INFORMATION AND CONTROL 10, 189–208 (1967)

Recognition and Parsing of Context-Free Languages in Time n^3*

DANIEL H. YOUNGER

General Electric Research and Development Center, Schenectady, New York

A recognition algorithm is exhibited whereby an arbitrary string over a given vocabulary can be tested for containment in a given context-free language. A special merit of this algorithm is that it is completed in a number of steps proportional to the "cube" of the number of symbols in the tested string. As a byproduct of the grammatical analysis, required by the recognition algorithm, one can obtain, by some additional processing not exceeding the "cube" factor of computational complexity, a parsing matrix—a complete summary of the grammatical structure of the sentence. It is also shown how, by means of a minor modification of the recognition algorithm, one can obtain an integer representing the ambiguity of the sentence, i.e., the number of distinct ways in which that sentence can be generated by the grammar.

The recognition algorithm is then simulated on a Turing Machine. It is shown that this simulation likewise requires a number of steps proportional to only the "cube" of the test string length.

LIST OF SYMBOLS

$\rightarrow$	"can be rewritten as"
$\Rightarrow$	"generates"
T	terminal vocabulary
N	nonterminal vocabulary
S	special nonterminal symbol
L	language
G	grammar
φ, ψ	strings over $N \cup T$
$S_t = s_1 \cdots s_n$	test string

* Some of this work was presented at the Seventh Annual Symposium on Switching Circuit Theory and Logical Design, Berkeley, California, October 26–28, 1966.

R	recognition matrix with typical entry $r(i, j, k)$
M	ambiguity matrix with typical entry $m(i, j, k)$
P	parsing matrix with typical entry $p(i, j, k)$
G_t	matrix of terminating rules of grammar with typical entry $g_t(k, l)$
G_c	matrix of binary constituent rules of grammar with typical entry $g_c(k, k_1, k_2)$
i, t	indices representing lengths of substrings of S_t
j	index representing position in S_t of first substring symbol
k, k_1, k_2	indices over set of nonterminal symbols
l	index over set of terminal symbols
n	length of S_t
q	number of nonterminal symbols
s	number of terminal symbols

I. INTRODUCTION

An important criterion for the choice of a language for use in computer programming is the ease with which a proposed sentence can be tested for proper formation. This paper exhibits a "recognition" algorithm which works for any context-free language; a special merit of this algorithm is that it is completed in a number of steps proportional to the "cube" of the number of symbols in the tested string. This efficiency encourages the further search for languages in the context-free class that are suitable for practical programming.

This algorithm involves the making of a detailed grammatical analysis of the tested string. As a by-product of this analysis, many additional facts about the grammatical structure of the string can be derived. By modifying the procedure in a small way, one can obtain an integer representing the ambiguity of the sentence, i.e., the number of distinct ways in which that sentence can be generated by the grammar. On the other hand, having successfully recognized a string as a sentence, one can obtain, by some additional processing not exceeding the "cube" factor of computational complexity, a complete summary of the grammatical structure of that sentence. Such a summary is called a parsing matrix.

Following the development of these concepts, the paper relates the recognition algorithm to a classification of languages based on the number of steps required for recognition by a Turing Machine. It is

perhaps surprising that the program of the algorithm for such a simple computer model as the Turing Machine requires a number of steps proportional to only the "cube" of the test string length.

Within the context of natural language analysis, there have been numerous parsing algorithms proposed; those known to the author are the Predictive Analyzer (Kuno and Oettinger, 1963; Kuno, 1965), Cocke's Parsing Algorithm (Hays, 1962), a parsing algorithm by Sakai (1962), and a parsing procedure of Kay (1963), the last of which is most closely related to the one presented here. For none of these algorithms is a definite estimate given of the time required for the analysis. A survey of the relative efficiencies, as determined empirically, of proposed context-free grammar recognizers is given by Griffiths and Petrick (1965).[1] On the other hand, whereas the algorithms presented in this paper are intended to be minimum in time required, an algorithm of Hartmanis, Lewis, and Stearns (1965) is noteworthy in that it requires, for an input test string of length n, an amount of intermediate storage proportional to $(\log_2 n)^2$, the least of any known algorithm.

Before presenting the recognition algorithm a short review will be made of elementary abstract language concepts; see Chomsky (1963) for an introduction to abstract languages.

II. BASIC CONCEPTS OF CONTEXT-FREE LANGUAGES

A *vocabulary* is a given set of symbols comprised of two disjoint subsets, T and N, called the *terminal* and *nonterminal* vocabularies, respectively. The elements of T, terminal symbols, are denoted by lowercase letters, or numerals, e.g., s, 0, 1. Nonterminal symbols are denoted by capital letters, such as A, B, etc.; N contains a special symbol, denoted exclusively by S. A *string over* $N \cup T$ is a finite sequence of symbols of $N \cup T$. The *length* of a string is the number of elements which it contains. A *language* L is any set of strings over T; each string contained in L is called a *sentence* of L.

A *grammar* G is a finite set of rules which recursively specifies the sentences of a language. For a *context-free* grammar, all *rules* are of the form $A \rightarrow \psi$, where A is a single nonterminal symbol and ψ is a nonnull string over $N \cup T$. By the rule $A \rightarrow \psi$, a string over $N \cup T$ of the form $\varphi A \varphi'$ *generates* string $\varphi\psi\varphi'$, where φ and φ' are strings over $N \cup T$. If, for a sequence $\varphi_1, \cdots, \varphi_n$ of strings over $N \cup T$, φ_i generates φ_{i+1} ac-

[1] The author has learned that Kasami has independently shown that context-free languages are recognizable in n^3 time, See Kasami (1966).

cording to a rule of G, for each i between 1 and $n - 1$, then it is said that φ_1 *generates* φ_n *by* G, written $\varphi_1 \Rightarrow \varphi_n$. The *language* $L(G)$ *generated by* G is the set of strings over T that S (i.e., the string of length one consisting of the special nonterminal symbol S) generates. A language generated by a context-free grammar is called a *context-free language.*

If all rules of a grammar G are of one of the generic forms $A \rightarrow BC$ or $A \rightarrow a$, where A, B, and C are arbitrary nonterminal symbols and a is an arbitrary terminal symbol, then G is said to be a *normal grammar.* For a normal grammar, rules of the form $A \rightarrow a$ are called *terminating rules,* those of the form $A \rightarrow BC$ are called *binary constituent rules.* Given a context-free grammar G, it is easy to reduce it to a normal grammar G' such that $L(G) \Rightarrow L(G')$. An example of this reduction follows: a proof is given by Chomsky (1959).

$$
\begin{aligned}
&G\text{:}\\
&\qquad S \rightarrow SBS\\
&\qquad S \rightarrow s\\
&\qquad B \rightarrow BB\\
&\qquad B \rightarrow BS\\
&\qquad B \rightarrow 0\\
&\qquad B \rightarrow 1
\end{aligned}
\tag{1}
$$

All the rules of this grammar are of the form required for a normal grammar except the first. This rule can be replaced by two rules:

$$
\begin{aligned}
S &\rightarrow SA\\
A &\rightarrow BS
\end{aligned}
$$

where A is a newly created nonterminal symbol, thereby yielding a normal grammar G':

Binary constituent rules	Terminating rules
$S \rightarrow SA$	$S \rightarrow s$
$A \rightarrow BS$	$B \rightarrow 0$
$B \rightarrow BB$	$B \rightarrow 1$
$B \rightarrow BS$	

(2)

Since the two rules $S \rightarrow SA$ and $A \rightarrow BS$ constitute a normal form representation of the rule $S \rightarrow SBS$, the normal grammar is not essen-

tially different from the context-free grammar from which it was obtained.

III. THE RECOGNITION MATRIX

Assume that a context-free grammar is given; assume further that this grammar has been reduced to normal form. This normal grammar is presented in the form of two sets: a set of ordered pairs that represent the terminating rules, and a set of ordered triples that represent the binary constituent rules. Also given is a string of symbols over the terminal vocabulary of the language. The task is to recognize whether or not this string is part of the language generated by the given grammar.

DEFINITION. An algorithm *recognizes* a given language L if, given any string over the terminal vocabulary of that language, it renders a correct decision after a finite number of steps to accept or to reject that string as a sentence of L.

This recognition algorithm will be framed in terms of a recognition matrix. This matrix lists, for each substring of the test string S_t, all the symbols in N which generate that substring. In particular, this matrix lists the symbols which generate the full string S_t: if special symbol S is contained in this list, the string S_t is then accepted as a sentence in the language; if not, it is rejected. Such a matrix contains not only information relevant to recognition, but also much additional information about the grammatical structure of that sentence. In the first part of this paper, we use this matrix only for recognition; later, however, we use it as the first stage in the construction of a parsing matrix.

The test string is of the form

$$S_t = s_1 s_2 \cdots s_n, \quad \text{with} \quad s_j \in T, \qquad j = 1, \cdots, n$$

The substring of length i whose first symbol is at position j is the string $s_j s_{j+1} \cdots s_{j+i-1}$.

For the purpose of this discussion, the elements of N, the nonterminal vocabulary of the normal grammar,[2] will be written as A_k, where $A_1 = S$

[2] This is generally a larger vocabulary than that for the context-free grammar from which the normal grammar was derived. The added symbols are of two types. The first are like the symbol C which enables the rule $A \to Bc$ to be replaced in the normal grammar by $A \to BC$ and $C \to c$. The added symbols of the second type are those that permit a rule of the form $A \to \psi$, where ψ is of length $l > 2$, to be replaced by $l - 1$ binary constituent rules.

is the special symbol in N, and index k runs from 1 to q (the number of elements in N).

DEFINITION. Given a normal grammar G over a vocabulary $T \cup N$, and given a string $S_t = s_1 s_2 \cdots s_n$ over T, a *recognition matrix* R for S_t in $L(G)$ is a three-dimensional binary matrix $R = [r(i, j, k)]_{nnq}$[3], where n is the length of S_t and q is the number of symbols in N, such that $r(i, j, k) = 1$ if and only if the substring of S_t of length i whose first symbol is s_j is generable by G from the kth symbol A_k in N, i.e. $r(i, j, k) = 1$ if and only if $A_k \Rightarrow s_j s_{j+1} \cdots s_{j+i-1}$.

By this definition, the entry $r(n, 1, 1)$ of R is 1 if and only if $A_1 \Rightarrow s_1 s_2 \cdots s_n$, i.e. if and only if S_t is in $L(G)$. That this entry is equal to 1 is the *recognition criterion* for S_t in $L(G)$.

To illustrate the recognition matrix recall the normal grammar specified by (2). Take as a test string $S_t = s0s10s$. The recognition matrix for S_t in $L(G)$ is the following:

length i	1	2	3	4	5	6
$S_t =$	s 0 s 1 0 s					
position j	1 2 3 4 5 6	1 2 3 4 5	1 2 3 4	1 2 3	1 2	1
symbol A_k $S(=A_1)$	1 . 1 . . 1	. . . . .	1 . . .	. . 1	. .	1
$A(=A_2)$	. . . . . .	. 1 . . 1	. . . 1	. . .	. 1	.
$B(=A_3)$	. 1 . 1 1 .	. 1 . 1 1	. 1 . 1	. 1 .	. 1	.

(3)

In this recognition matrix, the indices i, j, and k have the following meanings:

i = length of the substring,

j = position in the test string of the first substring symbol,

k = index over the nonterminal symbols of the normal grammar.

Each substring of the test string S_t is specified by a pair of indices i, j. Since pairs of index values for which $i + j > n + 1$ do not correspond to substrings, the corresponding matrix entries are omitted. Other zero entries have been indicated by a · for the sake of clarity. Furthermore, the separation of the planes of the matrix for different substring lengths i is done for the sake of presentation. Either of the other coordinates could have been shown separated. The significance of a typical entry of the above matrix R is as follows: consider the 1 entry at position

[3] By this notation is meant that indices i and j each run from 1 to n, and index k runs from 1 to q.

(4, 2, 3). That this entry is 1 signifies that the substring of S_t of length 4 starting at position 2 is generable from $A_3 = B$ by means of the grammar. The substring of length 4 starting at position 2 for $S_t = s0s10s$ is 0s10, and this can be generated from B as follows: $B \Rightarrow BB \Rightarrow BSB \Rightarrow BSBB \Rightarrow 0s10$.

IV. AN ALGORITHM FOR RECOGNITION OF CONTEXT-FREE LANGUAGES

The *algorithm for recognition* is an iterative procedure for computing the entries of R. The entries for $i = 1$ are obtained directly from the terminating rules of G. Specifically, for $i = 1$, $r(1, j, k) = 1$ if $A_k \Rightarrow s_j$, i.e., if $A_k \rightarrow s_j$ is a terminating rule of G, and is zero otherwise. Having employed the terminating rules to find the $i = 1$ plane, the remaining entries are found by employing the binary constituent rules. For example, the (2, 2, 2) entry of the recognition matrix is found to be 1 by the following reasoning. The nonterminal with associated index 2 is A. Hence the first step in a generation from A must employ a binary constituent rule with A on the left. From the grammar, it is seen that the only such rule is $A \rightarrow BS$. Since A is to generate a terminal string of length 2, then B and S must each generate strings of length 1. To see whether $B(= A_3)$ generates such a string the (1, 2, 3) entry is checked; to check $S(= A_1)$, the (1, 3, 1) entry is examined. Since both of these are 1, then $r(2, 2, 2) = 1$.

In general, we have the following

Iterative Procedure for Obtaining $R = [r(i, j, k)]_{nnq}$

(a) $r(1, j, k) = 1$ if $A_k \rightarrow s_j$ is a terminating rule of G, and $r(1, j, k) = 0$, otherwise;

(b) if entries have been computed for all $i' < i$, where i is an index greater than 1, then

$$r(i, j, k) = \sum_{(k_1, k_2) \in P_k} \sum_{t=1}^{i-1} r(t, j, k_1) r(i - t, j + t, k_2) \quad \text{(boolean sum)} \quad (4)$$

where P_k is the set of ordered pairs (k_1, k_2) such that $A_k \rightarrow A_{k_1} A_{k_2}$ is a rule of G.

In other words, the above boolean summation indicates that $r(i, j, k) = 1$ if for some rule $A_k \rightarrow A_{k_1} A_{k_2}$ in G and some t between 1 and $i - 1$, the previously computed entries $r(t, j, k_1)$ and $r(i - t, j + t, k_2)$ are both equal to 1; $r(i, j, k) = 0$, otherwise.

Proof of Validity of Iterative Procedure. Suppose that the procedure yields entries of the recognition matrix that are in accord with its definition for all $i < i_0$. To satisfy the definition for $i = i_0$, $r(i_0, j, k)$ should equal one just in case $A_k \Rightarrow s_j s_{j+1} \cdots s_{j+i_0-1}$. This is certainly true when $i_0 = 1$. For $i_0 > 1$, this holds if and only if there exists an index t between 1 and $i_0 - 1$ such that for some rule $A_k \rightarrow A_{k_1} A_{k_2}$ of the normal grammar, it follows that $A_{k_1} \Rightarrow s_j s_{j+1} \cdots s_{j+t-1}$ and $A_{k_2} \Rightarrow s_{j+t} s_{j+t+1} \cdots s_{j+i_0-1}$. The latter pair of relations can hold if and only if $r(t, j, k_1) = 1$ and $r(i_0 - t, j + t, k_2) = 1$, by the inductive hypothesis. In other words, $r(i_0, j, k) = 1$ if and only if the boolean sum (4) for $i = i_0$ is equal to 1.

Whereas the rules of the grammar presented as in (2) are quite suitable when the recognition matrix is calculated by hand, a more appropriate format must be chosen for automatic processing. Compatibility with the recognition matrix format suggests that the grammar be presented as a binary matrix. There are two matrices needed, one for the terminating rules, the other for binary constituent rules. The terminating rules are each an ordered pair of symbols, the first a nonterminal, the second a terminal. The rules can thus be presented as a binary matrix

$$G_t = [g_t(k, l)]_{qs}$$

where q is the number of nonterminal symbols and s is the number of terminal symbols in the vocabulary. An entry $g_t(k, l) = 1$ if $A_k \rightarrow a_l$ is a terminating rule of G, and $g_t(k, l) = 0$ otherwise. The binary constituent rules are ordered triples, each element of which is a symbol of N. They can be presented as a matrix $G_c = [g_c(k, k_1, k_2)]_{qqq}$. An entry $g_c(k, k_1, k_2) = 1$ if $A_k \rightarrow A_{k_1} A_{k_2}$ is a rule of G; otherwise $g_c(k, k_1, k_2) = 0$.

For example, recall the grammar specified by (2); the terminating rules can be represented by

$$G_t = \begin{array}{c|cc} k \backslash l & 0 & 1 \\ \hline S & . & . \\ A & . & . \\ B & 1 & 1 \end{array} \qquad \begin{array}{c} s \\ \hline 1 \\ . \\ . \end{array} \tag{5}$$

Similarly, the binary constituent rules are represented by the binary matrix:

$$G_c = \begin{array}{c|c|ccc|ccc|ccc} k & & S & & & A & & & B & & \\ \hline k_2 & & S & A & B & S & A & B & S & A & B \\ \hline k_1 & S & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ & A & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ & B & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & 1 & \cdot & 1 \end{array} \tag{6}$$

Using the matrix format for the grammar, the equation (4) used in calculating the entries of R iteratively can be rewritten as

$$r(i, j, k) = \sum_{t=1}^{i-1} \sum_{k_1=1}^{q} \sum_{k_2=1}^{q} g_c(k, k_1, k_2) r(t, j, k_1) r(i - t, j + t, k_2) \tag{7}$$

$$i > 1 \qquad \text{(boolean sum)}$$

Now we are in a position to estimate the number of elementary operations required to compute R. Let $T(R)$ be the number of boolean products required in finding all entries of R, with $T(r(i, j, k))$ the number required for the entry $r(i, j, k)$. For simplicity, the operations required to compute the $i = 1$ plane are omitted.

From (7) we get

$$T(r(i, j, k)) = (i - 1)q^2 \tag{8}$$

$$\begin{aligned} T(R) &= \sum_{\substack{i \geqq 2, j \geqq 1 \\ i+j \leqq n+1}}^{n} \sum_{k=1}^{q} T(r(i, j, k)) \\ &= \sum_{\substack{i \geqq 2, j \geqq 1 \\ i+j \leqq n+1}}^{n} (i - 1)q^3 = \tfrac{1}{6}q^3 n(n^2 - 1) < \tfrac{1}{6}q^3 n^3 \end{aligned} \tag{9}$$

Since the computation of all entries of the recognition matrix can be accomplished by finding $\frac{1}{6}q^3n^3$ boolean products, it follows that the recognition of an arbitrary context-free language can be accomplished by evaluting $\frac{1}{6}q^3n^3$ boolean products.[4]

In summary, the recognition algorithm is the following:

[4] In a personal communication, A. McKellar has pointed out that a general algorithm of Gilbert and Moore (1959) for assigning an optimum variable-length binary encoding of an alphabet of n symbols requires $\frac{1}{6}n(n^2 - 1)$ operations, a figure suggestive of a parallelism between that algorithm and the one given here. Indeed, although the objectives of the two algorithms are quite different, both employ a "bottom-up" approach, which accounts for the similarity in their computational complexities.

Recognition Algorithm: Find iteratively all entries of the recognition matrix for S_t in $L(G)$. If the $(n, 1, 1)$ entry of this matrix is 1, accept the string; if it is 0, reject the string as a sentence in $L(G)$.

V. THE AMBIGUITY OF A SENTENCE

The method of recognition can readily be extended to finding the ambiguity of a sentence in any context-free language. The *ambiguity* of a sentence with respect to a grammar G is the number of distinct ways in which it can be generated from S by G. A sentence is said to be *unambiguous* if it has ambiguity equal to 1; it is *ambiguous* if it has ambiguity greater than 1; the ambiguity is found exactly as the recognition was determined, with the distinction that ordinary arithmetic is used in computing the ambiguity rather than boolean arithmetic.

DEFINITION. Given a normal grammar G over a vocabulary $T \cup N$, and given a string S_t over T, the *ambiguity matrix* for S_t in $L(G)$ is a three-dimensional integer-valued matrix $M = [m(i, j, k)]_{nnq}$, where n is the length of S_t and q is the number of symbols in N, such that $m(i, j, k)$ is equal to the number of distinct ways in which the substring of S_t of length i whose first symbol is at position j is generated from the kth symbol A_k in N by grammar G.

From the definition of the ambiguity matrix, it follows that the ambiguity of a string S_t in $L(G)$ is equal to $m(n, 1, 1)$. If $m(n, 1, 1) = 0$, then S_t is not in $L(G)$.

Iterative Procedure for Obtaining $M = [m(i, j, k)]_{nnq}$

(a) $m(1, j, k) = 1$ if $A_k \rightarrow s_j$ is a terminal rule of G, and $m(1, j, k) = 0$, otherwise;

(b) if entries have been computed for all $i' \leqq i$, where i is an index greater than 1, then

$$m(i, j, k) = \sum_{(k_1, k_2) \epsilon P_k} \sum_{t=1}^{i-1} m(t, j, k_1) m(i - t, j + t, k_2) \qquad (10)$$

(arithmetic sum)

where P_k is the set of ordered pairs (k_1, k_2) such that $A_k \rightarrow A_{k_1} A_{k_2}$ is a rule of G.

The proof of the validity of this procedure is, with appropriate changes from boolean to ordinary arithmetic, the same as for the recognition matrix. To illustrate the procedure, consider again the grammar G given by (2) and the sequence $s_t = s0s10s$. The ambiguity matrix for

S_t in $L(G)$ is as follows:

length i		1	2	3	4	5	6
	$S_t =$	s 0 s 1 0 s					
position j		1 2 3 4 5 6	1 2 3 4 5	1 2 3 4	1 2 3	1 2	1
symbol A_k	$S(=A_1)$	1 . 1 . . 1	. . . . .	1 . . .	. . 1	. .	4
	$A(=A_2)$	. . . . .	. 1 . . 1	. . . 1	. . .	. 3	.
	$B(=A_3)$	. 1 . 1 1 .	. 1 . 1 1	. 1 . 1	. 2 .	. 5	.

(11)

The ambiguity of S_t in $L(G)$ is equal to 4, the value of $m(n, 1, 1)$

VI. THE PARSING MATRIX

A *parse* of a sentence in a language $L(G)$ is a description of how the sentence is generated by G; this description is generally in the form of a generation tree. There is for an ambiguous sentence more than one generation tree. In fact, there exist grammars G for which some sentences in the language generated by G have associated with them a number of distinct generation trees which is an exponential function of sentence length. An example for which this is true is any sentence from the language whose grammar consists of the two rules $S \rightarrow SS$ and $S \rightarrow s$.

Even if it were feasible to exhibit all the generation trees of a highly ambiguous sentence, that set of trees would not convey a great deal of information to the analyst. What is needed is a more compact form which summarizes the grammatical structure. A form which seems suited is the recognition matrix. There is a drawback to the recognition matrix, however: it does contain some information which is irrelevant. Specifically, it contains nonzero entries for nonterminal symbols that generate a given substring, when in fact no such nonterminal generates that substring in a derivation of the sentence. Fortunately, such irrelevant non-zero entries can be eliminated easily. This is done by working back down from S, retaining only entries that are generated by S. Specifically, we have the following definition:

DEFINITION. Given a normal grammar G over a vocabulary $T \cup N$, and given a string $S_t = s_1 s_2 \cdots s_n$ over T, a *parsing matrix* P for S_t in $L(G)$ is a three-dimensional binary matrix $P = [p(i, j, k)]_{nnq}$, where n is the length of S_t and q is the number of symbols in N, such that $p(i, j, k) = 1$ if and only if for nonterminal symbol A_k

1. $A_k \Rightarrow s_j s_{j+1} \cdots s_{j+i-1}$, and

2. either $S(=A_1) \Rightarrow s_1 \cdots s_{j-1}A_k s_{j+i} \cdots s_n$ or $i, j, k = n, 1, 1$.

As an example, consider again the grammar given by (2) and the string $s0s10s$. The parsing matrix is the following:

length i		1	2	3	4	5	6
	$S_t =$	s 0 s 1 0 s					
position j		1 2 3 4 5 6	1 2 3 4 5	1 2 3 4	1 2 3	1 2	1
symbol A_k	$S(=A_1)$	1 . 1 . . 1	. . . . .	1 . . .	. . 1	. .	1
	$A(=A_2)$	. . . . . .	. 1 . . 1	. . . .	. . .	. 1	.
	$B(=A_3)$	. 1 . 1 1 .	. 1 . 1 .	. 1 . .	. 1 .	. .	.

(12)

Given a recognition matrix, the entries in the parsing matrix (in particular, the parsing matrix just exhibited) are found by the following procedure:

Iterative Procedure for Obtaining $P = [p(i, j, k)]_{nnq}$ *from* $R = [r(i, j, k)]_{nnq}$

(a) $p(n, 1, 1) = r(n, 1, 1)$;

(b) if $p(i, j, k) = 1$, then for rules in G the form $A_k \rightarrow A_{k_1}A_{k_2}$ such that $r(t, j, k_1)r(i - t, j + t, k_2) = 1$ for some t, $1 \leqq t \leqq i - 1$, set $p(t, j, k_1)$ and $p(i - t, j + t, k_2)$ each equal to 1; all entries not equal to 1 by this rule are equal to zero.

Proof. Suppose that the procedure yields entries of the parsing matrix in accord with its definition for all $i > i_0$; we wish to show as a consequence that the definition is satisfied for $i = i_0$. This is certainly true for $i_0 = n$. For $i_0 < n$, suppose that the procedure yields an entry $p(i_0, j_0, k_0) = 1$. Then there must exist a rule $A_k \rightarrow A_{k_1}A_{k_2}$ (with either $k_1 = k_0$ or $k_2 = k_0$) and a nonzero entry $p(i, j, k)$ for $i > i_0$ from which the 1 entry in $p(i_0, j_0, k_0)$ is generated. By hypothesis, the entry $p(i, j, k)$ is in accord with the parsing matrix definition, i.e., $S \Rightarrow s, \cdots s_{j-1}\ A_k\ s_{j+i} \cdots s_n$. Hence $S \Rightarrow s, \cdots s_{j-1}\ A_{k_1}A_{k_2}\ s_{j+i} \cdots s_n$, from which it follows that the entry $p(i_0, j_0, k_0) = 1$ is in accord with the parsing matrix definition. Reversing the argument yields the converse, thereby validating the procedure.

VII. A TURING MACHINE RECOGNIZER AND PARSER

It has been shown above that all entries of a recognition matrix can be found by computing about $\frac{1}{6}q^3n^3$ boolean products. In arriving at this figure, the computation of the $i = 1$ plane was neglected, as well as

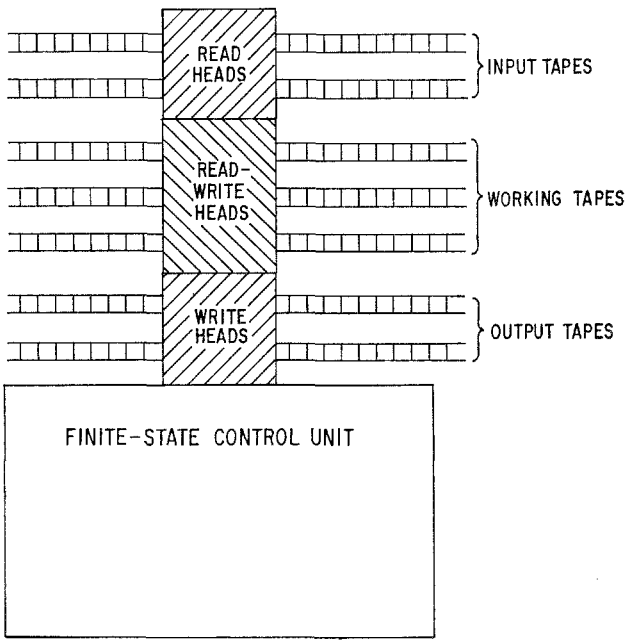

FIG. 1. A Turing Machine model.

the question of the difficulty of finding and comparing the arguments for a given boolean product. In order to define more precisely the complexity of calculating the entries of the recognition and parsing matrices, these calculations are simulated on a Turing Machine. The simulation developed in this section and the next is that of a universal context free language recognizer or parser, i.e., a Turing Machine that tests any string against a grammar specified as part of the input. That such a simulation exists is not surprising. But it is of interest to find that the computation of the recognition or parsing matrix on a Turing Machine can be accomplished (neglecting the $i = 1$ plane) in a number of steps differing only by a constant factor from the number of boolean products required. It is shown by Hartmanis and Stearns (1965) that a constant factor of computational complexity can be absorbed by an expansion of the tape vocabulary coupled with a recoding of the computation. Hence a Turing Machine can be designed which computes the entries of the recognition or parsing matrix in a number of steps equal to the number of boolean products required.

The form of the Machine employed is that developed for the work on computational complexity initiated by Hartmanis and Stearns (1965) and carried forward in conjunction with Hennie (Hennie and Stearns, 1965) and Lewis (Lewis, Stearns, and Hartmanis, 1965). In this work, computations are classified according to their difficulty of execution on a Turing Machine. This difficulty is measured in terms of time (i.e., number of Turing Machine steps) required to effect the computation, or alternatively in terms of the number of squares of working tape required. The "time" criterion is the one employed here; however, the Machine model employed is that which was developed in conjunction with study of the "space" criterion.

A *Turing Machine* consists of a finite-state *control unit* coupled to a finite number of infinite *tapes*; see Fig. 1. Each tape is partitioned into *squares*, each of which is either blank or contains a symbol from a given *finite tape vocabulary*, here assumed to consist of two symbols, 0 and 1. The coupling of the control unit to the infinite tapes is by means of read-write heads, one for each tape, each positioned over some square of tape. A Turing Machine step consists of the following sequence of three actions:

1. Each read-write head reads (identifies) the symbol contained in the square under scan;[5] the finite-state control unit changes state on the basis of symbols read.

2. Each head may over-write a symbol on the square under scan, or leave it as is.

3. Each head may independently shift its square under scan one to the left or one to the right.

The various tapes of the Turing Machine play specialized roles in a computation; three classes are identified:

1. Input tapes—These can be read from but not written upon. Hence each contains a fixed input pattern of symbols written upon its squares. At the start of a computation, the head associated with a given input tape is positioned over the leftmost nonblank symbol.

2. Working tapes—These may be read from and written upon. All squares of a working tape are initially blank.

3. Output tapes—These may be written upon but not read from. All squares of an output tape are initially blank.

A Turing Machine that effects the recognition algorithm is now de-

[5] An exception, as will be seen below, is made in the case of heads associated with output tapes, which cannot read symbols under scan.

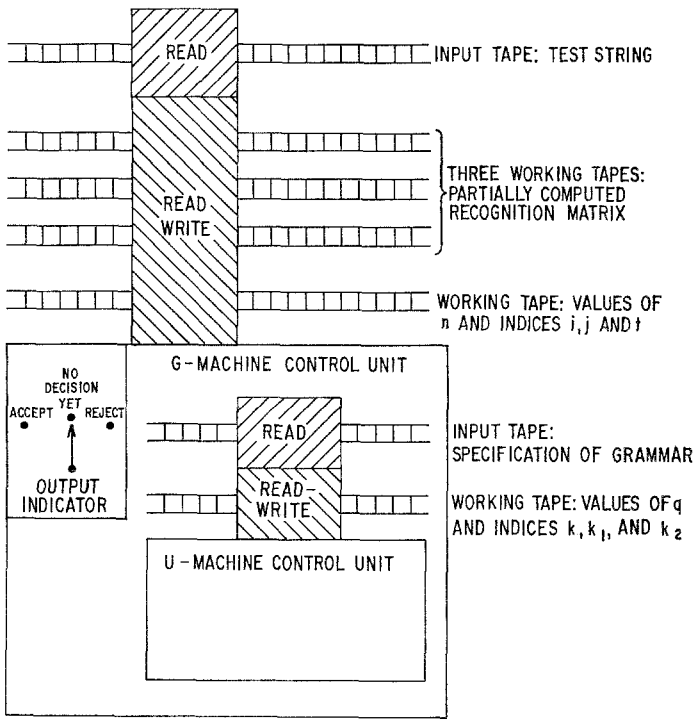

FIG. 2. Structure of Turing Machine recognizer.

scribed. For the present, it is assumed that the rules of grammar are stored in the finite state control unit, and hence the Machine is a recognizer for a specific grammar G. Let us call this a G-machine; see Fig. 2. In the next section, the control unit of the G-machine is dissected to reveal its internal structure, consisting of a control unit for a universal machine, an input tape on which the grammar is specified, and a working tape on which working parameters related to the grammar are temporarily stored.

The G-machine has one input tape. On this tape is inscribed the input string. The terminal symbols of this string are coded in terms of the tape vocabulary. For simplicity, this coding is assumed to be in unary notation; specifically, the terminal symbol a_l is represented by a string of l consecutive 1's followed by a blank square. The head associated with the input tape is positioned initially over the leftmost nonblank square. It need move only to the right, sensing the end of a symbol by the appear-

ance of a single blank square, and the end of the string by the appearance of two consecutive blank squares.

The G-machine has four working tapes. On three of these are developed, in the course of a recognition, three copies of the recognition matrix, laid out along the single tape dimension.[6] The fourth working tape is used to record values of n and indices i, j, and t.

The G-machine needs no output tape. The results of recognition are indicated by a three-state indicator, the three positions of the indicator dial being labeled "accept," "reject," and "no decision yet."

The format for the recognition matrix as it is developed on each of three working tapes is now explained in further detail. Recall that the entries in the recognition matrix are each 0 or 1. Within the limits scanned of the working tapes on which the recognition matrix are developed, no squares are blank; each contains a matrix entry, 0 or 1. The entry $r(i, j, k)$ of the recognition matrix is at position $(i - 1)nq + (j - 1)q + k$ of each of these working tapes, where position 1 is the square under scan before recognition begins, the squares being numbered left-to-right. According to this format, entries of the recognition matrix adjacent along the i-dimension are separated by $nq - 1$ squares when developed on tape, entries adjacent along the j-dimension are separated by $q - 1$ squares, and entries adjacent along the k-dimension are adjacent on tape.

We now describe the operation of the G-machine during recognition. This operation follows the *Iterative Procedure for Obtaining R*, given in Section IV. The recognition matrix entries for $i = 1$ are all developed first. To accomplish this the test string is read from left to right. For each string symbol read, the control unit examines its internal memory to find all nonterminal symbols which generate that terminal by means of a single terminating rule. If it is found that the jth string symbol is generated by nonterminal A_k, then recognition matrix entry $r(i, j, k) = 1$. All other entries for $i = 1$ are made 0. After having developed all $i = 1$ entries on one of the three working tapes devoted to the R-matrix, these entries are copied over into each of the other tapes.

After finding all recognition matrix entries for $i = 1$, the $i = 2$ plane is developed on one of the tapes, and again, these entries are transferred to the other tapes. Continuing, those of successively larger i are calcu-

[6] An alternative to the use of these three working tapes is the use of one working tape such that three heads couple to it. Such a tape would constitute an extension of the model allowed here.

lated, each time using the iterative relation (4). By that relation, each entry of the recognition matrix is determined from pairs of previously computed entries, the pairs which are appropriate being determined by the binary constituent rules of grammar. Two of the working tapes in question are used as memory from which to determine the values of these pairs of computed entries; on the third is recorded the entries resulting from an appropriate conjunction of 1's from among these previously computed entries.

All entries of the R-matrix for a given i are computed before any having index $i + 1$. The entries for a given i are computed in $i - 1$ separate passes, each pass corresponding to a value of index t in Eq. (4). On each pass entries are computed in order of increasing j. Furthermore, for a given i, t, and j, entries are computed in order of increasing index k. Hence, for a given pass t, the head recording the entries moves only from left to right. For each square that this head moves, the heads which gather information must run over all combinations of pairs of indices k_1 , k_2 . This then is the operation of a Turing Machine that effects the recognition algorithm.

The G-machine is readily modified to yield a *G-parser*. To accomplish this, an output tape is added in which is recorded the parsing matrix, with the format given for the recognition matrix. The G-parser first develops the recognition matrix as outlined above. If the $(n, 1, 1,)$ entry of the recognition matrix is 0, then the tested string is not a sentence, and hence the corresponding parsing matrix is all zero. If the $(n, 1, 1)$ entry is 1, however, the iterative relationship given in Section VI is employed to eliminate nonzero entries of the recognition matrix which are not relevant to the structure of the sentence. This refinement is a "top-down" procedure, working down from n with decreasing i. This procedure follows the general structure of that used for the recognition matrix, except that it is run "top-down." The number of steps required is not greater than that used for developing the recognition matrix.

VIII. A UNIVERSAL CONTEXT-FREE RECOGNIZER AND PARSER

Let us now dissect the control unit of the G-machine recognizer. It consists of a U-machine control unit (U stands for universal) and two associated tapes. In other words, by adding two additional tapes, each relating to manipulations on the grammar and modifying the finite state control unit, one can construct a U-machine, a machine which will per-

form the recognition test for *any* grammar that is context-free and in normal form.

One of the additional tapes is an imput tape on which the grammar is specified. This tape is composed of three sections, separated by single blank squares. On the first section is inscribed a string of 1's whose length is q, the number of nonterminal symbols in the normal grammar. The second section contains a specification of all terminating rules of grammar. This section occupies qs squares, where s in the number of terminal symbols of the grammar. The square located $(l-1)q+k$ squares from the left-hand boundary blank square contains a 1 if and only if $A_k \rightarrow a_l$ is a terminating rule of grammar. The third section contains a specification of all binary constituent rules of grammar. This section occupies q^3 squares; the one located $(k-1)q^2+(k_1-1)q+k_2$ squares from the left-hand boundary blank square contains a 1 if and only if $A_k \rightarrow A_{k_1}A_{k_2}$ is a binary constituent rule of grammar. The format of this input tape is designed to be compatible with the format chosen for the recognition matrix as it is realized on tape.

The second additional tape is a working tape on which a record is kept of the values of parameters relating to the grammar, i.e., q, k, k_1, and k_2.

We are now in a position to calculate the number of steps required for recognition by the U-machine. The number is similar to that obtained earlier, namely, $\frac{1}{6}q^3n^3$. However, one must also take into account the steps required to establish the $i=1$ plane of the matrix, the steps required for resetting various working and input tape heads to their proper positions, and the time required for making two extra copies of the recognition matrix; all these add to the number of steps required. Omitting details, the number of steps required to establish all entries in the $i=1$ plane for the U-machine is not greater than $(2s+1)qn$. The number of steps required to establish the rest of the entries is not greater than $\frac{1}{3}n^3(q^3+1)+2qn^2$. Together, this gives a total time for recognition not greater than

$$\tfrac{1}{3}n^3(q^3+1)+(2s+2n+1)qn.$$

This gives, for $q>1$ and $qn \gg s$, the following upper bound on the time complexity of recognition:

$$T_r = \tfrac{1}{2}n^3q^3. \tag{13}$$

For the U-parser the number of steps required is not greater than twice that for the recognizer and hence

$$T_p = n^3 q^3 \tag{14}$$

for $qn \gg s$ and $q > 1$.

IX. REMARKS

The algorithms for recognition and parsing in a number of elementary operations proportional to n^3 suggests the question as to whether or not there exist context-free languages which in fact require this many operations. This appears to be a difficult result to obtain. In fact, whereas Hartmanis and Stearns (1965) have exhibited a context-free language and a proof that it requires more than exactly time n to recognize, it has not been shown that context-free languages exist that require more time than is proportional to n, though intuition suggests that such do exist.

If a context-free language does exist that requires a factor of n^3 basic operations, that language is not generable by a metalinear grammar, since it can be shown by the matrix technique that all such languages can be recognized in a factor of n^2 basic operations.

Acknowledgment

The framework on computational complexity and its application to abstract language study out of which this work grew was developed by colleagues R. E. Stearns and P. M. Lewis II and by J. Hartmanis. The author is grateful for the many enlightening discussions he has had with each of them.

Received: June 23, 1966

REFERENCES

Chomsky, N. (1963), Formal properties of grammars. *In* "Handbook of Mathematical Psychology," Vol. II, R. D. Luce, R. R. Bush and E. Galanter, eds. Wiley, New York.

Chomsky, N. (1959), On certain formal properties of grammars. *Inform. Control* **2,** 137–167.

Gilbert, E. N. and Moore, E. F. (1959), Variable-length binary encodings. *Bell System Tech. J.* **38,** 933–967.

Griffiths, T. V. and Petrick, S. R. (1965), On the relative efficiencies of context-free grammar recognizers. *Commun. Assoc. Comput. Mach.* **8,** 283–300.

Hartmanis, J. and Stearns, R. E. (1965), On the computational complexity of algorithms. *Trans. Am. Math. Soc.* **117,** 285–306.

Hays, D. (1962), Automatic language-data processing. *In* "Computer Applications in the Behavioral Sciences," H. Borko, ed. Prentice Hall, Englewood Cliffs, New Jersey.

Hennie, F. C. and Stearns, R. E. (1965), Two-Tape Simulation of Multitape Turing Machines. Gen. Elec. Res. Lab. Report No. 65-RL-4020E.

Kay, M. (1963), A parsing procedure. *In* "Information Processing 1962." North Holland, Amsterdam.

KUNO, S. AND OETTINGER, A. E. (1963), Multiple-path syntactic analyser. *In* "Information Processing 1962." North Holland, Amsterdam.

KUNO, S. (1965), The predictive analyser and a path elimination technique. *Commun. Assoc. Comput. Mach.* **8**, 453–463.

KASAMI, T. (1966), An Efficient Recognition and Syntax-Analysis Algorithm for Context-Free Languages. University of Illinois Coordinated Science Lab. Rept. No. R-257.

LEWIS, P. M. II, STEARNS, R. E., AND HARTMANIS, J. (1965), Memory bounds for recognition of context-free and context-sensitive languages. *IEEE Conf. Record Switching Circ. Theory and Log. Des.*, pp. 191–202.

SAKAI, I. (1962), Syntax in universal translation. *Proc. 1961 Intern. Conf. Mach. Transl. Languages and Appl. Language Analysis* (Her Majesty's Stationery Office, London).

STEARNS, R. E., HARTMANIS, J., AND LEWIS, P. M., II. (1965), Hierarchies of memory limited computations. *1965 IEEE Conf. Record Switching Circ. Theory and Log. Des.*, pp. 179–190.