CASE FILE
COPY

CENTER FOR

COMPUTER AND INFORMATION SCIENCES

AND

DIVISION OF ENGINEERING

N71-35688

COMPUTATIONAL WORK AND TIME ON FINITE MACHINES*

by

J. E. Savage
Brown University

June, 1971

BROWN UNIVERSITY

Providence

Rhode Island

N71-35688

COMPUTATIONAL WORK AND TIME ON FINITE MACHINES*

by

J. E. Savage
Brown University

June, 1971

*This work was completed in part at Brown University and in part at the Jet Propulsion Laboratory, California Institute of Technology, Pasadena, Calif. The Brown University portion has been supported by NSF under Grant GK-13162 and by NASA under Grants NGR 40-002-082 and NGR 40-002-090. The JPL portion of this work was supported by NASA under Contract NAS 7-100.

COMPUTATIONAL WORK AND TIME ON FINITE MACHINES

by

J. E. Savage

Division of Engineering

and

Center for Computer and Information Sciences

Brown University

Providence, R. I. 02912

ABSTRACT

Measures of the computational work and computational delay required by machines to compute functions are given. Exchange inequalities are developed for random access, tape and drum machines to show that product inequalities between storage and time, number of drum tracks and time, number of bits in an address and time, etc., must be satisfied to compute finite functions on bounded machines. Criterea for the design and use of general purpose computers are developed and applications of the exchange inequalities are made to the language recognition problem.

1. Introduction

It is a fact, as Minsky notes [1], that very little is known about "possible exchanges between time and memory, tradeoffs between time and program complexity," and other important parameters of computation. While exchange relations are not the only form in which basic information about computational processes could be expressed, they could be one important representation of such information. In fact, Minsky has said that "the recognition of exchanges is often the conception of a science, if quantifying them is its birth" [1]. In this setting, this paper contributes to the conception and quantification of computer science by developing many exchange inequalities involving storage, time and other important parameters of computation.

In this paper we examine the computation of finite functions (functions whose domain is finite) on finite machines. In Section 2, two machine models are examined, sequential machines which are assumed to execute a fixed number of cycles and autonomous machines, which receive no external inputs during a computation and which execute a number of cycles dependent on their initial states. We show that for every autonomous machine there exists a sequential machine which computes the same function.

Two complexity measures are defined in Section 3 for finite functions. These are combinational complexity and time complexity and we assume that functions are computed by "straight-line" algorithms, that is, algorithms with no loops and limited branching. Two sets of basic inequalities are developed relating the combinational complexity and time complexity of sequential machines and the number of cycles which they execute to the combinational complexity and time complexity of the functions which they compute. These inequalities provide a natural definition of computational work and computational delay and the inequalities are interpreted as requiring that a minimum amount of work be done

and a minimum delay be experienced by machines which compute functions of given complexity. The time rate at which computational work is done by a machine is defined as its computing power and this measure finds application in later sections. Computational work has been previously introduced in the study of decoding for error correcting codes [2,3] and time complexity has been used in connection with studies of the time required to multiply and add [4,5].

The properties of the two complexity measures are surveyed in Section 4. Included in this survey are tests which can be applied to functions which when satisfied provide lower bounds on their complexity. Analytical methods which prove useful in later sections are also developed here.

Computational efficiency is discussed briefly in Section 5 in terms of the computational work and computational delay measures. It is shown that simple functions can be computed efficiently by many different sequential machines and that most Boolean functions are computable with modest efficiency on many machines also.

In Section 5, tight bounds are developed on the combinational complexity and time complexity of random access, tape and drum (or disk) storage units. These bounds are used in Section 6 together with the inequalities of Section 2 to provide exchange inequalities for general purpose computers using storage of these three types. It is shown, for example, that the computation of complex functions on random access or tape machines requires that the product of the storage capacity and execution time of these machines be large. The analogous result for drum machines shows that the product of the number of tracks and execution time be large, which is a much weaker inequality. Inequalities are also developed for on and off-line multitape machines. Among other things these inequalities show that the execution time required to compute a function of combinational complexity C must grow linearly with C on drum machines,

at least as fast as $\sqrt{C}$ on tape machines and it can also be shown that functions can be computed in time independent of their complexities on random access machines. These results suggest a hierarchy of storage units.

On the basis of computing power two rules of thumb are given for the use and acquisition of storage units. We also carry out a calculation of the computing powers of storage devices in a typical computing system to argue that bulk auxiliary storage can materially increase the execution time of programs that are forced to interact heavily with it.

Section 8 uses the language recognition problem as one illustration of the application of computational work and the inequalities derived in earlier sections. Bounds are derived on the work required to recognize languages in the Chomsky hierarchy and a language is given to suggest that the Chomsky hierarchy and a hierarchy based on the work required to recognize languages do not coincide.

Finally, in Section 9 inequalities which are developed in Section 2 are used to bound the complexity of the most complex function which can be computed by a quantum-mechanical computer with energy E in t seconds. This result is used to show that these computers cannot compute most Boolean function of 160 or more variables in one hour with a kilowatt of power.

2. Computation on Finite Machines

In this section, we introduce two models for computing machines and we define the finite functions which they compute. The first model is called a sequential machine and it executes a fixed number of cycles, The second model is called an autonomous machine and it executes a number of cycles which depends on its initial state. These models have been chosen because they are representative of the ways in which computing machines are used today. The automomous machine models stored programmed computers which act upon programs and data.

Definition 1 A (Moore) sequential machine is a sextuple $S = < S, I, \delta, \lambda, 0; T >$ where S, I and 0 are the finite state set, input alphabet and output alphabet, of the machine, respectively, δ: $S \times I \to S$ is the state transition function and λ: $S \to 0$ is the output function. The machine produces T outputs including that determined by its initial state and it is said that S executes T cycles. Let I^n be the n-fold cartesian product of I and let $(y_1, y_2, --, y_n) \varepsilon I^n$, $s \varepsilon S$. Then, the extensions $\delta^{(n)}$: $S \times I^n \to S$, $\lambda^{(n)}$: $S \times I^{n-1} \to 0$ of δ and λ are defined by $\delta^{(1)} = \delta$, $\lambda^{(1)} = \lambda$ and for $n = 2, 3$ ---

$$\delta^{(n)}(s; y_1, y_2, --, y_n) = \delta(\delta^{(n-1)}(s; y_1, y_2, --, y_{n-1}), y_n)$$

$$\lambda^{(n)}(s; y_1, y_2, ---, y_{n-1}) = \lambda(\delta^{(n-1)}(s; y_1, --, y_{n-1}))$$

Definition 2 An autonomous machine is a quintuple $A = < A, \delta, \lambda, P, 0 >$ where A and 0 are the finite state set and output alphabet, respectively, δ: $A \to A$ is the state transition function, λ: $A \to 0$ is the output function and P: $A \to \{0,1\}$ is the print function. The state set has distinguished states

$p \varepsilon A$ called print states and P is the characteristic function of a set F of print states, i.e. for $a \varepsilon A$

$$P(a) = \begin{cases} 1 & a \varepsilon F \\ 0 & a \notin F \end{cases}$$

The extensions $\delta^{(n)}$: $A \to A$, $\lambda^{(n)}$: $A \to 0$, $n = 1, 2, 3---$, of the functions δ and λ are defined by $\delta^{(1)} = \delta$, $\lambda^{(1)} = \lambda$ and

$$\delta^{(n)}(a) = \delta(\delta^{(n-1)}(a))$$

$$\lambda^{(n)}(a) = \lambda(\delta^{(n-1)}(a))$$

for $a \varepsilon A$. The function f_A: $A \to 0$ is then defined for $a \varepsilon A$ by

$$f_A(a) = \lambda^{(n)}(a)$$

where $n \geq 1$ is the smallest integer such that $\delta^{(n)}(a) \varepsilon F$. If no such integer exists, then $f_A(a)$ is undefined. Then, it is said that A computes the partial function f_A.

For technical reasons, we want to limit our attention to sequential machines. The following lemma is important because it demonstrates that every autonomous machine which eventually prints can be replaced by a sequential machine. The proof is constructive and it exhibits a simple machine which when adjoined to the autonomous machine creates a sequential machine. We show later that this simple machine, adds in a minor way to the complexity of the autonomous machine.

Lemma 1 Let $A = < A, \delta, \lambda, P, 0 >$ be an autonomous machine which computes f_A and which reaches a print state when started in any state. Then, there exists an integer T_A and a sequential machine $S_A = < S_A, ---, \delta_A, \lambda_A, 0; T_A >$

such that f_A is computed by S_A.

Proof Let $S_A = A \times \{0,1\} \times 0$ and let $(a, b, z) \in S_A$. If A is started in state a', S_A is started in state $(a', 0, \lambda(a'))$ if $P(a') = 0$ and in state $(a', 1, \lambda(a'))$, otherwise. Then, we define δ_A: $S_A \to S_A$ and λ_A: $S_A \to 0$ by

$$\delta_A(a, b, z) = (\delta(a), b', z')$$

$$\lambda_A(a, b, z) = z'$$

where

$$b' = \begin{cases} 1 & b = 1, \text{ or } P(a) = 1 \\ 0 & \text{otherwise} \end{cases}$$

$$z' = \begin{cases} z & b = 1 \text{ or } b = 0, P(a) = 0 \\ \lambda(a) & b = 0, P(a) = 1 \end{cases}$$

The first component of (a,b,z) records the present state of A. The second component assumes value 1 when A enters a print state and the third component retains the output of A at that time. Once $b = 1$, the (b, z) pair remains unchanged thereafter. Since A executes at most T_A cycles, the machine S_A will produce as its T_A th output the value of f_A. Thus,

$$\lambda_A^{(n)}(a,b,z) = f_A(a) \text{ for } n = T_A \text{ and all } a \text{ in } A.$$

Q.E.D.

In the proof above the second and third components of the state vector (a,b,z) represent a machine which has been adjoined to A to form S_A. Since every autonoumous machine can be replaced by a sequential machine, we assume in our discussions below, that all machines are sequential machines which execute fixed numbers of cycles unless explicitly stated otherwise.

Consider next the interconnection of a set of sequential machines $\{S_1, S_2, --, S_L,\}$ each of which has a clock and makes one state transition every clock cycle. Let the clock cycle lengths of the machines be $\tau_1, \tau_2, --, \tau_L$, respectively, and assume that each length is a multiple of some length τ_0. Since the clock cycle lengths may be different, it is important to know when a machine makes a state transition. We assume that a state transition in a machine occurs instantaneously at the end of its clock cycle and that the transition is determined by the state of the machine during that cycle and by the value of the input to the machine just preceding the end of the cycle. The last assumption is important because signals on input lines of a machine in an interconnected set of machines may change during one of its cycles. Note that a change in a machine output occurs only at the end of a cycle and if a machine completes the number of cycles it executes, then its state and output remain constant thereafter.

A sequential machine may have several input lines and several output lines. The input and output alphabets represent sets of possible configurations of signals on these lines.

An <u>interconnection rule</u> $\propto$ for machines $S_1, S_2, ---, S_L$ is a partition of the set of input and output lines of these machines into disjoint classes subject to the restriction that no two outputs are in the same class. Classes which contain no output lines are called <u>external inputs</u> and the remaining classes are called <u>internal inputs</u>.

We assume that each of the machines $S_1, --, S_L$ begins to execute simultaneously and that they execute $T_1, --, T_L$ cycles, respectively. Then, the interconnection rule $\propto$ and the timing information determine which outputs and external inputs in time are used by any given machine to make state transitions. When this information is available, the functions computed by $S_1, --, S_L$ can

be composed to form the functions computed by the collection of machines. We do not explicitly state the functions computed here because the notation required to do so would obscure the result.

3. Measures of Functional Complexity

In this section, we define "straight-line" or Ω-algorithms and we show the equivalence of the graph of such algorithms and combinational machines. We also define two measures of functional complexity, combinational complexity and time complexity, and we establish two inequalities involving them. These inequalities will be used repeatedly to derive many of the major conclusions of this paper.

Definition 3 Let Ω be a finite set of primitive operations $h_i \in \Omega$, $h_i: \Sigma^{n_i} \to \Sigma$ where Σ is some finite set. Let $\Gamma = \Sigma \cup \{X_1, X_2, --, X_N\}$ be the data set where each X_i is a variable over Σ. Then, a K-step Ω-algorithm with data set Γ is a K-tuple $\beta = (\beta_1, \beta_2, --, \beta_K)$ where at the kth step $\beta_k \in \Gamma$ or $\beta_k = (h_i; k_1, --, k_{n_i})$ for some $h_i \in \Omega$ and $k_j < k$, $1 \leq j \leq n_i$. If $\beta_k \in \Gamma$ we associate with it the function $\overline{\beta_k}$ which is either a constant or a function which identifies a variable X_i. If $\beta_k = (h_i; k_1, --, k_{n_i})$ we associate the recursively defined function $\overline{\beta_k} = h_i(\overline{\beta}_{k_1}, ---, \overline{\beta}_{k_{n_i}})$. The Ω-algorithm β is said to compute the functions $\overline{\beta}_{m_1}, \overline{\beta}_{m_2}, --\overline{\beta}_{m_q}$ where $\beta_{m_1}, \beta_{m_2}, --, \beta_{m_q}$ are any of the non-data steps of β. A set Ω is said to be complete if every function $f: \Sigma^n \to \Sigma$ can be realized by some Ω-algorithm (For example, the set Ω consisting of the 2-input AND, 2-input OR and the NOT functions is complete for Boolean functions.) We assume that Ω is complete.

To each Ω-algorithm we associate a directed, acyclic graph as indicated below.

Definition 4 The graph G of a K-step Ω-algorithm β is a set of nodes which are in a 1-1 correspondence with steps of β and a set of labeled directed edges

between nodes. If $\beta_k \epsilon \Gamma$, the corresponding node of G has no edges directed into it and is called a source node. If $\beta_k = (h_i; k_1, --, k_{n_i})$, the node corresponding to it has n_i edges directed into it from nodes corresponding to steps $\beta_{k_1}, --, \beta_{k_{n_i}}$. Furthermore, these edges are labeled to retain the order of the steps as arguments of h_i. A node which has no edges directed from it is called a terminal node.

In this paper, we limit our attention to Ω-algorithms where $\Sigma = \Sigma_2 = \{0,1\}$. Therefore, the primitives $h_i: \Sigma_2^{n_i} \to \Sigma_2$ are Boolean functions and the graph of an Ω-algorithm is commonly known as a combinational machine or a switching circuit. There will be very little loss of generality due to this restriction and most results derived will hold for Ω-algorithms over arbitrary but finite sets Σ.

We now state two measures of functional complexity.

Definition 5 The combinational complexity of the set of Boolean functions $\{f_1, f_2, ---, f_r\}$ with respect to the set of primitives Ω, $C_\Omega(f_1, f_2, --, f_r)$, is the smallest number of non-source nodes in any graph of an Ω-algorithm which computes these functions.

Definition 6 The length of an Ω-algorithm is the number of edges on the longest directed path of its graph. The time complexity of the set of Boolean functions $\{f_1, f_2, ---, f_r\}$ with respect to the set of primitives Ω, $D_\Omega(f_1, f_2, ---, f_r)$, is the length of an Ω-algorithm of shortest length which computes these functions.

Combinational complexity should be interpreted as the number of logical operations required to compute a set of functions. Since every logic element introduces some delay into a circuit, time complexity should be interpreted as the time required to compute a set of functions.

The functions computed by Ω-algorithms with Ω a set of Boolean primitives are Boolean. However, the functions computed by sequential machines are not Boolean,

in general. Thus, if the complexity of these functions is to be measured, we must create representations for them in terms of Boolean functions. To do this we first define an encoding of a set B, $h: B \to \Sigma_2^m$ to be a 1-1 map of B into Σ_2^m for some m. If $f: B_1 x \text{--} x B_p \to B_1' x \text{--} x B_q'$ is a non-binary function and if $(b_1, \text{---}, b_p) \varepsilon B_1 x \text{--} x B_p$, $(b_1', \text{--}, b_q') \varepsilon B_1' x \text{--} x B_q'$, then we define

$$f^*: \Sigma_2^{m_1} x \text{--} x \Sigma_2^{m_p} \to \Sigma_2^{m_1'} x \text{--} x \Sigma_2^{m_q'}$$

by

$$f^*(h_1(b_1), \text{---}, h_p(b_p)) = (h_1'(b_1'), \text{---} h_q'(b_q'))$$

where $f(b_1, b_2, \text{--}, b_p) = (b_1', b_2', \text{--}, b_q')$ and h_i is the encoding of B_i and h_j' is the encoding of B_j'.

The definition of the combinational complexity of the functions $f_1, \text{--}, f_r$ is now extended to non-binary functions and is called $C_\Omega^E(f_1, \text{--}, f_r)$. This is defined as the minimum over all encodings of $C_\Omega(f_1^*, \text{--}, f_r^*)$. The definition of time complexity is also extended to non-binary functions and is called $D_\Omega^E(f_1, \text{--}, f_r)$. Unless confusion is likely to arise, we shall use C_Ω for C_Ω^E and D_Ω for D_Ω^E.

We are now prepared to state the major results of this section.

Theorem 1 Let $S_\ell = \langle S_\ell, I_\ell, \delta_\ell, \lambda_\ell, O_\ell; T_\ell \rangle$ for $1 \le \ell \le L$ and let S_ℓ have cycle length τ_ℓ. Let the machines $S_1, S_2, \text{--}, S_L$ be interconnected according to rule α and let this collection of machines compute the functions $f_1, f_2, \text{---}, f_r$. Let E_ℓ be a set of encodings for the domains and ranges of S_ℓ: $S_\ell \times I_\ell \to S_\ell$ and λ_ℓ: $S_\ell \to O_\ell$ subject to the restriction that the same encoding is used for each occurence of S_ℓ. Let $C_\Omega(\delta_\ell, \lambda_\ell; E_\ell)$ be the combinational complexity of δ_ℓ and λ_ℓ with the encodings E_ℓ. Then,

$$C_\Omega(f_1, --, f_r) \leq \min_{E_1---E_L} \sum_{\ell=1}^{L} C_\Omega(\delta_\ell, \lambda_\ell; E_\ell)\, T_\ell$$

where the minimum is over the encodings with the restriction that all machine inputs in the same set of external or internal inputs be assigned the same encodings.

Proof Given an Ω-algorithm which computes δ_ℓ, λ_ℓ with encodings E_ℓ, an Ω-algorithm can be constructed which computes all of the functions computed by S_ℓ. This is done by constructing T_ℓ copies of the Ω-algorithm for $(\delta_\ell, \lambda_\ell)$ and connecting the state output of one copy to the state input of another so that a chain is formed. This algorithm has combinational complexity equal to $C_\Omega(\delta_\ell, \lambda_\ell; E_\ell)T_\ell$.

The interconnection rule and timing information determine which external and internal inputs in time are used by the various machines to make state transitions. We construct Ω-algorithms as indicated above for each of these machines and then form connections as indicated by the interconnection rule and the timing information. These connections do not create any loops and an Ω-algorithm has been created which computes the functions computed by the collection of machines. Note that in making connections, it is assumed that encodings of inputs and outputs which are connected are identical.

The theorem follows by choosing encodings which minimize the bounds on combinational complexity. Q.E.D.

Given a combinational machine which realizes the transition and output function of a sequential machine S, we could realize S as shown in Fig. 1 by connecting the combinational machine to a bank of binary memory cells. Then, a combinational machine equivalent to S can be constructed as shown in Fig. 2 by streching S. It is often helpful to think of machines in these terms.

Corollary Under the conditions of Theorem 1, let $C_\Omega(S_\ell)$ be the minimum over all encodings E_ℓ of $C_\Omega(\delta_\ell, \lambda_\ell; E_\ell)$. Then,

$$C_\Omega(f_1, --, f_r) \leq C_\Omega(S_{\ell_o}) + \min_{E_\ell,\ \ell \neq \ell} \sum_{\substack{\ell=1 \\ \ell \neq \ell_o}}^{L} C_\Omega(\delta_\ell, \lambda_\ell; E_\ell) T_\ell \qquad (2)$$

where the minimum is over all encodings E_ℓ, $\ell \neq \ell_o$, with the restriction that encodings of the external and internal inputs to machines S_ℓ, $\ell \neq \ell_o$ be consistent with the encodings which minimize $C_\Omega(\delta_{\ell_o}, \lambda_{\ell_o}; E_{\ell_o})$.

Proof The inequality is valid because the encodings to the Ω-algorithm constructed above for S_ℓ can be held fixed and the other encodings chosen to minimize the bound. Q.E.D.

Theorem 2 Under the conditions of Theorem 1, let $D_\Omega(\delta_\ell, \lambda_\ell; E_\ell)$ be the time complexity of $\delta_\ell, \lambda_\ell$ with encodings E_ℓ. Let

$$R_\Omega(S_1 --, S_L) = \min_{E_1, ---, E_L} \max_{1 \leq \ell \leq L} \frac{D_\Omega(\delta_\ell, \lambda_\ell; E_\ell)}{\tau_\ell}$$

where the minimum is over all encodings with the restriction that each machine input in the same set of external or internal inputs be assigned the same encoding. Then,

$$D_\Omega(f_1, f_2, --, f_r) \leq 2R_\Omega(S_1, ---, S_L)\ [\max_\ell T_\ell\ \tau_\ell] \qquad (3)$$

Proof The bound applies to the Ω-algorithm which was constructed in the proof of Theorem 1.

Consider the longest path P through the graph of this algorithm and suppose it passes consecutively through the graphs associated with machines

$S_{\ell_1}, S_{\ell_2}, \text{---}, S_{\ell_N}$. Let the path pass through the graph associated with S_{ℓ_j} at an input given to S_{ℓ_j} just prior to its m_{ℓ_j} th state transition and at an output produced by S_{ℓ_j} following its n_{ℓ_j} th transition, $1 \leq j \leq N$. Then, $1 \leq m_{\ell_j} \leq n_{\ell_j} \leq T_{\ell_j} - 1$ and $n_{\ell_j} \tau_{\ell_j} < m_{\ell_{j+1}} \tau_{\ell_{j+1}}$ because if this second condition is not satisfied, the $m_{\ell_{j+1}}$ th transition of $S_{\ell_{j+1}}$ cannot depend on the n_{ℓ_j} th output of S_{ℓ_j} .

In the graph which has been constructed for S_{ℓ_j} , the section of the path P passing through it has length $(n_{\ell_j} - m_{\ell_j} + 1) D_\Omega(\delta_{\ell_j}, \lambda_{\ell_j}; E_{\ell_j})$. But $n_{\ell_j} < m_{\ell_{j+1}} \frac{\tau_{\ell_{j+1}}}{\tau_{\ell_j}}$. Thus, the length of P , namely, D , is bounded by

$$D < \sum_{j=1}^{N} (m_{\ell_{j+1}} \tau_{\ell_{j+1}} - m_{\ell_j} \tau_{\ell_j} + \tau_j) D_\Omega(\delta_{\ell_j}, \lambda_{\ell_j}; E_{\ell_j})/\tau_{\ell_j}$$

where we have chosen to define $m_{\ell_{N+1}} \tau_{\ell_{N+1}} = n_{\ell_N} \tau_{\ell_N}$. Since $n_{\ell_N} \leq T_{\ell_N} - 1$, it follows that

$$D < \left[\max_{1 \leq \ell \leq L} \frac{D_\Omega(\delta_\ell, \lambda_\ell; E_\ell)}{\tau_\ell} \right] \left[\sum_{j=1}^{N-1} \tau_{\ell_j} + T_{\ell_N} \tau_{\ell_N} \right]$$

But if this chain of N machines is to exist, it must be that $\sum_{j=1}^{N-1} \tau_{\ell_j} \leq (T_{\ell_N} - 1)\tau_{\ell_N}$.

Using this inequality and minimizing over the encodings $E_1, \text{--}, E_L$ we have the result of the theorem. Q.E.D.

These two theorems suggest the following definitions which we shall use in interpreting these results.

Definition 7 A sequential machine $S = \langle S, I, \delta, \lambda, 0; T\rangle$ with cycle length τ is said to do computational work $W = C_\Omega^*(S)T$ and to introduce a computational delay $\Delta = D_\Omega^*(S)T$ where $D_\Omega^*(S)$ is the minimum of $D_\Omega(\delta, \lambda; E)$ over all encodings E. Also, the computing power $P_\Omega(S)$ is defined as $P_\Omega(S) = C_\Omega^*(S)/\tau$. The computational work W and computational delay Δ of a collection $S_1, S_2, --, S_L$ with cycle lengths $\tau_1, --, \tau_L$ and interconnection rule $\propto$ are given by the upper bounds in Theorems 1 and 2, respectively.

Computational work W is interpreted as the equivalent number of logical operations performed by sequential machines and is an analog of mechanical work. Theorem 1 states that a set of functions can only be computed by a set of machines which do at least as much computational work as the combinational complexity of these functions. Thus, complex functions must be computed by many machines (L large) or by a few machines each of which does a large amount of work. This in turn means that these machines execute many cycles, have many equivalent logic elements or both.

A machine which has cycle length τ and executes T cycles runs for a time $T\tau$. Thus, it does work at the rate of $P = \frac{W}{T\tau}$ and by analogy with mechanical power, we call P computing power.

Logic elements introduce delay in circuits, so we interpret computational delay Δ as the equivalent delay introduced by sequential machines. Then, Theorem 2 states that a set of functions can only be computed by a set of machines if the machines introduce a computational delay which is at least as large as the time complexity of these functions. Thus, to compute complex functions some machines must introduce a large delay or run for a long time.

We finish this section with a theorem which demonstrates that an inequality like that of Theorem 1 holds for autonomous machines.

Theorem 3 Let $A = \langle A, \delta, \lambda, P, 0\rangle$ be an autonomous machine which computes f_A and which prints when started in any state. Let $C_\Omega(\delta, \lambda, P; E)$ be the combinational complexity of δ, λ and P with respect to the encodings E which encode the range of P with the identity map and which encode 0 and A with the 1-1 encodings $h: 0 \to \Sigma_2^n$ and $g: A \to \Sigma_2^m$. Let $C_\Omega(A)$ be the minimum of $C_\Omega(\delta, \lambda, P; E)$ over all encodings E and let n_o be the integer in the minimizing encoding $h_o: 0 \to \Sigma_2^{n_o}$. Then, if Ω contains the 2-input AND element, the 2-input OR element and the NOT element, we have

$$C_\Omega(f_A) \leq (C_\Omega^*(A) + 5n_o + 1)\, T_A$$

where T_A is the maximum number of cycles executed by A when it first prints. Let $D_\Omega^*(A)$ be the minimum over encodings E of $D_\Omega(\delta, \lambda, P; E)$ which is the time complexity of δ, λ, P subject to the encodings E. Then,

$$D_\Omega(f_A) \leq (D_\Omega^*(A) + 3)\, T_A$$

Proof We use the machine S_A constructed in the proof of Lemma 1. Let $z \in 0$ and let $h_o(z) = (x_1, --, x_{n_o})$. Let $g_o: A \to \Sigma_2^{m_o}$ be the minimizing encoding g, let $a \in A$ and let $g_o(a) = (y_1, --, y_{m_o})$. Then, we represent δ_A, λ_A of S_A and P of A by δ_A^*, λ_A^* and P^* given by

$$\delta_A^*(g_o(a), b, h_o(z)) = (g_o(\delta(a)), b', h_o(z'))$$

$$\lambda_A^*(g_o(a), b, h_o(z)) = h_o(z')$$

$$P^*(g_o(a)) = P(a)$$

where

$$b' = b + P(a)$$

and if $h_o(z') = (x_i', x_2', --, x_{n_o}')$ and $h_o(\lambda(a)) = (x_1'', --, x_{n_o}'')$

then

$$x_j' = x_j \cdot (b + \overline{P(a)}) + x_j'' \cdot P(a) \cdot \bar{b}$$

Here . denotes AND, + denotes OR and $\bar{\ }$ denotes NOT. The graph of the Ω-algorithm that has been constructed which realized δ_A, λ_A has $5\,n_o + C^*_\Omega(A)$ non-source nodes, so $C^*_\Omega(S_A) \leq 5\,n_o + C^*_\Omega(A)$ and the first inequality follows. For the second, construct the functions δ^*_A, λ^*_A and P^* using the encodings which minimize $D_\Omega(\delta, \lambda, P; E)$. Then, the length of the Ω-algorithm realizing δ_A, λ_A is at most $D^*_\Omega(A) + 3$ (the length of the graph generating x'_j) and the second inequality follows. Q.E.D.

We have demonstrated that the function computed by an autonomous machine A which prints when started in any state can also be computed by a sequential machine and that inequalities like those in Theorems 1 and 2 can be established. The new inequalities have $C^*_\Omega(A)$ and $D^*_\Omega(A)$ increased by terms which are generally small. The number of cycles T_A involved is the maximum number of cycles A executes before first printing. Hereafter, we model all machines by sequential machines taking into account the results of Lemma 1 and Theorem 3.

4. Bounds on Complexity

A brief survey of important results concerning combinational complexity and time complexity is given in this section. We begin by stating two tests which can be applied to binary functions to derive lower bounds on their combinational and time complexity.

A Boolean function $f: \Sigma_2^n \to \Sigma_2$ is dependent on variable x_i if there exist values for x_1, -----, x_{i-1}, x_{i+1}, --, x_n such that $f(x_1, \text{-----}, x_{i-1}, 1, x_{i+1}, \text{--}, x_n) \neq f(x_1, \text{--}, x_{i-1}, 0, x_{i+1}, \text{--}, x_n)$.

Lemma 2 Let $f: \Sigma_2^n \to \Sigma_2$ be dependent on each of its variables. Then, if $\lceil x \rceil$ is the least integer no smaller than x,

$$C_\Omega(f) \geq \left\lceil \frac{n-1}{r-1} \right\rceil$$

$$D_\Omega(f) \geq \lceil \log_r n \rceil$$

where r is the fan-in or maximum number of variables of primitives of Ω.

The proof of the first equality is straightforward and follows from an accounting of the number of edges into and out of nodes in the directed graph of an Ω-algorithm computing f. The second inequality has been established by Winograd [4] for circuits which may have loops, so it applies to combinational circuits as we have defined them, since they do not have loops.

No test on Boolean functions is known which when satisfied provides a lower bound on combinational complexity which grows faster than linearly in the number of variables on which the function depends. This is true despite the fact that combinational complexity has been actively studied for well over twenty years. Similarly, no test is known which improves on the bound to time complexity by more than a constant multiplicative factor. However, it can be shown with the aid of Specker's theorem [6] that a function which violates the conditions of this theorem has a time complexity which is larger than $\lceil \log_r n \rceil$.

Now we state two results which are useful in bounding the complexity (time or combinational) of the most complex function in a class. They are not substitutes for tests on functions which provide bounds but in the absence of stronger tests they do provide some information of value.

Theorem 4 Consider a class of N binary functions $f: \Sigma_2^m \to \Sigma_2^n$ and fix ε, $0 < \varepsilon < 1$. Let Ω be a set of Boolean primitives of fan-in r and let the size of Ω be $|\Omega|$. Then the fraction F of these functions with combinational complexity exceeding

$$C = \frac{1-\varepsilon}{r} \frac{\log_2 N}{\log_2 \log_2 N} - \max(m+1, (n+1)/r)$$

is bounded below by $F \geq 1 - N^{-\varepsilon}$

Proof Consider the graph of an Ω-algorithm which has c non-source nodes and which computes $f: \Sigma_2^m \to \Sigma_2^n$. Source nodes can have any one of the following $m+2$ labels: $0, 1, x_1, x_2, --, x_m$. We attach distinct labels to the c non-source nodes and note that these have at most rc edges directed into them. An edge directed into a node can originate on at most one of $c + m + 1$ differently labeled nodes, since loops are not permitted in the graph. Therefore, there are at most $(c + m + 1)^{rc}$ distinct graphs.

Since a graph is to compute $f: \Sigma_2^m \to \Sigma_2^n$, n nodes must be identified as output nodes. Since there are at most c^n ways to choose these nodes, at most $(c + m + 1)^{rc + n}$ functions $f: \Sigma_2^m \to \Sigma_2^n$ can be computed by Ω-algorithms with c non-data steps. The number of such functions which can be computed with at most C non-data steps is bounded above by $(C + m + 1)^{(rC + n + 1)}$.

Using the value of C given above it is easy to show that the number of functions $f: \Sigma_2^m \to \Sigma_2^n$ computable in that many steps is bounded above by $N^{1-\varepsilon}$. Therefore, the fraction with combinational complexity less than or equal to C is bounded above by $N^{-\varepsilon}$. The conclusion follows directly. Q.E.D.

This theorem and its proof parallel and extend results by Shannon for contact networks [7].

Theorem 5 Consider a class of N Boolean functions $f: \Sigma_2^m \to \Sigma_2$ and fix ε, $0 < \varepsilon < 1$. Let Ω be a set of Boolean primitives of fan-in r and $\alpha = \log_2[|\Omega|^{1/(r-1)}2(m+2)]$. Then, the fraction F of these functions with time complexity exceeding

$$D = \log_r \left[\frac{(1-\varepsilon)\log_2 N}{\alpha}\right]$$

is bounded below by $F \geq 1 - N^{-\varepsilon}$.

Proof (Following [5], Section 7) The minimum length graph of an Ω-algorithm which

computes a Boolean function can be chosen to be a tree. There are at most $\left[\exp_{|\Omega|}(\frac{r^{\ell}-1}{r-1})\right](m+2)^{r^{\ell}}$ such trees of length ℓ since there are at most $\frac{r^{\ell}-1}{r-1}$ non-source nodes and at most r^{ℓ} edges which can be directed from source nodes labeled $0, 1, x_1, x_2, --, x_m$. Summing on ℓ, we see that there are no more than $(L+1)\left[\exp_{|\Omega|}(\frac{r^{L}-1}{r-1})\right](m+2)^{r^{L}}$ graphs of length less than or equal to L and at most this many Boolean functions $f: \Sigma_2^m \to \Sigma_2$ can be realized by them. The result follows by bounding the number of graphs by $[|\Omega|^{1/(r-1)}\ 2(m+2)]^{r^{L}}$ and equating the bound with $N^{1-\varepsilon}$. Q.E.D.

Theorem 4 is now applied to three classes of functions $f: \Sigma_2^m \to \Sigma_2^n$, the binary symmetric functions, the binary linear functions and the class of all such binary functions. There are $2^{(m+1)n}$ symmetric functions, 2^{mn} linear functions (with addition and multiplication modulo 2) and 2^{n2^m} binary functions. Therefore, for m and n sufficiently large, most binary symmetric functions have $C_\Omega(f) \gtrsim \frac{1-\varepsilon}{r}\ \frac{(m+1)n}{\log_2[(m+1)n]}$ and $D_\Omega(f) \gtrsim \log_r[\frac{(1-\varepsilon)(m+1)}{\log_2(m+1)}]$, most binary linear functions have approximately the same bounds and most binary functions have $C_\Omega(f) \gtrsim \frac{1-\varepsilon}{r}\ \frac{n2^m}{m+\log_2 n}$ and $D_\Omega(f) \gtrsim \log_r[(1-\varepsilon)\ \frac{2^m}{\log_2 m}]$.

Contrasting with these results it can be shown [8] that every binary symmetric function can be realized with combinational complexity no larger than $nm(3m+2)/2$ and time complexity bounded above by $2n + \lceil\log_2 n\rceil$ when Ω is the set of 2-variable Boolean functions. For the same set Ω, the linear functions can be realized with $C_\Omega(f) \lesssim n(2m-1)$ and $D_\Omega(f) \lesssim \lceil\log_2 2m\rceil$. Again for this set Ω, Lupanov [9] has shown that all of the binary functions can be realized with $C_\Omega(f) \lesssim n\frac{2^m}{m}(1+\varepsilon)$, $0 \lesssim \varepsilon \lesssim 1$, for $m \gtrsim M(\varepsilon)$ where $M(\varepsilon) \to \infty$ as $\varepsilon \to 0$. Also, a disjunctive normal form realization of binary functions has $D_\Omega(f) \lesssim n + \lceil\log_2 n\rceil$.

A few comments are in order concerning the sensitivity of the two complexity

measures to the set Ω of primitives. As Muller [10] has observed, replacement of a complete set of Ω by a complete set Ω' can be done by replacing each element of Ω be some fixed number of elements from Ω' and vice versa. Thus, $C_{\Omega}(f)$ and $C_{\Omega'}(f)$ can differ only by some constant factor. The same is also true for time complexity for similar reasons.

5. Efficiency

The two measures of computational work W and computational delay Δ and the inequalities of Theorems 1 and 2 provide two measures of efficiency. Work efficiency, ε_W, is defined as the ratio of the combinational complexity of a set of functions to be computed to the computational work performed by a set of machines to compute these functions. Delay efficiency, ε_Δ, is defined as the ratio of time complexity to computational delay.

There are many simple functions which are computed efficiently by many machines. For example the minterm (Boolean) functions $f(y_1, y_2, \ldots y_n) = y_1^{c_1} \cdot y_2^{c_2} \cdot \text{---} \cdot y_n^{c_n}$, where $c_i \in \{0,1\}$, $y^0 = \bar{y}$, the INVERSE of y, $y^1 = y$ and $\cdot$ denotes AND, can be realized with work efficiency $\frac{n-1}{4n} \leq \varepsilon_W \leq 1$ when Ω consists of 2-input elements by a machine that executes T cycles $1 \leq T \leq n$. The transition and output function of this machine are shown realized in Figure 3 which has an ℓ - input AND function (realized with $\ell - 1$ 2-input AND's), a 2-input AND function, ℓ 2-input EXCLUSIVE OR's for a total of 2ℓ logic elements and many binary cells offering unit delay. The variables $y_1, \text{---}, y_n$ and coefficients $c_1, \text{---}, c_n$ are grouped into sets of size ℓ, the machine executes $T = \lceil n/\ell \rceil$ cycles and does work $W = 2\ell \cdot \lceil n/\ell \rceil \leq 4n$. The minterm functions, however, can be shown to have combinational complexity equal to $n-1$ using Lemma 2. The same machine has computational delay of $\Delta = (\lceil \log_2 \ell \rceil + 2)T$ and if $1 \leq T \leq K$, K a constant, then for large n, $\frac{1}{2K} \leq \varepsilon_\Delta \leq 1$ since $D_\Omega(f) = \lceil \log_2 n \rceil$ from Lemma 1 and a construction argument.

It can also be shown that most Boolean functions of n variables can be computed with a work efficiency bounded below by $\varepsilon_W \geq 1/n^2$ over a large range of cycles by constructing a machine which realizes the minterm or disjunctive normal form decomposition of the function. While this bound can

undoubtedly be improved, it does illustrate that complex functions can be realized by sequential machines with a work efficiency that is not too small.

Work and delay efficiency may prove useful in measuring the performance of algorithms and machines. At this point, however, computational efficiency is a relatively undeveloped concept.

6. The Complexity of Storage Devices

Storage devices which are capable of selective recall have the power to compute, as we show in this section. We examine binary random access, tape and drum (or disk) storage devices.

A random-access storage device is a sequential machine $S_{ra} = < S_{ra}, I_{ra}, \delta_{ra}, \lambda_{ra}, O_{ra}; T_{ra} >$ where T_{ra} is arbitrary, S_{ra} is the set of M-tuples of stored words $(W_1, W_2, --, W_M)$, $W_j \varepsilon(\Sigma_2)^b$, I_{ra} is the set of triples of read-write command, addresses and input words (r, a, W_o), $r \varepsilon\{0,1\}$ $a \varepsilon\{1, 2, -, M\}$, $W_o\varepsilon(\Sigma_2)^b$ and O_{ra} is the set of output words $v \varepsilon(\Sigma_2)^b$. Also $\delta_{ra}(1, r, a, W_o, W_1, --, W_M) = (W_1', --, W_M')$ where $W_j' = W_j$ if $a \neq j$ and $W_j' = W_o$ if $a=j$ and $\delta_{ra}(0, a, W_o, ---, W_M) = (W_o, --, W_M)$. And $\lambda_{ra}(r, a, W_o, W_1, --, W_M) = v$ where $v = W_a$. This is called a random access device because any stored word can be accessed in any cycle.

A tape storage device is a sequential machine $S_t = < S_t, I_t, \delta_t, \lambda_t, O_t; T_t >$ where T_t is arbitrary, S_t is the set of (M+1)-tuples of stored words and head position p, $(W_1, W_2, --, W_M, p)$, where $W_j \varepsilon(\Sigma_2)^b$ and $p \varepsilon\{1,2, -,M\}$, I_t is the set of triples of read-write command, head position increment ρ and input word (r, ρ, W_o) where $r\varepsilon\{0,1\}$ $\rho\varepsilon\{-1,0,1\}$ and $W_o\varepsilon(\Sigma_2)^b$, O_t is the set of pairs (v,p') of output words $v\varepsilon(\Sigma_2)^b$ and head positions $p' \varepsilon\{1,2,-M\}$. Also, $\delta_t(1,p,W_o,W_1, --, W_M,p) = (W_1' --- ,W_M',p')$ where

$$p' = \begin{cases} 1 & p+\rho \leq 1 \\ M & p+\rho \geq M \\ p+\rho & \text{otherwise} \end{cases}$$

and $W'_j = W_j$ if $j \neq p'$, $W'_j = W_o$ if $j = p'$ and $\delta_t(0,p, W_o, ---, W_M) = (W_o, --, W_M)$. And $\lambda_t(\rho,W_o,W_1, ---,W_M,p) = (v,p')$ where $v = W_p$. Thus, the tape unit reads, increments by -1, 0 or 1 and writes a word and the new head position.

A drum storage device is a sequential machine $S_d = < S_d, I_d, \delta_d, \lambda_d, O_d; T_d >$ where T_d is arbitrary, S_d is the set of $(bM+1)$-tuples of stored bits (word organized) and head position p, $(W_{11}, W_{12}, --, W_{1b}, W_{21}, --, W_{2b}, --, W_{M1}, --, W_{Mb}, p)$, where $W_{ji} \in \Sigma_2$ and $p \in \{1,2, --, M\}$, I_t is the set of triples of write command r, input bit β and head address h, (r,β,h) where $r, \beta \in \Sigma_2$ and $h \in \{1,2, --, b\}$ and O_t is the set of pairs (β',p') of output bits $\beta' \in \Sigma_2$ and head positions $p' \in \{1,2, --, M\}$. Also, $\delta_d(r,\beta,h,W_{11}, --, W_{Mb},p) = (W'_{11}, ---, W'_{Mb},p')$ where $W'_{ji} = W_{j+1,i}$, $1 \leq j \leq M-1$, $1 \leq i \leq b$ and $W_{Mi} = \beta$ if $r = 0$ and $i = h$ or $W'_{Mi} = W_{1i}$, otherwise, and $p' = p+1$ if $p \leq M-1$ and $p' = 1$ if $p = M$. And $\lambda_d(r,\beta,h,W_{11}, --, W_{Mb},p) = (\beta',p')$ where $\beta' = W_{1h}$ if $r = 1$ and $\beta' = 0$ if $r = 0$. Thus, a drum has b tracks and rotates in one direction. A head is selected, and a bit is read from and into the track position under this head. The drum also has a counter which records the present position of the read-write heads.

The random access unit is a reasonably accurate model of core storage units. The tape and drum devices are abstractions of real tape and drum units since the latter usually access data in blocks and they also exhibit large delays before reacting to a command. Access in blocks is possible by the addition of an auxiliary machine which uses block addresses and the position of read-write heads to direct reading and writing from the storage units. Disk units and drum units are similar in operation and characteristics and the drum storage device is an adequate model for disk units.

Theorem 6 Let $C^*_\Omega(S_{ra})$, $C^*_\Omega(S_t)$, $C^*_\Omega(S_d)$ and $D^*_\Omega(S_{ra})$, $D^*_\Omega(S_t)$, $D^*_\Omega(S_d)$ be the combinational complexity and time complexity of the transition and output

functions of the random access, tape and drum storage devices, respectively. Let Ω be the set of Boolean functions with fan-in of $r = 2$. Then, if $b \geq 4$, $M \geq 17$ and b is a power of 2

$$(S\text{-}b) \leq C^*_\Omega(S_{ra}) \leq (7S+4M), \quad \lceil \log_r M \rceil \leq D^*_\Omega(S_{ra}) \leq (\lceil \log_r M \rceil + \lceil \log_r \log_2 M \rceil + 1)$$

$$(S\text{-}b) \leq C^*_\Omega(S_t) \leq (7S+5M), \quad \lceil \log_r M \rceil \leq D^*_\Omega(S_t) \leq (\lceil \log_2 M \rceil + 1)$$

$$(b\text{-}1) \leq C^*_\Omega(S_s) \leq (8b), \quad \lceil \log_r b \rceil \leq D^*_\Omega(S_d) \leq (\lceil \log_r b \rceil + \lceil \log_r \log_2 b \rceil + 2)$$

where $S = Mb$ is the storage capacity of each device.

The proof of this theorem is given in the Appendix. It is important to note that the lower bounds on combinational complexity for each device agree within a small multiplicative factor of the upper bounds. The bounds on time complexity are tighter yet. Asymptotically they are equal.

It is important to note that the bounds on complexity for tape and random access storage units have the same dependence on storage capacity and number of words. This would suggest that they are in some sense equivalent devices. Clearly, they are not equivalent for most problem solving, and in fact, a random access storage device could be used in such a way that it simulates a tape device but the reverse is not true. We postulate that the two storage devices are equivalent for some applications of a sequential nature and that this accounts for the same dependence on device parameters.

7. Computation on General Purpose Machines

In this section, we illustrate the use of Theorems 1, 2 and 3 by applying them to general purpose computers in which the principal storage medium is a random access, tape or drum device. We show that product relationships on storage, time, number of drum heads and other parameters, depending on the storage medium, must hold if functions of given complexity are to be computed.

In particular, we derive such relationships for multitape Turing Machines. We also indicate that sequential storage mediums are inherently less efficient than random access devices and indicate the size of this inefficiency. Finally, several rules of thumb are given for the use of storage devices based upon the size of their computing powers.

Consider a simple model for a general purpose computer consisting of a storage device and a second machine which acts as a central processor. Assume that inputs and outputs to the pair pass through the second machine so that the pair forms an autonomous sequential machine with its action determined by its initial state. Assume also that they both have the same cycle length and execute equal numbers of cycles while carrying out a computation. With this definition of a general purpose computer we have the following

Theorem 7 Consider three general purpose computers with random access, tape and drum storage devices. Let them execute at most T_{ra}, T_t and T_d cycles, to compute the finite functions f_{ra}, f_t and f_d, respectively, and assume that the random access and tape units have M words of b bits each, and that the drum unit has b_d tracks. Let b and b_d be powers of 2 and let Ω contain the 2-input Boolean primitives. Then, to compute f_{ra}, f_t and f_d the following inequalities must be satisfied when $b \geq 4$ and $M \geq 17$:

$$C_\Omega(f_{ra}) \leq (K_{ra}+7S+4M)T_{ra}$$
$$C_\Omega(f_t) \leq (K_t+7S+5M)T_f$$
$$C_\Omega(f_d) \leq (K_d+8b_d)T_d$$

$$D_\Omega(f_{ra}) \leq 2(\lceil\log_2 M\rceil+\lceil\log_2\log_2 M\rceil)T_{ra}$$
$$D_\Omega(f_t) \leq 2(\lceil\log_2 M\rceil+1)T_t$$
$$D_\Omega(f_d) \leq 2(\lceil\log_2 b_d\rceil+\lceil\log_2\log_2 bd\rceil+2)T_d$$

The second set of inequalities holds for large M and b_d. Here K_{ra}, K_t and K_d are constants of the machines and $S = Mb$ is the storage capacity of the

random access and tape storage devices.

The proof is a direct consequence of Theorems 1, 2, 3 and 6 and the fact that the central processor in each general purpose computer is fixed.

The interpretation of these inequalities is straightforward. If complex functions are to be computed on random access or tape machines which use a large amount of storage, then a storage-time product inequality must be satisfied as well as a product inequality involving time and the logarithm of the number of stored words. Experience teaches that some form of exchange relation must exist between storage and time on general purpose machinesand these inequalities support that experience. It is somewhat surprising that the inequalities for the drum (or disk) machine do not involve total storage capacity but only the number of tracks. This suggests that many more cycles are required on drum machines to compute functions than would be required on tape or random access machines with the same storage capacity. In fact, if the number of tracks on the drum is fixed, the number of cycles required by a drum machine must grow linearly with combinational complexity, and as we know from Section 4 this can be nearly exponential in the number of variables on which the functions depend.

The tape machine described above is a finite version of a machine known as a 1-tape, on-line Turing machine. We now look at finite versions of multitape Turing machines which are both on-line and off-line. An off-line tape machine has a finite control through which at least two tape units communicate. One of these tape units, called the input tape, has an input string written on it and it is used as a read only memory. An on-line tape machine has at least one working tape and no input tape. We assume that both types of tape machines produce outputs through their controls, that they act as autonomous machines and that their controls and tapes have equal length cycles and execute equal numbers of cycles.

Theorem 8 Let TM_1 be an on-line tape machine which computes f_1 and which has M b-bit words equally divided among m tapes. Let TM_2 be an off-line tape machine which computes f_2, which has M b-bit words equally divided among m working tapes and which has an input tape with n b-bit words. If TM_1 and TM_2 execute a maximum of T_1 and T_2 cycles to compute f_1 and f_2, respectively then there exists constants K_1 and K_2 such that the following inequalities must be satisfied:

$$C_\Omega(f_1) \leq [K_1 + 7(S + M)]T_1$$
$$C_\Omega(f_2) \leq [K_2 + 7(nb + S + n + M)]T_2$$

$$D_\Omega(f_1) \leq 2[\lceil \log_2 M/m \rceil + 1]T_1$$
$$D_\Omega(f_1) \leq 2[\lceil \log_2 \max(n, M/m) \rceil + 1]T_2$$

where $S = Mb$ and Ω contains the Boolean primitives of fan-in 2. The second set of inequalities apply for large M/m and n. In addition, if the tape heads are set at prechosen positions at the start of every computation, then the following set of inequalities must be satisfied if f_1 and f_2 are to be computed:

$$C_\Omega(f_1) \leq [K_1 + 7(b+1)m(2T_1+1)]T_1$$
$$C_\Omega(f_2) \leq [K_2 + 7(n(b+1) + (b+1)m(2T_2+1)]T_2$$

Proof The first two sets of inequalities follow directly from Theorems 1, 2, 3 and 6 and the fact that the controls are finite. The last two inequalities are a consequence of the fact that the tape heads always assume a set of prechosen positions before each computation. Let M_1 and M_2 be the smallest number of words with which f_1 and f_2 can be computed by the machines TM_1 and TM_2. Then, each working tape of these machines has M_1/m and M_2/m words, respectively, and the tape heads cannot reach all of these words from

their starting positions if $T_1 < (M_1/m-1)/2$, $T_2 < (M_2/m-1)/2$. Taking the converse of these inequalities, the result follows by invoking the first two inequalities. Q.E.D.

If the functions f_1 and f_2 are complex and if the word size and number of working tapes of the machines TM_1 and TM_2 are fixed, then T_1 and T_2 must grow at least proportionally with $\sqrt{C_\Omega(f_1)}$ and $\sqrt{C_\Omega(f_2)}$, respectively. We note from Theorem 7 that if f_3 is computed by a drum machine with fixed word size, then the number of cycles executed by it grows proportionally with $C_\Omega(f_3)$. Also, with "table look-up" any function can be computed on a random access machine in a time independent of its complexity and dependent only on the number of binary variables required to represent its domain and range. Thus, it would appear that a hierarchy of storage devices exists with the ordering determined by the manner in which words can be accessed. It would also appear that Turing machines are poor models for general purpose computers since the sequential nature in which they must search their tapes may seriously degrade performance over that available on random access machines.

In the study of the computation of recursive functions on Turing machines, time and tape complexity measures (the number of cycles and tape squares accessed) are usually considered. If these recursive functions are truncated so that they are functions on strings of length n or less, then Theorem 8 provides new relationships between time and tape complexity whose combined rate of growth may now be studied as a function of n .

To illustrate the accuracy of Theorems 7 and 8, consider the "fetch function" $f: \{1,2,---,M\} \to (\Sigma_2)^b$ given by $f(j; W_1, W_2,---, W_M) = W_j$. This function fetches one of M words and its complexity depends on values assumed by W_1---, W_M . For example, if each word has the same value, then $C_\Omega(f) = D_\Omega(f) = 0$. Since there are b^M such functions, we can, however, using Theorems 4 and 5,

show that for large M and fixed ε, $0 < \varepsilon < 1$, most fetch functions have

$$C_\Omega(f) \geq \frac{1-\varepsilon}{r} \frac{M\log_2 b}{\log_2(M\log_2 b)} \quad , \qquad D_\Omega(f) \geq \log_r[1 + \frac{(1-\varepsilon)}{\alpha} M\log_2 b]$$

where α is a constant defined in Theorem 5 and r is the fan-in of Ω.

The inequalities of Theorem 7 show that for large M at least one cycle is required on random access or tape machines to compute most fetch functions but that a number of cycles growing nearly linearly with M is required on drum machines. Linear growth for the tape machines is implied by the proof of Theorem 8 when the starting positions of the heads are preset. However, a fetch function can always be computed in one cycle on a random access machine using table look-up. On a tape machine, a fetch function can be computed in one cycle if the words are accessed in sequence while a fetch functions can be computed in a number of cycles linear in M on a drum machine or a tape machine with fixed starting positions for heads. Thus, the bounds appear to be well calibrated.

We turn now to the computing power of storage devices and develop several "rules of thumb" for their use in general purpose computers. Computing power $P_\Omega(S)$ was defined in Definition 7 as the ratio of the combinational complexity of a sequential machine S to its cycle length, $P_\Omega(S) = C^*_\Omega(S)/\tau$. The significance of computing power is that it is the rate at which computational work is done by a machine. Thus, if two machines are given equal lengths of time to compute a function, then the machine with the larger computing power will compute the function more quickly if the inequality of Theorem 1 is satisfied with near equality. These observations suggest several rules of thumb for the use of general purpose machines with several types of main and auxiliary storage.

Rules of Thumb

1. To prevent one storage unit from assuming most of the (computational) work load, choose units so that they all have about the same computing power.

2. Assign tasks to machines on the basis of the size of their computing powers.

In the absence of other information, computing power may be a useful basis on which to organize a computer and its operating system.

Now consider a typical general purpose computer which has as main memory a core (random access) unit having storage capacity of $S = 4.1 \times 10^6$ bits in words of 32 bits each and cycle time $\tau_c = 10^{-6}$ seconds. Let it have a drum with $b_{dr} = 200$ tracks, each with a capacity of 1.6×10^5 bits and a rotational speed of 3600 rpm. The bits are arranged serially on its track so its cycle time or the time to read one bit is $\tau_{dr} \cong 10^{-7}$ sec. Let it also have a disk (drum) unit with 16 disks, each with 4000 tracks and each containing 5.8×10^4 bits organized serially and a rotational speed of 3600 rpm. This unit has the equivalent of $b_d = 6.4 \times 10^4$ tracks and cycle a length of $\tau_d \cong 2.9 \times 10^{-7}$. Using the upper bounds of Theorem 6 we have for the computing power of the core P_c, the drum P_{dr} and the disk P_d the following

$$P_c = 2.9 \times 10^{13}, \quad P_{dr} = 1.6 \times 10^{10}, \quad P_d = 1.76 \times 10^{12}$$

Since the computing power of the drum and disk are at least an order of magnitude smaller than that of core, it should be expected that the channel connecting disk and drum to core and the central processor should act as a bottleneck for the machine. This correlates with experience.

8. Language Recognition

In this section we illustrate the use of computational work and inequalities developed in this paper by means of a brief examination of the work required to recognize languages. Language recognition is one function performed by most compilers or translators and if it were to require a large amount of work it could make compiling and translating costly.

A language L is a set of strings over some alphabet, say $\Sigma_a = \{0, 1, 2, --, a-1\}$, a an integer. The Chomsky hierarchy is well known and knowledge of this hierarchy is presumed here. With each language L over Σ_a we associate an infinite set of functions $\{f_L^n \mid n = 1, 2, 3, ---\}$ defined as follows: Let x be a string over Σ_a and let $\ell g(x)$ denote the length of x. Given n and a language L, we define $L_n = \{xb^{n-\ell g(x)} \mid \ell g(x) \leq n, x \in L\}$ where $b \notin \Sigma_a$ and b^i denotes the concatenation of b with itself i times. then, $f_L^n : (\Sigma_a \cup \{b\})^n \to \{0,1\}$ is defined by

$$f_L^n(y) = \begin{cases} 1 & y \in L_n \\ 0 & y \notin L_n \end{cases}$$

The function f_L^n has been defined as a function of n variables so that Lemma 1 can be used to lower bound its combinational complexity. In fact, we now show that if L contains a string x of length $\ell g(x) = k$, $2 \leq k \leq n-1$, then f_L^n depends on at least $n-2$ variables. Clearly f_L^n has value 1 on xb^{n-k} and 0 on $xb^j 0 b^{n-k-j-1}$, $1 \leq j \leq n-k-1$ so it depends on the last $n-k-1$ variables of f_L^n. Also, if we write $x = W_1 c W_2$ where $\ell g(c) = 1$ and $\ell g(W_2) \geq 1$, then f_L^n has value 1 on $W_1 c W_2 b^{n-k}$ and 0 on $W_1 b W_2 b^{n-k}$ since $W_1 b W_2 \notin L$. Thus, it also depends on its first $k-1$ variables or is dependent on at least $n-2$ variables. We conclude that

Lemma 3 If $x \in L$, $2 \leq \ell g(x) \leq n-1$, then $C_\Omega(f_L^n) \geq \lceil \frac{n-3}{r-1} \rceil$ where r is the fan-in of Ω.

Proof Lemma 2 would apply if the function f_L^n were binary. It is not, so some 1-1 encoding $h : \Sigma_a \cup \{b\} \rightarrow (\Sigma_2)^M$ is assumed. Then, the encoded version of f_L^n clearly depends on at least one component of at least $n-2$ encoded variables and the inequality follows directly from the application of Lemma 2. Q.E.D.

We examine the work required to recognize the regular, LR(k) and context-free languages. Context-free languages are exactly those recognized by off-line, push-down machines and LR(k) languages are those recognized by deterministic, off-line, push-down machines [11].

Theorem 9 Let L be a regular language. Then, there exists a constant A such that f_L^n can be computed with a computational work bounded by

$$W \leq A\, n$$

If L is LR(k), then there is a constant B such that f_L^n can be computed with a work bounded by

$$W \leq B\, n^2$$

If L is a context-free language, then there exists a constant K such that f_L^n can be computed with a work bounded by

$$W \leq K\, n^5$$

Proof The regular languages can be recognized by finite automata in real time. Thus, the first bound follows. Knuth's algorithm for the recognition of LR(k) languages [11] uses an off-line tape machine and uses a number of cycles and tape squares which are linear in n, the length of an input string. This algorithm need only be modified to test for b's are the end of a string (it can do this in linear time by reading from right to left first). Then using

Theorem 8, the second bound follows. The third bound follows in a similar fashion from Theorem 8 and the existence of the Younger algorithm [12] which recognizes an arbitrary context-free language in time proportional to n^3 and with a number of tape squares proportional to n^2. Q.E.D.

We use the following theorem to show that there are languages such that the constants in Theorem 9 must be arbitrarily large.

Theorem 10 Let L^* be a subset of $(\Sigma_2)^n$. Let $f_{L^*} : (\Sigma_2)^n \to \Sigma_2$ be the characteristic function of the set L^*. Then, for $0 < \varepsilon < 1$, a fraction $F \geq 1-2^{-\varepsilon 2^n}$ of the sets L^* have

$$C_\Omega(f_{L^*}) \geq \frac{1-\varepsilon}{r} \frac{2^n}{n} - (n+1)$$

for $n \geq |\Omega|^{1/r} \frac{(1-\varepsilon)}{r}$ where r is the fan-in of the set of primitives Ω.

This theorem follows from the direct application of Theorem 4 and the fact that there are 2^{2^n} subsets of $(\Sigma_2)^n$. Given any language L over Σ_2, the associated set L_n contains a subset L^* of Σ_2^n. Therefore, $C_\Omega(f_L^n) \geq C_\Omega(f_{L^*})$ for this set L^*. Also, the context-free, LR(k) and regular languages can include any finite set. Therefore, they can include one of the most complex sets for any given n. Since the most complex set L^* has $C_\Omega(f_{L^*})$ which grows nearly exponentially in n, the constants of Theorem 9 can be arbitrarily large.

The bounds of Theorem 9 may suggest to the reader that there is a 1-1 correspondence between the Chomsky hierarchy and a hierarchy established on the basis of computational work. To dispel such a notion, consider the language $L = \{b\,\underline{1}\,b\,c^{\ell_1}\,\underline{2}\,b\,c^{\ell_2}\,\underline{3}\,b\,c^{\ell_3} \text{ --- } \underline{i}\,b\,c^{\ell_i}\,b \mid i = 1, 2, 3, \text{---}\}$ where $\underline{i}$ is the dyadic representation of integer i, $b \neq c$, $b, c \notin \Sigma_2$ and ℓ_k is the length of $\underline{k}$ plus 1. L is context-sensitive because it can be recognized by a linear bounded automaton but not context-free since it is not closed under application of the "pumping lemma" for context-free languages. The function f_L^n can be

computed with a computational work bounded by

$$W \leq A\, n\, \ell og_2\, n$$

for some constant A since as we show, the set L_n can be recognized a tape machine which executes n cycles and uses no more than $\ell og_2 n$ binary tape squares. The tape machine stores $\underline{1}$ on its tape, ascertains that the first symbol is b, compares the digits until the next occurance of b with the digits of $\underline{1}$ stored on its tape (moving the tape head from left to right), uses the ℓ_1 occurences of c to add 1 to $\underline{1}$ to form $\underline{2}$ (moving the tape head from right to left using "carry ripple through" addition), comparing the set of digits until the next occurence of b with $\underline{2}$, etc.. If any of these tests fail, the input string is rejected. The number of binary tape squares required to process a string of length n certainly cannot exceed $\ell og_2 n$. Then the inequality follows from Theorem 8. Note that this language is processed in real time. Also note that the linear lower bound of Lemma 3 applies. Thus, the bound on computational work is tight for n such that $\ell og_2 n$ is small, that is, for n in the range 1 to 10^3, say.

9. A Quantum-Mechanical Bound on Complexity

In this section we derive a bound on the maximum complexity of any function that can be computed in t seconds with E units of energy under the assumption that the speed of operation of computers is so large that the quantum-mechanical limit is approached and that the computers must be realized with binary logic elements and binary memory cells. This assumption implies that the inequality of Theorem 1 applies to the actual machine.

The logic elements have several inputs and we view the action of one element as that of determining the state of each of its inputs by measuring energy levels

and computing and registering an output state. We assume (as is true for solids) that to discriminate between two energy levels with separation ΔE requires the expenditure of ΔE units of energy. Then, the maximum number of logic elements X which can be used if no more than E units of energy are to be expended satisfies

$$X \leq E/\Delta E$$

where ΔE is the minimum separationof energy levels in the computer.

Each logic element has a switching time Δt which cannot be less than the time to measure the states of its inputs. Then, the number of cycles which a machine can complete in t seconds, T satisfies

$$T \leq t/\Delta t \quad .$$

Also, ΔE and Δt are related by the Heisenberg uncertainty relation as follows:

$$\Delta E \Delta t \geq h/2\pi$$

where h is Planck's constant. That is, a reliable measurement of an energy difference ΔE requires at least Δt seconds where Δt satisfies this equation.

Then, for a function f to be computable in t seconds with E joules by a single machine with X logic elements in T cycles requires that

$$C_{\Omega}(f) \leq XT \leq (Et) \times 10^{34}$$

where Ω is the set of logic elements used for the realization of the "quantum-mechanical computers." It is doubtful whether this limit will ever be approached, nevertheless, it is instructive to observe the following:

Theorem 7 Subject to the conditions given above, most Boolean functions $f: \Sigma_2^p \to \Sigma_2$ with $p = 160$ or more cannot be computed in one hour with one kilowatt of power (1 joule = 1 watt-second).

This result follows as a consequence of Theorem 4.

While it is difficult to believe that one would want to compute the most complex Boolean functions of p variables, it is interesting that with $p = 160$ they cannot be computed with a very sizable amount of power in a considerable length of time.

10. Conclusions

In this paper, we have examined the computation of finite functions by sequential machines and have developed two measures of complexity, computational work and computational delay. Inequalities have been established showing that a minimum amount of work and delay is required to compute a set of functions and these minimums are the combinational complexity and time complexity, respectively, of this set of functions. These inequalities suggest two measures of computational efficiency which may prove useful in estimating the performance of computer programs and machines.

The inequalities involving work and delay have been applied to general purpose computers with random access units, tape units or drum units as principal storage to show (approximately) that for tape and random access units that the product of time and storage capacity must satisfy a lower bound determined by the complexity of the functions being computed. A similar but weaker inequality holds for drum machines. One conclusion drawn from these results is that the limited accessibility of stored words on tape requires that the number of cycles executed to compute functions with complexity C on tape machines must grow at least as fast as $\sqrt{C}$. On drum machines the number must grow linearly with C while on random access machines functions can be computed with a number of cycles which is independent of their complexities. This suggests clear hierarchy on storage units which corresponds to the hierarchy established on the basis of the ability of one device to mimick the behavior of another device.

The computing power of a machine has been defined and used to suggest rules of thumb for the use and acquisition of storage devices.

As an application of the measures and methods of analysis introduced in

this paper we have examined the work required to recognize strings of length n or less from regular, LR(k) and context free languages. Bounds are given on the work required to recognize such languages, all of which are algebraic in n. A context-sensitive language is given to suggest that the Chomsky hierarchy and a hierarchy based on computational work do not coincide.

Finally, a bound is given on the combinational complexity of the most complex function which can be computed by a quantum mechanical computer in one hour with a kilowatt of power.

The results of this paper may suggest new ways of treating the computational problems of greatest interest, namely, the computation of finite functions on finite machines.

Appendix

Proof of Theorem 6

The object is to bound $C^*_\Omega(S)$ and $D^*_\Omega(S)$ for the three machines. We begin with lower bounds.

A binary encoding is given to addresses, head positions and increments with a^* the encoding of a, p^* the encoding of p, h^* the encoding of h and ρ^* the encoding of ρ. Then, the binary representations for the output functions are $\lambda^*_{ra}(r, a^*, W_o, W_1, --, W_M) = W_a$, $\lambda^*_t(r, \rho^*, W_o, W_1, --, W_M, p^*) = (W_p, p^{*\prime})$, $\lambda^*_d(r, \beta, h^*, W_{11}, --, W_{Mb}, p^*) = (\beta', p^{*\prime})$. For the random access device, we form a new function which is the logical OR of the b digits of W_a. This function is easily shown to depend on all of the Mb variables in $W_1, W_2, --, W_M$ and from Lemma 1 its combinational complexity is at least as large as $Mb-1$. But $b-1$ 2-variable OR functions are sufficient to form the new function from λ^*_{ra} so the bound on P_{ra} follows. The lower bound on P_t follows an identical argument applied to λ^*_t modified by suppressing $p^{*\prime}$ (which cannot increase its complexity). The lower bound on P_d follows from Lemma 1 by observing that β' depends on the variables $W_{11}, W_{12}, --, W_{1b}$.

The lower bounds on Δ_{ra}, Δ_t and Δ_d follow also from Lemma 1 by observing that each component of W_a and W_p (which identify Boolean functions) depend on at least M variables and β' depends on at least b variables.

To develop upper bounds, we need a combinational machine which given a binary encoding of a word address a^*, an increment value ρ^*, or of a head address h^* produces a 1 output on a line corresponding to that address and a 0 output on all other lines. Let this machine accept one of at most 2^n binary n-tuple addresses a^* and let it produce a 1 on its <u>ath</u> line and 0 on other lines where a^* is the binary encoding of a. Call this machine $A(n)$. Then, if $\ell_j(a)$ is the output on the <u>jth</u> line of

$A(n)$ and if the binary encoding of j is $j^* = (c_1, --, c_n)$ then $\ell_j = X_1^{c_1} \cdot X_2^{c_2} \cdot --- \cdot X_n^{c_n}$ where $\cdot$ denotes AND, X^0 denotes $\bar{X}$, the INVERSE of X, and X^1 denotes X. Then $\ell_j(a)$ is equal to a minterm and the time complexity of $A(n)$ is clearly bounded above by $\lceil \ell og_2 n \rceil$. Also $A(n)$ can be produced from $A(n-1)$ with the addition of 2^n elements since each minterm $X_1^{c_1} --- X_{n-1}^{c_{n-1}} X_n^{c_n}$ of $A(n)$ can be produced from a minterm $X_1^{c_1} --- X_{n-1}^{c_{n-1}}$ of $A(n-1)$ by the addition of one element. Thus, the combinational complexity of $A(n)$ is bounded above by $2(2^n-1)$.

Consider the random access storage device with transition function $\delta_{ra}(r,a,W_0, W_1, ---, W_M) = (W_1', W_2', ---, W_M')$ and output function $\lambda_{ra}(r, a, W_0, W_1, ---, W_M) = v$. Each word is a b-tuple so let W_{ji} be the ith component of the jth word. Then, we have $W_{ji}' = W_{ji} \cdot (\bar{\ell}_j(a)+\bar{r}) + W_{oi} \cdot \ell_j(a) \cdot r$ $1 \leq i \leq b$, $1 \leq j \leq M$, and $v_i = \sum_{j=1}^{M} \ell_j(a) \cdot W_{ji}$, $1 \leq i \leq b$, where $+$ denotes OR and $\sum_{j=1}^{M}$ denotes the OR of M terms. Then, from these equations and the definition of $A(n)$, $C_\Omega^*(S_{ra}) \leq 5S + (M + M-1)b + 2(2^n-1)$ where $2^{n-1} \leq M \leq 2^n$. From this it follows that $C_\Omega^*(S_{ra}) < 7S + 4M$. Also, $D_\Omega^*(S_{ra}) \leq \max(3 + \lceil \ell og_2 n \rceil, 1 + \lceil \ell og_2 M \rceil + \lceil \ell og_2 n \rceil)$ where $n < \ell og_2 2M$. It follows that $D_\Omega^*(S_{ra}) \leq \lceil \ell og_2 M \rceil + \lceil \ell og_2 \ell og_2 2M \rceil + 3$.

Consider next the tape storage device with transition function $\delta_t(r,\rho,W_0, W_1, --, W_M, p) = (W_1', ---, W_M', p')$ and output function $\lambda_t(r,\rho, W_0, W_1, --, W_M, p) = (v, p')$. Represent the head positions with binary M-tuples $p^* = (q_1, q_2, ---, q_M)$ where $q_j(p) = 1$, $j=p$ and $q_j(p) = 0$, otherwise. Let $p'^* = (q_1', q_2', ---, q_M')$, represent ρ by a binary pair and consider the function $s^* = (s_1, s_2, s_3)$ realized by $A(2)$ where $s_j = 1$ if $j = 2 + \rho$, $s_j = 0$, otherwise. ($C_\Omega(s^*) \leq 6$, $D_\Omega(s^*) \leq 1$, since s^* is realized by $A(2)$). Then, we have $v_i = \sum_{j=1}^{M} q_j(p) \cdot W_{ji}$, $1 \leq i \leq b$, $q_j' = q_{j+1} \cdot s_1 + q_j \cdot s_2 + q_{j-1} \cdot s_3$, $2 \leq j \leq M-1$, $q_1' = q_1 \cdot (s_1 + s_2) + q_2 \cdot s_1$,

$q'_M = q_M \cdot (s_2 + s_3) + q_{M-1} \cdot s_3$ and $W'_{ji} = W_{ji} \cdot (\overline{q'_j} + \bar{r}) + W_{oi} \cdot q'_j \cdot r$, $1 \leq i \leq b$, $1 \leq j \leq M$. Therefore, $C^*_\Omega(S_t) \leq 6 + (M + M-1)b + 5(M-2) + 8 + 5Mb \leq 7S + 5M$, if $b \geq 4$. Also, it follows that $D^*_\Omega(S_t) \leq \max(1 + \lceil \ell og_2 M \rceil, 6, 6) \leq \lceil \ell og_2 M \rceil + 1$ since $\lceil \ell og_2 M \rceil \geq 5$.

Finally, consider the drum storage device with transition function $\delta_d(r, \beta, h, W_{11}, ---, W_{Mb}, p) = (W'_{11}, ---, W'_{Mb}, p')$ and output function $\lambda_d(r, \beta, h, W_{11}, ---, W_{Mb}, p) = (\beta', p')$. Represent the state of the counter by an M-tuple $p^* = (q_1, q_2, ---, q_M)$ (and $p'^* = (q'_1, q'_2, ---, q'_M)$) where $q_j = 1$ if $j = p$ and $q_j = 0$ otherwise. Then, we have $q'_j = q_{j-1}$, $2 \leq j \leq M$, and $q'_1 = q_M$ so the counter has zero combinational complexity. Let h^* be a binary $\lceil \ell og_2 b \rceil$-tuple and let it be used as input to $A(n)$, $n \leq \lceil \ell og_2 2b \rceil$. Let $\ell_i(h)$, $1 \leq i \leq b$, be the outputs. Then, $W'_{ji} = W_{j+1,i}$, $1 \leq j \leq M-1$, $1 \leq i \leq b$, $W'_{Mi} = \beta \cdot \bar{r} \cdot \ell_i(h) + W_{1i} \cdot \overline{(\bar{r} \cdot \ell_i(h))}$, $1 \leq i \leq b$ and $\beta = (\sum_{i=1}^{b} W_{1i} \cdot \ell_i(h)) \cdot r$. Then, it follows that $C^*_\Omega(S_d) \leq 4b + 2b + 2b = 8b$ and $D^*_\Omega(S_d) \leq \max(3 + \lceil \ell og_2 n \rceil, 2 + \lceil \ell og_2 b \rceil + \lceil \ell og_2 n \rceil)$ where $n \leq \lceil \ell og_2 2b \rceil$ so $D^*_\Omega(S_d) \leq \lceil \ell og_2 b \rceil + \lceil \ell og_2 \ell og_2 2b \rceil + 2$. Q.E.D.

References

1. Minsky, M., "Form and Computer Science," Journal, ACM, Vol. 17, No. 2 pp. 197-215, (April, 1970).

2. Savage, J. E., "Three Measures of Deooder Complexity," IBM Journal of Research and Development, Vol. 14, No 4, (July, 1970).

3. Savage, J. E., "The Complexity of Decoders - Part II: Computational Work and Decoding Time," IEEE Trans. on Information Theory, Vol. IT-17, No. 1, (January, 1971).

4. Winograd, S., "On the Time Required to Perform Addition," JACM, Vol. 12, No. 2, (April, 1965).

5. Winograd, S., "On the Time Required to Perform Multiplication," JACM, Vol. 14, No. 4, (October, 1967).

6. Hodes, L., "The Logical Complexity of Geometric Properties in the Plane," JACM, Vol. 17, No. 2, (April, 1970).

7. Shannon, C. E., "The Synthesis of Two-Terminal Switching Circuits," Bell System Technical J., Vol. 28, pp. 59-98, (1949).

8. Harrison, M. A., Introduction to Switching and Automata Theory, Chapter 6, McGraw-Hill, New York (1965).

9. Lupanov, O. B., "A Method of Circuit Synthesis," Izv. VUZ, Radiofizike, No. 1, 120 (1958).

10. Muller, D. E., "Complexity in Electronic Switching Circuits," IRE Trans. on Electronic Computers, Vol. EC-5, pp. 15-19, (March, 1956).

11. Knuth, D., "On the Translation of Languages From Left to Right," Information and Control, Vol. 8, pp. 607-639, (1965).

12. Younger, D. H., "Recognition and Parsing of Context-Free Languages in Time n^3," Information and Control. Vol. 10, No. 2, (1967).

JES/ljn:lb

ACKNOWLEDGMENT

The author acknowledges the encouragement and support provided by Drs. E. C. Posner and S. Butman as well as conversations with them and other members of the Communication Systems Research Section, JPL, and conversations with Drs. L. Kleinrock of UCLA and L. H. Harper, U.C. Riverside.

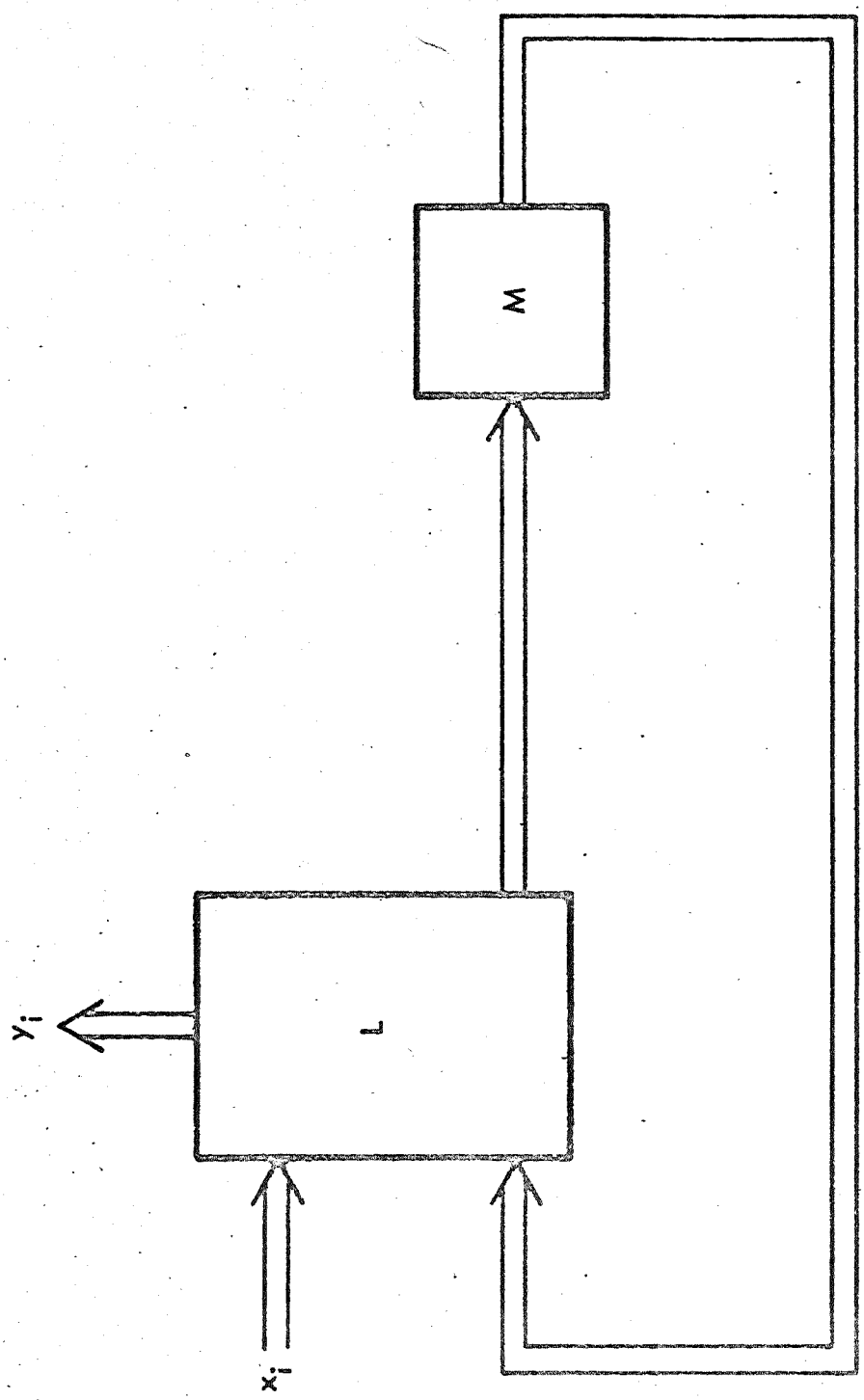

Fig. 1. Sequential Machine Model

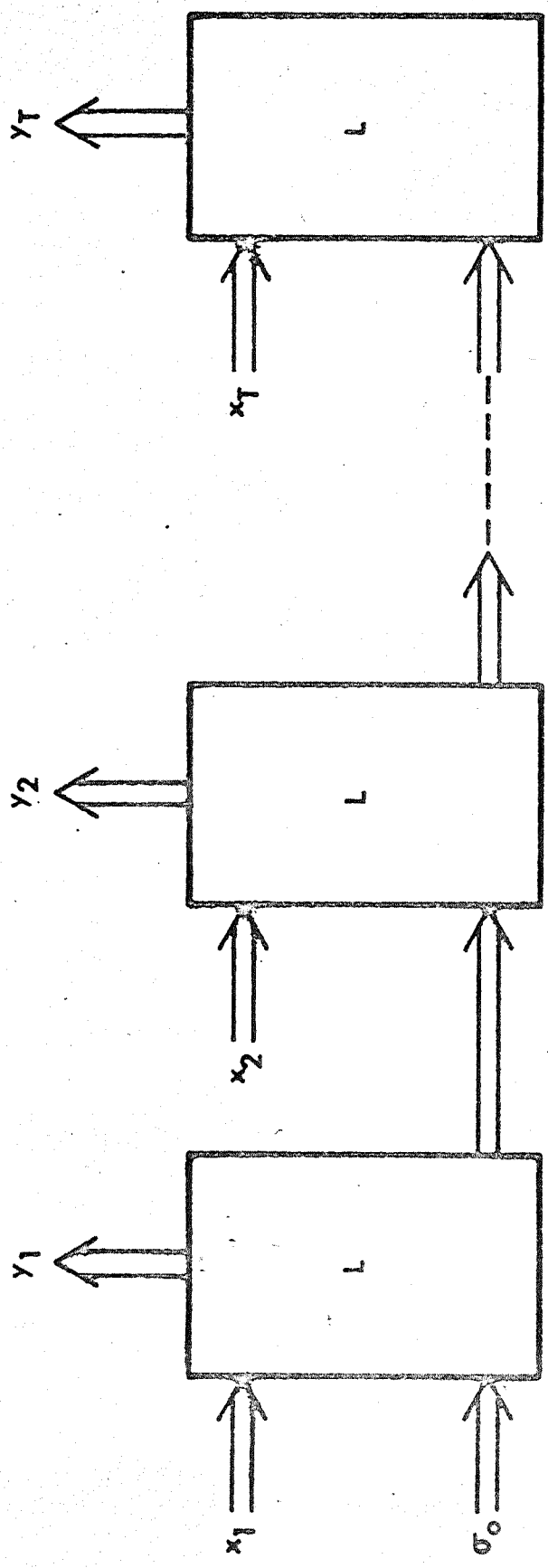

Fig. 2. Combinational Equivalent to Sequential Machine

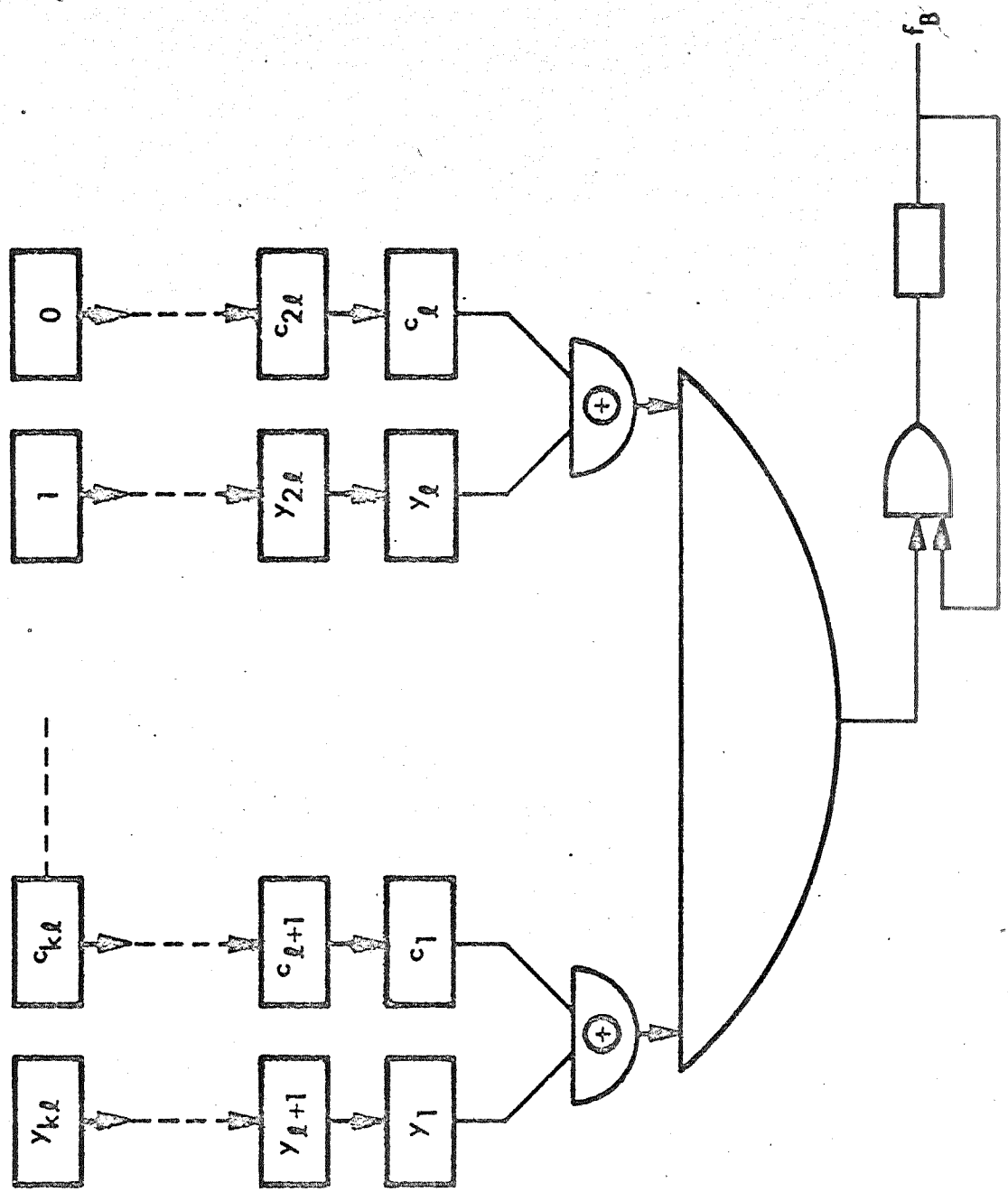

Fig. 3 Sequential Machine Which Computes f_B