Computational Work and Time on Finite Machines

J. E. SAVAGE

Brown University, Providence, Rhode Island

ABSTRACT. Measures of the computational work and computational delay required by ma chines 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, numbe of drum tracks and time, number of bits in an address and time, etc., must be satisfied to com pute finite functions on bounded machines.

KEY WORDS AND PHRASES: computational work, finite machines, automata, memor hierarchies, general purpose computer, computational efficiency, functional complexity

CR CATEGORIES: 5.22, 5.30, 6.1, 6.2

1. *Introduction*

It is a fact, as Minsky notes [1], that very little is known about "possible exchange between time and memory, tradeoffs between time and program complexity, and other important parameters of computation. While exchange relations are no the only form in which basic information about computational processes could b expressed, they could be one important representation of such information. I fact, Minsky has said that "the recognition of exchanges is often the conceptio of a science, if quantifying them is its birth" [1]. In this setting, this paper contribute to the conception and quantification of computer science by developing many ex change inequalities involving storage, time, and other important parameters c computation.

In this paper we examine the computation of finite functions (functions whos domain and range are finite) on finite machines. In Section 2 we examine machin models in which sequential machines play a central role. Two complexity measure are defined in Section 3 for finite functions, namely, combinational complexity an time complexity, and these are defined for "straight line" algorithms, that is, a gorithms with no loops or branching.

Two sets of basic inequalities are developed, relating the combinational complexit and time complexity of sequential machines and the number of cycles which the execute to the combinational complexity and time complexity of the function

This work was completed in part at Brown University and in part at the Jet Propulsion Lab oratory, California Institute of Technology, Pasadena, Calif. The Brown University portio has been supported by NSF under grant GK-13162 and by NASA under grants NGR 40-002-08 and NGR 40-002-090. The JPL portion of this work was supported by NASA under contrac NAS 7-100.

Author's address: Brown University, Division of Engineering, Providence, RI 02912.

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].

Computational efficiency is discussed briefly in Section 4 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 also that most Boolean functions are computable on many machines with modest efficiency.

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 is large, which is a much weaker inequality since storage capacity is not involved in the product. Inequalities are also developed for on-line 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 at least linearly with C on drum machines and at least as fast as $\sqrt{C}$ on tape machines. 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.

Computing power is discussed in Section 7 as a potential basis for choosing the type, size, and speed of storage units in a general purpose computer.

Finally, in Section 8, three applications of the inequalities of Section 2 are cited. They concern the measurement of the performance of algorithms, the recognition of languages, and computation on quantum mechanical computers. It is shown that most Boolean functions in p variables cannot be computed in one hour with a kilowatt of power if $p \geq 160$.

2. *Computation on Finite Machines*

As our basic model for computing machines, we shall use the conventional Moore sequential machine, modified to execute a fixed number of cycles. While such models do not capture the normal operation of a general purpose computer, it can be shown that computations on this type of machine can be carried out on sequential machines in the maximum number of cycles executed by the general purpose computer.

Definition 1. A *sequential machine* is a sextuple $S = \langle S, I, \delta, \lambda, O; T \rangle$ where S, I, and O are the finite *state set*, *input alphabet*, and *output alphabet* of the machine, respectively. δ: $S \times I \rightarrow S$ is the *state transition function* and λ: $S \rightarrow O$ 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 $(y_1, y_2, \cdots, y_n) \in I^n$, $s \in S$. Then the extensions $\delta^{(n)}$: $S \times I^n \to S$, $\lambda^{(n)}$: $S \times I^{n-1} \to O$ of δ and λ are defined by $\delta^{(1)} = \delta$, $\lambda^{(1)} = \lambda$, and for $n = 2, 3, \cdots$,

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

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

The machine is said to *compute* f_S : $S \times I^T \to O^T$, where

$$f_S(s; y_1, y_2, \cdots, y_T) = (z_1, z_2, \cdots, z_T) \quad \text{and} \quad z_n = \lambda^{(n)}(s; y_1, y_2, \cdots, y_n).$$

Clearly, if S computes f_S, it also computes all restrictions of f_S and all functions obtained by suppressing components of $(z_1, z_2, \cdots, z_T)$. In addition, f_S can be viewed as a collection of T (not necessarily independent) functions. It should also be noted that every finite function can be computed by some sequential machine.

To complete the description of a sequential machine S, we assume that the machine has a number of input and output lines and that the alphabets I and O are the sets of states which may exist on these lines, respectively. We also assume that S makes state transitions at a constant rate of one every τ seconds (τ is the cycle length) until T cycles have been completed, at which point the state of S remains fixed. Finally, we assume that S makes a state transition at the end of a clock cycle and that the new state and output are determined by the state during the cycle and the signals on input lines at the end of the previous cycle. It should be noted that this last assumption allows inputs to S to change during a cycle.

A single sequential machine is often an inadequate model for a computing machine. Therefore, we permit the interconnection of sequential machines in which output lines of machines may be connected to input lines of other machines. We require that interconnections satisfy the following two conditions: (1) every input line is connected to exactly one external input or output line of a machine; and (2) an output line and input line may be connected only if the signals which may exist on the output line are among those permitted on the input line. The machines in an interconnected set of machines may have different cycle lengths and execute different numbers of cycles.

An interconnected set of machines induces a map from initial states and external inputs to outputs of machines. This map is the function (or functions) computed by the set of machines. To avoid obscuring detail, no attempt is made to formally define such functions.

3. *Computational Work and Delay*

In this section we define "straight line" or Ω-algorithms, two measures of functional complexity, namely, 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 2. Let Σ be a finite set and let $\Omega = \{h_i \mid h_i: \Sigma^{n_i} \to \Sigma\}$ be a finite set of (primitive) operations over Σ. Let $\Gamma = \Sigma \cup \{x_1, x_2, \cdots, x_N\}$ 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 $\beta = (\beta_1, \beta_2, \cdots, \beta_K)$ where at the kth step $\beta_k \in \Gamma$ or $\beta_k = (h_i; k_1, \cdots, k_{n_i})$ 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 func-

tion $\bar{\beta}_k$ which is either a constant or a function which identifies a variable x_i. If $\beta_k = (h_i ; k_1, \cdots, k_{n_i})$, we associate the recursively defined function $\bar{\beta}_k = h_i(\bar{\beta}_{k_1}, \cdots, \bar{\beta}_{k_{n_i}})$. The Ω-algorithm β is said to compute the functions $\bar{\beta}_{m_1}, \bar{\beta}_{m_2}, \cdots, \bar{\beta}_{m_q}$ where $\beta_{m_1}, \beta_{m_2}, \cdots, \beta_{m_q}$ are any of the 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 3. 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 $\beta_k \in \Gamma$, the corresponding node of G has no edges directed into it and is called a *source node*. If $\beta_k = (h_i ; k_1, \cdots, k_{n_i})$, the node corresponding to it has n_i edges directed into it from nodes corresponding to steps $\beta_{k_1}, \cdots, \beta_{k_{n_i}}$. 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 4. The *combinational complexity*, $C_\Omega(f_1, f_2, \cdots, f_r)$, of the set of Boolean functions $\{f_1, f_2, \cdots, f_r\}$ with respect to the set of primitives Ω is the smallest number of nonsource nodes in any graph of an Ω-algorithm which computes these functions.

Definition 5. The length of an Ω-algorithm is the number of edges on the longest directed path of its graph. The *time complexity*, $D_\Omega(f_1, f_2, \cdots, f_r)$, of the set of Boolean functions $\{f_1, f_2, \cdots, f_r\}$ with respect to the set of primitives Ω 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 number of units of delay required to compute a set of functions.

The definitions of combinational complexity and time complexity are extended to non-Boolean functions in the obvious way. If $f: B_1 \times B_2 \times \cdots \times B_p \to B_1' \times B_2' \times \cdots \times B_q'$, then each nonbinary set B_i or B_j' is encoded with a 1–1 map into a set of binary tuples and the resulting collection of Boolean functions is denoted f^*. Then, the combinational complexity and time complexity of f, denoted $C_\Omega(f)$ and $D_\Omega(f)$, respectively, are taken as the minimums over all encodings of $C_\Omega(f^*)$ and $D_\Omega(f^*)$.

A survey of results on the combinational complexity of functions can be found in Harrison [6] and results on time complexity can be found in [4, 5].

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

THEOREM 1. *Let $S_l = \langle S_l, I_l, \delta_l, \lambda_l, O_l : T_l \rangle$ for $1 \leq l \leq L$ and let S_l have cycle length τ_l. Let the machines $S_1, S_2, \cdots, S_L$ be interconnected subject to the conditions stated above and let this collection of machines compute the functions $f_1, f_2, \cdots, f_r$. Let E_l be a set of encodings for the domains and ranges of $\delta_l : S_l \times I_l \to S_l$ and $\lambda_l : S_l \to O_l$ subject to the restriction that the same encoding is used for each occurrence of*

S_l. Let $C_\Omega(\delta_l, \lambda_l; E_l)$ be the combinational complexity of δ_l and λ_l with the encodings E_l. Then,

$$C_\Omega(f_1, \cdots, f_r) \leq \min_{E_1,\cdots,E_L} \sum_{l=1}^{L} C_\Omega(\delta_l, \lambda_l; E_l) T_l$$

where the minimum is over the encodings under the condition that input and output lines which are connected have compatible 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 (δ_l, λ_l) and connecting the state output of one copy to the state input of another so that a chain is formed. This algorithm is realized using $C_\Omega(\delta_l, \lambda_l; E_l) T_l$ primitive operations.

The interconnections and timing information determine which external inputs and internal outputs are used in time by the various machines to make state transitions. Ω-algorithms are constructed as indicated above for each of these machines and then connections are formed as indicated by the interconnections 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 compatible.

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

THEOREM 2. *Under the conditions of Theorem 1, let $D_\Omega(\delta_l, \lambda_l; E_l)$ be the time complexity of δ_l, λ_l with encodings E_l. Let*

$$R(S_1, \cdots, S_L) = \min_{E_1,\cdots,E_L} \max_{1\leq l\leq L} (D_\Omega(\delta_l, \lambda_l; E_l)/\tau_l)$$

where the minimum is over all encodings under the condition that input and output lines which are connected have compatible encodings. Then,

$$D_\Omega(f_1, f_2, \cdots, f_r) \leq R_\Omega(S_1, \cdots, S_L) [\max_l T_l \tau_l].$$

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 $S_{l_1}, S_{l_2}, \cdots, S_{l_N}$. Let the path pass through the graph associated with S_{l_j} at an input given to S_{l_j} at the end of its m_{l_j}th cycle and at an output produced by S_{l_j} at the end of its n_{l_j}th cycle, $1 \leq j \leq N$. Then, $1 \leq m_{l_j} < n_{l_j} - 1$ and $n_{l_j}\tau_{l_j} \leq m_{l_{j+1}}\tau_{l_{j+1}}$ because if this second condition is not satisfied, the $m_{l_{j+1}}$th transition of $S_{l_{j+1}}$ 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 $(n_{l_j} - m_{l_j}) \times D_\Omega(\delta_{l_j}, \lambda_{l_j}; E_{l_j})$. But $n_{l_j} \leq m_{l_{j+1}}(\tau_{l_{j+1}}/\tau_{l_j})$. Thus the length of P, namely, D, is bounded by

$$D \leq \sum_{j=1}^{N} (m_{l_{j+1}}\tau_{l_{j+1}} - m_{l_j}\tau_{l_j}) \times D_\Omega(\delta_{l_j}, \lambda_{l_j}; E_{l_j})/\tau_{l_j}$$

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

$$D < [\max_{1\leq l\leq L} (D_\Omega(\delta_l, \lambda_l; E_l)/\tau_l)] T_{l_N}\tau_{l_N}.$$

Using this inequality and minimizing over the encodings $E_1, \cdots, E_L$, the result of the theorem follows. Q.E.D.

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

Definition 6. A sequential machine $S = \langle S, I, \delta, \lambda, O; T\rangle$ 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 $C_\Omega^*(S)$ and $D_\Omega^*(S)$ are the minimums of $C_\Omega(\delta, \lambda; E)$ and $D_\Omega(\delta, \lambda; E)$, respectively, 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 an interconnected set of machines $S_1, S_2, \cdots, S_L$ with cycle lengths $\tau_1, \cdots, \tau_L$ are given by the upper bounds in Theorems 1 and 2, respectively.

We interpret $C_\Omega^*(S)$ and $D_\Omega^*(S)$ as the equivalent number of logical operations performed by S in one cycle and the equivalent delay introduced by S in one cycle, respectively. These are equivalent numbers because S may not be realized with elements from Ω and may contain elements not usually called logic elements, such as magnetic cores, which do perform a limited amount of computation.

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 $P = W/T\tau$, 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.

4. *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 $f(y_1, y_2, \cdots, y_n) = y_1^{c_1} \cdot y_2^{c_2} \cdot \cdots y_n^{c_n}$, 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 $(n - 1)/4n \leq \epsilon_W \leq 1$ when Ω consists of 2-input elements, by a machine that executes T cycles $1 \leq T \leq n$. The transition and output functions of this machine are shown realized in Figure 1, which has an

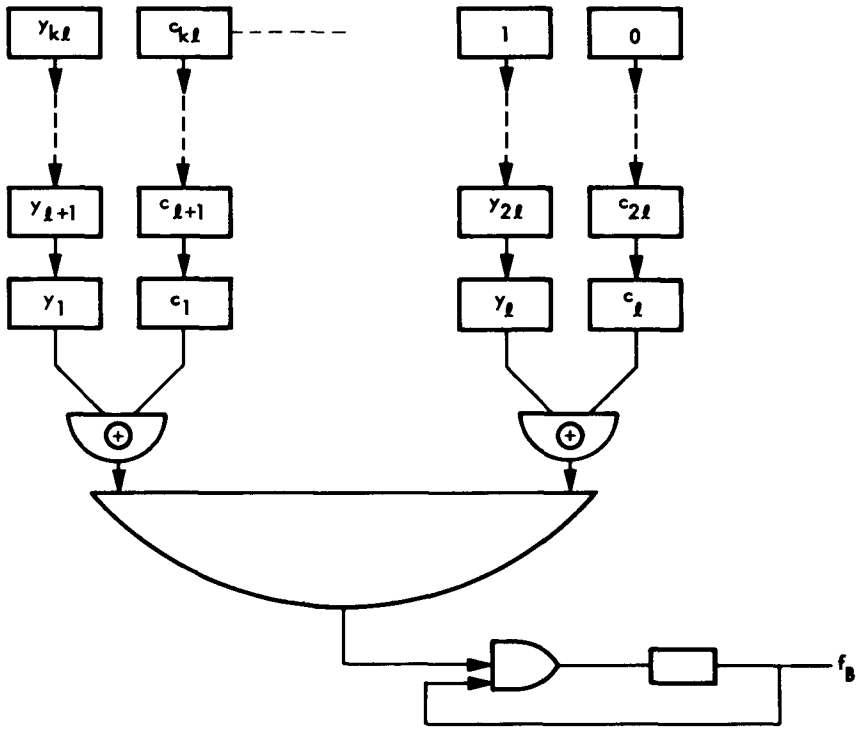

FIG. 1. Sequential machine which computes f

l-input AND function (realized with $l - 1$ 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 $y_1, \cdots, y_n$ and coefficients $c_1, \cdots, c_n$ are grouped into sets of size l; the machine executes $T = \lceil n/l \rceil$ cycles and does work $W = 2l \cdot \lceil n/l \rceil \leq 4n$. The minterm functions, however, can be shown to have combinational complexity equal to $n - 1$ when Ω consists of 2-input Boolean functions. The same machine has computational delay of $\Delta = (\lceil \log_2 l \rceil + 2)T$, and if $1 \leq T \leq K$, K a constant, then for large n, $1/K \leq \epsilon_\Delta \leq 1$ since it is easily shown that $D_\Omega(f) = \lceil \log_2 n \rceil$.

It can also be shown that most Boolean functions of n variables can be computed with a work efficiency bounded below by $\epsilon_W \geq 1/n^2$ 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.

5. *The Complexity of Storage Devices*

Storage devices which are capable of selective recall have the power to compute, as shown in this section. We examine binary random access, tape, and drum (or disk) storage devices, each containing $M = 2^m$ words for some integer m. We let $\underline{q} \in (\Sigma_2)^m$

be the dyadic representation of the integer q in $\{0, 1, \cdots, M-1\}$, and we let $\hat{q} \in (\Sigma_2)^M$ be the positional representation of $q \in \{0, 1, \cdots, M-1\}$; i.e. the $(q+1)$th component of $\hat{q}$ is 1 and all others 0.

A *random access* storage device is a sequential machine $S_{ra} = \langle S_{ra}, I_{ra}, \delta_{ra}, \lambda_{ra}, O_{ra}; T_{ra}\rangle$ where T_{ra} is arbitrary. S_{ra} is the set of $M+1$ tuples $(W_0, W_1, \cdots, W_M)$, $W_j \in (\Sigma_2)^b$, where $W_0, \cdots, W_{M-1}$ are stored words and W_M is the output word. I_{ra} is the set of triples $(s, \underline{a}, w)$ where $s \in \Sigma_2$, $a \in \{0, 1, \cdots, M-1\}$, and $w \in (\Sigma_2)^b$. If $s = 0$, the internal state is unchanged. If $s = 1$, the word at address a is replaced by W_0. $O_{ra} = (\Sigma_2)^b$ and

$$\delta_{ra}(W_0, \cdots, W_M; s, \underline{a}, w) = \begin{cases} (W_0, W_1, \cdots, W_M) & s = 0, \\ (W_0', W_1', \cdots, W_{M-1}', W_M') & s = 1, \end{cases}$$

where $W_M' = W_a$ and $W_j' = W_j$, $0 \le j \le M-1$ unless $j = a$, in which case $W_j' = w$. Also, $\lambda_{ra}(W_0, \cdots, W_M) = W_M$. 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 $S_t = \langle S_t, I_t, \delta_t, \lambda_t, O_t; T_t\rangle$ where T_t is arbitrary. S_t is the set of $M+2$ tuples $(W_0, \cdots, W_M, \hat{p})$, $W_j \in (\Sigma_2)^b$, where $W_0, \cdots, W_{M-1}$ are stored words, W_M is the output word, and $p \in \{0, 1, \cdots, M-1\}$ is the position of the read-write head. I_t is the set of triples $(s, \underline{e+2}, w)$ where $s \in \Sigma_2$ is the read-write command, $e \in \{-1, 0, 1\}$ is the incremental head position command and $w \in (\Sigma_2)^b$ is the input word. O_t is the set of pairs (v, p') of output words $v \in (\Sigma_2)^b$ and new head position $p' \in \{0, 1, \cdots, M-1\}$. Also,

$$\delta_t(W_0, \cdots, W_M, \hat{p}; s, \underline{e+2}, w) = \begin{cases} (W_0, \cdots, W_M, \hat{p}) & s = 0, \\ (W_0', \cdots, W_{M-1}', W_M', \hat{p}'), & s = 1, \end{cases}$$

where $W_M' = W_p'$ and $W_j' = W_j$, $0 \le j \le M-1$, unless $j = p'$, in which case $W_j = w$. Also,

$$p' = \begin{cases} 0 & p + e \le 0, \\ M-1 & p + e \ge M-1, \\ p+e & \text{otherwise}, \end{cases}$$

and $\lambda_t(W_0, \cdots, W_M, \hat{p}) = (W_M, \hat{p})$. The tape unit reads, and if $s = 1$, increments the head position, writes a new word at position p', and reads both p' and the old word at this position.

A *drum* storage device is a sequential machine $S_d = \langle S_d, I_d, \delta_d, \lambda_d, O_d; T_d\rangle$ where T_d is arbitrary. S_d is the set of $M+2$ tuples $(W_0, \cdots, W_{M-1}, \beta', \hat{p})$, $W_j \in (\Sigma_2)^b$, where $W_0, \cdots, W_{M-1}$ are stored words, β' is the output bit, and $p \in \{0, 1, \cdots, M-1\}$ is the position of the read-write heads on the drum. I_d is the set of triples $(s, \underline{h}, \beta)$ of read-write command $s \in \Sigma_2$, head address $h \in \{0, 1, \cdots, b-1\}$, b a power of 2, and input bit $\beta \in \Sigma_2$. O_d is the set of pairs $(\beta', \hat{p})$, $\beta' \in \Sigma_2$, and $p \in \{0, 1, \cdots, M-1\}$. Also,

$$\delta_d(W_0, \cdots, W_{M-1}, \beta', \hat{p}; s, \underline{h}, \beta) = \begin{cases} (W_{M-1}, W_0, \cdots, W_{M-2}, W_{0h}, p') & s = 0, \\ (W_0' \cdots, W_{M-1}', W_{0h}, \hat{p}') & s = 1, \end{cases}$$

where $p' = (p+1) \bmod M$. Here $W_j' = W_{(j-1) \bmod M}$ unless $j = p$, in which case $W_{ji}' = W_{(j-1) \bmod M, i}$, $i \ne h$, and $W_{jh}' = \beta$. (W_{ji} is the ith component of the jth word.) In addition, $\lambda_d(W_0, \cdots, W_{M-1}, \beta', \hat{p}) = (\beta', \hat{p})$. Thus the drum advances one word in each cycle and the device reads and writes onto the track under the hth head if $s = 1$.

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 they usually access data in blocks. 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 both in operation and in characteristics, and the drum storage device is an adequate model for disk units.

THEOREM 3. *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 2-input Boolean functions and let* M *and* b *be powers of 2 with* $M \geq 4$. *Then,*

$$(S - b) \leq C_\Omega^*(S_{ra}) \leq (6S + 2M),$$
$$\lceil \log_2 M \rceil \leq D_\Omega^*(S_{ra}) \leq (\lceil \log_2 M \rceil + \lceil \log_2 \log_2 M \rceil + 1),$$
$$(S - b) \leq C_\Omega^*(S_t) \leq 4(S + 3M),$$
$$\lceil \log_2 M \rceil \leq D_\Omega^*(S_t) \leq (\lceil \log_2 M \rceil + 7),$$
$$(b - 1) \leq C_\Omega^*(S_d) \leq (8b),$$
$$\lceil \log_2 b \rceil \leq D_\Omega^*(S_d) \leq (\lceil \log_2 b \rceil + \lceil \log_2 \log_2 b \rceil + 1),$$

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 with the upper bounds within a small multiplicative factor. The bounds on time complexity are tighter yet.

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 to simulate 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.

6. *Computation on General Purpose Machines*

In this section, we illustrate the use of Theorems 1 and 2 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.

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 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.

THEOREM 4. *Consider three general purpose computers with random access, tape, and drum storage devices, and fixed central processors. Let them execute* T_{ra}, T_t, *and* T_d *cycles to compute the finite functions* f_{ra}, f_t, *and* f_d, *respectively. Assume that the random access and tape units have* M *words of* b *bits each, and that the drum unit has* b_d *tracks, where* M, b, *and* b_d *are powers of* 2. *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* $M \geq 4$:

$$C_\Omega(f_{ra}) \leq (K_{ra} + 5S + 3M)T_{ra},$$

$$C_\Omega(f_t) \leq (K_t + 5(S + 2M))T_t,$$

$$C_\Omega(f_d) \leq (K_d + 8b_d)T_d,$$

$$D_\Omega(f_{ra}) \leq (\lceil \log_2 M \rceil + \lceil \log_2 \log_2 M \rceil + 1)T_{ra},$$

$$D_\Omega(f_t) \leq (\lceil \log_2 M \rceil + 7)T_t,$$

$$D_\Omega(f_d) \leq (\lceil \log_2 b_d \rceil + \lceil \log_2 \log_2 b_d \rceil + 4)T_d.$$

The second set of inequalities holds for large M *and* b_d. *Here* K_{ra}, K_t, *and* K_d *are constants which reflect the complexities of the central processors and* $S = Mb$ *is the storage capacity of the random access and tape storage devices.*

PROOF. The first set of inequalities follows from Theorems 1 and 3, where K_{ra}, K_t, and K_d are bounds on the combinational complexities of the central processors for the three machines. The second set of inequalities follows from Theorems 2 and 3 and the fact that the time complexities of the central processors are fixed. Q.E.D.

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 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 machine produce outputs through their controls and that their controls and tapes have equal length cycles and execute equal numbers of cycles.

THEOREM 5. *Let* TM_1 *be an on-line tape machine which computes* f_1 *and which has* M *b-bit words divided among* m *tapes. Let* TM_2 *be an off-line tape machine which computes* f_2, *which has* M *b-bit words 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 exist contsants* K_1 *and* K_2 *such that the following inequalities must be satisfied:*

$$C_\Omega(f_1) \leq [K_1 + 5(S + 2M)]T_1,$$

$$C_\Omega(f_2) \leq [K_2 + 5(nb + S + 2n + 2M)]T_2,$$

$$D_\Omega(f_1) \leq [\lceil \log_2 M/m \rceil + 7]T_1,$$

$$D_\Omega(f_1) \leq [\lceil \log_2 \max(n, M/m) \rceil + 7]T_2,$$

where $S = Mb$ *and* Ω *contains the Boolean primitives of fan-in* 2. *The second set of inequalities applies for large* M/m *and* n *when the* M *words are equally divided among the tapes. 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:

$$C_\Omega(f_1) \leq [K_1 + 5(b + 2)m(2T_1 + 1)]T_1,$$

$$C_\Omega(f_2) \leq [K_2 + 5(n(b + 2)) + (b + 3)m(2T_2 + 1)]T_2,$$

PROOF. The first two sets of inequalities follow directly from Theorems 1, 2, and 3, 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 the tape heads cannot reach all of these words from their starting positions if $T_1 < (M_1/m - 1)/2$, $T_2 < (M_2/m - 1)/2$ because, in the worst case, the words could be equally distributed among the tapes and the starting positions for the heads could be in the middle of each tape. Taking the converse of these inequalities, the result follows by invoking the first two inequalities. Q.E.D.

We conclude from Theorem 4 that functions cannot be computed on random access and tape machines unless their combinational complexities satisfy inequalities involving essentially a time-storage product. Other inequalities involving their time complexities and the product of time and the number of bits in a memory address must also be satisfied. These results suggest that storage can be exchanged for time, a common tenet of computing folklore.

The inequalities for drum machines in Theorem 4 involve only the number of drum tracks and time and suggest that much more time is required to compute functions when a drum is the principal storage medium. This is another of those commonly accepted notions.

In Theorem 5, the inequalities for multitape on-line machines are identical with those for 1-tape machines with the same storage capacity. Those for off-line machines differ in that the storage of the input tape must be added to that of the work tapes. These inequalities may be useful in studying the computation of recursive functions.

In the limit of large storage capacity, it is interesting to note the implications of the inequalities. Since any function of n variables can be computed on a random access machine by using the value of the variables as an address for the location containing the value of the function (i.e. by "table look-up"), functions can be computed in a number of cycles independent of their combinational complexities C. On tape machines with preset head positions, the story is different. Here the number of cycles must grow at least as fast as $\sqrt{C}$. The situation is more extreme yet on drums with fixed numbers of heads since the number of cycles must grow at least as fast as C. This suggests a hierachy of storage units which corresponds with the degree of flexibility in accessing stored data from these units.

7. *Computing Power*

The computing power $P_\Omega(S)$ of a sequential machine S with combinational complexity $C_\Omega^*(S)$ and cycle length τ is the rate at which S can do computational work, $P_\Omega(S) = C_\Omega^*(S)/\tau$. If two storage units have widely different computing powers, the more powerful unit may do the bulk of the work or be used inefficiently if it

depends on the less powerful unit for data. This often occurs between core and disk or drum units since the computing powers of the latter are proportional only to the number of tracks they contain, while the computing power of the core is proportional to its storage capacity. In fact, calculations of the upper bounds on computing power implied by Theorem 3 for some common commercial computer configurations show an order of magnitude difference between the computing power of core and disk. This might explain the bottlenecks noted between these devices by system programmers.

8. *Applications*

There are many applications for the inequalities derived in this paper, three of which are cited in this section. The combinational and time complexities of general purpose machines reflect the power of their machine language instruction sets. Therefore, these inequalities can be used to compare the performance of algorithms with different instruction sets, as well as that of algorithms over the same instruction set but which use different amounts of storage and time.

The inequalities can also be used to bound the complexities of functions for which algorithms exist. For example, let the language L be a subset of Σ^*, the free semigroup with identity generated by the finite set Σ. Then, the combinational complexity of the recognition function f_L of L ($f_L(x) = 1$ if $x \in L$ and 0 otherwise) when restricted to strings of length n, $C_\Omega({f_L}^n)$, can be overbounded for regular, LR(k), and context free languages (CFLs) using the storage-time product of algorithms which recognize them on tape or random access machines. The bounds are $A_1 n$, $A_2 n^2$, $A_3 n^5$, respectively, for constants A_1, A_2, A_3 since regular languages are recognized by finite state sequential machines, LR(k) languages are recognized in linear time on deterministic pushdown automata and CFLs can be recognized by Younger's algorithm [7]. It can be shown that no uniform bound on A_1, A_2, A_3 can be put for all regular, LR(k), and CF languages. Also, using the Lemma in the Appendix, a lower bound to $C_\Omega({f_L}^n)$ can be derived which is linear in n if L contains a string of length linear in n.

A third application of the inequalities can be made to quantum mechanical computers, machines which contain only binary logic elements and binary memory cells and in which binary signals are represented by energy levels in semiconductors. Here we let ΔE be the minimum separation between energy levels and let Δt be the minimum time to determine which of two levels an input to a logic element is in. Then, the Heisenberg uncertainty relation $\Delta E \Delta t \geq h/2\pi$ must be satisfied if an energy difference of ΔE is to be measured reliably in time Δt, where h is Planck's constant.

In solids, an energy difference of ΔE can be determined only with the expenditure of ΔE units of energy. Therefore, if energy E (joules) and time t (seconds) is available to compute f, we must have

$$C_\Omega(f) \leq XT \leq Et/\Delta E \Delta t \leq Et \times 10^{34}.$$

Since most Boolean functions in p variables have combinational complexity on the order of $2^p/p$ [6], we conclude that most Boolean functions with p variables cannot be computed in one hour with a kilowatt of power if $p \geq 160$.

9. *Conclusions*

The key results of this paper are the inequalities of Theorems 1 and 2 from which other results and conclusions follow. It is important to note that the inequalities apply to machines which can loop, branch, test for zero, etc., while the inequalities themselves are stated in terms of complexity measures on straight line algorithms with very limited instruction sets.

The inequalities of this paper may be useful in the study of compilers and interpreters. What is required are good lower bounds on the complexities of compiler and interpreter defined functions such as recognition and parsing functions.

The inequalities provide global information about the performance of algorithms and computing systems. As such they may also have application to operating systems.

Appendix

In the proof of Theorem 3, the following Lemma will be used. A function $f: \Sigma_2^n \to \Sigma_2$ is *dependent* on variable x_i if there exist values for x_j, $j \neq i$, $1 \leq j \leq n$, such that $f(x_1, \cdots, x_{i-1}, 1, x_{i+1}, \cdots, x_n) \neq f(x_1, \cdots, x_{i-1}, 0, x_{i-1}, 0, x_{i+1}, \cdots, x_n)$. Let $\lceil x \rceil$ denote the smallest integer greater than or equal to x and let ρ be the maximum number of variables on which elements of Ω are dependent (ρ is the *fan-in* of Ω). Then,

LEMMA. *Let $f: \Sigma_2^n \to \Sigma_2$ depend on each of its variables. Then,*

$$C_\Omega(f) \geq \lceil (n-1)/(\rho-1) \rceil, \qquad D_\Omega(f) \geq \lceil \log_\rho n \rceil.$$

PROOF. The proof of the first inequality 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. Q.E.D.

PROOF OF THEOREM 3. The object is to bound $C_\Omega^*(S)$ and $D_\Omega^*(S)$ for the three machines. We begin with lower bounds and recall that $\rho = 2$ in the conditions of Theorem 3.

Lower bounds to $C_\Omega^*(S)$ are obtained by lower bounding the complexities of δ_{ra}, δ_t, and δ_d. Define δ_{ra}', δ_t', and δ_d' by suppressing all but the $(M+1)$th components in the range of δ_{ra}, δ_t, δ_d. (The $(M+1)$th component is the output word for the random access and tape devices and the output bit for the drum device.) Let $f_{\text{OR}}: \Sigma_2^b \to \Sigma_2$ be the OR of b variables; then $C_\Omega(f_{\text{OR}}) = b - 1$. For the random-access and tape units form $f_{\text{OR}}(\delta_{ra}')$ and $f_{\text{OR}}(\delta_t')$. Each of these functions depends on the b components of each of the M stored words. Thus, $C_\Omega(f_{\text{OR}}(\delta_{ra}')) \geq Mb - 1$, $C_\Omega(f_{\text{OR}}(\delta_t')) \geq Mb - 1$ follow from the Lemma. But $C_\Omega(f_{\text{OR}}(\delta')) \leq C_\Omega(f_{\text{OR}}) + C_\Omega(\delta')$, from which the bounds to $C_\Omega^*(S_{ra})$ and $C_\Omega^*(S_t)$ follow. The lower bound to $C_\Omega^*(S_d)$ is a consequence of the fact that δ_d' depends on the b variables under the read-write head.

The lower bounds to $D_\Omega^*(S_{ra})$, $D_\Omega^*(S_t)$, and $D_\Omega^*(S_d)$ follow from the observation that each component of δ_{ra}' and δ_t' depends on the corresponding M components of stored words and that δ_d' depends on the b components under the read-write head.

Upper bounds are developed by construction. We now define the straight-line

algorithm $A(m)$ which will be used in later constructions. Let $A(m)$ have m input variables $x_1, \cdots, x_m$ and 2^m outputs $l_j(x_1 \cdots x_m)$, $0 \leq j \leq 2^m - 1$. Let $l_j(x_1, \cdots, x_m) = x_1^{c_1} \cdot x_2^{c_2} \cdot \cdots \cdot x_m^{c_m}$ where $\underline{j} = (c_1, c_2, \cdots, c_m)$, $\cdot$ denotes AND, $x^1 = x$ and $x^0 = \bar{x}$, the INVERSE of x. The Boolean functions l_j, $0 \leq j \leq 2^m - 1$, are minterms in $x_1, \cdots, x_m$, and $l_j(x_1, \cdots, x_m) = 1$ for $\underline{j} = (x_1, \cdots, x_m)$ and $l_j(x_1, \cdots, x_m) = 0$, otherwise. Clearly the time complexity of $A(m)$ is $\lceil \log_2 m \rceil$. The combinational complexity of $A(m)$ is bounded above by $2(2^m - 1)$ since each of the minterms of $A(m)$ can be constructed from a minterm of $A(m-1)$ by the addition of one operation.

Since the output functions of each of the storage devices are projection operators, they have zero combinational complexity.

The transition function δ_{ra} can be realized by $W'_{ji} = W_{ji} \cdot \overline{(l_j(\underline{a}) \cdot s)} + w_i \cdot (l_j(\underline{a}) \cdot s)$ for $0 \leq j \leq M - 1$, $1 \leq i \leq b$, and $W'_{Mi} = \sum_{j=0}^{M-1} l_j(\underline{a}) \cdot W_{ji}$, $1 \leq i \leq b$. Here $+$ and $\sum$ denote OR and $\overline{}$ denotes INVERSE. Then, from these equations and the definition of $A(m)$, we have $C_\Omega^*(S_{ra}) \leq 3Mb + (2M-1)b + 3M - 2 < 5Mb + 3M$ and $D_\Omega^*(S_{ra}) \leq \log_2 M + \log_2 \log_2 M + 1$ if $M \geq 4$.

The transition function δ_t can be realized by the following formulas:

$$\hat{p}_i' = (\hat{p}_{i+1} \cdot l_1(\underline{e+2}) + \hat{p}_i \cdot l_2(\underline{e+2}) + \hat{p}_{i-1} \cdot l_3(\underline{e+2})) \cdot s + \hat{p}_i \cdot \bar{s}, \quad 2 \leq i \leq M-1,$$

$$\hat{p}_1' = (\hat{p}_1 \cdot (l_1(\underline{e+2}) + l_2(\underline{e+2})) + \hat{p}_2 \cdot l_1(\underline{e+2})) \cdot s + \hat{p}_1 \cdot \bar{s},$$

$$\hat{p}_M' = (\hat{p}_M \cdot (l_2(\underline{e+2}) + l_3(\underline{e+2})) + \hat{p}_{M-1} \cdot l_3(\underline{e+2})) \cdot s + \hat{p}_M \cdot \bar{s},$$

$$W'_{ji} = W_{ji} \cdot \overline{(\hat{p}_j' \cdot s)} + w_i \cdot (\hat{p}_j' \cdot s), \quad 0 \leq j \leq M-1, \ 1 \leq i \leq b,$$

$$W'_{Mi} = (\textstyle\sum_{j=0}^{M-1} W_{ji} \cdot \hat{p}'_{j+1}) \cdot s + W_{mi} \cdot \bar{s}, \quad 1 \leq i \leq b.$$

Note that $A(2)$ is used to generate $l_j(\underline{e+2})$, $1 \leq j \leq 3$. Then it follows by simple enumeration and addition that

$$C_\Omega^*(S_t) \leq 4 + 8M - 2 + 5Mb + M \leq 5Mb + 10M \quad \text{for} \quad M \geq 4.$$

Also,

$$D_\Omega^*(S_t) \leq \max(1+5+3, 1+5+1+\lceil \log_2 M \rceil) = \lceil \log_2 M \rceil + 7 \quad \text{if} \quad M \geq 4.$$

The transition function δ_d can be realized by the following formulas:

$$\hat{p}_i' = \hat{p}_{(i-1) \bmod M} \qquad 1 \leq i \leq M$$

$$W'_{ji} = W_{j-1,i} \qquad 1 \leq j \leq M-1, \ 0 \leq i \leq b-1$$

$$W'_{0i} = W_{M,i} \cdot \overline{(l_i(\underline{h}) \cdot s)} + \beta \cdot (l_i(\underline{h}) \cdot s), \qquad 0 \leq i \leq b-1$$

$$W_{0h} = \textstyle\sum_{i=0}^{b-1} W'_{0i} \cdot l_i(\underline{h})$$

It follows that $C_\Omega^*(S_d) \leq 2(b-1) + 4b + 2b - 1 \leq 8b$ and that $D_\Omega^*(S_d) \leq \log_2 b + \lceil \log_2 \log_2 b \rceil + 1$. Q.E.D.

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

Fiduccia of S.U.N.Y. at Stonybrook are acknowledged as well as the contribution of R. Waters of Brown University to Theorem 2. The author also acknowledges a careful reading of the manuscript by E. Lamagna of Brown University.

REFERENCES

1. MINSKY, M. Form and content in computer science. *J. ACM 17*, 2 (Apr. 1970), 197–215.
2. SAVAGE, J. E. Three measures of decoder complexity. *IBM J. Res. Devel. 14*, 4, (July 1970), 417–425.
3. SAVAGE, J. E. The complexity of decoders—Part II: Computational work and decoding time. *IEEE Trans. on Information Theory, IT-17*, 1, (Jan., 1971). 77–85.
4. WINOGRAD, S. On the time required to perform addition. *J. ACM 12*, 2 (Apr. 1965), 277–285.
5. WINOGRAD, S. On the time required to perform multiplication. *J. ACM 14*, 4 (Oct. 1967), 793–802.
6. HARRISON, M. A. *Introduction to Switching and Automata Theory*. McGraw-Hill, New York, 1965, Ch. 6, 7.
7. YOUNGER, D. H. Recognition and parsing of context-free languages in time n^3. *Information and Control 10*, 2, (1967), 189–208.

RECEIVED SEPTEMBER 1970; REVISED NOVEMBER 1971