approximately the entropy H of the ensemble, which is defined by:

$$H = \sum_{\substack{s \in \mathbf{S} \\ p(s) \neq 0}} p(s) \log \frac{1}{p(s)}$$

The following example shows that this is not general for arbitrary (finite) ensembles.

Example. Let $X = \{ x_0, x_1, \cdots, x_{2^n} \}$ and let p be the probability measure defined on X by $p(x_0) = \frac{1}{2}$ and $p(x_i) = \frac{1}{2} 2^{-n}$ for $i = 1, \cdots, 2^n$. It is simple to show that the entropy $H = 1 + \frac{n}{2}$, which is considerably less than $\log|X|$; but no matter how large $|X|$ is, there is no way to capture most of the probability without taking practically all the points in X.

In spite of this example, many practical situations have $\hat{H}(\varepsilon)$ significantly less than N. In these cases, the following theorem becomes particularly useful.

Theorem (8)

Let f be a Boolean function whose support is a subset of the N-set S. Then $K_\varepsilon(f) \leq \hat{H}(\varepsilon) * N$.

Proof. We shall construct a function g satisfying $\Pr(f \neq g) \leq \varepsilon$ and $K(g) \leq \hat{H}(\varepsilon) * N$. Partition S into $S_T \cup S_A$ according to the definition of $\hat{H}(\varepsilon)$. Define the function g to be equal to f for the states in S_T and equal to 0 for the states in S_A. Now, $\Pr(f \neq g) \leq \Pr(S_A) \leq \varepsilon$. Also, g assumes the value 1 in at most $2^{\hat{H}(\varepsilon)}$ states in S. Hence the deterministic entropy (section < 3.4>) of g is at most $\hat{H}(\varepsilon)$ and the proof follows from theorem 7. |

For most problems in pattern recognition, theorem 8 guarantees low ε-complexity since $\hat{H}(\varepsilon)$ is typically far less than N.

< 4.2> Complexity-Reliability:-

Theorem 8 might suggest that looking for low-complexity approximations of high-complexity functions is a good policy. It is conceivable that in the maximum-entropy case (uniform probability measure), some other technique for constructing g, like the source coding method of the previous section, will work. In this section, we show that this is not the case.

Definition. An ε-error pattern e is any Boolean function satisfying $\Pr(e = 1) \leq \varepsilon$.

Each ε-error pattern can be thought of as the locations in the Karnaugh map where an error is made in approximating a function f by a function g with error probability less than ε. We now prove a basic lemma.

Lemma 5. Let the probability measure on **S** be uniform. Given $\varepsilon > 0$ and $0 \leq \delta < \frac{1}{2}$, there exists a positive integer N_0 such that for $N \geq N_0$, the number of functions f having $K_\delta(f) \leq N(1-\varepsilon)$ is less than $\varepsilon\, 2^{2^N}$.

Proof. By theorem 5, the statement is clear for $\delta = 0$. Suppose that $0 < \delta < \frac{1}{2}$. By theorem 5 again, we estimate the number of functions g of complexity $K(g) \leq (1-\varepsilon)N$ to be at most $2^{2^{N(1-(\varepsilon/2))}}$. From the definition of error pattern, it is obvious that the function g which approximates some function f with error $\leq \delta$ must satisfy $f = g \oplus e$ for some δ-error pattern e. The number of 1's in e is at most $\delta 2^N$ since the probability measure is uniform. Therefore, by lemma 2, the number of different δ-error patterns is at most $\delta 2^N \times \exp\left[2^N H(\delta)\right]$ where $H(\delta) < 1$ is the uncertainty function (section **< 1.3>**) evaluated at $\delta < \frac{1}{2}$. Hence, the number of functions f that can be approximated by some function g of complexity $K(g) \leq (1-\varepsilon)N$ is at most $\left[2^{2^{N(1-(\varepsilon/2))}}\right]\left[\delta 2^N \times \exp\left[2^N H(\delta)\right]\right]$ which can be made less than $\varepsilon\, 2^{2^N}$ by taking N large enough. This completes the proof. |

This lemma says that the functions of high complexity have their 1's and 0's in the Karnaugh map so scattered that there is no way to reduce the complexity significantly by placing *don't care*'s in less than half the blocks of the map. This yields the following complexity-reliability theorem.

Theorem (9)

Let the probability measure on **S** be uniform. Given $\varepsilon > 0$ and $0 \leq \delta \leq 1$, the following holds:-

a. If $\delta \geq \frac{1}{2}$, then $K_\delta(f) = 0$ for any function f.

b. If $\delta < \frac{1}{2}$, there exists a positive integer N_0 (function of ε and δ) such that for $N \geq N_0$, the fraction of functions f having $K_\delta(f) \leq K(f) - \varepsilon N$ is less than ε.

Proof. The proof for part (**a**) follows from the discussion in section **< 4.1>** and for part (**b**) from lemma 5. |

Although theorem 9 assumes uniform probability distribution, a similar statement can be proved for non-uniform distributions by considering only the typical blocks in the Karnaugh map.

< 4.3> Applications:-

In this section, we give two examples of the application of the theory in pattern recognition and coding theory. The examples provide insight into some standard problems using the new notions of complexity. We start by an informal description of coding theory.

A block code (section 3.1 in [10]) is a fixed collection of M binary strings (codewords) each of length N bits. The rate of the code is defined by $R = \frac{\log M}{N}$. In communication systems, the transmitter sends one of the M codewords to convey to the receiver $\log M$ facts or decisions, each of which could be one of two things. The channel is a medium which changes some of the bits in the transmitted codeword according to some probability measure. However, when $R < 1$, there are fewer codewords than there are binary strings of length N and the receiver may be able to guess from the received erroneous version of the codeword what the actual transmitted codeword was by looking for the codeword closest to what he has received (the codeword that differs from the received word in the least number of bits). The channel coding theorem in information theory (theorem 11 in [20]) assigns a number $C < 1$, called the *capacity*, to the channel and states that if $R < C$ then by taking N large enough, the optimal-decoding error probability can be made arbitrarily small for almost all choices of the code.

It is desirable to use as high a rate as affordable. However, when the rate is small, the probability of error goes to zero faster, and long codes, which mean costly decoding, are avoided. The following example shows that not only does the code become shorter, but the decoding complexity also becomes smaller w.r.t. to the code length. We shall use some basic relations of information theory (section 2.3 in [10]).

Example. Let x be the transmitted bit, y be the received bit. The binary symmetric channel (BSC) produces y from x according to the rule $y = x \oplus n$ where n is a binary random variable which is 1 only ε of the time [15]. When several bits are transmitted sequentially, the different n's are statistically independent (from one another and from the x's). We denote the operation on an N-binary string $\mathbf{x}$ to produce an N-binary string $\mathbf{y}$ by $\mathbf{y} = \mathbf{x} \oplus \mathbf{n}$ where $\mathbf{n}$ is a binary N-tuple of noise bits generated by the channel. Denote the ensembles of $\mathbf{x}, \mathbf{y}, \mathbf{n}$ by $\mathbf{X}, \mathbf{Y}, \mathbf{N}$ respectively. We have $H(\mathbf{Y}) = H(\mathbf{Y} \mid \mathbf{X}) + I(\mathbf{X};\mathbf{Y})$ where H denotes the self-information and I the mutual information. But $H(\mathbf{Y} \mid \mathbf{X}) = H(\mathbf{N} \mid \mathbf{X}) = H(\mathbf{N})$ since $\mathbf{N}$ is statistically independent of $\mathbf{X}$ and, given $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{n}$ are deterministic functions of one another. Hence $H(\mathbf{Y}) = H(\mathbf{N}) + I(\mathbf{X};\mathbf{Y})$. Suppose we use a random collection of codewords (in the sense of chapter 5 in [10]) of rate $R < C$ for communication on the BSC. By the channel coding theorem, we are almost sure that it is a good code and by theorem 5 (complexity distribution), we are almost sure (a priori) that the decoding function (best estimate for each bit of information in terms of the received codeword) has complexity near N. However, if C is near 1 and we use a rate much less than C, we know that $I(\mathbf{X};\mathbf{Y})$ is at most RN and that $H(\mathbf{N}) = (1-C)N$. Therefore, $H(\mathbf{Y}) \leq (1-C+R)N$ will be significantly less than N and by theorem 8 and the statistics of the BSC, the reliable decoding complexity cannot exceed the order of $(\sqrt{1-C+R})\,N$. A direct relation between reliability, rate, and complexity (instead of codeword length) is established.

In a typical pattern recognition problem, an image (an array of binary data or a point in S) is given and it is required to decide whether this image belongs to a certain class. Let us denote the optimal classification decision by D. D is a Boolean function of the Boolean variables in S (the pixels) and can a priori have $K(D)$ approximately N. However, since $\hat{H}(\varepsilon)$ is typically much smaller than N, theorem 8 tells us that the complexity of D cannot be bigger than the order of $\sqrt{N\hat{H}(\varepsilon)}$. Since $\hat{H}(\varepsilon)$, and hence $H(\mathbf{S})$, is crucial to the complexity, one should use whatever information that may be available to reduce the entropy. Such procedure is a common preprocessing step in pattern recognition and is known as *normalization*.

Definition. A *normalization procedure* is a mapping from $\mathbf{S}$ to $\hat{\mathbf{S}}$ such that $I(\hat{\mathbf{S}};D) = I(\mathbf{S};D)$ and $H(\hat{\mathbf{S}}) < H(\mathbf{S})$.

Notice how this definition is quite general and does not depend on any specific classification strategy to be followed after normalization. The normalization step uses the properties of the pattern which are known to us to get rid of the *false* entropy, i.e., the variations in the pattern which are not "random."

< 4.4> Epilogue:-

We conclude this work by pointing out some observations about complexity. Suppose that a "random" definition of complexity is introduced which satisfies a distribution like that of theorem 5. It is obvious that part (**b**) of the universality theorem (theorem 6) still applies since this complexity measure has the right count of functions. However, part (**a**) need not hold since the random measure does not guarantee any implementation cost and this nullifies its significance. On the other hand, if we define the complexity of a function to be simply its rank, both parts of theorem 6 apply. However, this definition does not have any class of low-complexity functions like the modulo-two sum or the low-entropy functions and this nullifies its practical significance.

Since the complexity distribution is dictated by lemma 4, it is obvious that *any* other consistent definition of $K(f)$, based on universal gates, will not differ significantly from our definition except in a negligible fraction of all functions. However, it turns out that this fraction is of practical interest and the definition should accommodate as many low-complexity functions as it can without violating part (**a**) of theorem 6. Lemma 4 is therefore universal and puts a rigid requirement on any measure of complexity that is related to cost in a direct manner.

The methodology can also be applied when other devices are available. Based on the implementation power of the device in question and its cost, it is possible to derive the counterpart of lemma 4 and predict the necessary distribution of complexity, then propose a model for reducibility based on these devices and introduce the new definition of complexity. Another approach that accommodates special-purpose devices within our framework is the *conditional* complexity. The nature of the device and its cost will not be explicitly specified, but the *function* it implements will represent it as far as the analysis is concerned.

Definition. The *conditional complexity* of a Boolean function f given a set of Boolean functions $G = \{g_1, \cdots ,g_d\}$, denoted by $K(f \mid G)$, is the complexity $K(f)$ given that the configuration $C = <S_1, \cdots ,S_L>$ is such that each S_i is a subset of $\mathbf{U} \cup G$ (instead of $\mathbf{U}$ only).

This definition assumes that the definition of Boolean mappings has been technically modified to accommodate functions as arguments. Notice that the definition of $K(f \mid G)$ enlarges the minimization domain of the definition of $K(f)$. It follows that $K(f \mid G) \leq K(f)$. The g_i's are meant to be the outputs of special devices and they capture the role of such devices in reducing (possibly to a triviality) the complexity of extracting the function f from the independent Boolean variables in $\mathbf{U}$. This definition can also be used to evaluate the usefulness of the device to the particular implementation of f by comparing $K(f \mid G)$ to $K(f)$.

References

[1] H. Abelson, "Towards a theory of local and global in computation," *Theoretical Computer Science*, Vol. 6, pp. 41-67, 1978.

[2] H. Abelson et al., "Compositional complexity of Boolean functions," *Discrete Applied Mathematics*, Vol. 4, pp. 1-10, 1982.

[3] Y. Abu-Mostafa, "Information-theoretic characterization of linear feature spaces," to be published, 1982.

[4] A. Aho et al., *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[5] E. Berlekamp et al., "On the inherent intractability of certain coding problems," *IEEE Trans. on Information Theory*, Vol. IT-24, pp. 384-386, 1978.

[6] A. Borodin, "On relating time and space to size and depth," *SIAM J. Comput.*, Vol. 6, pp. 733-744, 1977.

[7] G. Chaitin, "Information-theoretic computational complexity," *IEEE Trans. on Information Theory*, Vol. IT-20, pp. 10-15, 1974.

[8] G. Chaitin, "A theory of program size formally identical to information theory," *J. of the ACM*, Vol. 22, pp. 329-340, 1979.

[9] R. Duda et al., *Pattern Classification and Scene Analysis*, Wiley-Interscience, 1973.

[10] R. Gallager, *Information Theory and Reliable Communication*, John Wiley, 1968.

[11] D. Knuth, "An almost linear recurrence," *Fibonacci Quarterly*, Vol. 4, pp. 117-128, 1966.

[12] Z. Kohavi, *Switching and Finite Automata Theory, second edition*, McGraw-Hill, 1978.

[13] A. Kolmogorov, "On the representation of continuous functions of several variables by superpositions of continuous functions of a smaller number of variables," *American Math. Soc. Transl. (2)*, Vol. 17, pp. 369-373, 1961.

[14] C. Liu, *Introduction to Combinatorial Mathematics*, McGraw-Hill, 1968.

[15] R. McEliece, *The Theory of Information and Coding*, Addison-Wesley, 1977.

[16] C. Mead et al., *Introduction to VLSI Systems*, Addison-Wesley, 1980.

[17] J. Peatman, *Digital Hardware Design*, McGraw-Hill, 1980.

[18] W. Peterson et al., *Error Correcting Codes*, MIT Press, 1972.

[19] H. Ryser, *Combinatorial Mathematics*, the Mathematical Association of America, 1963.

[20] C. Shannon, "A mathematical theory of communication," *Bell Sys. Tech. J.*, Vol. 27, pp. 379-423, 1948.

[21] C. Shannon, "The synthesis of two-terminal switching circuits," *Bell Sys. Tech. J.*, Vol. 28, pp. 59-98, 1949.

[22] S. Skyum et al., "A complexity theory based on Boolean Algebra," *Proc. 22nd Symp. on Foundations of Computer Science*, IEEE, pp. 244-253, 1981.

[23] L. Valiant, "Completeness classes in Algebra," *Proc. 11th ACM Symp. on Theory of Computing*, pp. 249-261, 1979.

[24] J. van Lint, private correspondence, 1983.

[25] J. Wolfowitz, *Coding Theorems of Information Theory*, third edition, Springer-Verlag, 1978.