MEMORY BOUNDS FOR RECOGNITION OF CONTEXT-FREE AND CONTEXT-SENSITIVE LANGUAGES

P.M. Lewis II, R.E. Stearns, and J. Hartmanis
General Electric Research Laboratory, Schenectady, New York

I. INTRODUCTION

An important part of the theory of abstract languages deals with the processing of such languages by various machine models. Results in this area, in addition to yielding insights into the basic nature of languages, also have immediate implications relating to the processing of computer languages.

This paper is concerned with the recognition of context-free and context-sensitive languages by machine models with various organizations.

A machine is said to recognize a given language if, whenever a string of symbols is placed on its input tape, the machine processes this string and eventually halts, indicating either "Yes" the string is in the language or "No" it is not.

Experience indicates that some languages are intrinsically more difficult to recognize than others, and that this difficulty may depend strongly on the organization of the machine that does the processing. The difficulty of a computation can, of course, be measured in a number of ways. In this paper we use as the complexity measure, the amount of memory space required for the computation: more specifically the L(n)-tape measure defined in [1]. To study the dependence of this complexity measure on machine organization, we consider the four machine models defined in [1]: namely the off-line and on-line Turing machine and the off-line and on-line pushdown machine. Thus if any one of these models recognizes a given language and processes each input string of length n using no more than L(n) squares on its working tape, then we say that that language is L(n)-tape recognizable or more simply L(n)-recognizable. Since we are interested in tape functions as a rate of growth, we conveniently restrict ourselves to monotone non-decreasing tape functions.

The results of [1] make it natural to define a partial ordering "$\leq\cdot$" on the tape functions. We write $L(n) \leq\cdot Q(n)$ if and only if there is a constant C such that $L(n) \leq C\,Q(n)$ for all n. We write $L(n) \doteq Q(n)$ if and only if $L(n) \leq\cdot Q(n)$ and $Q(n) \leq\cdot L(n)$. Finally, we write $L(n) <\cdot Q(n)$ if $Q(n) \leq\cdot L(n)$ is false. From [1], we know that $L(n) \leq\cdot Q(n)$ means that all Q(n)-recognizable sets are L(n)-recognizable, $L(n) \doteq Q(n)$ means that the class of Q(n)-recognizable sets and the class of L(n)-recognizable sets are identical, and $L(n) <\cdot Q(n)$ means that some Q(n)-recognizable sets are not L(n)-recognizable provided Q(n) is constructable. Since $\leq\cdot$ is only a partial ordering, it is quite possible that $L(n) <\cdot Q(n)$ and $Q(n) <\cdot L(n)$.

II. DEFINITIONS OF ABSTRACT LANGUAGES

The abstract languages in which we are interested are those proposed by Chomsky [2]. They are defined in terms of the grammars which generate them. A grammar G consists of two disjoint, finite sets of symbols,

N = {S,A,B,C,...} the set of non-terminal symbols,

T = {a,b,c,...} the set of terminal symbols,

S a designated starting symbol in N, and

$\mathcal{P}$ a finite set of productions.

We consider three-types of grammars characterized by the form of their productions:

1. A context-sensitive grammar is one that has productions of the form

$$\alpha A \beta \longrightarrow \alpha \gamma \beta$$

where α, β and γ are strings of terminals and non-terminals and γ is non-empty. This production is to be interpreted as follows: In any string in which the

non-terminal A appears between the strings α and β, A can be replaced by γ.

2. A context-free grammar is one that has productions of the form

$$A \longrightarrow \gamma$$

3. A finite state or regular grammar is one that has productions which are all of the form

$$A \longrightarrow aB$$

$$A \longrightarrow a$$

or all of the form

$$A \longrightarrow Ba$$

$$A \longrightarrow a$$

that is: a single non-terminal can be replaced by a single terminal or by a single terminal with a single non-terminal consistently on its left or right.

The language L(G) defined by a grammar G is the set of strings in T* that can be generated by that grammar starting from the start symbol S. ("*" is always the Kleene star operator.) A language is called context sensitive (or context free or finite state) if there is at least one context sensitive (or context free or finite state) grammar that generates it.

The class of context free languages is of particular interest because it includes ALGOL and other programming languages.

III. RECOGNITION OF ABSTRACT LANGUAGES

There are a number of well-known results on the recognition of abstract languages.

A. Context Sensitive Languages

A set of sequences is a context sensitive language if and only if it can be recognized by a non-deterministic linear bounded automaton (Landweber [3] and Kuroda [4]).

In the present terminology a linear bounded automaton is an off-line Turing machine with L(n) = n. Therefore the above result implies that any one of the machine models with $L(n) \leq \cdot\ n$ recognizes a context-sensitive language.

The results of [1] then imply that the on-line and off-line Turing machine models and the off-line push-down machine model each define an infinite hierarchy of distinct complexity classes of context sensitive languages. For the off-line Turing machine and off-line push-down machines these hierarchies span a doubly exponential range for L(n) in

$$\log \log n \leq \cdot\ L(n) \leq \cdot\ n,$$

and for the on-line Turing machine there is an exponential range for L(n) in

$$\log n \leq \cdot\ L(n) \leq \cdot\ n$$

Recall for example from [1] the sequences

$$\mathcal{E}_1 = \{s\ b_1\ s\ b_2\ s \ldots s\ b_m\ s \mid m = 1,2,\ldots\}$$

$$\mathcal{E}_2 = \{s\ b_1^a\ s\ b_2^a\ s \ldots s\ b_m^a\ s \mid m = 1,2,\ldots\}$$

where b_j is the binary expansion of the integer j and b_j^a is the same expansion with the digits relatively addressed. The sets $\mathcal{E}_1$ and $\mathcal{E}_2$ form context sensitive and not context free languages that can be recognized with

$$L(n) = \log \log n$$

tape by an off-line Turing machine and off-line push-down machine, respectively.

B. Context Free Languages

Context free languages are often studied in conjunction with push-down machines because of the following:

1. A set of sequences is a context free language if and only if it can be recognized by a non-deterministic on-line push-down machine. There exist context free languages that cannot be recognized by a deterministic on-line push-down machine (Chomsky [5]).

Thus the class of languages that can be recognized by on-line push-down machines form a proper subset of the class of context free languages. As shown in [1], there is only one complexity class of non-regular sets on these machines,

namely the class of L(n) = n-recognizable sets.

2. Off-line push-down machines can recognize context free and context sensitive languages. For example the set of sequences

$$\{a^n b^n c^n \mid n = 1,2,\ldots\}$$

is a context sensitive but not a context free language that can be recognized by an off-line push-down machine. We conjecture that the set of sequences recognized by off-line push-down machines is properly contained in the set of context sensitive languages and that it does not contain all context free languages.

3. All context free languages can be recognized by a linear bounded automaton (Kuroda [4]).

Thus all context free languages can be recognized by off-line Turing machines with $L(n) \leq \cdot\, n$. The results in the next section give a more exact account as to which L(n) permit the recognition of all context free languages by off-line Turing machines.

C. Finite State Languages

A language is finite state if and only if it can be recognized by a finite state machine (Chomsky [2]). This implies that finite state languages can be recognized by any machine model with L(n) = 1.

IV. HIERARCHIES OF CONTEXT FREE LANGUAGES

In this section we demonstrate that off-line and on-line Turing machines and off-line push-down machines each define separate infinite hierarchies of context free languages. As discussed earlier, the on-line push-down machine has only one complexity class L(n) = n and hence does not define a hierarchy. Of these machines, only the Turing machines have been shown to include all context free languages in their hierarchies.

To give a complete picture of the different hierarchy results, we state them all now and give the proofs later.

A. For Off-line Turing Machines

1. There exist non-regular context free languages that are L(n) = log log n-recognizable.

2. There exist context free languages that require $L(n) \doteq \log n$.

3. There is an infinite hierarchy of context free languages with certain tape functions L(n) which satisfy

$$\log \log n \leq \cdot\, L(n) \leq \cdot\, \log n \ .$$

4. All context free languages can be recognized with

$$L(n) = (\log n)^2 \ .$$

The hierarchy has only been demonstrated in the region between log log n and $\log_2 n$ and not up to the bound of $(\log n)^2$ given by 4. It is an open question whether there are any context free languages that require more than L(n) = log n on an off-line Turing machine. We conjecture that there are such languages and that $(\log n)^2$ is the real bound.

B. For On-line Turing Machines

1. There exist non-regular context free languages that are L(n) = log n-recognizable.

2. There exist context free languages that require $L(n) \doteq n$.

3. There is an infinite hierarchy of context free languages with certain tape functions L(n) which satisfy

$$\log n \leq \cdot\, L(n) \leq \cdot\, n \ .$$

C. For Off-line Push-down Machines

1. There exist non-regular context free languages that are L(n) = log log n-recognizable.

2. There exist context free languages that require L(n) = n.

3. There is an infinite hierarchy of context free languages with certain tape functions L(n) which satisfy

$$\log \log n \leq \cdot\, L(n) \leq \cdot\, n \ .$$

We now define three languages which we will show are examples of non-regular context free languages that can be recognized with the L(n) given respectively in statements A1, B1, and C1.

Let L_1 be the language whose complement is

$$\overline{L_1} = \{\overrightarrow{b_1}\ s\ \overleftarrow{b_2}\ s\ \overrightarrow{b_3}\ s \ldots s\ \overleftarrow{b_{2m}} \mid m = 1,2,\ldots\}$$

where $\overrightarrow{b_j}$ is the binary expansion of j written from left to right, $\overleftarrow{b_k}$ is the binary expansion of k written from right to left, and s is a space symbol.

Let L_2 be the language whose complement is

$$\overline{L_2} = \{\overrightarrow{b_1^{\alpha}}\ s\ \overleftarrow{b_2^{\alpha}}\ s\ \overrightarrow{b_3^{\alpha}} \ldots s\ \overleftarrow{b_{2m}^{\alpha}} \mid m = 1,2,\ldots\}$$

where $\overrightarrow{b_j^{\alpha}}$ is the binary expansion of j written from left to right with its digits relatively addressed and these addresses written in alternating directions (See also examples from [1]). For instance

$$\overrightarrow{b_{25}^{\alpha}} = \text{uzu 1 zzu 1 uu 0 zu 0 u 1}$$

where in the relative addressing z stands for 0 and u for 1. The string $\overleftarrow{b_k^{\alpha}}$ is similarly defined.

Finally let

$$L_3 = \{a^n b^n \mid n = 1,2,\ldots\} .$$

Theorem 1: The sets L_1, L_2 and L_3 are non-regular context free languages. The language L_1 is

$$L(n) = \log \log n$$

recognizable on an off-line Turing machine. The language L_2 is

$$L(n) = \log \log n$$

recognizable on an off-line push-down machine. The language L_3 is

$$L(n) = \log n$$

recognizable on an on-line Turing machine.

Proof: The languages L_1, L_2 and L_3 are clearly not regular languages.

L_1 and L_2 can both be recognized on their respective machines with L(n) = log log n using obvious modifications of the procedures worked out for the examples of [1]. It remains only to prove that L_1 and L_2 are context free languages.

We give the proof for L_1; the proof for L_2 is quite similar.

It is readily verified that $\overline{L_1}$ can be written as the intersection of two other languages

$$\overline{L_1} = L_4 \cap L_5$$

where

$$L_4 = \overrightarrow{b_1}\ s\ \left(\overleftarrow{b_j}\ s\ \overrightarrow{b_{j+1}}\ s\right)^*\ (0 + 1 + s)^*$$

$$L_5 = \left(\overrightarrow{b_k}\ s\ \overleftarrow{b_{k+1}}\ s\right)^* .$$

Thus L_1 can be written

$$L_1 = \overline{(L_4 \cap L_5)} = \overline{L_4} \cup \overline{L_5} .$$

We now show that L_4 and L_5 are both context free languages that can be recognized by an on-line push-down machine. Therefore their complements are also context free languages (because the output of the machine can be complemented). Since the union of two context free languages is a context free language, this implies that L_1 is context free.

To demonstrate, for example, that L_5 can be recognized by an on-line push-down machine, consider the following informal description of the recognition procedure: As the machine scans $\overrightarrow{b_k}$, it stores it on the push-down tape. After the s symbol is scanned, the sequence $\overrightarrow{b_k}$ is popped off the tape one digit at a time and applied to a serial adder (in the finite state part of the machine) which adds 1 to it. The output of the serial adder can be compared digit by digit with $\overleftarrow{b_{k+1}}$, thus completing

the recognition. The procedure is repeated for each parenthesis within the * operator.

A similar procedure can be used to recognize L_4 with an on-line push-down machine. This completes the proof for the off-line machines.

Clearly the language

$$L_3 = \{a^n b^n \mid n = 1,2,\ldots\}$$

is context free and can be recognized with L(n) = log n on an on-line Turing machine since the log n storage can be used as a counter to count the a's from 0 to n and compare this number with the number of b's.■

Theorem 2:

(1) The context free language

$$L_3 = \{a^n b^n \mid n = 1,2,\ldots\}$$

is L(n) = log n-recognizable on an off-line Turing machine and cannot be recognized with

$$L(n) <\cdot \log n.$$

(2) The context free language L_3 is L(n) = n-recognizable on an off-line push-down machine and cannot be recognized with

$$L(n) <\cdot n.$$

(3) The context-free language

$$L_6 = \vec{w}\, s\, \overleftarrow{w} \quad ,$$

where $\vec{w}$ is any finite sequence of binary symbols and $\overleftarrow{w}$ is the mirror image of $\vec{w}$, can be recognized on an on-line Turing machine with L(n) = n and cannot be recognized with L(n) <· n.

Proof: From Theorem 1 we know that L_3 is L(n) = log n-recognizable on an off-line Turing machine. We now outline the proof that L_3 cannot be recognized on an off-line Turing machine with

$$L(n) <\cdot \log n.$$

The proof is by contradiction; we show that any machine M which allegedly accepts the set of sequences $\{a^m b^m\}$ with L(n) <· log n must also accept the sequence $a^m b^{m+m!}$ for some m. Let p be a number such that, for all positive integers j,

$$p^j > Q \cdot T \cdot j \cdot T^j$$

where Q is the number of states in the control and T is the number of symbols in the working tape alphabet. Because L(n) <· log n, there must be some m for which $L(2m) < \log_p m$ and M has less than $m < p^{L(2m)}$ different memory configurations with which to process $a^m b^m$. If the machine is processing the string $a^m b^m$ and the reading head is crossing the entire sequence of b's then there must be two b's for which when they are scanned on this crossing, M is in the same configuration. If the two b's with this property are separated by ℓ tape squares, then it follows that if, instead of m b's there were $(m + k\cdot\ell)$b's where k is any positive integer, then M would arrive at the other end of this string of b's in the same configuration as for the string of m b's. This is true for each crossing of the string of b's, although the distance ℓ can be different for each crossing. Still $m + m!$ is of the form $m + k\cdot\ell$ for all $\ell \leq m$, and thus on no crossing can the machine distinguish between

$$a^m b^m \quad \text{and} \quad a^m b^{m+m!}$$

Note that if the machine starts to scan the string of b's but does not reach the other end, then again it treats $a^m b^m$ and $a^m b^{m+m!}$ identically. Thus $a^m b^{m+m!}$ must also be accepted and we have achieved the contradiction we were seeking.

The push-down case is essentially a special case of the lemma in [1] so we omit any further details.

Part 3 of the theorem is self evident. If the reading head of the on-line Turing machine crosses the s center marker as its n-th input, an amount of information sufficient to reconstruct $\vec{w}$ must be stored on the working tape and this clearly requires n squares.■

Next we show that each machine model gives an infinite hierarchy of context free languages (statements A3, B3 and C3).

Theorem 3: There exist infinite hierarchies of context free languages: for off-line Turing machines with

$$\log \log n \leq\cdot L(n) \leq\cdot \log n$$

for on-line Turing machines with

$$\log n \leq\cdot L(n) \leq\cdot n$$

and for off-line push-down machines with

$$\log \log n \leq\cdot L(n) \leq\cdot n$$

Proof: To prove that there exist infinitely many complexity classes of context free languages for off-line Turing machines, we show that for any rational number $p/q \geq 1$ there exists a context free language that can be recognized with

$$L(n) = [\log \log n]^{p/q}$$

but not with

$$L(n) <\cdot [\log \log n]^{p/q} \quad .$$

We give a proof for the case $p/q = 2$ which reveals the basic reasoning.

Consider the language L_7 whose complement is given by

$$\overline{L_7} = \Big\{ \overrightarrow{b_1^{\alpha}} \; s \; \overleftarrow{b_2^{\alpha}} \; s \ldots s \; \overleftarrow{b_{2m}^{\alpha}} \; ss \; 1^{[\log \log 2m]} \; s \; 1^{[\log \log 2m]-1} \; s \ldots s \; 1^2 \; s \; 1 \; sss \; 1 \; s \; 1^2 \; s \; 1^4 \; s \; 1^8 \; s \ldots s \; 1^{2^{[\log \log 2m]^2}} \Big\} .$$

For each m, let n_m be the length of the corresponding sequence in $\overline{L_7}$. Using reasoning as in Theorem 2, this m-th sequence will require some $\log [2^{[\log \log 2m]^2}]$ tape squares to process in order that the machine not be "fooled" by the same sequence with some additional number of ones added at the end. Thus, we must have

$$L(n_m) \doteq \log [2^{[\log \log 2m]^2}]$$

$$\doteq [\log \log 2m]^2 .$$

The length n_m may be shown to be less than $2m \log m$ and using the monotonicity of L, we see that

$$L(2m \log m) \geq L(n_m) \doteq [\log \log 2m]^2$$

$$\doteq [\log \log [2m \log m]]^2$$

or

$$L(n) \cdot\geq [\log \log n]^2 .$$

From our previous experience with $\overline{L_2}$, it is clear that we can recognize L_7 with

$$L(n) \leq\cdot [\log \log n]^2$$

and so $L(n) \doteq [\log \log n]^2$ is demonstrated.

It remains only to show that L_7 is a context free language. The method of Theorem 1 is used; $\overline{L_7}$ is expressed as the intersection of several context free languages that can be recognized by on-line push-down machines. In particular $\overline{L_7}$ is the intersection of the following seven languages.

$$L_a = \overrightarrow{b_1^{\alpha}} \; s \; \left(\overleftarrow{b_j^{\alpha}} \; s \; \overrightarrow{b_{j+1}^{\alpha}} \; s \right)^* \; (0 + 1 + s + u + z)^*$$

$$L_b = \left(\overrightarrow{b_k^{\alpha}} \; s \; \overleftarrow{b_{k+1}^{\alpha}} \; s \right)^* \; ss \; (0 + 1 + s)^*$$

$$L_c = (0 + 1 + s + u + z)^* \; (u + z)^q \; ss \; 1^q s \; (1^{\ell} \; s \; 1^{\ell-1} s)^* \; 1^* sss \; (0 + 1 + s)^*$$

$$L_d = (0 + 1 + s + u + z)^* \; ss \; (1^p \; s \; 1^{p-1})^* \; 1^* \; sss \; (0 + 1 + s)^*$$

$$L_e = (0 + 1 + s + u + z)^* \; ss \; (1 + \tfrac{s}{2})^r \; sss \; (1^* \; s)^r \quad ,$$

where $(1 + \frac{s}{2})^r \; sss \; (1^* s)^r$ designates an arbitrary string of 1's and s's followed by sss followed by r occurrences of 1^*s where r is equal to the number of 1's plus half the number of s's appearing in the arbitrary string.

$$L_f = (0 + 1 + s + u + z)\ sss\ 1\ s\ (1^t\ s\ 1^{2t})^*\ (0 + 1 + s)^*$$

$$L_g = (0 + 1 + s + u + z)\ sss\ (1^\nu\ s\ 1^{2\nu})^*\ (0 + 1 + s)^*$$

In a manner similar to the proof of Theorem 1, L_a and L_b produce the initial part of $\underline{L_7}$ before the first occurrence of ss. Since the $b_{2m}^{\overleftarrow{\alpha}}$ is relatively addressed, directly before the ss is a number of u and z symbols equal to [log log 2 m]. L_c then produces a number of 1's directly after the ss also equal to [log log 2m]. L_c and L_d then produce that portion of $\underline{L_7}$ between ss and sss. In this portion the number of 1's is

$$[\log \log 2\ m] + [(\log \log 2\ m) - 1] + \ldots + 2 + 1$$

$$= \frac{1}{2}\left\{[\log \log 2\ m]^2 - [\log \log 2\ m]\right\}$$

If we add to this number, every other s marker, there are then exactly

$$[\log \log 2\ m]^2$$

symbols. L_e ensures that there are exactly this number, $[\log \log 2\ m]^2$ of s markers after the sss. L_f and L_g then produce that part of the sequence after the sss. The only remaining part of the proof is to demonstrate that all seven languages are in fact recognizable by an on-line push-down machine. This is relatively straightforward.

Similar methods give all rational powers greater than unity of log log 2 m and all rational fractional powers of log 2 m as well as many other functions. The reason that the hierarchy for off-line Turing machines is limited by this method to L(n) = log n can now be seen. To prove that some function L(n) of tape squares is needed, we construct a sequence that has $2^{L(n)}$ 1's at the end. But this number of 1's must certainly be insignificant compared with n the length of the sequence; hence

$$2^{L(n)} \leq \cdot\ n$$

or

$$L(n) \leq \cdot\ \log n$$

Thus if context free languages that require L(n) $\cdot>$ log n are to be found a different method must be used.

Similar methods can be used to prove this theorem for the other two machine models.∎

As stated before, we cannot show (and conjecture it is not true) that all context free languages can be recognized with L(n) = log n. We do however have an algorithm that will recognize all context free languages and requires $L(n) = [\log n]^2$.

Theorem 4: All context free languages can be recognized on an off-line Turing machine with $L(n) = [\log n]^2$.

Proof: The proof is by exhibiting the algorithm. Again for brevity we give an informal description.

Assume for the sake of this simplified discussion that all productions of the language have only two symbols (i.e. $A \longrightarrow BC$ or $A \longrightarrow aB$). We first need an auxiliary result. Consider any n symbol sentence in such a language. We first show that there must be at least one substring in the sentence with length ℓ

$$n/3 \leq \ell \leq 2n/3$$

such that the substring is exactly the descendents of a single non-terminal symbol. To see this consider the initial production that generated the string

$$S \longrightarrow AB$$

If neither A or B are non-terminals with the desired property than the descendents of one of them, say B, has length greater than 2n/3. Now consider the direct descendent of B,

$$B \longrightarrow CD$$

At least one of these, say D, must have descendents with lengths greater than 1/2 (2n/3) = n/3. Thus if C and D both do not have the desired property, then

D must have length greater than 2n/3 and the process can be repeated, until finally a non-terminal with the desired property is determined. This completes the proof of the auxiliary result.

The initial step in the recognition is to guess at the location of the descendents of the non-terminal with this property,

1. Given a string of symbols of length n, select by a systematic procedure a substring with length ℓ

$$n/3 \leq \ell \leq 2n/3$$

The selection procedure must be systematic such that if the initial guess turns out to be incorrect, a new guess, different from all previous guesses, can be made in some simple manner such as incrementing indices, etc.

2. The $(\log n)^2$ memory is divided into log n blocks each log n long. In the first block of memory store the location within the string (i.e. the addresses) of the two end symbols of the selected substring.

3. Repeat the above two steps on the selected substring. That is within this substring select a new substring whose length is between 1/3 and 2/3 of that of the first substring and store its address in the second block of memory.

4. Repeat this process again within the new substring and continue until a substring of only two symbols is obtained. Clearly no more than $\log_{2/3} n$ blocks of memory will be required for the entire procedure.

The first four steps of the procedure are shown diagrammatically in Fig. 1.

5. At this point it can be determined whether or not the final selection corresponds to an actual production. If not, then a new final selection is made. If no possible final selection corresponds to a production, a different next to last selection is made, and the final selection process repeated. If no next to last selection is satisfactory, the selection one level back is changed and the process continued until a final selection is found which does correspond to an actual production. (If all selections are exhausted without this situation occurring, the string is not in the language.)

6. If the final selection does correspond to a production, the non-terminal corresponding to that production is written into the last block of memory.

7. Now it can be determined whether or not the next to last selection corresponds to a production. If not, a new next to last selection is made and a procedure similar to step 5 is followed. If all second level and higher selections have been exhausted without finding a production, a new final selection is made and step 5 is repeated.

8. If the next to last selection does correspond to one or more productions the non-terminals corresponding to these productions are written into the next to last memory block.

9. Now the last memory block is no longer needed and is erased.

10. As shown in Fig. 2, a new selection of a subsequence is made inside the third from last selection, and its address is recorded in the newly available last memory block.

11. The procedure is repeated and larger and larger subsequences are identified. As each subsequence is identified all memory blocks that were used to record information about productions within the subsequence are no longer needed and are made available for making other selections as in step 10. If the given sequence is in the language, the procedure will finally identify the entire sequence as descending from the starting symbol S.∎

V. COMPARISON OF CONTEXT FREE LANGUAGE HIERARCHIES

It should be emphasized that the hierarchies of context free languages defined by the four machine models are different. Languages which are easy to recognize on one model may be difficult on another and conversely.

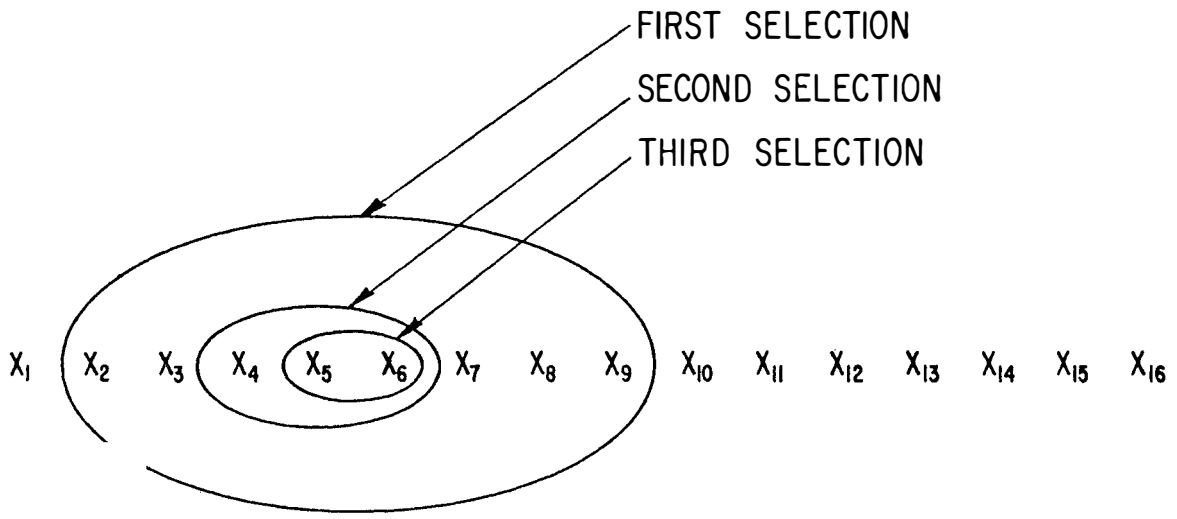

FIGURE 1. THE FIRST FOUR STEPS OF THE RECOGNITION ALGORITHM.

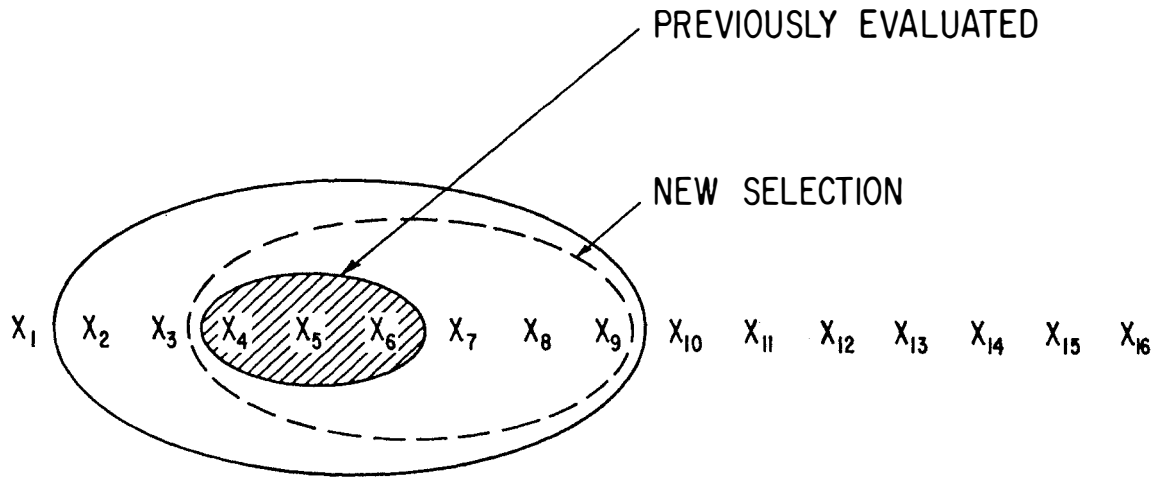

FIGURE 2. THE TENTH STEP OF THE ALGORITHM.

The most striking result of this type concerns the on-line push-down machine and the off-line Turing machine. Any non-regular context free language that can be recognized at all on an on-line push-down machine requires at least L(n) = log n on an off-line Turing machine. Thus all the infinite hierarchy of context free languages that are "easy" on the off-line Turing machine, i.e. for which

$$\log \log n \leq\cdot L(n) <\cdot \log n,$$

are impossible to recognize on an on-line push-down machine. And all those context free languages that can be recognized on the on-line push-down machine are "difficult" on the off-line Turing machine i.e. require at least L(n) = log n. State more precisely

Theorem 6: All non-regular context free languages that can be recognized on an on-line push-down machine require at least $L(n) \doteq \log n$ to be recognized on an off-line Turing machine.

Proof: At least three different types of on-line push-down machines have been defined in the literature:

(1) A machine which can push down or pop up only one symbol on its tape for each input symbol and for which the push-down tape must be empty when a sequence is recognized.

(2) A machine similar to 1 except that the tape need not be empty on recognition.

(3) A machine similar to 2 except that an arbitrary number of symbols can be popped off the tape for each input symbol.

The theorem is true for all three types (Type 3, the most general one, is the one defined in [1]). However, even a sketch of the proof for Type 3 machines is too long to be included here. Therefore we sketch a proof for Type 2 machines.

The basic idea of the proof is this: We first show that if a non-regular language is recognized by an on-line push-down machine, there are sequences that require that any arbitrary length of push-down tape be written on, and later read. We then show that such a sequence must contain

(1) some word, say w_A which is repeated an indefinite number of times, causing the push-down tape to be written on and increase in length,

(2) some other word, say w_B, which is repeated the same number of times as w_A and causes the tape to pop up. The recognition then depends on comparing the number of appearances of w_A and w_B and by Theorem 2, for an off-line Turing machine this requires $L(n) \doteq \log n$.

We now give the details of the proof.

If a non-regular context free language can be recognized by an on-line push-down machine, then there must be sequences that require an arbitrarily large amount of push-down tape. Stated precisely, given any length ℓ there must be a word w_ℓ, which when applied to the machine (as the initial part of a sequence) leaves the tape with some $\ell_o \geq \ell$ symbols on it, and furthermore any other word, say w' which utilizes less than ℓ_o tape squares can be distinguished from w_ℓ. That is, there is some other word w_1 which can be applied after w_ℓ and w' to give two sequences

$$S_1 = w_\ell \, w_1$$

$$S_2 = w' \, w_1$$

such that S_1 is in the language and S_2 is not or vice versa.

Since ℓ can be arbitrarily large, we choose it equal to

$$\ell = Q \, I \, T + Q^2 \, I^2 \, T + 2$$

where Q is the number of states of the finite state portion of the machine and I and T are the number of symbols in the input and working tape alphabets respectively. The Q I T + 1 squares at the bottom of the tape (that is those written

on first) are called the bottom of the tape. The remaining squares are called the top of the tape.

We first show that for this ℓ and corresponding w_ℓ, there is some w' which requires fewer tape squares but for which the push-down machine must visit the bottom of the push-down tape for w_ℓ to distinguish w' from w_ℓ.

Since the bottom of the tape for w_ℓ contains Q I T + 1 squares it must contain two squares say t_1 and t_2 for which when w_ℓ is applied:

(a) The same tape symbol is (finally) printed on t_1 and t_2, and

(b) when these symbols are printed the finite state portion of the machine is in the same state and the input symbols (in w_ℓ) are the same.

Thus w_ℓ can be written as

$$w_\ell = w_\alpha \, s_1 \, w_\beta \, s_1 \, w_\Gamma$$

where the two appearances of s_1 are the symbols that cause the (final) symbols to be printed on t_1 and t_2. Now consider as w'

$$w' = w_\alpha \, s_1 \, w_\Gamma$$

This word will cause exactly the same push-down tape to be generated as w_ℓ except that the portion between t_1 and t_2 (including t_2) will be omitted. Since the tapes for w' and w_ℓ differ only at the bottom, then to distinquish w_ℓ from w', that is to distinguish

$$S_1 = w_\ell \, w_1$$

from

$$S_2 = w' \, w_1$$

the machine must visit the bottom of the tape. Thus in processing S_1, for example, the machine must first print the entire tape, bottom and top, and must then "pop off" the entire top in order that it may read the bottom.

Now consider the top of the tape generated in processing S_1. Since w_ℓ utilizes $\ell_o > \ell = Q\ I\ T + Q^2\ I^2\ T + 2$ tape squares the top of its tape contains at least $Q^2\ I^2\ T + 1$ tape squares. Hence, it must contain at least two squares say t_3 and t_4 such that in processing S_1

(a) The same tape symbols are (finally) printed on t_3 and t_4, and

(b) when (finally) printing the symbols on t_3 and t_4 and later when reading the symbols on t_3 and t_4, (at all four times) the finite state part of the machine is in the same state and the input symbols are the same.

Thus S_1 can be written

$$S_1 = w_4 \, s_2 \, w_5 \, s_2 \, w_6 \, s_2 \, w_7 \, s_2 \, w_8$$

where the first two appearances of s_2 cause the symbols to be (finally) written on t_3 and t_4 and the last two appearances of s_2 occur when t_4 and t_3 are read.

Now consider what happens when a sequence of the form

$$\delta_1 = w_4 \, s_2 \, (w_5 \, s_2)^n \, w_6 \, s_2 \, (w_7 \, s_2)^n \, w_8$$

is applied. The computation is very similar to that for S_1. The additional inputs in $(w_5\ s_2)^n$ cause additional tape squares to be generated identical to those originally between t_4 and t_3. If the tape for S_1 is of the form

$$\tau_1 \, t \, \tau_2 \, t \, \tau_3$$

(where t is the symbol printed on t_3 and t_4) the tape for δ_1 will be

$$\tau_1 \, t \, (\tau_2 \, t)^n \, \tau_3$$

However the inputs in $(w_7\ s_2)^n$ cause these additional tape squares to be popped off exactly as before. Thus the result of the computation is the same, and all

the sequences in $\mathcal{S}_1$ are accepted (or rejected) if the sequence S_1 is accepted (or rejected).

While the machine is on the top of the tape, it cannot distinguish S_2 from S_1. Thus the tape squares t_3 and t_4 have the same properties in processing S_2 as S_1. In particular then, S_2 can be written

$$S_2 = w_9\ s_2\ w_5\ s_2\ w_6\ s_2\ w_7\ s_2\ w_{10}$$

where w_9 and w_{10} account for the different bottom of the tape but s_2, w_5, w_6, and w_7 are the same as for S_1. It also follows then that the set of sequences

$$\mathcal{S}_2 = w_9\ s_2\ (w_5\ s_2)^n\ W_6\ s_2\ (w_7\ s_2)^n\ w_{10}$$

is accepted or rejected as S_2 is accepted or rejected.

Thus we have shown that $\mathcal{S}_1$ is in the language accepted by the machine and $\mathcal{S}_2$ is not or conversely. Now consider the two sets of sequences

$$\mathcal{S}_1' = w_4\ s_2\ (w_5\ s_2)^{n_1}\ w_6\ (w_7\ s_2)^{n_2}\ w_8$$

$$\mathcal{S}_2' = w_9\ s_2\ (w_5\ s_2)^{n_1}\ w_6\ (w_7\ s_2)^{n_2}\ w_{10}$$

where $(n_2 - n_1) \geq K$ and K is large enough such that in processing $\mathcal{S}_1'$ and $\mathcal{S}_2'$ the entire top of the tape is not popped up (recall we are only considering machines that can only pop up one tape symbol for each input symbol). Since the machine does not read the bottom of the tape in processing $\mathcal{S}_1'$ and $\mathcal{S}_2'$ it cannot distinguish them and they are either both accepted or both rejected.

Assume now, for example, that the language is such that $\mathcal{S}_1$ is in the language and $\mathcal{S}_2$, $\mathcal{S}_1'$ and $\mathcal{S}_2'$ are not. If a Turing machine is to recognize this language and distinguish $\mathcal{S}_1$ from $\mathcal{S}_1'$ it must compute $(n_2 - n_1)$ and compare it with K. Then by the same reasoning as in Theorem 2, it must essentially be able to count up to n and requires at least $L(n) \doteq \log n$ tape squares. Similar reasoning applies to the other possibilities. ■

Theorem 6 states that it requires at least $L(n) \doteq \log n$ tape squares on an off-line Turing machine to simulate an on-line push-down machine performing a non-regular recognition. Theorem 5 states that it requires at most $L(n) \doteq (\log n)^2$, even if the push-down machine is non-deterministic. As previously noted it is an open question whether the $(\log n)^2$ bound is real. We conjecture that it is, even for deterministic push-down machines.

Other open questions concern similar results for off-line push-down machines; what L(n) function is required on an off-line Turing machine to simulate an off-line push-down machine doing a non-regular recognition. Here the result may depend on the L(n) for the push-down machine.

REFERENCES

1. R.E. Stearns, J. Hartmanis, and P.M. Lewis II, "Hierarchies of Memory Limited Computations." These Proceedings.

2. N. Chomsky, "Three Models for the Description of Language," IRE Transactions on Information Theory, v. IT-2, 1956. pp. 113-124.

3. P.S. Landweber, "Three Theorems on Phrase Structure Grammars of Type I," Information and Control v. 6, 1963. pp. 131-137.

4. S.Y. Kuroda, "Classes of Languages and Linear-Bounded Automata," Information and Control, v. 7, 1964. pp. 207-223.

5. N. Chomsky, "Formal Properties of Grammars," Chapter 12 of Handbook of Mathematical Psychology, Volume II, Edited by R.D. Luce, R.R. Bush and G. Galanter. John Wiley and Sons, Inc., 1963.