The Complexity of Information Extraction

YASER S. ABU-MOSTAFA

This paper is dedicated to the memory of Herbert J. Ryser.

Abstract—**How difficult are decision problems based on natural data, such as pattern recognition? To answer this question, decision problems are characterized by introducing four measures defined on a Boolean function f of N variables: the implementation cost $C(f)$, the randomness $R(f)$, the deterministic entropy $H(f)$, and the complexity $K(f)$. The highlights and main results are roughly as follows. 1) $C(f) \approx R(f) \approx H(f) \approx K(f)$, all measured in bits. 2) Decision problems based on natural data are partially random (in the Kolmogorov sense) and have low entropy with respect to their dimensionality, and the relations between the four measures translate to lower and upper bounds on the cost of solving these problems. 3) Allowing small errors in the implementation of f saves a lot in the low entropy case but saves nothing in the high-entropy case. If f is partially structured, the implementation cost is reduced substantially.**

I. Introduction

THE ACCESSIBILITY of available information is the central issue in decision-making based on natural data. In a typical pattern recognition problem, for example, we have more than enough information to make the correct decision, but this is precluded by the high complexity of extracting the right bits of information from the data.

Pattern recognition problems are unique in their computational demands. In contrast with the structured nature of the problems in computational complexity, the problems which are based on "natural" data are inherently random, that is, so unstructured that they have no concise effective definition. An algorithm that tells us whether or not there is a tree in a given picture contains, at least implicity, the lengthy definition of the object tree. The purpose of this work is to define and study the complexity of solving decision problems of random nature.

To do so, we define four parameters to measure the complexity in the same way we measure the information. In terms of these parameters, we address questions like: What is the cost of solving a high-complexity problem? Should we hope to find a tricky algorithm or a compact system to solve such a problem with low cost? What is the impact of the dimensionality of the problem? How significant is the partial structure of a problem in reducing its complexity? Is there a system that is capable of solving a wide class of problems optimally? These questions and their answers are the core of this work.

Manuscript received August 8, 1984; revised October 23, 1985. This paper was presented in part at the IEEE International Symposium on Information Theory, Brighton, England, June 1985.

The author is with the Departments of Electrical Engineering and Computer Science, 116-81, California Institute of Technology, Pasadena, CA 91125, U.S.A.

IEEE Log Number 8608098.

A. Main Results

Although the definitions and relations are motivated by decision problems, the results can be stated entirely in the context of the complexity of Boolean functions. We can summarize our approach as follows. We characterize a Boolean function f by four measures $C(f)$, $R(f)$, $H(f)$, $K(f)$. The values of these measures are normalized to range approximately from 0 to N bits for a function of N variables. Roughly speaking, $C(f)$ measures the cost of implementing f based on memory devices (related to combinational complexity [15], [23], [25]), $R(f)$ measures the randomness or lack of structure in f (based on the Kolmogorov–Chaitin complexity [8], [13]), $H(f)$ measures the entropy or essential dimensionality of the independent variables of f (related to Shannon's entropy [24]), and $K(f)$ measures the rank of f among all Boolean functions as far as simple decomposition is concerned (based on compositional complexity [2]). An important preliminary result about these measures is that they share the common distribution of Fig. 1, so that approximately 2^{2^K} functions have the value of each measure in the neighborhood of K bits.

The main results of the paper are the pairwise relations between the four measures C, H, R, and K and the interpretation of these relations. The convenience of having these four measures on the same scale makes the form of these relations surprisingly simple. All of them point in one direction; the values of these completely different measures are practically the same. More accurately, we have 12 inequalities:

$$
\begin{aligned}
&R(f) \le K(f) + o(N), && \text{for all } f\\
&K(f) \le R(f) + o(N), && \text{for almost all } f\\
&H(f) \le K(f) + o(N), && \text{for almost all, but not all, } f\\
&K(f) \le H(f) + o(N), && \text{for almost all } f\\
&C(f) \le K(f) + o(N), && \text{for all } f\\
&K(f) \le C(f) + o(N), && \text{for almost all } f\\
&H(f) \le R(f) + o(N), && \text{for almost all, but not all } f\\
&R(f) \le H(f) + o(N), && \text{for all } f\\
&C(f) \le R(f) + o(N), && \text{for almost all } f\\
&R(f) \le C(f) + o(N), && \text{for all } f\\
&C(f) \le H(f) + o(N), && \text{for all } f\\
&H(f) \le C(f) + o(N), && \text{for almost all, but not all } f.
\end{aligned}
$$

In these relations, $o(N)$ is some positive function of N

0018-9448/86/0700-0513$01.00

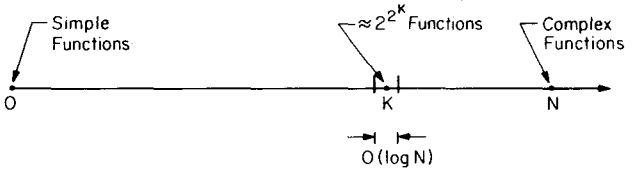

Fig. 1. Common distribution of four measures.

which is asymptotically negligible with respect to N (mostly of the order log N), and "almost all" means all but an asymptotically negligible fraction. The exact statements are given in Section III. The proofs are based on enumeration arguments, explicit constructions, and simulation techniques.

B. Previous Work

Among the several approaches for defining complexity, [2], [25], and [26] had the most relevant notions. The complexity and cost models were inspired by [1] and [19], and by the relations between time and circuit complexity (e.g., [6]). The key to the characterization of randomness is provided in [8] and [13]. Some background textbooks are [14] and [22] for combinatorial methods, [9] for pattern recognition, [10] and [17] for information theory, [12] and [19] for digital logic, and [5] and [23] for computational complexity. See [3] for many related results.

Previous work on compositional complexity of Boolean functions [2] gives a complexity definition close to $K(f)$ and proves some of the basic properties of what we call normal-form input configurations. What is new here is the uniform definition of $K(f)$ for any number of variables on a bit scale, the distribution of $K(f)$, and the relation to implementation cost (combinational complexity). Previous work on combinational complexity of Boolean functions [15], [25] can be translated in terms of $C(f)$ as $C(f) = N - o(N)$ for almost all f. What is new here is the estimation of how many functions have a certain value for $C(f)$ over the whole range from 0 to N. The work of Pippenger [21] implicitly incorporates the notions of deterministic entropy and approximation, and the relations between $C(f)$ and $H(f)$ and between $C(f)$ and $C_\delta(f)$ can be derived as corollaries of his main theorem. What is new here is the derivation of similar results for $K(f)$ and $R(f)$. Previous definitions of the Turing complexity of Boolean functions [23] are different from the definition of $R(f)$.

C. Outline

Section II contains the preliminary definitions and properties used in the rest of the paper, especially in connection with the complexity measure $K(f)$. The main results are in Section III, where the four measures are defined and related, and their relations are interpreted. Section IV discusses the complexity in a probabilistic context, where small computational errors are allowed. Finally, false entropy is introduced in Section V to characterize the partial structure of practical problems.

Lemmas are stated and proved in the appendices and are used technically in other proofs. Propositions are intermediate results about the notions of this paper and are used in the proofs of the theorems. They are stated in the text and proved in the appendices. The three theorems are the main results, and they are stated and proved in the text. While Theorem 1 relates $K(f)$ to $C(f)$, the other pairwise relations are contained in the discussion in Sections III-D and III-E as they follow the same line of argument.

D. Notation

Bits are used as the units throughout this work, and all logarithms (log) and exponentials (exp) are to the base 2. As usual, $\lfloor K \rfloor$ stands for the largest integer less than or equal to K, while $\lceil K \rceil$ stands for the smallest integer greater than or equal to K.

We shall use the notion of a *multiset*, which is a collection of objects where repetition is allowed and the order does not 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 $\langle \cdot \rangle$. 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 sum multiplicities.

As in most asymptotic results, an error term exists, denoted by $o(N)$. A statement involving this symbol will be interpreted as follows: a function n from natural numbers to natural numbers exists, satisfying $\lim_{N \to \infty}(n(N)/N) = 0$, and it will make the statement true when substituted for $o(N)$.

II. The Normal Form

This section is devoted to the development of the basic notions and relations used in the definition of the complexity and cost measures in the next section.

A. Boolean Functions

Here, we set 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 because their domains are different, but they are the same "function" in our definition. Furthermore, 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.

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

Let $\boldsymbol{U}$ be a universal set of independent Boolean variables assuming the values 0 or 1 only. The cardinality of $\boldsymbol{U}$ is potentially infinite. We refer to any specific assignment of 0's and 1's to all Boolean variables in $\boldsymbol{U}$ as the *state* of the system. Let $S = \{s_1, \cdots, s_N\}$ be any nonempty finite subset of $\boldsymbol{U}$ for some positive integer N. A *Boolean mapping on S*, 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 variables whose value is determined by the values of the independent variables in S. In contrast to the Boolean mapping $f_n(\cdot)$ 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; 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 nonempty subsets of $\boldsymbol{U}$) is denoted by $\boldsymbol{M}$. The cardinality of $\boldsymbol{M}$ is potentially infinite. Some elements of $\boldsymbol{M}$ 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 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 $\boldsymbol{M}$ 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.

Clearly, $\equiv$ is an equivalence relation; hence it induces a partition of $\boldsymbol{M}$ 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 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 $\boldsymbol{M}$. The set of all Boolean functions is denoted by $\boldsymbol{F}$.

Notice that any Boolan function f depends on a finite number of Boolean variables in $\boldsymbol{U}$, because the definition of Boolean mappings on S applies only to finite sets. 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 liberal 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 $\boldsymbol{U}$ 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, that is, whose rank is N, is potentially infinite.

B. Configurations

Proposing a valid and useful measure of complexity involves two considerations from theory and practice that are often conflicting. 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 must capture both aspects. We introduce the components of our complexity measure here, but the full justification of the definition is reflected in the theorems of the following sections.

Our building blocks are n-input "universal gates," for example, programmable devices [19, sec. 2.12] such as a programmable read-only memory (PROM) with n address lines and one data line. Although any function of n variables can be simulated by this universal gate, the cost of implementing such a gate (in terms of the number of standard switching devices or the number of memory locations) is exponential in the number of inputs. Many functions of n variables can be simulated using less powerful devices of n inputs or several smaller universal gates interconnected together. The *normal form* (Fig. 2) is the simplest way of breaking down a function in this manner. It consists of a first stage of (primary) universal gates which take their inputs directly from the input Boolean variables and a second stage with one (secondary) universal gate which takes the outputs of the primary gates and produces the function being simulated. A normal form can be thought of as an interconnection of devices analogous to the disjunctive or conjunctive normal form [12] where the AND's and OR's are now replaced by universal gates. It also resembles the 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.

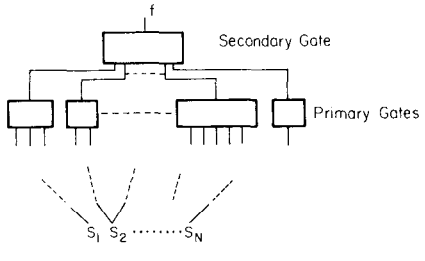

Fig. 2. Normal form.

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 of *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 global classification decision based on local extracted features. The normal form structure is based on support systems which were introduced in the context of local and global computation [1]. We now give the definition of configurations that formalizes this structure.

Definition: A *normal-form input configuration* (or simply a *configuration*) C is a finite multiset of finite nonempty subsets of U, $C = \langle S_1, \cdots, S_L \rangle$. The S_i are called the *components* of the configuration C. The *support* of C, denoted by $T(C)$, is defined as $\bigcup_{l=1}^{L} S_l$ (all Boolean variables which appear in any component of the configuration). The cardinality of the support is called the *rank* of C, 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 (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).

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 are then input to the secondary gate whose output becomes the overall simulated function.

Definition: A nonempty configuration $C = \langle S_1, \cdots, S_L \rangle$ is said to *admit* a Boolean function f if there exist Boolean mappings $F_{S_1}^1, \cdots, F_{S_L}^L$ on the subsets $S_1, \cdots, S_L$ of U (the superscript distinguishes between the F's) and a Boolean mapping $D_L(\cdot)$ on the integer L such that, for all states of the system, $f = D_L(F_{S_1}^1, \cdots, F_{S_L}^L)$. The empty configuration admits the constant functions only.

A function admitted by a configuration is one that can be simulated 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 $\boldsymbol{F}(C)$. Two configurations C_1 and C_2 are *equivalent* if they admit the same functions, that is, if $\boldsymbol{F}(C_1) = \boldsymbol{F}(C_2)$. The number of functions admitted by C is denoted by $N(C) (= |\boldsymbol{F}(C)|)$.

We observe immediately that the support $T(f)$ of a function admitted by a configuration must be a subset of the support of the configuration $T(C)$. The set $\boldsymbol{F}(C)$ contains all Boolean functions that can be simulated on a normal form using the input configuration C. The number of functions $N(C)$ admitted by a configuration C expresses the power of C. Notice that each function is counted as a single vote regardless of its "complexity." Since the gates of the normal form are universal, the configuration that 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. We now develop some relations between the different parameters of the configuration as well as some structural properties.

C. Properties of Configurations

The parameters of a configuration C are interrelated. The following proposition 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 section and will provide insight into the nature of configurations.

Proposition 1: Let $C = \langle S_1, \cdots, S_L \rangle$ 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) $d(C) \leq \log\log N(C) \leq \max(l(C), d(C)) + \log(\max(l(C), d(C)) + 1)$; b) $\sqrt{r(C)} \leq \log\log N(C) \leq r(C)$.

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 $N(C)$ essentially 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 (possible the secondary one) must be at least $\sqrt{r(C)}$.

Although the configuration is just a multiset of subsets, it has the functional interpretation of simulating Boolean functions on normal forms. This makes configurations distinct from hypergraphs, for example. We shall use this fact to characterize configurations in a way similar to [2].

Definition: A configuration $C = \langle S_1, \cdots, S_L \rangle$ is *redundant* if one of the S_i can be omitted without diminishing $\boldsymbol{F}(C)$.

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 $C = \langle \{s_1, s_2\}, \{s_1, s_2, s_3\} \rangle$ without diminishing $\boldsymbol{F}(C)$. The following proposition describes nonredundant configurations.

Proposition 2: If a configuration $C = \langle S_1, \cdots, S_L \rangle$ is not redundant, then for all subsets λ of $\{1, \cdots, L\}$, the following condition holds:

$$\left| \bigcup_{i \in \lambda} S_i \right| \geq |\lambda|.$$

Proposition 2 says that if several components of the configuration have only a few variables between them, then these variables must be *overrepresented*, and some of these components may be omitted without damaging the

information passed on to the secondary gate about the variables. This result is used in proving that the configuration does not have to be very long and hence in estimating the number of distinct configurations.

Proposition 3: a) If C is a configuration with $l(C) > r(C)$, then a configuration C^* with $l(C^*) \leq r(C^*)$ exists which is equivalent to C. b) Let S be an N-subset of the universal set $\boldsymbol{U}$. There are at most 2^{N^2} possible values for $\boldsymbol{F}(C)$ over all configurations C whose support $T(C)$ is a subset of S.

Proposition 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. We are now in a position to define complexity in terms of normal-form input configurations.

III. Four Measures

In this section, we introduce four measures defined on Boolean functions: the complexity $K(f)$, the cost $C(f)$, the randomness $R(f)$, and the entropy $H(f)$. We derive and interpret the pairwise relations between these measures.

A. Complexity

Suppose we have a number of objects which possess a certain property to different degrees. We want 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 how much 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 a *comparative* approach.

To apply comparativeness to define a measure for the complexity of Boolean functions, we need to order these functions according to their complexity. The notion of *reducibility* is a natural way of comparing the complexity of two procedures. If procedure A can be carried out by transforming it in a simple way to procedure B and then carrying out the procedure B instead, A cannot be more complex than B. In our case, we use admittance to configurations as a basis for reducibility, which resembles other forms of reducibility such as projection [26]. A similar approach was introduced in [2]. The point is that if we take the smallest configuration C that simulates a Boolean function f, then the other functions admitted by C are reducible to f since they can be simulated by the smallest structure that simulates f.

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

$$K(f) = \log\log \min \{ N(C) | C \text{ admits } f \}.$$

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

We first dispose of the log log as being a scale down for $N(C)$ which is typically of the form 2^{2^K}. The definition says that we consider all configurations C that admit the function f, choose the minimal configuration 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). Also, $K(f) \leq N$ for any function f which depends on N variables, since f must be admitted to a configuration C whose support consists of these N variables only and hence has $N(C) \leq 2^{2^N}$. Notice that the normal form served as a "catalyst" in the definition of complexity.

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 maximal disjunctive and conjunctive normal forms [12].

One of the "health" properties of any complexity measure is that it should resolve different levels of complexity. The following proposition estimates the number of functions at different levels of complexity.

Proposition 4: Let $\boldsymbol{F}_S$ be the set of all Boolean functions f whose support is a subset of nonempty N-set S. Define $N_K = |\{ f \in \boldsymbol{F}_S | K(f) \leq K \}|$. For $0 \leq K \leq N$, we have

$$K - 1 \leq \log\log N_K \leq K + 2\log N.$$

This means that N_K is approximately 2^{2^K} functions (with respect to K with an error of $\pm o(N)$). Although N_K is the total number of functions whose complexity is at most K, almost all of these functions are very close to K on the complexity scale. This is because $2^{2^K} - 2^{2^{K-\epsilon N}}$ is approximately 2^{2^K}, where ϵ is arbitrarily small and N is sufficiently large. Hence the number of functions whose complexity is between $K - \epsilon N$ and K is approximately 2^{2^K}.

B. Cost

To have practical significance, the complexity measure $K(f)$ should be related to the cost of implementing f. We start by defining the cost.

Definition: The (*denormalized*) *cost* of a universal gate of n inputs ($n \geq 0$) is defined to be 2^n "cells." The cost of an interconnection is zero cells. The cost of a collection of gates and interconnections is the sum of the costs of the components.

This definition is motivated by the actual number of cells in an integrated-circuit PROM, and by the fact that implementing an n-input universal gate requires an exponential number of standard gates. Notice that, in practice, an interconnection has a nonzero cost. However, this fact can only strengthen the main results to be proved shortly.

A normal form whose primary gates have $n_1, \cdots, n_L$ inputs costs $2^L + \sum_{i=1}^{L} 2^{n_i}$ cells (the secondary gate has L inputs). For example, the normal form corresponding to the empty configuration has $L = 0$ and costs 1 cell. We now show that the cost of normal form implementation of a function f is directly related to the complexity $K(f)$.

Proposition 5: Let f be a Boolean function of a complexity $K(f) = K$. Then, a) any normal form implementation of f costs at least 2^K cells, and b) there is a normal form implementation of f which costs at most $2^{K+\log(1+K)}$ cells.

This relation between the complexity of f and the cost of its normal form enables us to think of $K(f)$ as the cost of this restricted implementation of f. To contrast this with the unrestricted implementation, we introduce a normalized version of combinational complexity.

A collection of Q universal gates (Fig. 3(a)) with $n_1, \cdots, n_Q$ inputs costs $\sum_{i=1}^{Q} 2^{n_i}$ cells. We use these gates together with the input lines $s_1, \cdots, s_N$ to build a *combinational circuit* Γ to simulate a given function. A combination circuit (Fig. 3(b)) is an unrestricted loop-free interconnection of gates (with unlimited fan-out). Γ simulates f if f is one of the gate outputs $y_1, \cdots, y_Q$ in Γ.

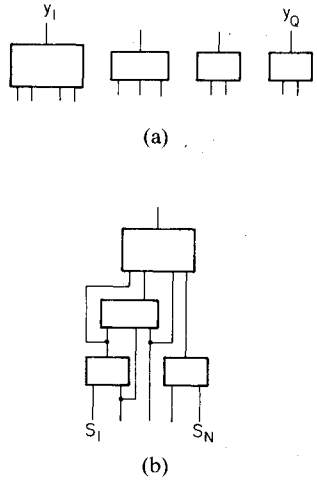

Fig. 3. (a) Collection of Q universal gates. (b) Combinational circuit made out of gates in (a).

Definition: The (*normalized*) *cost* of a Boolean function f, denoted by $C(f)$, is defined by

$$C(f) = \log \min \{\text{cost of } \Gamma : \Gamma \text{ simulates } f\}.$$

The units of $C(f)$ are bits.

A constant function f is the output of a universal gate with zero inputs. Such a gate costs $2^0 = 1$ cell. Hence $C(f) = 0$ bits for the two constant functions, and $C(f) > 0$ for all other functions. Also, $C(f) \leq N$ for any function f which depends on N variables, since a universal gate with N inputs (2^N cells) can simulate any such function. This normalization of cost simplifies the form of the relations to be derived and emphasizes the order of magnitude of the cost. Notice that $C(f)$ differs by at most a constant from the normalized cost based on any other complete basis of switching devices such as two-input NAND gates. To see this, one can simulate universal gates using the complete basis and vice versa.

Although $C(f)$ is based on the cost of an unrestricted circuit that simulates f without assigning any cost to the design of the circuit or its wiring, the distribution of $C(f)$ is very close to that of $K(f)$, which was based on a very structured circuit. This distribution, estimated in the following proposition, is a key factor in relating $C(f)$ to $K(f)$.

Proposition 6: Let $\boldsymbol{F}_S$ be the set of all Boolean functions f whose support is a subset of a nonempty N-set S. Define $\hat{N}_K = |\{f \in \boldsymbol{F}_S | C(f) \leq K\}|$. For $0 \leq K \leq N$, we have

$$K - 1 \leq \log\log \hat{N}_K \leq K + \log(8 + N).$$

The situation is similar to Proposition 4. Again, the number of functions whose cost is between $K - \epsilon N$ and K is approximately 2^{2^K} (with respect to K with an error of $\pm o(N)$). A corollary of Proposition 6 is that almost all functions of N variables have $C(f) \geq N - o(N)$, which is a known result (e.g., in [25]).

C. Complexity Versus Cost

The contrast between the definitions of complexity and cost is clear: $K(f)$ is based on the theoretical principles of reducibility and comparativeness applied to the simplest decomposition of f, while $C(f)$ is based on the actual cost in the most general decomposition of f. However, these two quantities turn out to be closely related. Inherent complexity is to implementation cost what mass is to weight, an intrinsic property that is different from, but directly related to, a practical impact.

Theorem 1: Given $\epsilon > 0$, a positive integer N_0 exists such that for any Boolean function f whose support is a subset of a fixed N-set S, where $N \geq N_0$, and for $0 \leq K \leq (1 - \epsilon)N$, the following holds. a) If $K(f) = K$, then $C(f) \leq K + \epsilon N$ bits. b) The fraction of functions f in the complexity range $K \leq K(f) \leq K + \epsilon N$ which have $C(f) \leq K$ is less than ϵ.

Proof: We shall use Propositions 4, 5, and 6. a) From Proposition 5, a normal form implementation of f exists which costs at most $2^{K+\log(1+K)}$ cells. Taking N large enough, ϵN will be greater than $\log(1 + K)$ since $K \leq N$, and the result follows by taking the logarithm.

b) From Proposition 4, we have the following estimates: $N_K \leq \exp 2^{K+2\log N}$ and $N_{K+\epsilon N} \geq \exp 2^{K+\epsilon N-1}$. Therefore, taking N large enough, the number of functions whose complexity is between K and $K + \epsilon N$ is at least $\exp 2^{K+(\epsilon/2)N}$. From Proposition 6, at most $\exp 2^{K+\log(8+N)}$ functions have $C(f) \leq K$. By taking N large enough, $\log(8 + N)$ will be less than $(\epsilon/4)N$ and the ratio of $\exp 2^{K+(\epsilon/4)N}$ to $\exp 2^{K+(\epsilon/2)N}$ can be made less than ϵ which completes the proof. Q.E.D.

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 cut the cost, you will fail in almost all cases. Notice that for very low-complexity functions, the error term ϵN becomes significant, and hence the theorem does not say much. This has the nice interpretation that if the function is very simple, it may pay to look for a compact unsystematic implementation.

D. Randomness

Theorem 1 says that the complexity measure $K(f)$ is almost identical to the cost measure $C(f)$. It turns out that the two measures are also related to the algorithmic information (Kolmogorov–Chaitin complexity [8], [13]) applied to Boolean functions, which we call the randomness of the function. We establish these relations and discuss their interpretation.

Let U be a universal Turing machine [27] with input alphabet $\{0,1\}$, and let $\boldsymbol{p}$ denote the binary program supplied to the tape of U. If, given $\boldsymbol{p}$, U halts and leaves the binary string w on the tape, we say that $w = U(\boldsymbol{p})$. Let $\tau(f)$ be a listing of the truth table of the Boolean function f, that is, $\tau(f) = \tau_0, \tau_1, \cdots, \tau_{2^N-1}$, where τ_k is the value of f when the inputs are the N-bit binary representation of the number k. The measure of randomness is defined in terms of the shortest length of a program that generates $\tau(f)$. This measure will have a large value if we cannot describe the truth table in a concise way.

Definition: The *randomness* of a Boolean function f, denoted by $R(f)$, is defined by

$$R(f) = \log \min \{|\boldsymbol{p}| \geq 1 | U(\boldsymbol{p}) = \tau(f)\}.$$

The units of $R(f)$ are bits.

Since any string $\tau(f)$ can be generated by a program whose length is a constant (the code of a trivial Turing machine) plus the length of the string (namely 2^N), $R(f)$ is at most $\approx N$ bits. In contrast with the $K(f)$ and $C(f)$, $R(f)$ is an uncomputable function.

The versatility of a universal Turing machine enables us to find short programs to generate $\tau(f)$ whenever f has a compact normal form (small $K(f)$) or a compact combinational circuit (small $C(f)$). For example, we can construct a program $\boldsymbol{p}$ of length $|\boldsymbol{p}| \leq 2^{K(f)+o(N)}$ that generates $\tau(f)$, thereby showing that $R(f) \leq K(f) + o(N)$. Given $K(f)$, a normal form exists that simulates f and has at most $K(f)$ inputs per gate. The program is based on this normal form and consists of three parts. The first part is a constant-length routine for generating $\tau(f)$, bit by bit, given the full specification of the normal form and the truth tables of its gates. The second part of the program is an encoding of the normal form input configuration, the length of this encoding is bounded by a polynomial in N. The third part is a listing of the truth tables of the gates, at most $K(f) + 1$ tables each of length at most $2^{K(f)}$ bits. Putting the three parts together, it is clear that $|\boldsymbol{p}| \leq 2^{K(f)+o(N)}$. We can also show that $R(f) \leq C(f) + o(N)$ by constructing a program to generate $\tau(f)$ based on the smallest circuit that simulates f. We fix a lexicographic ordering of *all* circuits, with the less costly circuits coming first. The program $\boldsymbol{p}$ includes the smallest index of a circuit that simulates f (by Proposition 6, this index will be at most $2^{C(f)+o(N)}$ bits long), a constant-length routine to "decode" the circuit from its index, and a constant-length routine to generate $\tau(f)$ from the circuit. The details are straightforward.

On the other hand, for any K, there are at most $\sum_{i=0}^{2^K} 2^i < 2^{2^K+1}$ programs $\boldsymbol{p}$ of length $|\boldsymbol{p}| \leq 2^K$ bits, and hence at most that many functions f with $R(f) \leq K$. Using the same argument in part b) of Theorem 1, the other direction of the relations between $K(f)$ and $R(f)$ and between $C(f)$ and $R(f)$ is established.

An interesting interpretation can be gained from these relations. While $C(f)$ measures the size of a purely combinational implementation of f, $R(f)$ measures the size of a purely sequential implementation. An important merit of $K(f)$ is its intimate relation to both of these measures. The distribution of these measures was a key factor in the arguments; the number of functions having $K(f) \leq K(C(f) \leq K$, or $R(f) \leq K)$ is approximately 2^{2^K}. While this approach relates complexity measures in an "almost always" sense, it is of interest to investigate which measures are *pointwise* identical, that is, different by $o(N)$ for *every* function f [4].

E. Deterministic Entropy

Proposition 4 may raise the question as to whether or not there is an important class of functions whose complexity is less than K, other than those functions which depend on less that K variables. The answer to this question is fortunately *yes*. The class we are concerned with here is the class of *low-entropy functions*.

Definition: Let S be a fixed nonempty N-subset of $\boldsymbol{U}$. If a function f, whose support is a subset of S, assumes the value 1 (or the value 0) in $h \leq 2^{N-1}$ states of the variables in S, then f is said to be of (*deterministic*) *entropy* $H = \log(1 + h)$. The units of entropy are *bits*.

Functions of low entropy have relatively few 1's or 0's in their Karnaugh maps [12]. The motivation for this terminology will become apparent shortly. Notice that the definition of H depends on (the fixed) N. We are interested in estimating the complexity of the functions of entropy H. Without loss of generality, we shall consider only the functions with h 1's.

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

f can have at most $h = 2^H - 1$ positive states for any subset S_i, since only h 1's are in the Karnaugh map of the function. Therefore, as far as f is concerned, the state of the variables in S_i can be encoded using $\lceil \log(1 + h) \rceil = \lceil H \rceil$ binary variables (the extra 1 represents "the state is not positive"). Taking $|S_i| > \lceil H \rceil$, this encoding constitutes information compression, since we represent a number of variables by a smaller number of variables. Furthermore, in terms of the new variables (the com-

pressed variables from S_i together with the rest of the variables outside S_i), the entropy of the function remains the same, and hence we can repeat this compression. This fact is used to implement low-entropy functions on circuits and normal forms of moderate size.

Circuit Implementation: Consider an arbitrary Boolean function f of N variables whose entropy is H bits. Since we can compress any number of variables into $\lceil H \rceil$ variables, we repeatedly compress $\lceil H \rceil + 1$ variables into $\lceil H \rceil$ variables, each time using $\lceil H \rceil$ universal gates of $\lceil H \rceil + 1$ inputs (Fig. 4(a)). We thus reduce the N variables to $N-1, N-2, \cdots$, down to any number of variables, say $\lceil H \rceil + 1$ variables (Fig. 4(b)). We can then implement the function f in terms of these $\lceil H \rceil + 1$ variables using one universal gate of $\lceil H \rceil + 1$ inputs. The compression from N to $\lceil H \rceil + 1$ variables takes $(N - \lceil H \rceil - 1) \times \lceil H \rceil$ universal gates of $\lceil H \rceil + 1$ inputs, and then we have the final gate with $\lceil H \rceil + 1$ inputs too. Therefore, the cost of this circuit is $(1 + (N - \lceil H \rceil - 1) \times \lceil H \rceil) 2^{\lceil H \rceil + 1}$ cells. Since $0 \le \lceil H \rceil \le N$, this cost is $\le 2^{H + o(N)}$ cells.

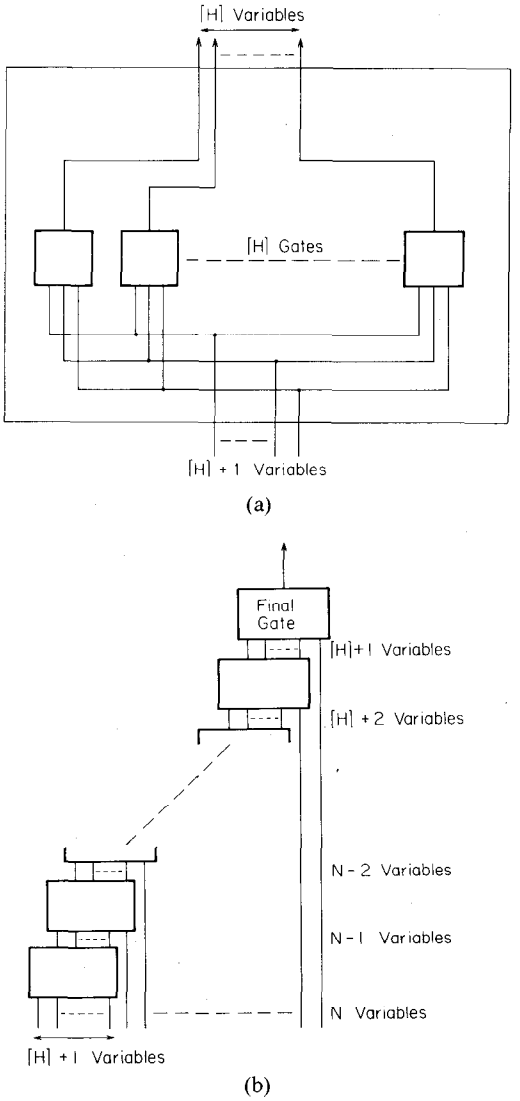

Fig. 4. (a) Compressing $\lceil H \rceil + 1$ variables into $\lceil H \rceil$ variables. (b) Circuit for implementing any function of N variables with entropy H using block in (a).

Normal-Form Implementation: Suppose that $H \ll N$. Partition S into $\sqrt{N/H}$ subsets each of cardinality $\sqrt{HN}$ (approximately). Consider the configuration C that has $\lceil H \rceil$ duplicates of each of these subsets. Each primary gate has $\sqrt{HN}$ inputs, and the secondary gate also has $\lceil H \rceil \times \sqrt{N/H} \approx \sqrt{HN}$ inputs (Fig. 5(a)). Therefore, $\log \log N(C)$ will be approximately $\sqrt{HN}$ (by Proposition 1). Furthermore, C admits any function f of entropy H since the $\lceil H \rceil$ (duplicate) primary gates can be programmed to encode the (relevant) state of their inputs, and the secondary gate can then decide whether $f = 1$ (matched positive states) or $f = 0$ (unmatched positive states, or some non-positive state). If H is not much smaller than N, a configuration C can still be constructed using $\lceil H \rceil$ duplicates of a subset of S having approximately $(N + H)/2$ elements together with the rest of the elements in S appearing as singletons (Fig. 5(b)). This configuration has $\log \log N(C) \approx (N + H)/2$.

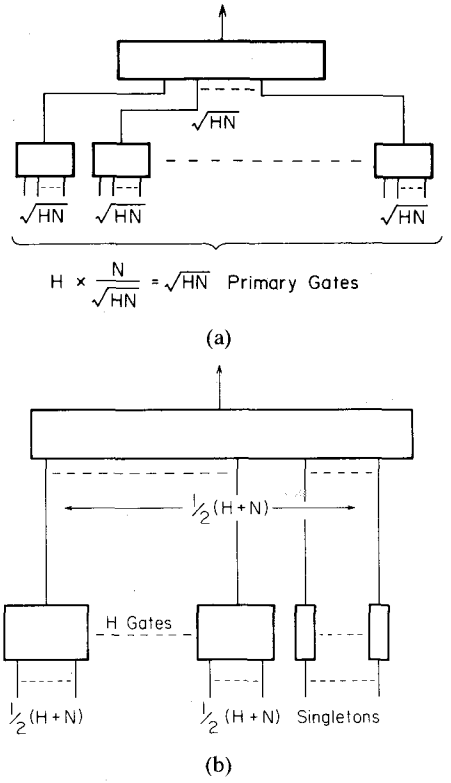

Fig. 5. (a) Normal form for implementing any function of N variables with low entropy H. (b) Normal form for implementing any function of N variables with entropy H.

Notice that in all these implementations, the *same* circuit or normal form can simulate *any* function f of entropy H by proper programming of the gates. This makes them universal structures for all problems of dimensionality N and entropy H. It is appropriate to ask whether we can do substantially better for *every* function. The following theorem proves the contrary for circuits, but the question regarding normal forms is still open [4].

Theorem 2: Let S be a nonempty N-subset of $\boldsymbol{U}$, and let $\boldsymbol{F}_S(H)$ be the set of functions of entropy H (where

$2^H - 1 \leq 2^{N-1}$ is an integer). Define $C_H = \max\{C(f) | f \in F_S(H)\}$ and $K_H = \max\{K(f) | f \in F_S(H)\}$. Then, a) $H - o(N) \leq C_H \leq H + o(N)$; b) $H - o(N) \leq K_H \leq (H + N)/2 + o(N)$.

Proof: We shall use Lemmas 1 and 2 (Appendix I) and Proposition 4. a) The right-hand side (RHS) follows from the circuit implementation described above. For the left-hand side (LHS), we take $H > 3$, without loss of generality (since $C_H \geq 0 \geq 3 - o(N)$). The number of different f's with $2^H - 1$ 1's is $\binom{2^N}{2^H - 1}$, which is more than $\binom{2^N}{2^{H-1}}$. By Lemma 2, this is at least $(2^{-(2+(H-1)/2)}) \times \exp(2^N H(2^{H-1-N}))$. By Lemma 1, this is at least $(2^{-(2+(H-1)/2)}) \times \exp((N - H + 1)2^{H-1})$. Since $H > 3$, this is greater than $\exp((N - H)2^{H-1}) \geq \exp 2^{H-2}$. From Proposition 6, since $\log(8 + N) = o(N)$, at least one of the functions f of entropy H has $C(f) \geq H - o(N)$, and hence $C_H \geq H - o(N)$. b) The RHS follows from the normal form implementation described earlier. The LHS follows by an argument similar to that of part a), using Proposition 4 instead of Proposition 6. This completes the proof. Q.E.D.

For any function f of entropy H, one can construct a program of length $\leq 2^{H+o(N)}$ to generate the truth table $\tau(f)$ by specifying the index of f among all functions of entropy H. This shows that the randomness, like the cost, satisfies $R(f) \leq H(f) + o(N)$. It is also simple to extend Theorem 2 and show that *almost all* functions of entropy H have complexity, cost, and randomness at least $H - o(N)$. This makes deterministic entropy an essential parameter in characterizing a function. Finally, we note that $H(f)$ cannot be bounded by $K(f)$, $C(f)$, or $R(f)$, since simple functions of high entropy exist such as the modulo-two sum.

IV. Approximation

In this section, the complexity results of the previous section are discussed in a probabilistic context. An error-tolerant version of the complexity and cost measures are defined and related to information-theoretic entropy and decision reliability.

Consider the case where the Boolean variables $s_1, \cdots, s_N$ become random variables under some probability measure. Let S be a fixed nonempty N-subset of the universal set U. Define $\boldsymbol{S}$ 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 probability measure on $\boldsymbol{S}$. This measure induces a probability distribution on all Boolean functions f whose support $T(f)$ is a subset of S, that is, these functions become (dependent) random variables under this measure. We shall refer to the pair $(\boldsymbol{S}, p)$ as the *ensemble*.

Suppose that we can afford a nonzero probability of error δ in implementing the function f. It is conceivable that we can reduce the complexity (cost) of f by approximating f by another function g which is less complex (costly) than, but does not *often* differ from, f.

Definition: The *δ-complexity* and *δ-cost* of a Boolean function f, denoted by $K_\delta(f)$ and $C_\delta(f)$, respectively, are defined for $0 < \delta < 1$ by

$$K_\delta(f) = \min\{K(g) | \Pr(f \neq g) \leq \delta\}$$
$$C_\delta(f) = \min\{C(g) | \Pr(f \neq g) \leq \delta\}.$$

Since the minimization domain includes f itself, it follows that $K_\delta(f) \leq K(f)$. Also, it is obvious that for $\delta \geq 1/2$, $K_\delta(f) = 0$ 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 $K_\delta(f)$ significantly less than $K(f)$ for small δ. The same remarks apply to $C_\delta(f)$ and can be extended to a similar definition of $R_\delta(f)$.

A. Low-Entropy Case

We start with a definition that links deterministic entropy to information-theoretic entropy.

Definition: The *δ-entropy* $H_\delta(\boldsymbol{S})$ is defined for $0 < \delta < 1$ by

$$H_\delta(\boldsymbol{S}) = \log \min\{|\boldsymbol{S}_T| \boldsymbol{S}_T \subseteq \boldsymbol{S}, p(\boldsymbol{S}_T) \geq 1 - \delta\}.$$

In other words, H_δ is the minimum number such that $\boldsymbol{S}$ can be partitioned into $\boldsymbol{S}_T \cup \boldsymbol{S}_A$ (T for typical and A for atypical), where $|\boldsymbol{S}_T| \leq 2^{H_\delta}$ and $p(\boldsymbol{S}_A) \leq \delta$. $H_\delta(\boldsymbol{S})$ becomes very significant when the partition is such that $\boldsymbol{S}_T$ has most of the probability (small δ), while $\boldsymbol{S}_A$ has most of the points (small H_δ). In pattern recognition, for example, the set of meaningful images is an exceedingly small subset of the set of all "two-dimensional data arrays."

For many probability measures of interest [28], such as the independent identically distributed s_i, H_δ is asymptotically equal to Shannon's entropy H of the ensemble:

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

We now apply Theorem 2 and relate H_δ to K_δ and C_δ. For any function f, no matter how complex, we argue that $C_\delta(f) \leq H_\delta(\boldsymbol{S}) + o(N)$ and $K_\delta(f) \leq (H_\delta(\boldsymbol{S}) + N)/2 + o(N)$. This is because we can construct a function g satisfying $\Pr(f \neq g) \leq \delta$ and also having $C(g) \leq H_\delta(\boldsymbol{S}) + o(N)$ and $K(g) \leq (H_\delta(\boldsymbol{S}) + N)/2 + o(N)$. To do this, we partition $\boldsymbol{S}$ into $\boldsymbol{S}_T \cup \boldsymbol{S}_A$ according to the definition of H_δ. If f is always 0 or always 1 for the states in $\boldsymbol{S}_T$, we take the approximating function g to be a constant 0 or 1, respectively. Since $\Pr(f \neq g) \leq p(\boldsymbol{S}_A) \leq \delta$ and $K(g) = C(g) = 0$, the result follows. On the other hand, if f assumes both values 0 and 1 for the states in $\boldsymbol{S}_T$, then define the function g to be equal to f for the states in $\boldsymbol{S}_T$ and equal to 0 for the states in $\boldsymbol{S}_A$. Again, $\Pr(f \neq g) \leq p(\boldsymbol{S}_A) \leq \delta$. Furthermore, g assumes the value 1 in at most

$2^{H_\delta} - 1$ states in $\boldsymbol{S}$. Hence the deterministic entropy of g is at most H_δ and the result follows from Theorem 2.

B. High-Entropy Case

The reliable low-cost approximation of all functions in the low-entropy case may persuade us to look for a good approximation of any function in general. Conceivably, even in the maximum-entropy case (uniform probability measure), some other technique for constructing g will work. We will show that this is not the case.

Definition: A δ-error pattern e is any Boolean function satisfying $\Pr(e = 1) \leq \delta$.

Each δ-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 at most δ). The following theorem describes how approximation affects the complexity and cost of most functions.

Theorem 3: Let the probability measure on $\boldsymbol{S}$ be uniform. Given $\epsilon > 0$ and $0 < \delta < 1$, the following hold. a) If $\delta \geq 1/2$, then $K_\delta(f) = 0$ for any function f. b) If $\delta < 1/2$, a positive integer N_0 (function of ϵ and δ) exists such that for $N \geq N_0$, the fraction of functions f having $K_\delta(f) \leq K(f) - \epsilon N$ is less than ϵ. c) The same results apply to the δ-cost function $C_\delta(f)$.

Proof: We shall use Lemma 2 (Appendix I) and Propositions 4 and 6. a) One of the two constant functions $g = 0$ or $g = 1$ must be in the minimization domain. Since $K(g) = 0$ for both functions, the result follows.

b) By Proposition 4, we estimate the number of functions g of complexity $K(g) \leq (1 - \epsilon)N$ to be at most $2^{2^{N(1-(\epsilon/2))}}$. 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 $\delta 2^N \times \exp(2^N H(\delta))$ where $H(\delta) < 1$ is the uncertainty function (Appendix I) evaluated at $\delta < 1/2$. Hence the number of functions f that can be approximated by some function g of complexity $K(g) \leq (1 - \epsilon)N$ is at most $(2^{2^{N(1-(\epsilon/2))}})(\delta 2^N \times \exp(2^N H(\delta)))$, which can be made less than $\epsilon 2^{2^N}$ by taking N large enough.

c) The argument is similar to parts a) and b) using Proposition 6 instead of Proposition 4. This completes the proof. Q.E.D.

This theorem says that most functions have approximately 2^{N-1} 1's and 2^{N-1} 0's scattered in the Karnaugh map, and no way exists to reduce the complexity or the cost significantly by placing "*don't care's*" in less than half the blocks of the map. For these functions, it is not worth saving when we want to implement them. Although Theorem 3 assumes uniform probability distribution, a similar statement can be proved for some nonuniform distributions by considering only the typical blocks in the Karnaugh map.

V. False Entropy

In a typical pattern recognition problem, a point in the ensemble $\boldsymbol{S}$ (an image, or a binary matrix) is given, and it is required to decide whether or not this image belongs to a certain class. The optimal classification decision D is a Boolean function of the Boolean variables in S (the pixels). Typically, the entropy H_δ is much smaller than N, and the reliable implementation cost of D is within $H_\delta \pm o(N)$. Therefore, the entropy of $\boldsymbol{S}$ is crucial to the cost, and one should use whatever information may be available to *reduce* the entropy.

Preprocessing procedures to reduce the entropy in pattern recognition include segmentation and normalization. These procedures can be formalized in terms of the average mutual information and the entropy.

Definition: An *entropy-reduction procedure* is a mapping from $\boldsymbol{S}$ to $\hat{\boldsymbol{S}}$ such that $I(\hat{\boldsymbol{S}}; D) = I(\boldsymbol{S}; D)$ and $H(\hat{\boldsymbol{S}}) < H(\boldsymbol{S})$.

In words, we retain all the information relevant to the decision but discard some of the irrelevant variations in the ensemble. This step uses the properties of the pattern which are known to us to get rid of the *false* entropy, that is, the variations in the pattern which are not random. For example, all images which are rotated versions of one another usually belong to the same class. When we normalize an image such that its contents have a specific orientation, we get rid of the false entropy associated with rotational variations. Such a procedure usually takes linear or polynomial computation time, but it reduces the computation demand tremendously because it decrements the *exponent* in the total cost.

False entropy is not an absolute measure, it depends on what is considered to be a regularity. It is possible to formalize the concept based on universal Turing machines. The randomness of a function of entropy H can, in principle, be as high as $\max\{R(g) | H(g) = H\}$. If it is less, the difference will be due to the (partial) structure or regularity of the function.

Definition: Let S be a fixed N-subset of $\boldsymbol{U}$. The *false entropy* of a function f, whose support is a subset of S, is defined by

$$\Delta(f) = \max\{R(g) | T(g) \subseteq S, H(g) = H(f)\} - R(f).$$

The units of $\Delta(f)$ are bits.

Notice that $\max\{R(g) | T(g) \subseteq S, H(g) = H(f)\}$ is the maximum randomness (algorithmic information) for this level of entropy (combinatorial information). This maximum is close to $R(g)$ of most functions g of entropy $= H(f)$ and is approximately equal to $H(f)$. Hence $\Delta(f) \approx 0$ for most functions. However, for highly structured functions of maximum entropy such as the modulo-two sum, practically all the entropy is false; $\Delta(f) \approx N$. In practical problems, preprocessing steps take care of the partial structure in the problem. The resulting ensemble is then expected to meet the bounds of Theorems 1, 2, and 3, and any system that will do the recognition task reliably will have to meet the cost requirements. A pattern recognition

system that solves problems with entropy H must cost the order of 2^H cells.

ACKNOWLEDGMENT

I gratefully acknowledge Dr. D. Psaltis, who contributed significantly to this work. I also wish to thank Drs. R. McEliece, C. Mead, and E. Posner for their help.

APPENDIX I
LEMMAS 1, 2, AND 3

The following general (and not particularly strong) lemmas are used to prove the main propositions and theorems.

The uncertainty (entropy) function [24] is defined by $H(x) = x\log(1/x) + (1-x)\log 1/(1-x)$ for $0 < x < 1$ and $H(0) = H(1) = 0$. The following lemma estimates $H(x)$.

Lemma 1: For $0 < x < 1$, we have $x\log(1/x) < H(x) < x(2 + \log(1/x))$.

Proof: The LHS follows from the positivity of $(1-x)\log 1/(1-x)$. For the RHS, we estimate this term. $(1-x)\log 1/(1-x) = (1-x)/\ln 2 \ln 1/(1-x)$, where ln denotes the natural logarithm. Since ln 2 is greater than 0.5, we have the overestimate $2(1-x)\ln 1/(1-x) = 2(1-x)(x + (x^2/2) + (x^3/3) + \cdots)$. 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 $\alpha\sqrt{n}(n/e)^n$, where α is between 2 and e (converging to $\sqrt{2\pi}$). The following lemma uses this formula to estimate $\binom{n}{r}$. Similar estimates are found in [20, app. A].

Lemma 2: Let $\binom{n}{r} = n!/r!(n-r)!$. a) If $0 < r < n$, then $(1/4\sqrt{r})2^{nH(r/n)} \le \binom{n}{r} \le 2^{nH(r/n)}$. b) If $0 < t < n/2$, then $(1/4\sqrt{t})2^{nH(t/n)} \le \sum_{r=0}^{t}\binom{n}{r} \le t2^{nH(t/n)}$.

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 check that $\alpha_1/\alpha_2\alpha_3$ is between $1/4$ and $1/\sqrt{2}$. Also, $(n/r(n-r))^{1/2}$ is between $1/\sqrt{r}$ and $\sqrt{2}$. Therefore, their product is between $1/4\sqrt{r}$ and 1. We now evaluate $((n/e)^n/(r/e)^r((n-r)/e)^{n-r})$. This can be rearranged as $((r/n)^r((n-r)/n)^{n-r})^{-1}$. Taking the logarithm, we get $nH(r/n)$ 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 overestimates each term by the last term (since $0 < t < n/2$) and then applies the RHS in b). This completes the proof. Q.E.D.

The following lemma estimates the number of different multisets of a special class.

Lemma 3: Let ρ_M be the number of different multisets $X_M = \langle n_1, \cdots, n_Q\rangle$ where the n_i 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, except $X_{M+1} = \langle M+1\rangle$. 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 $\langle M, M\rangle$ because at most a single 1, a single $2, \cdots$, a single $M-1$ are 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, which proves the claim. Hence ρ_{M+1} is at most $1 + (\rho_M(\rho_M+1)/2)$. 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 overestimating ρ_1 and ρ_2 by 2^{2^1} and 2^{2^2}, respectively. Q.E.D.

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

APPENDIX II
PROOFS OF PROPOSITIONS 1, 2, AND 3

These propositions are concerned with the properties of normal-form input configurations.

Proposition 1: Let $C = \langle S_1, \cdots, S_L\rangle$ 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) $d(C) \le \log\log N(C) \le \max(l(C), d(C)) + \log(\max(l(C), d(C)) + 1)$; b) $\sqrt{r(C)} \le \log\log N(C) \le r(C)$.

Proof: We first maintain that $\log\log N(C)$ exists (and is nonnegative) by observing that $N(C) \ge 2$ since all configurations admit the two constant functions (including the empty configuration by definition). All statements are trivial for the empty configuration, and now assume that C is nonempty. a) Let S_r be a component of maximal cardinality in the configuration. By definition, $|S_r| = d(C)$. Now C can simulate at least the $2^{2^{d(C)}}$ different functions whose support is a subset of S_r. 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 simulated by the primary gate of S_i is $2^{2^{|S_i|}}$ and the number of mappings that can be simulated by the secondary gate in terms of its inputs is $2^{2^{l(C)}}$. Therefore, $N(C)$ is at most $\exp(\sum_{i=1}^{l(C)} 2^{|S_i|} + 2^{l(C)})$. We increase this number by substituting for each $|S_i|$ and for $l(C)$ by $M = \max(l(C), d(C))$. This gives $N(C) \le 2^{(M+1)2^M}$, 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 $C^* = \langle S_1^*, \cdots, S_{L^*}^*\rangle$ be the configuration defined by $S_1^* = S_1$ and $S_i^* = S_i \setminus \bigcup_{j=1}^{i-1} S_j$ (if nonempty; otherwise, skip and relabel) for $i = 2, \cdots, L^*$ $(L^* \le L)$. We shall prove the LHS inequality in b) for C^*. If $d(C^*) \ge \sqrt{r(C^*)}$, we are done (LHS inequality in a)), so we assume that $d(C^*) < \sqrt{r(C^*)}$. However, $l(C^*)d(C^*) \ge r(C^*)$ (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^* are disjoint and nonempty, C^* can simulate at least $\exp 2^{\sqrt{r(C^*)}}$ functions (those simulated 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 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, $N(C) \ge N(C^*)$ and the LHS inequality in b) follows. This completes the proof. Q.E.D.

Proposition 2: If a configuration $C = \langle S_1, \cdots, S_L \rangle$ is not redundant, then for all subsets λ of $\{1, \cdots, L\}$, the following condition holds:

$$\left| \bigcup_{i \in \lambda} S_i \right| \geq |\lambda|.$$

Proof: We shall use Hall's theorem [22] 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. Suppose that the condition does not hold. Then a subset λ of $\{1, \cdots, L\}$ exists for which $|\cup_{i \in \lambda} S_i| < |\lambda|$. Let λ be a minimal subset satisfying this condition. Let j be any element of λ (λ is nonempty since $|\lambda| > 0$). Let $\lambda_j = \lambda \setminus \{j\}$. Since λ is minimal, we have $|\cup_{i \in \lambda_j} S_i| \geq |\lambda_j|$. However, $|\lambda_j| = |\lambda| - 1$ and $|\lambda| > |\cup_{i \in \lambda_j} S_i| \geq |\cup_{i \in \lambda_j} S_i|$. This forces both $\cup_{i \in \lambda} S_i$ and $\cup_{i \in \lambda_j} S_i$ to have cardinality $|\lambda| - 1$ ($= |\lambda_j|$) and forces S_j to be contained in $\cup_{i \in \lambda_j} S_i$. 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 $\cup_{i \in \lambda_j} S_i$. Hence, the SDR covers all the elements of the S_i with i in λ_j. Now S_j is a subset of $\cup_{i \in \lambda_j} S_i$, and we can omit it without diminishing $\boldsymbol{F}(C)$ 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 $\cup_{i \in \lambda} S_i$. Hence C is redundant and the proof is complete. Q.E.D.

Proposition 3: a) If C is a configuration with $l(C) > r(C)$, then a configuration C^* with $l(C^*) \leq r(C^*)$ exists which is equivalent to C. b) Let S be an N-subset of the universal set $\boldsymbol{U}$. At most 2^{N^2} possible values for $\boldsymbol{F}(C)$ exist over all configurations C whose support $T(C)$ is a subset of S.

Proof: a) C must be redundant since $l(C) > r(C)$ means that the condition of Proposition 2 does not hold for C with $\lambda = \{1, \cdots, l(C)\}$. We keep omitting the unnecessary S_j 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 at most N such components exist. Therefore, the total number of different configurations is at most $\prod_{i=1}^{N} 2^N = 2^{N^2}$. Since the different $\boldsymbol{F}(C)$'s can be at most that many, the proof follows. Q.E.D.

Appendix III
Proofs of Propositions 4, 5, and 6

These propositions are concerned with the properties of the complexity and cost measures.

Proposition 4: Let $\boldsymbol{F}_S$ be the set of all Boolean functions f whose support is a subset of a nonempty N-set S. Define $N_K = |\{f \in \boldsymbol{F}_S | K(f) \leq K\}|$. For $0 \leq K \leq N$, we have

$$K - 1 \leq \log\log N_K \leq K + 2\log N.$$

Proof: We shall use Proposition 3. Obviously, all functions f depending on at most $\lfloor K \rfloor$ variables have $K(f) \leq K$. There are at least $2^{2^{K-1}}$ such functions in $\boldsymbol{F}_S$ and the LHS inequality follows by taking the logarithm twice. From Proposition 3, at most 2^{N^2} different $\boldsymbol{F}(C)$'s exist with $T(C)$ a subset of S. Each function f of complexity $K(f) \leq K$ must belong to an $\boldsymbol{F}(C)$ whose cardinality is at most 2^{2^K}. Therefore, $N_K \leq 2^{N^2} 2^{2^K}$ and the RHS inequality follows by taking the logarithm twice. This completes the proof. Q.E.D.

Proposition 5: Let f be a Boolean function of complexity $K(f) = K$. Then, a) any normal form implementation of f costs at least 2^K cells, and b) a normal form implementation of f exists which costs at most $2^{K + \log(1+K)}$ cells.

Proof: We shall use Propositions 1 and 2. a) Let $C = \langle S_1, \cdots, S_L \rangle$ 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 $N(C) \leq \exp(2^L + \sum_{i=1}^{L} 2^{n_i})$ (same argument as in Proposition 1). By taking the logarithm, we get part a).

b) Let C be a minimal configuration (with respect to $N(C)$) that admits f. If C is redundant, omit unnecessary S_i until the condition of Proposition 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 rearrangement. Q.E.D.

Proposition 6: Let $\boldsymbol{F}_S$ be the set of all Boolean functions f whose support is a subset of a nonempty N-set S. Define $\hat{N}_K = |\{f \in \boldsymbol{F}_S | C(f) \leq K\}|$. For $0 \leq K \leq N$, we have

$$K - 1 \leq \log\log \hat{N}_K \leq K + \log(8 + N).$$

Proof: Since $\lfloor K \rfloor \leq N$, there are at least $\exp 2^{\lfloor K \rfloor}$ functions that can be simulated by a single universal gate of $\lfloor K \rfloor$ inputs. The LHS follows since $\lfloor K \rfloor > K - 1$. For the RHS, we can take $K \geq 1$ since the statement is clear for $K < 1$ (constant functions only). Let $M = \lceil K \rceil$. We overestimate $\hat{N}_K$ by $\hat{N}_M$. To do this, we shall estimate the number of different ways we can choose a collection of gates given the total cost of 2^M cells, the number of circuits that can be formed using a given collection of gates, and the number of Boolean functions that can be simulated on a given circuit. We restrict the gates to have a positive number of inputs, since zero-input gates contribute only the constant functions which can be simulated otherwise. Restricting the cost to be *exactly* 2^M cells is justified by adding 1-input gates (all costs involved are even) without using their outputs.

1) A collection of gates is isomorphic to a multiset of numbers (the number of inputs in each gate). The number of multisets $\langle n_1, \cdots, n_Q \rangle$ with positive n_i 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 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 collection of gates with $n_1, \cdots, n_Q$ inputs, 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 functions because the gates are universal). By Lemma 2, substituting for each term in the product, this number is at most $\exp\left((Q+N)\sum_{i=1}^{Q} H(n_i/Q+N)\right)$ where $H(x)$ is the uncertainty function defined in Appendix I. Substituting for each H using Lemma 1, this number is at most $\exp\left((Q+N)\sum_{i=1}^{Q}(n_i/(Q+N))(2+\log(Q+N)/n_i)\right)$. 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 $\sum_{i=1}^{Q} = 2^M$, the maximum value of $\sum_{i=2}^{Q} n_i$ occurs when all the n_i are 1's and is 2^{M-1}. Hence we get the following overestimate: $\exp((1 + (1/2)\log(2^{M-1} + N))2^M)$.

3) For each of the foregoing circuits, there are $\prod_{i=1}^{Q} 2^{2^{n_i}}$ ways to program the universal gates, and for each of these we have at most Q implemented functions out of the Q gates. Hence 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, and 3, the number of functions f whose support is a subset of S and can be implemented within cost 2^M cells is at most the product of the three estimates, namely, $\exp\big((4 + (1/2)\log(2^{M-1} + N))2^M\big)$. Since $M < K + 1$ and also $M \leq N$, this is at most $\exp\big((8 + \log(2^{N-1} + N))2^K\big)$. Since $N \leq 2^{N-1}$ (N is an integer), this is at most $\exp\big((8 + \log(2^{N-1} + 2^{N-1}))2^K\big)$, which reduces to $\exp\exp(K + \log(8 + N))$, and the proof follows by taking the logarithm twice. Q.E.D.

References

[1] H. Abelson, "Towards a theory of local and global in computation," *Theoret. Comput. Sci.*, vol. 6, pp. 41–67, 1978.

[2] H. Abelson *et al.*, "Compositional complexity of Boolean functions," *Discrete Appl. Math.*, vol. 4, pp. 1–10, 1982.

[3] Y. Abu-Mostafa, "Complexity of information extraction," Ph.D. dissertation, California Inst. Technol., Pasadena, 1983.

[4] ——, "Pointwise universality of the normal form," in *Fundamental Problems in Communication and Computation*, B. Gopinath and T. Cover, Eds. New York: Springer-Verlag, 1985.

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

[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. Inform. Theory*, vol. IT-20, pp. 10–15, 1974.

[8] ——, "A theory of program size formally identical to information theory," *J. Ass. Comput. Mach.*, vol. 22, pp. 329–340, 1975.

[9] R. Duda and P. Hart, *Pattern Classification and Scene Analysis*. New York: Wiley-Interscience, 1973.

[10] R. Gallager, *Information Theory and Reliable Communication*. New York: Wiley, 1968.

[11] D. Knuth, "An almost linear recurrence," *Fibonacci Quart.*, vol. 4, pp. 117–128, 1966.

[12] Z. Kohavi, *Switching and Finite Automata Theory*, 2nd ed. New York: McGraw-Hill, 1978.

[13] A. Kolmogorov, "Three approaches for defining the concept of information quantity," *Inform. Transmission*, vol. 1, pp. 3–11, 1965.

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

[15] O. Lupanov, "Complexity of formula realization of logical algebra," *Probl. Cybern.*, vol. 3, pp. 782–811, 1960.

[16] P. Martin-Löf, "The definition of random sequences," *Inform. Contr.*, vol. 9, pp. 602–619, 1966.

[17] R. McEliece, *The Theory of Information and Coding*. Reading, MA: Addison-Wesley, 1977.

[18] C. Mead and L. Conway, *Introduction to VLSI Systems*. Reading, MA: Addison-Wesley, 1980.

[19] J. Peatman, *Digital Hardware Design*. New York: McGraw-Hill, 1980.

[20] W. Peterson and E. Weldon, *Error Correcting Codes*. Cambridge, MA: MIT Press, 1972.

[21] N. Pippenger, "Information theory and the complexity of Boolean functions," *Math. Syst. Theory*, vol. 10, pp. 129–167, 1977.

[22] H. Ryser, *Combinatorial Mathematics*. Math. Ass. Amer., 1963.

[23] J. Savage, *The Complexity of Computing*. New York: Wiley-Interscience, 1976.

[24] C. Shannon, "A mathematical theory of communication," *Bell. Syst. Tech. J.*, vol. 27, pp. 379–423, 1948.

[25] ——, "The synthesis of two-terminal switching circuits," *Bell Syst. Tech. J.*, vol. 28, pp. 59–98, 1949.

[26] S. Skyum and L. Valiant, "A complexity theory based on Boolean algebra," in *Proc. 22nd IEEE Symp. Foundations of Computer Science*, 1981, pp. 244–253.

[27] A. Turing, "On computable numbers with an application to the Entscheidungsproblem," in *Proc. London Math. Soc.*, vol. 42, pp. 230–265, 1936.

[28] J. Wolfowitz, *Coding Theorems of Information Theory*, 3rd ed. New York: Springer-Verlag, 1978.