Log Space Recognition and Translation of Parenthesis Languages

NANCY LYNCH

University of Southern California, Los Angeles, California

ABSTRACT It is shown how to determine membership in any parenthesis context-free language in log space As an application, the evaluation of Boolean sentences is shown to be log space computable Log space translation of parenthesis languages is similarly shown to be possible, thus log space translators among various representations of Boolean formulas may be constructed

KEY WORDS AND PHRASES log space, parenthesis grammars, parenthesis languages

CR CATEGORIES· 5 23, 5 25, 5 26

1. *Introduction*

The time-complexity classification of the membership problem for general context-free languages [3, 15] and for subclasses of the context-free languages [1] has been carefully examined because of its importance in parsing. Of less practical significance, but of equal theoretical interest, is the corresponding space-complexity classification, both for general context-free languages and for natural subclasses.

It has been shown [3] that the membership problem for every context-free language is solvable with space at most the square of the log of the input length being used. It is an open question whether this bound may be improved to nondeterministic log space. In fact, it is conceivable that the bound might be improved to deterministic log space, but [12] this improvement would imply that unlikely result that *any* problem solvable in nondeterministic log space is also solvable in deterministic log space. A more likely possibility is that the membership problem for every deterministic context-free language is solvable in deterministic log space. Although it is not yet known whether this is so, this problem is reduced to simpler subcases in [13]. For example, it is shown there that there exists a simple precedence language which is "at least as hard as" all deterministic context-free languages; so the above question for deterministic context-free languages is equivalent to the corresponding question for simple precedence languages.

In the present paper attention is restricted to one particular class of languages, namely, those which are log space reducible, in the sense of [14] and [5], to parenthesis context-free languages [6, 10]. It is shown in Section 3 that the membership problem for any parenthesis language is solvable in deterministic log space; thus the membership problem for any language log space reducible to a parenthesis language is also solvable in deterministic log space.

As examples of such languages, we consider in Section 4 the set of all Boolean sentences (in any reasonable notation) which evaluate to 1 (true). It is concluded that the evaluation of Boolean sentences may be done in log space.

A part of this work was done while the author was a visitor at the IBM Thomas J Watson Research Center, Yorktown Heights, N Y.

This work was supported in part by the National Science Foundation under Grant DCR-92373

Author's present address· School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA 30332.

There have been several other recent papers on space classification of various subclasses of the context-free languages. In [11] it is shown that the Dyck languages have their membership problems solvable in log space. In [8] a similar result is proved for the word problem for free groups with finitely many generators. In [9] Theorem 1 of this paper is independently presented; in [2] are several log space *lower* bounds for context-free languages. And in [4] several subclasses of the deterministic context-free languages defined by restrictions on the automaton model are shown to have log space membership algorithms.

It should be noted that the original motivation for the work in the present paper came not from context-free language classification but from previous work by the author and others on log space reducibilities. A reasonable generalization of the log space reducibility of [14] and [5] is the log space truth-table reducibility proposed and studied in [7]. The naturalness of the definition in [7] depends on the ability to evaluate truth-table conditions (i.e. Boolean sentences) in log space. Further discussion of this application of the present work appears in the other paper.

An operation seemingly related to the evaluation of Boolean sentences is the translation of Boolean formulas from infix notation to prefix or postfix notation, and vice versa. We consider in Section 5 the analogous syntax directed translation from a parenthesis language to any context-free language. It is shown that such a translation may be performed in log space. In particular, a "parse" may be obtained in log space for any word in a parenthesis language Also, as a consequence, translation among various representations of Boolean formulas is shown in Section 6 to require no more than log space.

It is known that deterministic context-free languages may be parsed in linear time and (therefore) linear space. The algorithms in this paper for membership and translation all require higher degree polynomial amounts of time for the languages to which they apply. It would be interesting to show that this trade-off is inherent.

Another interesting question arises from consideration of the distinction between the membership question (a language problem) and the determination of a parse for a word (a grammar problem) for context-free languages. Does the log space computability of the former problem necessarily imply the existence of a grammar for which the latter problem is also log space computable?

2. *Notation and Definitions*

A context-free grammar G is written as $(\mathcal{V}, \mathcal{T}, \mathcal{P}, S)$, where $\mathcal{V}$ is the set of variables, $\mathcal{T}$ the set of terminal symbols, $\mathcal{P}$ the set of rules, and S the start variable. $L(G)$ represents the language generated by grammar G.

A *parenthesis (context-free) grammar* [10] is a context-free grammar $(\mathcal{V}, \mathcal{T}, \mathcal{P}, S)$ having two distinguished terminal symbols "(" and ")", with every rule in $\mathcal{P}$ of the form $A \rightarrow (x)$, where $x \in (\mathcal{V} \cup \mathcal{T})^*$ and x contains no occurrences of "(" or ")". A *parenthesis (context-free) language* is a language which is $L(G)$ for some parenthesis grammer G.

$|x|$ represents the length of string x. λ represents the empty string.

A *log space machine* M is a deterministic Turing machine having the following properties: M has three tapes. One is a two-way read-only input tape (with end markers) which may contain symbols from a finite *input alphabet*. The second is a two-way read-write worktape which may contain symbols from a finite *worktape alphabet*. The third is a right-moving write-only output tape, on which may be written symbols from a finite *output alphabet*. If M is started with any input x with the input head at the left, then M eventually halts with at most $\log_2(|x|)$ worktape squares having been visited during the computation; if y is the contents of the output tape when M halts, then M *computes* y *on input* x.

If $\mathcal{A}$, $\mathcal{B}$ are finite alphabets and $f:\mathcal{A}^* \to \mathcal{B}^*$, then f is *log space computable* if there is a log space machine M with input alphabet $\mathcal{A}$ and output alphabet $\mathcal{B}$ such that M computes $f(x)$ on input x for all $x \in \mathcal{A}^*$.

If $A \subseteq \mathcal{A}^*$ for alphabet $\mathcal{A}$, $C_A^{\mathcal{A}}$ is defined by:

$$C_A^{\mathcal{A}}(x) = \begin{cases} 1 & \text{if } x \in A, \\ 0 & \text{if } x \in \mathcal{A}^* - A. \end{cases}$$

A set A is *log space computable* if $C_A^{\mathcal{A}}$ is log space computable for some $\mathcal{A}$.

If $A \subseteq \mathcal{A}^*$, $B \subseteq \mathcal{B}^*$ for alphabets $\mathcal{A}$, $\mathcal{B}$, and if A, B, $\tilde{A}$, $\tilde{B} \neq \varnothing$, then we write $A \leq_m^{\mathcal{L}} B$ (A *is log space many-one reducible to* B) provided there is a log space computable function $f:\mathcal{A}^* \to \mathcal{B}^*$ such that $x \in \mathcal{A}$ iff $f(x) \in B$. It is seen in [14] that if $A \leq_m^{\mathcal{L}} B$ and B is log space computable, then A is log space computable.

3. *Recognition of Parenthesis Languages*

We prove our main result:

THEOREM 1. *All parenthesis languages are log space computable.*

PROOF. Let $G = (\mathcal{V}, \mathcal{T}, \mathcal{P}, S)$ be any parenthesis grammar. We describe the operation of a log space machine M which computes $C_{L(G)}^{\mathcal{T}}$.

Let m be the largest number of variable occurrences on the right of any single rule in $\mathcal{P}$. M's worktape will be divided into several tracks. The first m tracks will be called *storage tracks* and will be used to keep account of partial determinations of parses. The remaining tracks will be used for bookkeeping operations, primarily counting parentheses.

M first performs a preliminary check that its input string x is of the appropriate form. Specifically, M checks that x contains equal numbers of "(" and ")", that x begins with "(" and ends with ")", and that x cannot be written as x_1x_2, x_1, $x_2 \neq \lambda$, where the number of "(" in x_1 is less than or equal to the number of ")" in x_1. If this check is successful, it is then known that each parenthesis has a well-defined matching parenthesis. M then checks that there is no substring of x of the form $(x_1(y_1)x_2(y_2) \cdots (y_{m+1})x_{m+2})$, where $x_1x_2 \cdots x_{m+2}$ contains no "(" or ")", and where the pairs of parentheses surrounding $y_1, \ldots, y_{m+1}$ and the entire expression are all matching pairs. If x is not of the appropriate form, M outputs 0. If x is of the appropriate form, it is meaningful to use certain descriptive terminology in presenting the remainder of the construction:

When the input head rests on a square containing "(", we call that symbol the *current parenthesis*. A *phrase* is a substring of x which begins and ends with matching parentheses. The phrase beginning at the current parenthesis is the *current phrase*. The maximal subphrases of each phrase are its *children*; the minimal phrase containing a proper subphrase of x is the *parent* of that proper subphrase. Children of a common parent are *siblings*. The children of each phrase are ordered according to length, the longest subphrase first in this ordering. If two are of equal length, the leftmost precedes the rightmost.

Symbols will be placed in *columns* consisting of corresponding squares on the m storage tracks. An auxiliary variable *Vars*, which can assume any value from $2^{\mathcal{V}}$ (the set of subsets of $\mathcal{V}$), is kept in M's finite control.

M places its input head on the leftmost symbol of x and proceeds to step 1 below. At the outset, all storage tracks are blank (i.e contain only a designated symbol b).

1 See if the current phrase contains any "(" other than the current parenthesis

1 1 If so, move the input head to the leftmost symbol of the first child of the current phrase Return to 1. (Note "first" refers to the ordering of children established above)

1 2 If not, set *Vars* equal to the set of all variables appearing on the left of productions in $\mathcal{P}$ for which the current phrase is the right side Go to 2

2. See if the current phrase is x.
   2.1. If so, then see if S is in *Vars*.
      2.1.1. If so, M outputs 1 and halts
      2.1.2. If not, M outputs 0 and halts.
   2.2 If not, see if the current phrase is a first but not an only child, a middle child, a last but not an only child, or an only child.
      2.2.1 If the current phrase is a first but not an only child, then record the current value of *Vars* in the topmost square of the first blank column of the storage tracks Move the input head to the leftmost symbol of the next sibling of the current phrase. Return to 1.
      2.2.2 If the current phrase is a middle child, then record the current value of *Vars* in the topmost blank square of the last nonblank column of the storage tracks Move the input head to the leftmost symbol of the next sibling of the current phrase. Return to 1
      2.2.3. If the current phrase is a last but not an only child, then record the current value of *Vars* in the topmost blank square of the last nonblank column of the storage tracks Move the input head to the leftmost symbol of the parent of the current phrase Set *Vars* equal to the set of all variables appearing on the left of productions in $\mathcal{P}$ for which the right side is

$$(x_1V_1x_2V_2 \cdots V_nx_{n+1}), \quad x_i \in \mathcal{T}^*, \ 1 \leq i \leq n+1, \ V_i \in \mathcal{V}, \ 1 \leq i \leq n,$$

      where the current phrase is

$$(x_1(y_1)x_2(y_2) \cdots (y_n)x_{n+1})$$

      (all the pairs of parentheses shown being matching pairs), and for all i, j, if (y_i) is the jth child of the current phrase, then V_i is a member of the set represented on the jth track of the last nonblank column Erase the last nonblank column Return to 2.
      2.2.4 If the current phrase is an only child, record the current value of *Vars* in the topmost square of the first blank column Move the input head to the leftmost symbol of the parent of the current phrase Set *Vars* equal to the set of all variables appearing on the left of productions in $\mathcal{P}$ for which the right side is

$$(x_1Vx_2), \quad x_1, x_2 \in \mathcal{T}^*, \ V \in \mathcal{V},$$

      where the current phrase is

$$(x_1(y)x_2)$$

      (both the pairs of parentheses shown being matching pairs), and V is a member of the set represented in the topmost square of the last nonblank column Erase the last nonblank column Return to 2

It should be clear that M is simply performing a bottom-up parse of x, parsing phrases in order of length. At any time, any column of the storage tracks contains only values of *Vars* arising from parsing sibling phrases. Since no parent has more than m children, the m storage tracks are sufficient.

The reader may verify that all the required bookkeeping operations may be performed in space $\log_2(|x|)$. It remains to show that $\log_2(|x|)$ columns of the storage tracks suffice.

By the construction, each symbol placed on any of the storage tracks is a value of *Vars*, which is the set of variables in $\mathcal{V}$ which generate (in G) a particular proper subphrase of x. Denote by $x_{i,j,t}$ the phrase which corresponds naturally to the symbol in square j of track i after t steps of M's computation on input x. Then after any number t of steps of M's computation, the following are true:

(a) For any j, all $x_{i,j,t}$ which are defined are siblings. Moreover, for any j, $x_{1,j,t}$ (if it is defined) is a first child.

(b) If $x_{1,j+1,t}$ is defined, then $x_{1,j,t}$ is defined and the parent of $x_{1,j+1,t}$ is a (not necessarily proper) subphrase of a sibling of $x_{1,j,t}$.

Proof of (a) and (b) is by induction on t. With (a) and (b), it is straightforward to show that for any j, t, the length of the parent of $x_{1,j+1,t}$ (if $x_{1,j+1,t}$ is defined) is less than half of the length of the parent of $x_{1,j,t}$. Then the number of columns used after any number t of steps is at most $\log_2(|x|)$.

It is easy to see that the time for this algorithm is $O(n^2)$, where n is the length of the input string.

4. *Corollaries: Evaluation of Boolean Sentences*

Let $\Omega = \{\omega_i | 1 \leq i \leq 16\}$. Define an *infix Boolean sentence* as a word in the language of grammar $(\{S\}, \{0, 1, (,)\} \cup \Omega, \{S \to S\omega S, S \to (S\omega S) | \omega \in \Omega\} \cup \{S \to 0, S \to 1\}, S)$. Each $\omega \in \Omega$ represents one of the 16 binary Boolean functions. We assume that if $i < j$ then operation ω_i has the same or greater precedence than ω_j. Among operation symbols of equal precedence, association is from the left. $val(x)$, for any infix Boolean sentence x, represents the value of x according to the interpretations of the elements of Ω and the given precedence rules.

COROLLARY 1. *There exists a log space machine M with input alphabet* $\{0, 1, (,)\} \cup \Omega$ *which computes* $val(x)$ *for any infix Boolean sentence input* x.

PROOF. Let $A = \{\text{infix Boolean sentences } x | val(x) = 1\}$. Let B be the parenthesis language generated by the grammar $(\{S, Z\}, \{0, 1, (,)\} \cup \Omega, \mathscr{P}, S)$, where $\mathscr{P}$ consists of the following rules (for all $\omega \in \Omega$):

$$
\begin{array}{llll}
S \to (S\omega S) & \text{if } val(1\omega 1) = 1, & S \to (S\omega Z) & \text{if } val(1\omega 0) = 1, \\
S \to (Z\omega S) & \text{if } val(0\omega 1) = 1, & S \to (Z\omega Z) & \text{if } val(0\omega 0) = 1, \\
Z \to (S\omega S) & \text{if } val(1\omega 1) = 0, & Z \to (S\omega Z) & \text{if } val(1\omega 0) = 0, \\
Z \to (Z\omega S) & \text{if } val(0\omega 1) = 0, & Z \to (Z\omega Z) & \text{if } val(0\omega 0) = 0,
\end{array}
$$

$S \to (1)$, and $Z \to (0)$.

That is, B consists of all the infix Boolean sentences x having $val(x) = 1$ with a full parenthesis structure superimposed.

It suffices to show $A \leq_m^{\mathscr{L}} B$. But this is simply the statement that there exists a log space machine which fully parenthesizes an infix Boolean sentence, which the reader may verify. □

Similarily, define a *prefix Boolean sentence* as a word in the language of grammar $(\{S\}, \{0, 1\} \cup \Omega, \{S \to \omega SS | \omega \in \Omega\} \cup \{S \to 0, S \to 1\}, S)$. $val(x)$, for any prefix Boolean sentence x, represents the value of x according to the usual rules.

COROLLARY 2. *There exists a log space machine M with input alphabet* $\{0, 1\} \cup \Omega$ *which computes* $val(x)$ *for any prefix Boolean sentence input* x.

PROOF. The proof is very similar to that of Corollary 1 and is left to the reader. □

Clearly, a result similar to Corollaries 1 and 2 is true for postfix representation of Boolean sentences.

Note that the apparent time requirement for evaluation of (infix, prefix, or postfix) Boolean sentences according to the given algorithms is $O(n^4)$. This is because parenthesizing any type of Boolean sentence seems to require time $O(n^2)$, as does the evaluation algorithm. Straightforward composition of the given log space algorithms (as in [14]) requires time proportional to the product of the times of the component algorithms.

5. *Translation of Parenthesis Languages*

A syntax-directed translation schema (SDTS) [1] is a 5-tuple $T = (\mathscr{V}, \mathscr{T}, \mathscr{T}', \mathscr{P}, S)$, where

(1) $\mathscr{V}$ is a finite set of variables,

(2) $\mathscr{T}$ and $\mathscr{T}'$ are finite *input* and *output alphabets*, respectively,

(3) $\mathscr{P}$ is a finite set of rules of the form $A \to x, x'$, where $x \in (\mathscr{V} \cup \mathscr{T})^*$, $x' \in (\mathscr{V} \cup \mathscr{T}')^*$, and the variables of x' are a permutation of the variables of x. To each occurrence of a variable x is *associated* an occurrence of an identical variable in x', in a one-to-one manner, and

(4) $S \in \mathscr{V}$ is the start symbol.

We define a *translation form* of T:

(1) (S, S) is a translation form, and the two S's are associated, and

(2) if $(wAx, w'Ax')$ is a translation form in which the two instances of A are associated, and if $A \to y, y'$ is a rule in $\mathscr{P}$, then $(wyx, w'y'x')$ is a translation form of T. The variables in the new translation form have the natural association deriving from the

association in the previous form and in the rule.

We write in the preceding situation $(wAx, w'Ax') \underset{T}{\Rightarrow} (wyx, w'y'x')$; $\overset{+}{\underset{T}{\Rightarrow}}$ is the transitive closure of $\underset{T}{\Rightarrow}$. The *translation* $\tau(T)$ defined by an *SDTS* T is $\{(x, x') \mid (S, S) \overset{+}{\underset{T}{\Rightarrow}} (x, x'), x \in \mathcal{T}^*, x' \in (\mathcal{T}')^*\}$. The domain of the translation *dom* $\tau(T)$ is $\{x \mid (\exists x')[(x, x') \in \tau(T)]\}$.

A *parenthesis (syntax-directed translation) schema* is an SDTS $(\mathcal{V}, \mathcal{T}, \mathcal{T}', \mathcal{P}, S)$ in which $\mathcal{T}$ contains two distinguished symbols, "(" and ")", and each rule of $\mathcal{P}$ is of the form $A \rightarrow (x), x'$, where $A \in \mathcal{V}$, $x \in (\mathcal{V} \cup \mathcal{T}')^*$, and x and x' contain no occurrences of "(" or ")".

THEOREM 2. *For any parenthesis schema $T = (\mathcal{V}, \mathcal{T}, \mathcal{T}', \mathcal{P}, S)$, there exists a log space machine M with input alphabet $\mathcal{T}$ such that on any input $x \in$ dom $\tau(T)$, M computes some value x' with $(x, x') \in \tau(T)$.*

PROOF. We may assume that M's input x is in *dom* $\tau(T)$. The terminology *current parenthesis*, *phrase*, *current phrase*, *child*, *parent*, and *sibling* is used as in Theorem 1. However, the ordering of children established in Theorem 1 will not be used in the description of M.

We select a derivation of x by ordering the rules of T in some fixed way. Every phrase of x then has a *corresponding rule*, determined as follows:

(1) The rule corresponding to x is the first rule $S \rightarrow y, y'$, such that $(\exists x')[(y, y') \overset{+}{\underset{T}{\Rightarrow}} (x, x')]$, and

(2) if z is the kth child (from the left) of phrase y, and $A \rightarrow vBw, v'Bw'$ is the rule corresponding to y (where B is the kth variable of vBw), then the rule corresponding to z is the first rule $B \rightarrow u, u'$ such that $(\exists y')[(u, u') \overset{*}{\underset{T}{\Rightarrow}} (y, y')]$.

M will require the following subroutine N, which starts with the input head on a "(" and determines the rule corresponding to the current phrase. A variable *Rule*, which can assume as its value any rule in $\mathcal{P}$, is kept in the finite control.

Subroutine N: Record the current input head position on track 1 and the position of the left end of the input on track 2

1 By simulating the machine of Theorem 1, determine the rule corresponding to the phrase beginning at the position recorded on track 2 (If *Rule* has a previously defined value, this value will be used here.) Set *Rule* = the newly determined rule. See if the head positions recorded on tracks 1 and 2 are identical.
  1.1. If not, then determine the position of the leftmost symbol of that child of the phrase beginning at the position now recorded on track 2, which contains the phrase beginning at the position now recorded on track 1
  Record this newly determined position on track 2
  Return to 1
  1.2. If so then the current value of *Rule* is the needed value.

Subroutine N simply follows the inductive definition of the corresponding rule; it obtains the rule corresponding to the current phrase by first obtaining the rule corresponding to the entire input and then doing the same for successive children which contain the current phrase.

We now describe the main construction of M. M begins at the leftmost symbol of x, at stage 1 below..

1 Call subroutine N to determine the rule $A \rightarrow w, w'$ corresponding to the current phrase See if w contains any variables
  1.1. If so, output all symbols of w' up to and not including the first variable If the kth variable (from the left) of w is associated with the first variable of w', move the input head to the first symbol of the kth child (from the left) of the current phrase
  Return to 1
  1.2 If not, then output w'
  Go to 2.
2 See if the current phrase is x
  2 1. If so, halt.
  2 2. If not, then call subroutine N to determine the rule, $A \rightarrow w, w'$, corresponding to the parent of the current phrase If the current phrase is the kth child, and if the lth variable of w' is associated with the kth variable of w, then see if the lth variable of w' is the last variable of w'

2 2 1 If not, then output the terminal symbols of w' between its lth and $(l + 1)$-th variables. If the mth variable of w is associated with the $(l + 1)$-th variable of w', move the input head to the first symbol of the mth child of the parent of the current phrase.
Return to 1

2 2 2 If so, output the terminal symbols of w' following the last variable Move the input head to the first symbol of the parent of the current phrase
Return to 2.

Verification of the correctness of the construction is left to the reader; it is obvious that all steps can be done in log space.

It is easy to see that the time for this algorithm is $O(n^3)$.

With appropriate definitions, Theorem 2 implies that any word in a parenthesis language can be assigned a parse (for a particular parenthesis grammar) in log space.

6. *Corollaries: Translation of Boolean Formulas*

Define an infix Boolean formula as a word in the language of grammar $(\{S, V\}, \{0, 1, (,)\} \cup \Omega, \{S \to S\omega S, S \to (S\omega S) | \omega \in \Omega\} \cup \{S \to V, V \to V1, V \to V0, V \to \lambda\}, S)$. Here V generates binary strings which represent Boolean variables. Define a *prefix. Boolean formula* as a word in the language of grammar $(\{S, V\}, \{0, 1, \#\} \cup \Omega, \{S \to \omega SS | \omega \in \Omega\} \cup \{S \to V, V \to V1, V \to V0, V \to \#\}, S)$. Two Boolean formulas are *equivalent* if they have the same values under the interpretation of the elements of Ω, the given precedence rules, and all values of the variables.

COROLLARY 3. *There is a log space machine M with the input alphabet $\{0, 1, (,)\} \cup \Omega$ which computes, for any infix Boolean formula, an equivalent prefix Boolean formula.*

PROOF. Consider the parenthesis schema $T = (\{S, V\}, \{0, 1, (,)\} \cup \Omega, \{0, 1, \#\} \cup \Omega, \mathscr{P}, S)$, where $\mathscr{P}$ consists of the following rules (for all $\omega \in \Omega$):

$$S \to (S\omega S), \omega SS, \quad S \to (V), V, \quad V \to (V1), V1, \quad V \to (V0), V0, \quad \text{and } V \to (\ ), \#.$$

M simulates the composition of two log space machines M' and M''. M', on input an infix Boolean formula x, fully parenthesizes x. M'' produces, from a fully parenthesized infix Boolean formula, an equivalent prefix Boolean formula (by Theorem 2). □

COROLLARY 4. *There is a log space machine M with input alphabet $\{0, 1, \#\} \cup \Omega$ which computes, for any prefix Boolean formula, an equivalent infix Boolean formula.*

PROOF. Left to the reader.

Similar results hold for the postfix representation of Boolean formulas. The time performance in each case is $O(n^5)$.

ACKNOWLEDGMENTS. This paper owes much to the suggestions of Albert Meyer and to discussions with Ron Rivest and Richard Lipton.

REFERENCES

1 AHO, A , AND ULLMAN, J *The Theory of Parsing, Translation and Compiling, Vol. I.* Prentice-Hall, Englewood Cliffs, N J , 1973

2 ALT, H , AND MEHLHORN, K Lower bounds for the space requirement of some families of context-free languages Tech Rep , U des Saarlandes, Saarbrucken, West Germany, 1975.

3 HOPCROFT, J , AND ULLMAN, J *Formal Languages and Their Relation to Automata* Addison-Wesley, Reading, Mass , 1969.

4 IGARASHI, Y Tape bounds for some subclasses of deterministic context-free languages Tech Rep No 80, Centre for Computer Studies, U of Leeds, Leeds, England, 1976.

5 JONES, N D , AND LAASER, W T Complete problems for deterministic polynomial time. Proc Sixth Annual ACM Symp on Theory of Computing, 1974, pp 40-46

6 KNUTH, D A characterization of parenthesis languages *Inform. and Contr 11* (1967), 269-289.

7 LADNER, R , AND LYNCH, N Relativization of questions about log space computability In *Math Syst. Theory 10* (1976), 19-32

8. LIPTON, R., AND ZALCSTEIN, Y Word problems solvable in logspace. *J. ACM 24,* 3 (1977), 522-526

9 MEHLHORN, K Bracket-languages are recognizable in logarithmic space Tech. Rep., U. des Saarlandes, Saarbrucken, West Germany, Sept. 1975

10. MCNAUGHTON, R Parenthesis grammars. *J. ACM 14,* 3 (July 1967), 490-500.

11. RITCHIE, R.W., AND SPRINGSTEEL, T.M. Language recognition by marking automata. *Inform. and Contr. 20,* 4 (May 1972), 313-330.
12. SUDBOROUGH, I.H. On tape-bounded complexity classes and multi-head finite automata Proc 14th Annual IEEE Symp on Switching and Automata Theory, 1973, pp. 138-144.
13. SUDBOROUGH, I.H. On deterministic context-free languages, multihead automata and the power of an auxiliary pushdown store Proc. Eighth Annual ACM Symp on Theory of Comptng, 1976, pp. 141-148.
14. STOCKMEYER, L., AND MEYER, A Word problems requiring exponential time: Preliminary report Proc. Fifth Annual ACM Symp on Theory of Comptng , 1973, pp. 1-9
15 VALIANT, L General context-free recognition in less than cubic time. *J. Comptr. Syst Sci. 10,* 2 (1975), 308-315.

RECEIVED AUGUST 1975; REVISED OCTOBER 1976